VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

INTEGRACE JMS POSKYTOVATELŮ TŘETÍCH STRAN

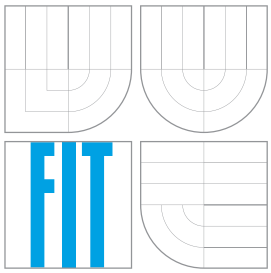
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. NIKOLETA ŽIAKOVÁ

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

INTEGRACE JMS POSKYTOVATELŮ TŘETÍCH STRAN

INTEGRATION OF 3RD PARTY JMS PROVIDERS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. NIKOLETA ŽIAKOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VENDULA HRUBÁ

BRNO 2013

Abstrakt

Tato práce se zabývá komunikací pomocí zasílání zpráv a integrací JMS poskytovatelů třetích stran do aplikačního serveru JBoss. Cílem práce bylo navrhnout a implementovat univerzální testovací sadu, která navrženou integraci jednoduše otestuje. Nejprve bylo potřebné nastudovat klíčové technologie důležité pro integraci, ke kterým patří standard Java Message Service, Java EE Connector Architecture, vybraní JMS poskytovatelé a JBoss AS. Konkrétní postupy integrace byly navrženy a popsány pro JMS poskytovatele Apache ActiveMQ, IBM WebSphere MQ a Red Hat MRG Messaging. Pro ověření funkčnosti integrace byla navržena a implementována testovací sada zaměřená na čtyři oblasti – transakce, clustering, vysokou dostupnost a výkonnost. Testování bylo automatizováno pomocí nástroje průběžné integrace Jenkins. Na závěr byla testovací aplikace použita na vyhodnocení funkčnosti integrace a porovnání jednotlivých JMS poskytovatelů a různých verzí JBoss AS.

Abstract

This thesis deals with messaging and integration of third-party JMS providers into JBoss application server. The aim of the thesis was to design and implement a general-purpose testsuite to verify the proposed integration. First requirement was to get familiar with key technologies for integration including Java Message Service, Java EE Connector Architecture, selected JMS providers and JBoss AS. Specific procedures of integration were designed and described for JMS providers Apache ActiveMQ, IBM WebSphere MQ and Red Hat MRG Messaging. The testsuite implemented to verify the functionlity of the integration focuses on four areas – transactions, clustering, high availibility and performance. The process of testing was automated using continuous integration tool Jenkins. Finally the test application was used to evaluate functionality of the integration and compare different JMS providers and various versions of JBoss AS.

Klíčová slova

zasílání zpráv, JMS, JCA, adaptér zdrojů, WebSphere MQ, MRG Messaging, ActiveMQ, JBoss AS, Maven, testovací sada, transakční testy, clusterové testy, testy vysoké dostupnosti, výkonnostní testy, MDB, Servlet

Keywords

messaging, JMS, JCA, resource adapter, WebSphere MQ, MRG Messaging, ActiveMQ, JBoss AS, Maven, testsuite, transaction tests, clustering tests, high availibility tests, performance tests, MDB, Servlet

Citace

Nikoleta Žiaková: Integrace JMS poskytovatelů třetích stran, diplomová práce, Brno, FIT VUT v Brně, 2013

Integrace JMS poskytovatelů třetích stran

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně pod vedením Ing. Venduly Hrubé.

.....
Nikoleta Žiaková
16. mája 2013

Poděkování

Ráda bych poděkovala vedoucí této práce Vendule Hrubé a technickému vedoucímu Martinovi Večeřovi za jejich čas, cenné rady a podporu, které mi při řešení diplomové práce poskytli.

Obsah

Kapitola 1

Úvod

Stále zvyšujúca sa distribuovanosť a rozsah softwarových systémov kladie rovnako zvyšujúce sa nároky na ich komunikačnú infraštruktúru, ktorá zabezpečuje komunikáciu medzi aplikáciami a prenos dát z jedného systému do iného systému. Komunikačné systémy majú vysoké požiadavky na spoľahlivosť, výkonnosť, škálovateľnosť a bezpečnosť. Komunikujúce strany sú navyše často heterogénne – využívajú rôzne protokoly, bežia na rôznych operačných systémoch a rôznych hardwarových platformách, používajú rôzne formáty dát apod. Klasická komunikácia medzi aplikáciami v podobe vzdialeného volania procedúr alebo komunikácia na úrovni operačného systému nedokáže uspokojiť všetky kľúčové požiadavky. Preto sa vyvinul spôsob komunikácie pomocou zasielania správ, ktorého hlavnými výhodami sú asynchrónnosť a oddelenie komunikujúcich strán. Odosielateľ správy nemusí čakať na jej doručenie či odpoveď, jednoducho pokračuje vo svojej práci.

Existuje veľké množstvo systémov pre zasielanie správ. Táto práca sa venuje aplikačnému serveru JBoss, ktorý ponúka dva systémy – JBoss Messaging a HornetQ. Mnohí zákazníci nahrádzajú drahé komerčné aplikačné servery open source aplikačným serverom JBoss. Napriek tomu si potrebujú ponechať existujúcu infraštruktúru. K tejto infraštruktúre patrí okrem iného systém komunikácie pomocou správ a z toho dôvodu je potrebná jeho integrácia do aplikačného servera JBoss. Táto práca sa zaoberá vlastnou integráciou JMS poskytovateľov a overením jej funkčnosti pomocou univerzálnej testovacej sady. Hlavným cieľom práce je popísať postup integrácie JMS poskytovateľov tretích strán do aplikačného servera JBoss, navrhnúť vhodnú testovaciu sadu a pomocou nej integráciu overiť.

Text práce je rozdelený do šiestich kapitol. Druhá kapitola pripravuje teoretický základ práce, popisuje technológie potrebné pre pochopenie problematiky. Vysvetľuje pojmy ako komunikácia pomocou zasielania správ, štandard Java Message Service a Java EE Connector Architecture. Ďalej obsahuje popis vybraných JMS poskytovateľov tretích strán aj JMS poskytovateľov od JBoss, popis ich kľúčových vlastností a stručné predstavenie aplikačného servera JBoss.

Tretia kapitola je zameraná na integráciu JMS poskytovateľov tretích strán do aplikačného servera JBoss a základné overenie jej funkčnosti. JMS poskytovateľmi vybranými pre integráciu sú komerčný IBM WebSphere MQ a dva open source poskytovatelia Apache ActiveMQ a Red Hat MRG Messaging.

Štvrtá kapitola je venovaná testovacej sade. Je tu uvedený návrh testovacej sady s podrobným popisom jednotlivých testovacích scenárov, ale aj vlastná implementácia. Súčasťou popisu implementácie je postup prenosu testovacej aplikácie z JBoss AS 5 na JBoss AS 7. Táto kapitola vysvetľuje rôzne možnosti spustenia testov a postup automatizácie procesu testovania pomocou nástroja priebežnej integrácie Jenkins.

Piata kapitola obsahuje výsledky navrhnutých testov pre jednotlivých JMS poskytovateľov a zvolené verzie JBoss AS a popisuje postup testovania, ktorým boli dané výsledky získané. Výsledky testov nie sú uvedené len pre JMS poskytovateľov tretích strán, ale aj pre JBoss Messaging a HornetQ, ktoré sú dodávané s JBoss AS. Na základe výsledkov testov sú porovnaní jednotliví JMS poskytovatelia a rôzne verzie JBoss AS.

Posledná kapitola obsahuje zhrnutie dosiahnutých výsledkov a predstavuje niekoľko možností ďalšieho pokračovania tejto práce. Nakoniec je uvedený zoznam použitých skratiek a obsah priloženého CD.

Kapitola 2

Použité technológie

Táto kapitola obsahuje prehľad technológií, ktoré sú v práci využité. Prvá podkapitola je venovaná základným pojmom ako komunikácia pomocou zasielania správ a štandard Java Message Service, na ktorých je táto práca postavená. Ďalšia podkapitola sa zaoberá štandardom Java EE Connector Architecture, ktorý hrá hlavnú úlohu pri integrácii JMS poskytovateľov do aplikačného servera. V tretej podkapitole sú predstavení vybraní JMS poskytovatelia a ich základné vlastnosti. Sú tu uvedení JMS poskytovatelia od JBoss aj poskytovatelia tretích strán. Nasleduje časť o aplikačnom serveri JBoss a porovnanie jeho dvoch verzií, ktoré sú v práci použité. Nakoniec je venovaná krátka podkapitola aplikácii Maven, ktorá je použitá na automatizovanú správu testovacej aplikácie.

2.1 Štandard Java Message Service

Java Message Service (JMS) [13, 10] je rozhranie pre programovanie aplikácií v programovacom jazyku Java (Java Application Programming Interface, Java API). JMS je súčasťou špecifikácie Java Platform, Enterprise Edition (Java EE) [12] od firmy Sun a patrí do skupiny *Message Oriented Middleware (MOM)* [6], ktorý umožňuje aplikáciám komunikovať zasielaním správ. Programy môžu používať JMS rozhranie bez ohľadu na konkrétnu implementáciu jednotlivých JMS poskytovateľov. Oproti napríklad elektronickej pošte, ktorá slúži na komunikáciu človeka s aplikáciou alebo medzi ľuďmi navzájom, JMS slúži na komunikáciu medzi aplikáciami alebo ich komponentmi. V tejto podkapitole sú vysvetlené základné princípy komunikácie pomocou správ, načrtnutá štruktúra JMS API, popísané súčasti JMS štandardu a vysvetlené úlohy JMS v Java EE.

2.1.1 Komunikácia pomocou zasielania správ

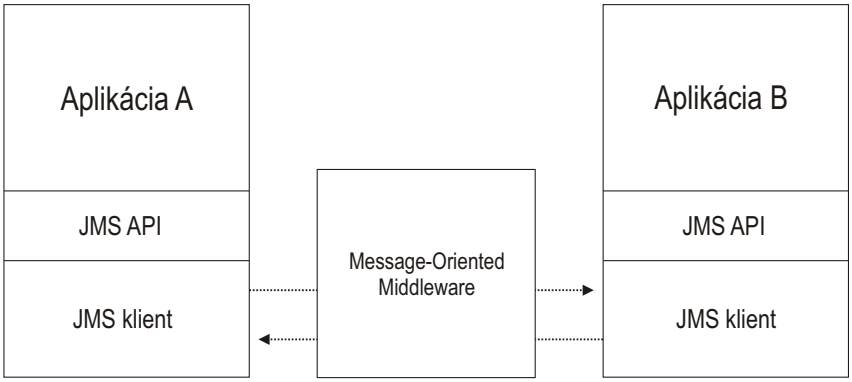
Zasielanie správ [13] slúži na asynchrónne posielanie dát z jedného systému do druhého cez počítačovú sieť. Asynchrónnosť komunikácie a teda aj asynchrónnosť zasielania správ umožňuje odosielateľovi po odoslaní správy pokračovať v práci bez toho, aby musel čakať na jej doručenie alebo odpoveď. Hlavnou oblasťou využitia tohto druhu komunikácie je prepojenie heterogénnych systémov.

Každá správa je samostatnou jednotkou, ktorá obsahuje všetky potrebné informácie k jej spracovaniu. JMS API poskytuje metódy na jej vytvorenie, odoslanie odosielateľom a prijatie na strane jedného alebo viacerých adresátov. Komunikácia pomocou zasielania správ využíva určité architektúry a modely, ktoré budú následne popísané.

Architektúra

Systémy komunikácie pomocou správ sa skladajú z klientov a servera. Klienti posielajú správy serveru, ktorý správy ďalej posiela iným klientom. Architektúru systému zasielania správ ukazuje Obrázok 2.1. JMS klient je aplikácia, ktorá používa JMS API na odosielanie a prijímanie správ. Klient, ktorý správu odosiela, sa nazýva *producent (producer)* a klient, ktorý správu prijíma sa nazýva *konzument (consumer)*. Aplikácia môže byť zároveň producentom aj konzumentom.

Server tvorí prostredníka medzi jednotlivými klientmi. Existujú rôzne typy architektúry systémov zasielania správ, ktoré sa líšia použitím servera a spôsobom smerovania správ – *centralizovaná*, *decentralizovaná* a *hybridná architektúra*.



Obrázok 2.1: Architektúra JMS.

Systémy s *centralizovanou architektúrou* využívajú server správ na smerovanie a doručovanie správ klientom. Server tak oddeľuje odosielajúceho klienta od prijímajúcich, všetci klienti komunikujú priamo len so serverom a o ostatných klientoch nemusia vedieť. To umožňuje pridávanie a odoberanie klientov do systému bez narušenia systému ako celku. V najjednoduchšom prípade systém obsahuje jeden centrálny server, ku ktorému je pripojený každý klient. Takáto topológia umožňuje komunikáciu klienta s ľubovoľným iným klientom pri najmenšom možnom objeme prepojovacej siete. V praxi sa často používa skupina distribuovaných serverov, ktoré pracujú ako jedna logická jednotka – *výpočtový cluster*.

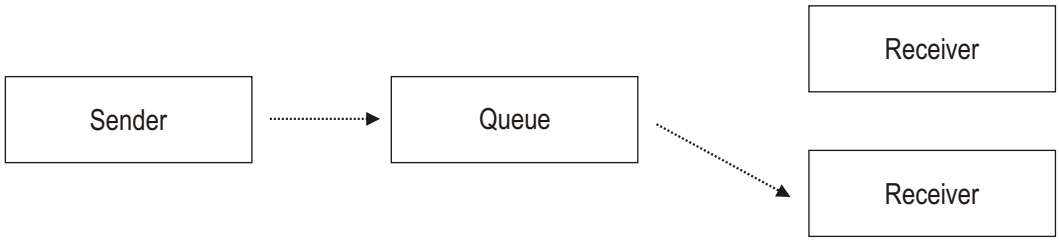
Decentralizovaná architektúra nepotrebuje server na smerovanie správ, ale využíva Internet Protocol (IP) multicast [11] na sieťovej vrstve, ktorá to robí automaticky. Aplikácie sa môžu pripojiť do jednej alebo viacerých multicastových skupín, z ktorých každá používa vlastnú IP adresu. Odosielajúca aplikácia posiela správy na konkrétnu multicastovú IP adresu a klientom, ktorí sú pripojení do danej skupiny, bude správa doručená. Pojmom server sa v tomto prípade označuje funkcionalita servera, ktorá musí byť lokálnou súčasťou klientov, napríklad podpora transakcií, perzistencie alebo bezpečnosti.

Hybridná architektúra sa používa na komunikáciu medzi centralizovanými a decentralizovanými systémami. Kým decentralizované systémy používajú protokol IP, centralizované systémy používajú protokol Transmission Control Protocol (TCP) [11]. V hybridnej architektúre existuje proces – démon, ku ktorému sa klienti pripájajú pomocou TCP a ten následne komunikuje s ostatnými démonmi pomocou IP protokolu.

Modely komunikácie

JMS poskytuje dva modely komunikácie, ktoré sa líšia tým, či je správa určená pre jedného alebo viacerých príjemcov. V prípade, keď producent posiela správu práve jednému konzumentovi (komunikácia jedného s jedným), sa v terminológii JMS hovorí o *point-to-point (p2p)* modeli. V druhom modeli *publish-and-subscribe (pub/sub)* môže byť správa určená jednému alebo viacerým konzumentom (komunikácia jedného s mnohými).

V point-to-point modeli sa producentovi hovorí *odosielateľ (sender)* a konzumentovi *prijímateľ (receiver)*. Komunikácia prebieha cez virtuálny kanál – *frontu (queue)*. Schéma tohto modelu je na Obrázku 2.2. Odosielateľ vytvorí správu a pošle ju do fronty. Na správy v jednej fronte môže čakať niekoľko potenciálnych príjemcov, avšak len jednému z nich je možné správu skutočne doručiť. Point-to-point komunikácia zaručuje doručenie každej správy práve jednému príjemcovi. Príjemca nedostáva správy z fronty automaticky, ale musí o ne požiadať (*pull model*). Správy sú vo fronte zoradené a odosielané príjemcovi v poradí, v akom boli do fronty umiestnené, poradie správ na odoslanie príjemcovi je možné zmeniť len nastavením priority. Prijatie správy z fronty prijímateľom spôsobí odstránenie prečítanej správy z fronty.



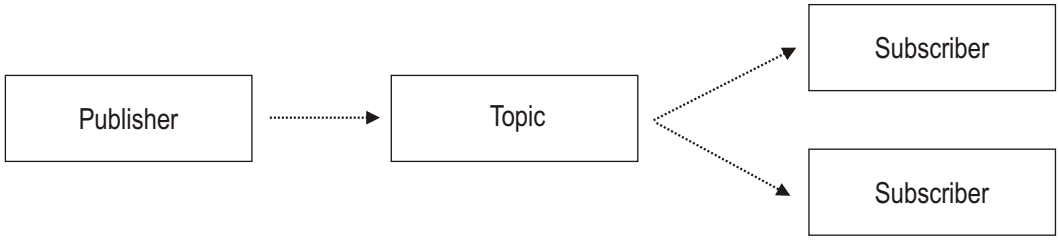
Obrázok 2.2: Point-to-point model komunikácie.

Pri tomto spôsobe komunikácie sú komunikujúce strany viac viazané ako v publish-and-subscribe modeli, keďže odosielateľ zvyčajne vie, komu a na aký účel je správa určená. Preto je možné tento model použiť aj synchrónne a komunikovať spôsobom *požiadavka/odpoveď (request/reply)*. V takomto prípade odosielateľ pošle správu do vstupnej fronty a potom čaká na odpoveď na výstupnej fronte.

Point-to-point model umožňuje aj ďalšie funkcie, ktoré nedefinuje štandard JMS, ale mnohí JMS poskytovatelia ich implementujú. Medzi ne patrí napríklad schopnosť rozloženia záťaže alebo prehliadač fronty. *Rozloženie záťaže (load balancing)* znamená, že na jednej fronte čakajú viacerí prijímatelia, ktorí sa v prijímaní striedajú, a tým rozdeľujú záťaž spojenú so spracovaním správy. Spôsob, akým sa konzumenti striedajú, závisí od daného JMS poskytovateľa. Pri pripojení viacerých konzumentov na jednu frontu však nemôže byť zaručené absolútne poradie prijatia správ kvôli rôznej rýchlosti jednotlivých konzumentov. Ak zlyhá potvrdenie správy, môže byť daná správa znovu doručená inému konzumentovi, akému bola pôvodne. V prípade, že konzument medzitým prijal správu, ktorá bola odoslaná neskôr, sa znovudoručená správa dostane mimo svoje pôvodné poradie. Vždy ale musí byť zaručené relatívne poradie správ. To znamená, že správa, ktorá sa vo fronte nachádza dlhšie, bude konzumentom odobraná skôr. *Prehliadač fronty (queue browser)* je nástroj na prehliadanie fronty pred prijatím správy bez toho, aby bola správa odobraná z fronty. Slúži na monitorovanie obsahu fronty, jej veľkosti, prípadne vyhľadanie konkrétnej správy

vo fronte. Prehliadač fronty obsahuje kópie správ, ktoré sa nachádzajú vo fronte v okamihu jeho vzniku. Zmeny, ktoré vo fronte nastanú v čase medzi vznikom prehliadača a jeho použitím, sa neprejavia.

Obrázok 2.3 zobrazuje druhý model komunikácie pomocou správ – publish-and-subscribe. V tomto modeli sa producent volá *vydavateľ (publisher)* a konzument *predplatiteľ (subscriber)*. Producent publikuje správy do virtuálneho kanála, ktorému sa hovorí *topik (topic)*. Všetci konzumenti, ktorí sa prihlásia k danému topiku dostanú kópiu každej správy, ktorá je doňho poslaná. Tu sa využíva mechanizmus, kedy konzument nemusí žiadať o každú správu z topiku, ale správy sú mu po "predplatení"doručované automaticky (*push model*).



Obrázok 2.3: Publish-and-subscribe model komunikácie.

Pri komunikácii jedného s mnohými odosielajúci systém väčšinou nepozná konkrétnych adresátov správy, nezaujíma ho, ako bude správa spracovaná alebo koľko konzumentov správu dostane. Takáto forma komunikácie má uplatnenie v prípade, kedy je potrebné zaslať informáciu o nejakej udalosti viacerým konzumentom.

V tomto modeli existuje niekoľko rôznych typov predplatiteľov, kú ktorým patria:

- *Nestály (non-durable)* – správy sú mu doručované len vtedy, keď je aktívne pripojený k danému topiku.
- *Stály (durable)* – správy dostane aj v prípade, keď nie je pripojený, a teda dostáva kópiu každej správy v danom topiku. To znamená, že ak v čase publikovania správy nie je stály predplatiteľ aktívny, správa je uložená a doručená po jeho pripojení k topiku.
- *Spravovaný (administered durable)* – je statický konzument, ktorý je známy JMS poskytovateľovi. Vytvára sa pomocou konfiguračného súboru alebo administračného nástroja, ktorý daný poskytovateľ poskytuje.
- *Dynamický stály (dynamic durable)* – je definovaný za behu programu, správy sú mu z topiku zasielané, pokiaľ sám nezruší ich prijímanie.

Pri výbere konkrétneho typu predplatiteľa je potrebné zvážiť požiadavky pre danú aplikáciu. Napríklad, či je nevyhnutné prijímať všetky správy, koľko úložného priestoru pre správy je k dispozícii alebo dĺžku platnosti zasielaných správ. V prípade, kedy je stály predplatiteľ po dlhú dobu nečinný, môžu ukladané správy zbytočne zaberať priestor v JMS úložisku.

2.1.2 Java Message Service API

Aktuálne existujú dve verzie JMS API, prvá verzia JMS 1.0.2b[1] a druhá verzia JMS 1.1[2]. V tejto práci je používaná najnovšia verzia JMS 1.1. JMS API sa skladá z troch základných častí, ktoré sa navzájom líšia svojím použitím:

- *všeobecné API* – posielanie a prijímanie správ cez fronty aj topiky,
- *point-to-point API* – posielanie a prijímanie správ výhradne použitím front,
- *publish-and-subscribe API* – posielanie a prijímanie správ výhradne použitím topikov.

Každé z týchto API sa ďalej delí, každé na sedem častí. Vzájomné vzťahy medzi týmito časťami JMS API pre všeobecné API sú zobrazené na Obrázku 2.4. Objekty `ConnectionFactory` a `Destination` musia byť podľa JMS špecifikácie získané od JMS poskytovateľa. Na to je možné použiť adresárovú službu Java Naming and Directory Interface (JNDI) [10], vďaka ktorej odosielateľ nemusí vedieť, kde je skutočné umiestnenie cieľa správy v sieti. JMS klient pracuje len s identifikátorom, ktorý získa z JNDI. Pre vytvorenie ostatných objektov existujú metódy v jednotlivých častiach – z `ConnectionFactory` je vytvorené spojenie, z ktorého je potom vytvorené sedenie a zo sedenia sú napokon vytvorení producenti, konzumenti a samotné správy. Vzťahy medzi jednotlivými časťami v point-to-point API a publish-and-subscribe API sú analogické k tým vo všeobecnom API, ale špecializované na konkrétny model komunikácie (viď Tabuľka 2.1).

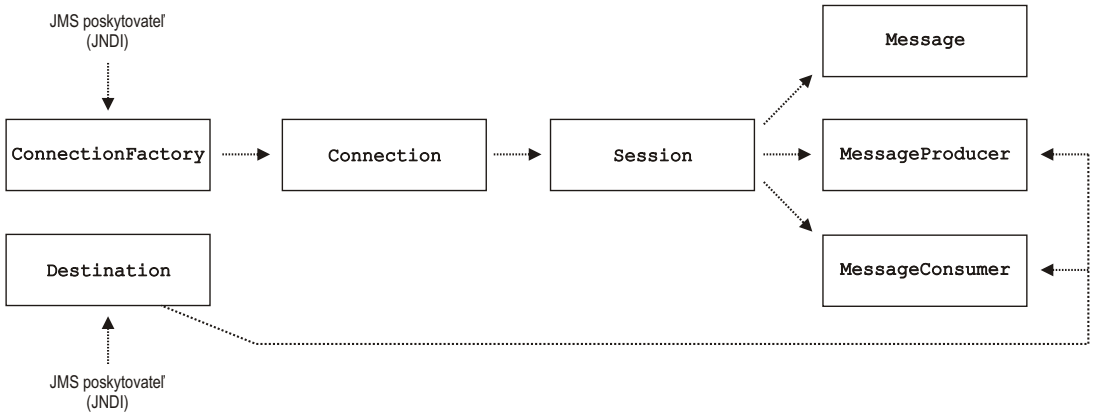
všeobecné API	point-to-point API	publish-and-subscribe API
`ConnectionFactory`	`QueueConnectionFactory`	`TopicConnectionFactory`
`Destination`	`Queue`	`Topic`
`Connection`	`QueueConnection`	`TopicConnection`
`Session`	`QueueSession`	`TopicSession`
`Message`	`Message`	`Message`
`MessageProducer`	`QueueSender`	`TopicPublisher`
`MessageConsumer`	`QueueReceiver`	`TopicSubscriber`

Tabuľka 2.1: Tabuľka JMS API

`ConnectionFactory` je objekt, ktorý používa klient na pripojenie k serveru. Parametre pre konfiguráciu tohto spojenia definuje administrátor. `Destination` určuje adresu virtuálneho kanála typu fronta alebo topik pre posielanie a prijímanie správ. Syntax adries nie je definovaná štandardom, ale je závislá od JMS poskytovateľa. `Connection` reprezentuje fyzické spojenie s JMS poskytovateľom. Vytvára sa pomocou metódy z rozhrania `ConnectionFactory`. Spojenie je pomerne náročné na zdroje, preto je vhodné vytvárať pre každý objekt `ConnectionFactory` len jedno. `Session` (sedenie) je jednovláknový kontext, ktorý vytvára producentov, konzumentov a správy. Sedenie spravuje transakcie počas odosielania a prijímania správ, sleduje potvrdenie o doručení správy konzumentom, serializuje doručenie správ. Objekt `Session` je vytváraný pomocou rozhrania `Connection`, pre jedno spojenie sa väčšinou používa niekoľko sedení, pretože tie tvoria transakčnú jednotku práce v komunikácii pomocou správ. `Message` je rodičovské rozhranie pre všetky typy správ, ktorým je venovaná nasledujúca podkapitola 2.1.3. `MessageProducer` posiela správy na

[1] http://download.oracle.com/otndocs/jcp/7543-jms-1.0.2b-spec-oth-JSpec/
[2] http://download.oracle.com/otndocs/jcp/7195-jms-1.1-fr-spec-oth-JSpec/

konkrétnu adresu – do fronty alebo topiku. Pri vytváraní producenta sedením je možné definovať predvolenú adresu, inak je potrebné adresu určiť pri odoslaní každej správy. `MessageConsumer` môže prijímať správy synchrónne alebo asynchrónne. Na synchrónne prijímanie slúžia tri obdoby metódy `receive()`. V prípade asynchrónneho prijímania je potrebný objekt `MessageListener`, ktorý prijíma správy vtedy, keď prídu.



Obrázok 2.4: Vzťahy medzi jednotlivými časťami JMS API.

2.1.3 Java Message Service správa

Správa je najdôležitejšou časťou JMS špecifikácie. Všetky dáta a udalosti sú komunikované pomocou správ a ostatné časti JMS existujú len preto, aby napomáhali tomuto procesu. Správa nesie dáta a oznamuje, že sa vyskytla nejaká udalosť. Na rozdiel od systémov založených na vzdialenom volaní procedúr JMS správa nie je príkaz na vykonanie nejakej procedúry, ani neblokuje odosielateľa, kým dostane odpoveď. Kvôli záruke úspešného odoslania a prijatia správy sú tieto činnosti potvrdzované na strane servera aj klienta. JMS správa sa skladá z troch komponentov, ktoré sú popísané v nasledujúcej časti.

Architektúra JMS správy

JMS správu tvorí *hlavička (header)*, *vlastnosti (properties)* a *telo (payload)*. JMS definuje 6 typov správ, ktoré sú určené typom obsahu v tele správy:

- `Message` – má prázdne telo, používa sa na jednoduché signalizovanie udalostí.
- `TextMessage` – telom správy je reťazec typu `java.lang.String`, ktorý môže obsahovať jednoduchý text, ale aj komplexné dáta. Môže sa použiť napríklad na prenos XML dokumentov.
- `ObjectMessage` – nesie serializovateľný objekt a slúži na výmenu Java objektov.
- `StreamMessage` – telo správy obsahuje prúd primitívnych typov (`int`, `char`...). môže byť využitý napríklad pre prenos veľkých správ, ktoré môžu presiahnuť veľkosť pamäte.

- `BytesMessage` – nesie pole bajtov. Je užitočný pre výmenu dát v natívnom formáte používajúcej aplikácie, napríklad v prípade, kedy je JMS použitý výhradne na prenos dát medzi dvomi systémami.

- `MapMessage` – telom správy je množina dvojíc (meno, hodnota), kde hodnota musí byť Java primitívum. Využitie má ako kľúčovaná správa, kedy meno predstavuje kľúč.

Hlavička správy obsahuje metadáta pre identifikáciu správy, nastavenie atribútov a informácie pre smerovanie. Polia hlavičky sa delia do dvoch skupín – *automaticky prideľované (automatically assigned)* a *prideľované programátorom (developer-assigned)*. Väčšina polí je prideľovaných automaticky JMS poskytovateľom pri odosielaní správy, ale niektoré musia byť explicitne definované objektu `Message` pred odoslaním. Medzi automaticky prideľované atribúty patria:

- `JMSDestination` – predstavuje adresu, kam správa smeruje, teda konkrétnu frontu alebo topik.

- `JMSDeliveryMode` – určuje jeden z dvoch možných spôsobov doručenia správy. Perzistentné správy sú doručené práve jedenkrát, čo znamená, že v prípade výpadku servera je správa doručená po jeho obnovení a nestratí sa. Neperzistentné správy sú doručené maximálne jedenkrát, preto sa môžu pri zlyhaní servera stratiť.

- `JMSMessageID` – obsahuje reťazec, ktorý správu jednoznačne identifikuje. Konkrétny spôsob identifikácie a miera jednoznačnosti závisí od JMS poskytovateľa.

- `JMSTimestamp` – je čas (meraný v milisekundách od 1.1.1970), kedy je správa prijatá JMS poskytovateľom.

- `JMSExpiration` – znamená čas, kedy správa stratí platnosť. Tento atribút sa používa pre správy, ktorých dáta sú platné len po vymedzenú dobu a zabraňuje doručeniu správy po jej uplynutí. Predvolená hodnota je nula, čo znamená, že platnosť správy neskončí nikdy.

- `JMSRedelivered` – hodnota `true` označuje, že správa bola konzumentovi znovu doručená. To sa môže stať, ak konzument nepotvrdil predchádzajúce doručenie správy alebo JMS poskytovateľ z nejakého iného dôvodu nevie, či bola správa doručená.

- `JMSPriority` – je priorita správy. Existujú dve kategórie priorít – normálna (stupne $0 - 4$) a zvýšená (stupne $5 - 9$). Správy so zvýšenou prioritou sú doručované pred tými s normálnou prioritou.

Nasledujúce atribúty sú prideľované programátorom:

- `JMSReplyTo` – určuje adresu, na ktorú má konzument v prípade potreby poslať odpoveď na danú správu. Neznamená to však, že konzument je povinný odpovedať, ak je tento atribút nastavený.

- `JMSCorrelationID` – je identifikátor, ktorý slúži na prepojenie aktuálnej správy s nejakými predchádzajúcimi správami alebo aplikačne špecifický identifikátor. Najčastejšie sa používa na označenie odpovede identifikátorom pôvodnej správy.

- `JMSType` – definuje štruktúru správy a typ obsahu jej tela.

Vlastnosti sú rozšírenia hlavičky správy, ktoré umožňujú programátorovi určiť jej ďalšie atribúty. Štandard JMS definuje tri kategórie vlastností v hlavičke správy:

- *Aplikačne špecifické vlastnosti (application-specific)* – sú definované a používané programátorom aplikácie. Môžu to byť ľubovoľné vlastnosti, ktoré daná aplikácia potrebuje.
- *Vlastnosti definované JMS (JMS-defined)* – definuje štandard JMS a jednotliví JMS poskytovatelia ich môžu, ale aj nemusia, podporovať.
- *Vlastnosti definované JMS poskytovateľom (vendor-specific)* – definuje každý JMS poskytovateľ a slúžia na podporu charakteristických vlastností daného poskytovateľa.

Filtrovanie správ

Bez použitia filtrovania správ prijímateľ prijme vždy nasledujúcu správu z fronty a predplatiteľ prijme všetky správy odoslané do určitého topiku. Avšak niekedy klient nepotrebuje všetky prijímané správy. Na takéto prípady slúži *selektor správ (message selector)*, bez ktorého by klient sám musel implementovať dodatočnú logiku na filtrovanie nežiaducich správ. Použitie selektora má ale oproti takému filtrovaniu ďalšie výhody. Správy odfiltrované pomocou selektora klient vôbec neprijme, preto ušetrí zdroje potrebné na ich spracovanie. Ak je takých správ veľké množstvo, môže sa výrazne zvýšiť efektivita programu. V point-to-point modeli je oproti publish-and-subscribe navyše výhoda v tom, že neprijaté správy ostávajú vo fronte a môžu byť prijaté iným konzumentom. Bez použitia selektora by skutočnosť, že ide o nepotrebnú správu, bola zistená až po jej prijatí, a teda po jej odobratí z fronty.

Selektor správ je definovaný v konzumentovi pre konkrétnu frontu alebo topik a následne konzument prijíma len tie správy, ktoré vyhovujú definovanému filtru. Neprijaté správy ostanú vo fronte alebo topiku, kým neskončí ich platnosť a môžu ich prijať iní konzumenti. Ak by správa nemala nastavenú dobu platnosti, môže ostať vo fronte alebo topiku navždy.

Selektory vyžívajú hlavičky a vlastnosti JMS správy v podmienených výrazoch. Obsah tela správy nemôže byť na túto úlohu použitý. Syntax selektorov je založená na podmnožine podmienených výrazov štandardu Structured Query Language SQL-92[3]. Skladajú sa z troch častí – identifikátorov, literálov a porovnávacích operátorov.

- *Identifikátor* – je časť podmieneného výrazu, ktorá je porovnávaná. Môže to byť len hlavička alebo vlastnosť JMS správy.
- *Literál* – môže byť reťazec, číslo alebo logická hodnota.
- *Porovnávacie operátory* – porovnávajú identifikátory s literálmi a vytvárajú logické výrazy. Môžu byť skladané pomocou logických spojok `AND` a `OR`. Medzi porovnávacie operátory patria: algebrické porovnávacie operátory, `LIKE`, `BETWEEN`, `IN`, `NOT` a `IS NULL`. Selektory môžu použiť aj aritmetické operátory, ktoré umožňujú vyhodnocovanie za behu programu.

Iný prístup k filtrovaniu správ ako pomocou selektorov je použitie viacerých destinácií – front alebo topikov. Tento prístup aplikuje filtrovanie správ pred ich odoslaním. V systéme existuje niekoľko destinácií pre rôzne správy, rozlíšené podľa nejakého kritéria. Producent

[3] http://www.contrib.andrew.cmu.edu/ shadow/sql/sql1992.txt

sám určuje, kam ktorú správu pošle a konzument potom nemusí používať filtrovanie. Keďže producent je ten, kto rozhoduje o filtrovaní, musí mať na to dostatočné znalosti. Čím viac musí producent vedieť, ako budú jeho správy použité, tým viac sú na seba producent a konzumenti naviazaní. Pri použití filtrovania pomocou selektorov je väzba medzi nimi slabšia a konzumenti rozhodujú o filtrovaní správ.

Samozrejme, je možný aj kombinovaný spôsob, kedy producent posiela správy do viacerých destinácií a konzumenti následne na jednotlivých destináciách vykonávajú filtrovanie pomocou selektorov. Tento prístup odstraňuje nevýhody oboch samostatne použitých spôsobov filtrovania správ.

Potvrdzovanie správ

Potvrdzovanie správ je kľúčovým prvkom záruky pri komunikácii pomocou správ. Server potvrdzuje prijatie správy od producenta, konzument potvrdzuje prijatie správy od servera. Protokol potvrdzovania správ umožňuje sledovať, či bola správa úspešne odoslaná a prijatá. JMS poskytovateľ tak môže garantovať jej doručenie. JMS špecifikuje niekoľko modelov potvrdzovania.

- `AUTO_ACKNOWLEDGE` – je implicitné potvrdenie správy po jej prijatí bez potreby konať zo strany klienta. Zaručuje prijatie správy práve jedenkrát.
- `DUPS_OK_ACKNOWLEDGE` – indikuje, že je v poriadku, ak je správa doručená jednému konzumentovi viac ako jedenkrát. Aplikácie, ktorým nevadí viacnásobné doručenie správy, môžu mať lepšiu výkonnosť, pretože strácajú nadbytočnú záťaž podmienky práve jedného doručenia pri možnosti `AUTO_ACKNOWLEDGE`.
- `CLIENT_ACKNOWLEDGE` – je spôsob potvrdenia, ktorý má klient vo vlastných rukách. Konzument explicitne informuje JMS server o doručení správy a môže to urobiť v ľubovoľnom čase, nielen bezprostredne po jej prijatí.

Transakčné doručovanie správ

JMS tiež podporuje posielanie viacerých správ v rámci jednej transakcie. To znamená, že zo skupiny správ budú doručené buď všetky alebo ani jedna. JMS transakcie sú, rovnako ako klasické posielanie správ, rozdelené na dve časti – odoslanie skupiny správ od producenta k JMS serveru a ich následné prijatie z JMS servera konzumentom. Vďaka tomuto rozdeleniu je možné kombinovať transakčné odosielanie s ne-transakčným prijímaním správ a naopak.

Transakčné odosielanie správ pre JMS server znamená, že musí ukladať správy a neprepposielať ich konzumentovi, kým producent nevyvolá `commit()`. V prípade výskytu chyby alebo vyvolania `rollback()`, sú všetky správy z danej transakcie zrušené a konzument nedostane ani jednu z nich.

Transakčné prijímanie správ je analogické k odosielaniu. JMS server musí znova ukladať odosielané správy, kým konzument nevyvolá `commit()`. Ak počas prijímania skupiny správ nastane chyba alebo konzument vyvolá `rollback()`, server sa pokúsi znova poslať celú skupinu správ.

2.1.4 Ďalšie prvky Java Message Service

JMS podporuje ďalšie vlastnosti, ktorých použitie a možnosti závisia od jednotlivých poskytovateľov. Sú to napríklad *mosty*, *vysoká dostupnosť*, *exspiračné fronty* a *mŕtve fronty*.

Niektorí JMS poskytovatelia umožňujú spájanie jednotlivých JMS serverov alebo clusterov pomocou tzv. *mostov (bridge)*. JMS mosty sa používajú typicky pre nespoľahlivé spojenia, napríklad cez rozľahlú sieť Wide Area Network (WAN) alebo Internet. Takéto spojenie je možné len pri použití point-to-point modelu komunikácie. Most najprv skonzumuje správy z fronty na jednom serveri a potom ich smeruje do fronty na inom serveri.

Vysoká dostupnosť (high availability) pri práci v clusteri znamená schopnosť systému pokračovať v práci aj pri výskyte chyby na jednom alebo viacerých serveroch. Súčasťou vysokej dostupnosti je tzv. *fail-over*, čo predstavuje schopnosť klienta pripojiť sa po výpadku servera k inému serveru. Opačný proces je tzv. *fail-back*, teda pripojenie klienta naspäť k pôvodnému serveru po jeho zotavení sa z výpadku.

Napriek všetkým zárukám, ktoré JMS poskytuje pre doručenie správy, sa môže stať, že správa nemôže byť doručená. Ak má správa nastavený atribút doby platnosti a táto doba uplynie pred jej doručením, správa je z fronty odstránená. Avšak v prípade, kedy je platnosť správy neobmedzená, mohla by vo fronte ostať navždy. Aby sa vo frontách nehromadili nedoručiteľné správy a nezahlcovali JMS prostriedky, niektorí JMS poskytovatelia implementujú tzv. *mŕtve fronty (dead letter queue)*. Mŕtve fronty majú za úlohu sa nejakým spôsobom postarať o také správy, ktoré sa nedajú doručiť v rozumnom čase. V najjednoduchšom prípade ide len o smerovanie týchto správ do mŕtvej fronty. Rozšírením môže byť upozornenie klienta na vloženie správy do tejto fronty. Ak JMS poskytovateľ podporuje mŕtvu frontu, pre aplikáciu je potrebné, aby obsahovala konzumenta, ktorý sleduje jej obsah a prijíma z nej správy. Niektorí JMS poskytovatelia podporujú aj tzv. *exspiračnú frontu (expiry queue)*. Je to fronta, do ktorej sú umiestnené všetky správy, ktorým skončila doba platnosti pred ich prijatím.

2.1.5 Java Message Service a Java Enterprise Edition

Java EE špecifikácia určuje, že každý aplikačný server podporujúci Java EE 4 alebo neskoršiu verziu, musí byť zároveň JMS poskytovateľom. To znamená, že v prostredí Java EE nie je potrebný externý JMS poskytovateľ, hoci jeho použitie nie je vylúčené. JMS má v Java EE dve úlohy – je to služba a zároveň tvorí základ komponent nazývaných *Message Driven Bean (MDB)* [12]. MDB je v podstate JMS klient, ktorý vie asynchrónne prijímať a odosielať JMS správy. Nie je ale viazaný na JMS, môže implementovať ľubovoľný typ komunikácie pomocou správ. MDB komunikuje pomocou správ nielen s inými Java EE komponentmi, ale aj s aplikáciami, ktoré nepoužívajú Java EE technológie.

Na rozdiel od klasického JMS klienta MDB čerpá výhody Java EE komponentu, napríklad súbežné spracovanie. Po nasadení na aplikačný server začne MDB počúvať na fronte alebo topiku a čakať príchodzie správy. V skutočnosti je počas behu programu vytvorených niekoľko inštancií MDB a uložených do bloku. Pri príchode správy je jeden MDB z bloku vybraný na jej spracovanie. V prípade príchodu viacerých správ súčasne je vybraný pre každú správu jeden MDB. Takto môžu byť správy spracované súbežne a MDB vykazuje vyššiu priepustnosť ako bežný JMS klient.

Podľa Enterprise Java Beans špecifikácie verzie 3 (EJB3) [2] existujú dva spôsoby definície MDB, a to pomocou *anotácií* priamo v triede MDB alebo pomocou XML súboru nazývaného *deployment descriptor*. Na rozpoznanie komponentu ako MDB slúži anotácia `@MessageDriven`, ktorá obsahuje jednotlivé atribúty, každý atribút je definovaný pomocou anotácie `@ActivationConfigProperty`. Pomocou týchto anotácií je možné definovať typ destinácie, adresu destinácie, typ potvrdzovania správ, selektory pre filtrovanie správ apod. Tieto atribúty sa dajú rovnako definovať v príslušnom XML súbore.

Trieda MDB musí podľa JMS špecifikácie implementovať rozhranie `MessageListener`, ktoré definuje jedinú metódu `onMessage()`. Táto metóda je automaticky vyvolaná pri príchode správy a slúži na jej spracovanie, čo spôsobí odstránenie jednej inštancie MDB z bloku. Po skončení metódy `onMessage()` je inštancia MDB vrátená naspäť do bloku a pripravená na spracovanie ďalšej správy.

EJB ponúka ešte jeden typ komponentu, ktorý dokáže pracovať s JMS správami, a to tzv. session bean. Session bean môže byť použitý na posielanie správ, ale na rozdiel od MDB vie správy prijímať len synchrónne.

Na integráciu JMS poskytovateľa do aplikačného servera slúži Java EE Connector Architecture, kedy je k JMS poskytovateľovi možné pristupovať pomocou tzv. adaptéra zdrojov. Vďaka tomuto prístupu môžu byť JMS poskytovatelia používaní v rôznych aplikačných serveroch a aplikačné servery môžu podporovať použitie rôznych JMS poskytovateľov. Štandard Java EE Connector Architecture je predstavený v nasledujúcej podkapitole 2.2.

2.2 Java EE Connector Architecture

Java EE Connector architecture (JCA) [3, 12] definuje štandardnú architektúru pre pripájanie platformy J2EE k heterogénnym systémom nazývaným *Enterprise Information System (EIS)* [12], a tým umožňuje komunikáciu aplikačného servera so systémom EIS.

EIS je informačný systém, ktorý poskytuje informačnú infraštruktúru podniku a ponúka klientom služby. Zahŕňa rôzne typy systémov, ako napríklad enterprise resource planning (ERP), mainframe transaction processing alebo databázové systémy. Každý EIS potrebuje len jednu implementáciu JCA, čím JCA zjednodušuje integráciu rozdielnych EIS. Keďže implementácia dodržiava špecifikáciu JCA, je prenosná na všetky zhodné Java EE servery. JCA ďalej formalizuje vzťahy, vzájomnú komunikáciu a balíčkovanie integračnej vrstvy, a tým umožňuje integráciu enterprise aplikácií.

Architektúra JCA sa skladá z dvoch častí. Prvou je adaptér zdrojov od dodávateľa EIS a druhou je aplikačný server, ktorý umožňuje inštaláciu adaptéra zdrojov. Po jednorazovom rozšírení aplikačného servera o podporu JCA je zaručená konektivita s viacerými EIS. Podobne dodávateľ EIS musí poskytnúť len jeden štandardný adaptér zdrojov, ktorý môže byť inštalovaný do ľubovoľného aplikačného servera podporujúceho JCA.

2.2.1 Adaptér zdrojov

Adaptér zdrojov (resource adapter) je Java EE komponent, ktorý implementuje JCA pre špecifický EIS. Pre každý typ databázy alebo iný EIS existuje zvláštny adaptér zdrojov. Adaptér zdrojov sa nainštaluje do aplikačného servera a poskytuje konektivitu medzi EIS, aplikačným serverom a enterprise aplikáciou. Komunikácia môže byť príchodzia, odchodzia aj obojsmerná. Aplikačný server používa adaptér zdrojov pre odchodziu komunikáciu smerom k EIS a pre príchodziu komunikáciu od EIS. Adaptér zdrojov takto poskytuje enterprise aplikáciám prístup k zdrojom mimo aplikačný server.

Moduly adaptéra sú balené v Java archívoch (JAR) s príponou .rar (resource adapter archive). Obsahujú rozhrania a triedy programovacieho jazyka Java, natívne knižnice a deployment descriptor adaptéra zdrojov. Tieto archívy môžu byť nasadené na ľubovoľný Java EE aplikačný server, podobne ako archívy Enterprise Archive (EAR) alebo Java EE aplikácie. RAR súbor môže byť nasadený samostatne alebo sa môže nachádzať vovnútri EAR súboru.

2.2.2 Klientske rozhranie

Komponenty enterprise aplikácií pristupujú k EIS prostredníctvom adaptéra zdrojov pomocou klientskeho API, ktoré definuje metódy typické pre prístup k dátam. Existujú dva druhy API, ktoré môžu využiť:

- Common Client Interface – je štandardné API, ktoré definuje JCA. Popis tohto rozhrania sa nachádza nižšie.
- Špecifické klientske API – sú rôzne API špecifické pre konkrétny typ adaptéra zdrojov a jeho EIS. Príkladom môže byť Java Database Connectivity (JDBC) API [4] pre prístup k relačným databázam.

Common Client Interface

Common Client Interface (CCI) je jednotné rozhranie pre vzájomnú komunikáciu s heterogénnymi EIS. Bolo navrhnuté tak, aby bolo nezávislé od špecifík konkrétneho EIS ako napríklad dátových typov. Cieľom CCI je dopĺňať štandardné JDBC API pre EIS iné ako relačné databázy. Štandard JCA neprikazuje podporu CCI ako klientskeho rozhrania adaptéra zdrojov.

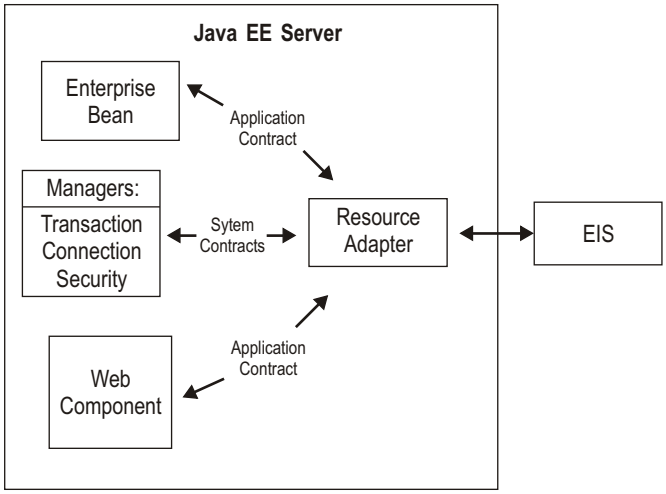
CCI sa skladá z niekoľkých častí, ktoré sú súčasťou implementácie adaptéra zdrojov. Patria medzi ne:

- `ConnectionFactory` – je rovnako ako v JMS API továrenská trieda slúžiaca na vytvorenie objektov `Connection`. Komponenty ju používajú na získanie spojenia s EIS.
- `ConnectionSpec` – poskytuje prostriedky pre predávanie atribútov objektu `ConnectionFactory` potrebných na vytvorenie spojenia. Tieto atribúty by mali byť viazané na klienta (napríklad užívateľské meno, heslo, jazyk, apod.) a nie na konfiguráciu cieľového EIS (ako číslo portu alebo meno servera).
- `Connection` – reprezentuje vlastné spojenie s EIS a je používaný na nasledujúce interakcie s EIS. Jedna inštancia `Connection` môže byť asociovaná s niekoľkými inštanciami `Interaction`.
- `Interaction` – používajú komponenty na vykonávanie funkcií EIS (napríklad procedúry uložené v databáze).
- `InteractionSpec` – definuje atribúty objektu `Interaction`.
- `LocalTransaction` – používajú komponenty na vymedzenie lokálnych transakcií na aplikačnej úrovni.
- `Record` – je Java reprezentácia dátovej štruktúry používanej ako vstup a výstup funkcií EIS. Je to rodičovská trieda pre rôzne druhy záznamov EIS. Komponenty môžu záznamy čítať podobne ako dáta z databázových tabuliek alebo ich zapisovať. Tieto činnosti vykonáva pomocou rôznych inštancií rozhrania `Record` – buď `MappedRecord`, `IndexedRecord` alebo `ResultSet`.
- `RecordFactory` – vytvára inštancie rozhrania `Record` podobne ako sú inštancie rozhrania `Connection` vytvárané pomocou objektu `ConnectionFactory`.
- `ConnectionMetaData`, `ResourceAdapterMetaData` a `ResultSetInfo` – triedy týkajúce sa metadát o implementácii adaptéra zdrojov a spojení s EIS.

2.2.3 Kontrakty adaptéra zdrojov

Adaptér zdrojov predstavuje prostredníka v komunikácii medzi aplikačným serverom a EIS. Túto úlohu vykonáva prostredníctvom *kontraktov (contract)*, ktoré definuje JCA a adaptér zdrojov ich musí podporovať na to, aby mohol byť inštalovaný do aplikačného servera. Na Obrázku 2.5 sú znázornené typy kontraktov, ktoré existujú medzi adaptérom zdrojov a rôznymi komponentmi. Aplikačný kontrakt definuje API, pomocou ktorého Java EE komponenty (napr. enterprise bean) pristupujú k EIS. Systémový kontrakt spája adaptér zdrojov s dôležitými službami, ktoré sú spravované aplikačným serverom. JCA definuje niekoľko druhov systémových kontraktov medzi aplikačným serverom a EIS – kontrakt na umožnenie riadenia, odchodzej komunikácie i príchodzej komunikácie. Adaptér zdrojov a systémový kontrakt sú Java EE komponentu transparentné.

Ku kontraktom o riadení patrí kontrakt na riadenie životného cyklu adaptéra zdrojov (Lifecycle Management Contract) a riadenie práce adaptéra zdrojov s vláknami (Work Management Contract). Kontrakty o odchodzej komunikácii zahŕňajú správu spojenia (Connection Management Contract), správu transakcií (Transaction Management Contract) a správu zabezpečenia (Security Contract). Kontrakty týkajúce sa príchodzej komunikácie poskytujú mechanizmy vyvolávania komponentov MDB z adaptéra zdrojov (Message Inflow Contract) a propagácie transakčných informácií z EIS do aplikácie v EJB kontajneri (Transaction Inflow Contract).



Obrázok 2.5: Java Connector Architecture.

Lifecycle Management Contract

Životný cyklus adaptéra zdrojov je riadený aplikačným serverom. Kontrakt o riadení životného cyklu poskytuje aplikačnému serveru mechanizmus zavádzania adaptéra zdrojov a adaptéru zdrojov poskytuje signalizáciu, že je ukončené jeho nasadenie v aplikačnom serveri.

Aplikačný server zavádza inštanciu adaptéra zdrojov do svojho adresového priestoru buď pri svojom spustení alebo pri nasadení adaptéra zdrojov do aplikačného servera. Počas

tohto procesu musí aplikačný server zavolať metódu `start()` inštancie adaptéra zdrojov, čo má za následok vytvorenie špecifických objektov a vláken a nastavenie sieťových spojení.

Keď skončí činnosť adaptéra zdrojov alebo samotného aplikačného servera, aplikačný server o tom informuje adaptér zdrojov. Ukončenie adaptéra zdrojov sa deje v dvoch fázach. Najprv sa aplikačný server musí uistiť, že všetky aplikácie používajúce danú inštanciu adaptéra zdrojov sú zastavené. Ukončenie prvej fázy zaručí, že aplikácie nebudú používať túto inštanciu adaptéra zdrojov ani v prípade, keď niektoré jeho objekty ostanú v pamäti. Rovnako zaručuje ukončenie všetkých aktivít daných aplikácií vrátane transakcií. V druhej fáze aplikačný server zavolá metódu `stop()` triedy `ResourceAdapter` a jej inštancia tak prestane pracovať, aby mohla byť bezpečne uvoľnená. Zastavenie inštancie adaptéra zdrojov zahŕňa uzavretie spojení, uvoľnenie vláken a špecifických objektov, ukončenie transakcií a zápis všetkých dát z vyrovnávacej pamäte do EIS.

Inštancia adaptéra zdrojov sa môže stať nefunkčnou aj pred zavolaním metódy `stop()`, ak zlyhá EIS. V takom prípade adaptér zdrojov vyhodí výnimku pri pokuse o odchodziu komunikáciu aplikáciou alebo aplikačným serverom.

Work Management Contract

Aplikačný server riadi aj prácu adaptéra zdrojov s vláknami. Nesprávna práca s vláknami, napríklad tvorenie príliš veľkého množstva vláken alebo neuvoľňovanie vytvorených vláken, môže spôsobiť zhoršenie výkonu celého aplikačného servera prípadne jeho zastavenie. Aplikačný server vytvára pre adaptér zdrojov blok zdieľaných vláken, z ktorého mu poskytuje potrebné vlákna. Po dokončení práce s daným vláknom ho adaptér zdrojov vráti naspäť aplikačnému serveru, ktorý ho môže vrátiť naspäť do bloku na ďalšie použitie alebo uvoľniť.

Connection Management Contract

Kontrakt o riadení spojenia umožňuje aplikačnému serveru pripájať sa k EIS. Riadenie spojenia podporuje ukladanie zdieľaných spojení do bloku podobne ako pri vytváraní vláken. Mechanizmus zdieľania spojení umožňuje správu spojení, ktoré sú náročné na vytvorenie a zničenie. Aplikačné prostredie sa tak stáva škálovateľným a podporuje veľké množstvo klientov vyžadujúcich prístup k EIS. Táto činnosť je aplikácii úplne transparentná.

Adaptér zdrojov poskytuje aplikácii inštancie tried `ConnectionFactory` a `Connection`. Aplikácia najprv vyhľadá inštanciu `ConnectionFactory` v mennom priestore JNDI a potom ju použije na získanie spojenia s EIS. `ConnectionFactory` deleguje požiadavku na vytvorenie spojenia inštancii `ConnectionManager` v aplikačnom serveri.

`ConnectionManager` umožňuje aplikačnému serveru poskytovať rôzne úrovne kvality služieb vrátane správy transakcií, zabezpečenia, zaznamenávanie a trasovanie chýb apod. Po prijatí žiadosti o spojenie začne vyhľadávanie v bloku spojení, ktoré poskytuje aplikačný server. Ak v bloku existuje spojenie, ktoré vyhovuje požiadavku, použije ho. V opačnom prípade aplikačný server vytvorí nové fyzické spojenie s EIS a pridá ho do bloku zdieľaných spojení.

Transaction Management Contract

JCA definuje kontrakt o riadení transakcií medzi aplikačným serverom, adaptérom zdrojov a jeho správcom transakcií, ktorý podporuje transakčný prístup k správcom zdrojov v EIS. *Správca transakcií (transaction manager)* riadi transakcie rôznych adaptérov zdrojov

a podporuje propagáciu transakčného kontextu naprieč distribuovaným systémom. Tento kontrakt má dve časti rozlíšené podľa typov transakcií, ktoré adaptér zdrojov podporuje:

- XA transakcie – sú kontrolované a koordinované správcom transakcií mimo správcu zdrojov.
- Lokálne transakcie – sú spravované interne správcom zdrojov bez potreby použitia externého správcu transakcií.

Security Contract

Bezpečnostný kontrakt rozširuje kontrakt o riadení spojenia o detaily týkajúce sa zabezpečenia. Poskytuje mechanizmy, ktoré ochraňujú EIS a v ňom uložené informácie pred neautorizovaným prístupom. Medzi tieto mechanizmy patria:

- *Identifikácia a autentizácia* – overenie užívateľov, že sú skutočne tými, za ktorých sa vydávajú. Vytvorenie nového fyzického spojenia vyžaduje prihlásenie sa k inštancii EIS. Podobne zmena bezpečnostného kontextu v existujúcom fyzickom spojení vyžaduje opätovnú autentizáciu.
- *Autorizácia a kontrola prístupu* – rozhodnutie, či užívateľ má právo pristupovať k aplikačnému serveru alebo EIS.
- *Bezpečná komunikácia* – medzi aplikačným serverom a EIS. Komunikácia prebiehajúca cez nezabezpečené spojenie môže byť chránená použitím protokolu (napríklad Kerberos), ktorý poskytuje služby autentizácie, integrity a utajenia. Ďalšou možnosťou je použitie zabezpečeného spojového protokolu (napríklad SSL).

Message Inflow Contract

Kontrakt o príchodzích správach umožňuje adaptéru zdrojov asynchrónne doručovať správy koncovým bodom v aplikačnom serveri. Nie je závislý na konkrétnom type zasielania správ ani na sémantike správ. JCA tak rozširuje možnosti komponentov MDB o schopnosť spracovať ľubovoľné správy nezávisle od použitej komunikácie, napríklad Java Message Service (JMS) alebo Java API for XML Messaging (JAXM). Takto sa môžu s aplikačným serverom spájať externé systémy s rôznymi typmi posielania správ. Ďalšou úlohou tohto kontraktu je umožnenie inštalácie rôznych poskytovateľov komunikácie pomocou správ do aplikačného servera.

Základnou črtou asynchrónneho zasielania správ je voľná viazanosť odosielateľa a príjemcu správy, komunikujúce strany nezdieľajú žiadny výpočtový kontext. Odosielateľa vo všeobecnosti nezaujíma výstup spracovania správy príjemcami, ani nemusí o príjemcoch vôbec vedieť. Správa môže byť prijatá jedným alebo viacerými príjemcami. Doručenie správy je vyžiadané, to znamená, že príjemca musí vopred prejaviť záujem o prijímanie správ.

Kontrakt o príchodzích správach je využitý v rôznych fázach životného cyklu komponentu MDB – jeho nasadenie, doručovanie správ a koniec nasadenia. V týchto fázach vystupujú nasledujúci aktéri:

- *Koncový bod* – aplikácia s MDB, ktorá je nasadená na aplikačný server a asynchrónne prijíma správy od dodávateľa správ. Rovnako môže správy odosielať a prijímať synchrónne.

- *Adaptér zdrojov* – je systémový komponent umiestnený v aplikačnom serveri schopný doručovať správy. Môže byť samostatný a zdieľaný viacerými aplikáciami alebo zabalený spolu s aplikáciou, ktorá obsahuje MDB.
- *Aplikačný server* – poskytuje prostredie pre beh aplikácie. Aktivuje koncový bod pri jeho nasadení a deaktivuje pri ukončení nasadenia.
- *Nasadzovateľ* – človek, ktorý pozná detaily aplikácie aj aplikačného servera, na ktorý bude nasadená. Má na starosti výber vhodného adaptéra zdrojov, konfiguráciu a nastavenia.
- *Dodávateľ správ* – je typicky externý systém, ktorý vystupuje ako zdroj správ.

Transaction Inflow Contract

JCA definuje aj spôsob propagácie transakčného kontextu z EIS. EIS napríklad začne transakciu, v rámci ktorej je vytvorené spojenie s Java EE komponentom v aplikačnom serveri cez adaptér zdrojov. Java EE komponent potom môže pokračovať v práci v rovnakom transakčnom kontexte a nakoniec transakciu komitnúť. JCA špecifikuje aj spôsob, akým kontajner ošetruje zotavenie po poruche na ochranu integrity dát. Kontrakt o príchodzích transakciách zabezpečuje dodržanie ACID vlastností transakcií – atomicitu (Atomicity), konzistenciu (Consistency), izoláciu (Isolation) a trvanlivosť (Durability).

2.3 JMS poskytovatelia

Ako už bolo spomenuté v kapitole 2.1, každý JMS poskytovateľ poskytuje vlastnú implementáciu JMS API, čím umožňujú vytvárať aplikácie, ktoré komunikujú zasielaním správ. V tejto kapitole je predstavených niekoľko JMS poskytovateľov od rôznych výrobcov, komerčné aj open source. Pri výbere konkrétneho JMS poskytovateľa hrajú dôležitú úlohu vlastnosti ako spôsob perzistencie dát, podpora rôznych transportných protokolov, možnosti clusteringu a správy systému.

JBoss ponúka v súčasnosti dvoch JMS poskytovateľov. V JBoss AS 4.3 a 5 je predvolený JMS poskytovateľ JBoss Messaging. V JBoss AS od verzie 7 ho vystriedal HornetQ. Oba systémy sú open source a napísané v programovacom jazyku Java. Príkladmi JMS poskytovateľov tretích strán sú komerčný WebSphere MQ od IBM a open source Apache ActiveMQ alebo Red Hat MRG Messaging.

2.3.1 JBoss Messaging

JBoss Messaging[4] [7] implementuje robustné výkonné jadro navrhnuté pre požiadavky ako Sevice Oriented Architecture (SOA) [5] alebo Enterprise Service Bus (ESB) [5] a ďalšie. Jeho návrh je zameraný na výkon, spoľahlivosť a škálovateľnosť s vysokou priepustnosťou a nízkou latenciou. Zahŕňa JMS front-end, čo znamená, že môže podporovať ďalšie protokoly posielania správ v budúcnosti.

JBoss Messaging poskytuje rozsiahle administračné rozhranie Java Management Extensions (JMX) [10]. JMS objekty (ako `ConnectionFactory` alebo destinácie) a mosty sú konfigurované ako Managed Bean (MBean) služby [12] v aplikačnom serveri. Vďaka integrácii s JBoss Transactions[5] podporuje plné zotavenia z transakcie. Bezpečnosť v JBoss

[4] http://www.jboss.org/jbossmessaging
[5] http://www.jboss.org/jbosstm

Messaging je založená na službe Java Authentication and Authorization Service (JAAS) [10].

Nasledujúce odseky vysvetľujú niektoré služby systému JBoss Messaging. Každá služba má vlastný konfiguračný súbor, ktorý je uložený v adresári `$JBOSS_HOME/server/$PROFILE/deploy/messaging`. Základná konfigurácia tzv. ServerPeer sa nachádza v súbore `messaging-service.xml`.

Perzistencia

JBoss Messaging používa ako úložište perzistentných dát databázu. Do databázových tabuliek ukladá dáta o vnútornom stave systému, perzistentné správy a ich stav, stav transakcií a bezpečnostné informácie. Okrem týchto dát navyše podporuje automatické stránkovanie správ do úložišťa, ktoré umožňuje použitie veľkých front, aké by sa nezmestili do operačnej pamäte.

Predvolenou databázou je Hypersonic[6], ktorá však nie je podporovaná pre použitie v produkcii a je vhodná len pre vývoj a testovanie kvôli niektorým známym problémom. K týmto problémom patrí napríklad neschopnosť práce v clusteri, nestabilnosť pri veľkej záťaži alebo neschopnosť podpory izolácie transakcií.

Všetka konfigurácia týkajúca sa predvolenej perzistencie dát sa nachádza v súbore `hsql-persistence-service.xml`. Výmena použitej databázy spočíva vo výmene tohto súboru za súbor <`typ použitej databázy`>`-persistence-service.xml`. Je možné použiť väčšinu najpoužívanejších databáz v súčasnosti: Oracle[7], DB2[8], Sybase[9], MS SQL Server[10], PostgreSQL[11], MySQL[12].

Transport

Konfigurácia transportu sa nachádza v súbore `remoting-bisocket-service.xml`. Všetka komunikácia medzi klientmi a serverom sa deje pomocou frameworku JBoss Remoting[13]. Implicitný typ transportu používa protokol TCP, ktorý je vhodný v prípade, kedy je povolená len príchodzia komunikácia na serveri alebo odchodzia na klientoch.

Medzi ďalšie možnosti patrí HyperText Transport Protocol (HTTP) [11], pre použitie s bránami firewall, ktoré podporujú len komunikáciu pomocou HTTP. Klient sa pravidelne dotazuje servera pre prijatie správ, kvôli čomu má tento typ transportu horšiu výkonnosť. Ak je potrebná vyššia miera zabezpečenia, je možné použiť Secure Sockets Layer (SSL) [11]. Poslednou možnosťou je komunikácia pomocou správ cez dedikovaný Java Servlet [12].

Clustering

Ďalšou skúmanou vlastnosťou je clustering, ktorý umožňuje spoluprácu distribuovaných serverov. Pri použití clusteru serverov JBoss Messaging rovnomerne rozdeľuje záťaž a vyvažuje cykly procesora každého uzla. Pri zlyhaní jedného uzla sa klient automaticky pripojí k inému uzlu z daného clusteru bez vyhodenia výnimky. Toto správanie je konfigurovateľné

[6] http://hsqldb.org/
[7] http://www.oracle.com/us/products/database/overview/index.html
[8] http://www-01.ibm.com/software/data/db2/
[9] http://www.sybase.com/
[10] http://www.microsoft.com/sqlserver/en/us/default.aspx
[11] http://www.postgresql.org/
[12] http://www.mysql.com/
[13] http://www.jboss.org/jbossremoting

a je možné ho zakázať a znovu vytvoriť spojenie manuálne. Clustery serverov, ktoré sú geograficky vzdialené je možné spájať pomocou mostov.

Striktné zoradenie správ

Implementáciou striktného zoradenia správ sú tzv. *ordering groups* – skupiny správ, v rámci ktorých sú správy doručené vždy v presnom poradí ich príchodu do cieľovej fronty. To znamená, že správa z danej skupiny je odoslaná až po úspešnom doručení predchádzajúcej správy z tejto skupiny. Keď je podpora ordering groups povolená, priority správ nemajú vplyv na poradie ich doručenia.

Povoliť ordering groups je možné dvomi spôsobmi:

- V objekte `ConnectionFactory` – všetci producenti vytvorení z tohto objektu budú používať spoločnú skupinu správ pre striktné zoradenie.
- V producentovi – producent má svoju vlastnú skupinu. V prípade, že objekt `ConnectionFactory`, z ktorého bol daný producent vytvorený, má tiež definovaný ordering group, uprednostní sa skupina definovaná v producentovi.

Ordering groups je možné použiť len s frontami. Nepoužívajú sa s topikmi, selektormi správ a ani s clusterovými frontami.

2.3.2 HornetQ

HornetQ[14] [16] je open source Message Oriented Middleware projekt vyvíjaný komunitou JBoss. Je to multiprotokolový clusterový asynchrónny systém posielania správ. Namiesto databázy používa pre perzistenciu vlastný žurnálový systém. Na rozdiel od systému JBoss Messaging, kde sú JMS objekty a mosty implementované ako MBean, v HornetQ sú implementované ako objekty Plain Old Java Object (POJO) [12].

HornetQ poskytuje okrem možnosti použiť JMS API aj vlastné klientske rozhranie na dosiahnutie funkcionality prekračujúcej JMS API. Toto rozhranie sa nazýva *Core client API* a zahŕňa všetku funkcionalitu JMS. Core client API je jednoduchšie ako JMS API, pretože odstraňuje rozdiely medzi frontami, topikmi a predplatiteľmi. Topik je implementovaný ako adresa, na ktorú je pripojených nula alebo viac front, z ktorých každá vedie k jednému predplatiteľovi. V prípade, kedy klient používa JMS API, sú všetky operácie najprv preložené do Core Client API a následne vykonané. HornetQ server tak pracuje len s Core Client API. Nižšie sú opäť uvedené dôležité vlastnosti systému HornetQ.

Administrácia

HornetQ poskytuje niekoľko administračných rozhraní, ktoré podporujú rovnakú funkcionalitu – umožňujú meniť nastavenie servera, vytvárať, prezerať a spravovať fronty a topiky, apod.

- *JMX* – štandardný spôsob správy Java aplikácií,
- *Core API* – administračné operácie sú zasielané HornetQ serveru Core správami,
- *JMS API* – administračné operácie sú zasielané HornetQ serveru JMS správami.

[14] https://www.jboss.org/hornetq

Perzistencia

Každý HornetQ server má svoj vlastný výkonný žurnál na perzistenciu, ktorý umožňuje dosiahnuť výkon, aký by s použitím relačnej databázy nebolo možné dosiahnuť. Žurnál je optimalizovaný pre použitie na komunikáciu pomocou zasielania správ. Poskytuje len operáciu zápisu – je tzv. append only. Skladá sa zo súborov na disku, ktoré sú predpripravené na konkrétnu veľkosť. Záznamy operácií ako pridanie alebo odobratie správy sa postupne dopisujú do žurnálu, kým aktuálny súbor nie je plný, potom sa začne používať nový súbor. O tom, či môže byť súbor žurnálu znovu použitý po zmazaní všetkých v ňom uložených dát, rozhoduje algoritmus automatickej správy pamäte garbage collector.

Väčšina žurnálu je implementovaná v jazyku Java. Operácie interakcie so súborovým systémom sú ale abstrahované, aby mohli podporovať rôzne implementácie. HornetQ poskytuje dve implementácie:

- *Java Non-blocking IO (NIO)* – používa štandardné operácie zápisu a čítania z jazyka Java. Beží na každej platforme, ktorá podporuje Javu verzie 6 alebo neskoršej.
- *Linux Asynchronous IO (AIO)* – používa linuxovú asynchrónnu knižnicu na zápis a čítanie. Poskytuje vyššiu výkonnosť, ale vyžaduje knižnicu `libaio`, ktorá je dostupná len na operačnom systéme Linux[15].

Na prehliadanie žurnálu poskytuje HornetQ nástroj na export dát zo žurnálu do XML súboru a naopak import dát z XML súboru naspäť do žurnálu.

Transport

Transportná vrstva systému HornetQ je flexibilná a podporuje niekoľko rôznych spôsobov transportu. Nepoužíva JBoss Remoting ale Netty[16] – výkonnú sieťovú knižnicu, ktorá môže byť použitá na transport pomocou TCP soketov, SSL, HTTP, HTTPS a servletov. TCP transport je nešifrovaný a všetky spojenia sú iniciované zo strany klienta. Šifrované spojenie poskytujú protokoly SSL a HTTPS.

Na serveri sú používané objekty `acceptor`, ktoré definujú spôsob prijímania spojenia. Objekty `connector` sú zas používané klientmi na definovanie spôsobu pripojenia k serveru. Oba typy objektov sú definované na serveri a štandardne používajú port 5445.

Clustering

HornetQ cluster je skupina HornetQ serverov, ktoré medzi sebou rozdeľujú záťaž spojenú so spracovaním správ. Každý aktívny server tvorí jeden uzol clusteru. Vnútorne sú jednotlivé uzly spojené Core mostom, ktorý je vytvorený automaticky a nie je potrebné ho explicitne definovať pre každé spojenie.

Vyvažovanie záťaže pracuje na základe počtu konzumentov pripojených k jednotlivým uzlom a podľa toho, či sú pripravení na prijatie správ. HornetQ dokáže automaticky prerozdeliť správy medzi uzly v clusteri tak, aby sa predišlo strate správ.

Existuje niekoľko rôznych topológií, do ktorých môžu byť uzly vytvárajúce cluster spojené. Najčastejšie používané sú symetrický a reťazový cluster.

[15] www.linux.org
[16] http://netty.io/

- *Symetrický cluster* – každý uzol je spojený so všetkými ostatnými uzlami clusteru, teda vzdialenosť medzi ľubovoľnými dvomi uzlami je jedna. Každý uzol vie o všetkých frontách a konzumentoch na všetkých ostatných uzloch, čo je dostatočná znalosť pre rozdelenie záťaže a preposielanie správ medzi jednotlivými uzlami.
- *Reťazový cluster* – neobsahuje všetky uzly spojené priamo, ale jednotlivé uzly tvoria reťaz za sebou idúcich uzlov. Uzly tejto reťaze tvoria prostredníka ostatným uzlom, ktoré sa pripájajú vždy len na predchádzajúci a nasledujúci uzol reťaze.

Vysoká dostupnosť

HornetQ poskytuje možnosť vytvoriť páry aktívnych (živých) a neaktívnych (záložných) serverov. Každému živému serveru prináleží jeden záložný, každý záložný server môže prináležať len jednému živému. Záložný server nepracuje, až kým nenastane fail-over. Vtedy sa záložný server stane aktívnym a prípadný ďalší neaktívny server sa stane záložným. Po obnovení pôvodného živého servera bude mať tento server prioritu stať sa živým, ak znovu nastane fail-over. Záložný server môže byť nakonfigurovaný tak, aby detekoval obnovu pôvodného živého servera a predať mu prácu naspäť.

V súčasnosti HornetQ podporuje len použitie zdieľaného úložiska dát pre živý aj záložný server. To znamená, že oba servery majú prístup k tomu istému adresáru so žurnálom a ďalšími perzistentnými dátami. Keď nastane fial-over, záložný server tieto dáta načíta a pokračuje v práci živého servera. Výhodou tohto spôsobu je, že nie je potrebné kopírovanie dát medzi servermi, ktoré by vyžadovalo určitú réžiu.

Mosty

HornetQ poskytuje dva typy mostov. Oba podporujú filtráciu pomocou selektorov na prenos konkrétnych správ a môžu konzumovať správy z front aj topikov. *JMS most* slúži na prepojenie ľubovoľných dvoch JMS poskytovateľov a používa JMS API. Pri použití *Core mostu* musí byť na oboch stranách HornetQ server.

2.3.3 ActiveMQ

ActiveMQ[17] [15] je open source MOM zhodný s JMS 1.1 od Apache Software Foundation. Cieľom ActiveMQ je poskytnúť integráciu aplikácií založenú na posielaní správ, ktorá by podporovala čo najviac rôznych programovacích jazykov a platform. Okrem implementácie JMS poskytuje ďalšie vlastnosti, ako napríklad virtuálne destinácie na zníženie počtu spojení so serverom, transformáciu správ, zrkadlené fronty na jednoduchšie riadenie, funkciu spätného konzumenta pre prijatie starých správ po pripojení k topiku a ďalšie. Okrem programovacieho jazyka Java ActiveMQ podporuje množstvo ďalších jazykov ako C/C++, .NET, Perl, Python alebo Ruby. Táto vlastnosť poskytuje možnosti širšieho využitia ActiveMQ. Napriek tomu, že server beží vo virtuálnom stroji Java Virtual Machine (JVM), klient môže byť napísaný v ľubovoľnom z podporovaných jazykov.

Konektivita

Konektor je mechanizmus, ktorý poskytuje komunikáciu medzi klientmi a serverom a medzi servermi navzájom. V ActiveMQ existujú dva typy konektorov, ktoré sa líšia komunikujúcimi stranami, a to transportný a sieťový konektor.

[17] http://activemq.apache.org/

Transportný konektor slúži na komunikáciu klienta so serverom. Podporuje širokú škálu protokolov ako HTTP, HTTPS, JXTA, SSL, TCP, UDP, XMPP a ďalšie. Z pohľadu servera je transportný konektor mechanizmus na čakanie a prijatie spojenia od klientov. ActiveMQ server dokáže súčasne podporovať niekoľko transportných protokolov, pre každý čakajúc na spojenie na inom porte. Každý konektor má okrem mena definovanú adresu servera. Na adresovanie servera sa používa identifikátor Uniform Resource Identifier (URI). URI konektora je potom použitý na vytvorenie spojenia so serverom na strane klienta.

Sieťový konektor slúži na vzájomnú komunikáciu serverov a vytváranie clusterov serverov. Takáto komunikácia môže byť jednosmerná aj obojsmerná. V prípade jednosmernej komunikácie sa jedná o preposielanie správ z jedného servera na iný server. V niektorých prípadoch ale jednosmerná komunikácia nie je dostatočná, a preto je potrebný obojsmerný komunikačný kanál medzi dvomi servermi.

Clustering

Pri použití clusteringu ActiveMQ podporuje fail-over klienta pri výpadku servera na iný server v clusteri. Cluster môže byť definovaný staticky alebo dynamicky, rovnako sa delia aj spôsoby hľadania dostupných serverov. Klienti hľadajú dostupné servery, s ktorými by mohli nadviazať spojenie, servery zas hľadajú dostupné servery pre vytvorenie clusteru. Pre staticky definovaný cluster je potrebné vedieť adresy všetkých serverov. Tieto adresy tvoria jeden zložený identifikátor URI. Oproti tomu v dynamicky definovanom clusteri sú dostupné servery a služby hľadané pomocou agentov.

ActiveMQ poskytuje možnosť vytvárania párov serverov, ktoré sa nazývajú *master/slave*. Klient je pripojený k serveru *master* a v prípade jeho výpadku nastane okamžitý fail-over na server *slave* bez straty správ. Správy sú medzi servermi priebežne replikované, prípadne je použité zdieľané úložisko dát.

Perzistencia

ActiveMQ poskytuje niekoľko spôsobov perzistencie dát, z ktorých je možné si vybrať – relačná databáza, žurnál v súborovom systéme alebo dokonca databáza v operačnej pamäti. Implicitným nastavením perzistencie je použitie ActiveMQ úložiska správ.

Úložisko AMQ (AMQ Message Store) je transakčné úložište založené na súboroch, navrhnuté pre veľmi rýchle a jednoduché ukladanie správ. Rýchlosť úložiska AMQ zabezpečuje kombinácia troch súčastí, ktorými sú transakčný žurnál, optimalizované úložište referencií správ a vyrovnávacia pamäť. Transakčný žurnál obsahuje záznamy správ a príkazov uložené v súboroch, umožňuje spoľahlivú perzistenciu, ktorá zvládne poruchu systému. Po dosiahnutí danej maximálnej dĺžky aktuálneho súboru je vytvorený nový súbor. Pre každú správu v žurnále je udržiavaný počet referencií, súbor môže byť odstránený až vtedy, keď žiadne dáta v ňom už nie sú potrebné. Referencie na správy v žurnále sú v úložišti referencií indexované identifikátorom správy. Vyrovnávacia pamäť obsahuje správy pre rýchly prístup po ich zapísaní do žurnálu. Pravidelne aktualizuje úložište referencií na základe identifikátorov správ, ktoré práve obsahuje a ich pozícii v žurnále. Po každej aktualizácii môžu byť správy z vyrovnávacej pamäte odstránené.

Iným typom úložišťa správ založeným na súboroch je *Kaha Message Store*. Oproti úložisku AMQ je približne o polovicu rýchlejšie, ale na úkor spoľahlivosti, pretože nepodporuje zotavenie po poruche systému.

Ďalším spôsobom perzistencie dát v ActiveMQ je použitie relačnej databázy pomocou

rozhrania JDBC. Predvolenou databázou je Apache Derby[18]. Medzi ostatné podporované databázy patria: MySQL, PostgreSQL, Oracle, SQLServer, Sybase, Informix[19] a MaxDB[20]. Tento spôsob perzistencie je možné použiť aj v kombinácii so žurnálom, ktorý môže výrazne zvýšiť výkonnosť servera.

V prípade, kedy je potrebné ukladať len konečné množstvo správ na krátku dobu, je možné použiť úložište dát v operačnej pamäti. Vhodné je ale len pre malé testovacie prípady, pretože nezahŕňa ukladanie dát do vyrovnávacej pamäte a je nutné dávať pozor na limity pamäte aj JVM.

Administrácia

ActiveMQ poskytuje niekoľko spôsobov administrácie – JMS alebo JMX API pre komunikáciu so serverom, pomocné správy a interné aj externé administračné nástroje.

Pomocné riadiace správy (advisory messages) reprezentujú administratívne príkazy, ktoré môžu oznamovať klientom výskyt dôležitých udalostí na serveri. Tieto správy sú odosielané do špeciálnych topikov, každý topik určený na nejaký typ správ. Klient sa potom môže pripojiť k tým topikom, ktoré obsahujú pre neho zaujímavé správy.

Interné administračné nástroje sú zahrnuté v binárnej distribúcii ActiveMQ. K nástrojom príkazového riadku patrí skript *activemq-admin*. Tento skript slúži na spustenie a zastavenie servera ActiveMQ, vymenovanie dostupných serverov, dotazovanie sa na informácie týkajúce sa stavu servera a prehliadanie destinácií na serveri. Ďalším interným nástrojom je agent, ktorý umožňuje vydávať administračné príkazy serveru pomocou jednoduchých JMS správ – tzv. *command agent*. Tento agent čaká na správy s príkazmi v špecifickom topiku. Po spracovaní príkazov posiela výsledky do rovnakého topiku. Nástroj *web console* poskytuje základné riadiace funkcie pomocou webového prehliadača. Príkladom externého nástroja slúžiaceho na administráciu servera je klientska aplikácia *JConsole*.

2.3.4 MRG Messaging

MRG Messaging[21] [18] je súčasťou produktu Enterprise MRG od spoločnosti Red Hat, ktorého ďalšími súčasťami sú Realtime a Grid. MRG Messaging je open source systém komunikácie pomocou správ založený na projekte Apache Qpid, ktorý používa na ukladanie, preposielanie a distribúciu správ. Implementuje špecifikáciu *Advanced Message Queuing Protocol (AMQP)* [17]. AMQP je otvorený protokol aplikačnej vrstvy pre Message Oriented Middleware (MOM). Na rozdiel od JMS nešpecifikuje API, ale formát posielaných dát. MRG Messaging poskytuje dve navzájom kompatibilné implementácie. Prvá z nich je napísaná v programovacom jazyku Java, je prenosná na rôzne platformy a operačné systémy. Druhá implementácia je optimalizovaná na operačný systém Linux a je napísaná v C++.

Clustering

Clustering v MRG Messaging je implementovaný pomocou frameworku OpenAIS[22], ktorý poskytuje spoľahlivý multicast. Dva MRG Messaging servery sú spolu v clusteri, keď majú

[18] http://db.apache.org/derby
[19] http://www-01.ibm.com/software/data/informix
[20] http://maxdb.sap.com
[21] http://www.redhat.com/products/mrg/
[22] freecode.com/projects/openais

rovnaké meno clusteru a používajú rovnakú OpenAIS multicastovú adresu, UDP port a adresu, na ktorú sú servery naviazané.

Zmeny na ľubovoľnom serveri v clusteri sú replikované na všetky ostatné servery v danom clusteri. Cluster môže poskytovať svoje služby v prípade, kedy je viac ako polovica jeho uzlov aktívna. V prípade, kedy uzol stratí kontakt so zvyškom clusteru, sa kvôli zabráneniu vzniku nekonzistencií MRG server v tomto uzle vypne. Klienti sa potom pripoja k zvyšku clusteru, ak ostala viac ako polovica uzlov aktívna. Toto chovanie zabraňuje rozdeleniu clusteru pri poruche siete. Dve časti clusteru by spolu nemohli komunikovať, ale pristupovali by k zdieľaným zdrojom, čo by mohlo viesť k narušeniu integrity.

Perzistencia

Na ukladanie perzistentných správ a iných dát používa MRG Messaging žurnál, ktorý je implementovaný ako cyklická fronta na disku s vyrovnávacou pamäťou pre zápis a čítanie v operačnej pamäti. Každá fronta má vlastný žurnál, do ktorého je zaznamenaný príchod správ, ich vyňatie z fronty a transakčné operácie. Jedna cyklická fronta na disku pozostáva z množiny súborov. Súbory žurnálu sú pripravené pri deklarácii asociovanej fronty, čo spôsobuje dlhšiu dobu potrebnú pre deklarovanie fronty, ale zaručuje alokovanie dostatočného priestoru.

Vyrovnávacie pamäte sú stránkované. Veľkosť stránky ovplyvňuje výkonnosť systému – malé stránky znižujú oneskorenie, väčšie stránky zas zvyšujú priepustnosť zmenšením počtu operácií zápisu na disk. Pri príchode perzistentnej správy do fronty je najprv správa zapísaná do vyrovnávacej pamäte pre zápis. Stránky vyrovnávacej pamäte sa plnia, kým nedosiahnu maximálnu veľkosť alebo po dopredu určenú dobu. Potom je stránka zapísaná do cyklickej fronty na disku. Vo vyrovnávacej pamäti pre zápis sa nachádzajú správy, ktoré príjemca nemôže prečítať, pretože ešte neboli potvrdené odosielateľovi.

Administrácia

MRG Messaging poskytuje niekoľko nástrojov príkazového riadka na administráciu a diagnostiku servera. Sú to nasledujúce nástroje:

- *qpid-config* – zobrazenie a konfigurácia servera, front a väzieb,
- *qpid-tool* – prístup k nastaveniam, štatistikám a ovládaniu servera,
- *qpid-queue-stats* – monitorovanie veľkostí front a pomerov počtov príchodov správ do front a ich vyňatí,
- *qpid-cluster* – konfigurácia a prezeranie informácií o clusteroch, ich serveroch, odpojenie klientov, ukončenie servera v clusteri alebo ukončenie celého clusteru,
- *qpid-route* – konfigurácia siete serverov spojených mostmi a smerovania medzi nimi, zistenie informácií o ich stave a topológii,
- *qpid-perftest* – meranie priepustnosti pre rozličné scenáre pomocou nastaviteľných parametrov,
- *qpid-stat* – zobrazenie štatistík pre rôzne objekty servera,
- *qpid-printevents* – predplatenie udalostí na serveri a výpis detailov o týchto udalostiach,

- *qpid-cluster-store* – obnovenie perzistentných dát po nie čistom zastavení clusteru.

2.3.5 WebSphere MQ

WebSphere MQ[23] [1] od spoločnosti IBM je rodina komerčných softwarových produktov spadajúcich do skupiny MOM. Poskytuje vlastné API – *Message Queue Interface (MQI)*, ktoré poskytuje aplikácii možnosť pripojiť sa k správcovi front, otvoriť fronty a topiky a posielať a prijímať správy. Je dostupný na množstve platform nielen od IBM a prístupný z rôznych programovacích jazykov.

Objekty a ich správa

Na definíciu a správu WebSphere MQ objektov existuje niekoľko spôsobov – príkazy Programmable Command Formats (PCF), skriptovacie príkazy MQ Script (MQSC) alebo grafické rozhranie nástroja MQ Explorer. Medzi takto spravované objekty patria:

- *Správca front* – definuje atribúty ostatných objektov a poskytuje aplikáciám služby WebSphere MQ. Na to, aby mohla aplikácia využívať jeho služby, musí sa k nemu najprv pripojiť.
- *Fronty* – sú objekty, do ktorých aplikácie vkladajú správy alebo z nich správy vyberajú. Každá fronta patrí niektorému správcovi front, jednému správcovi front môže patriť niekoľko front s unikátnym názvom. Fronta musí byť pred jej použitím vytvorená a otvorená. Frontu je potrebné otvoriť na dopredu určené použitie – prezeranie správ, vyberanie správ, ukladanie správ, zisťovanie atribútov fronty, nastavovanie atribútov fronty. WebSphere MQ poskytuje niekoľko typov front – lokálne alebo vzdialené, zdieľané, dynamické a clusterové.
- *Objekty pre správu topikov* – sú objekty, ktoré umožňujú priradiť topikom špecifické atribúty.
- *Zoznamy mien* – obsahujú zoznamy názvov clusterov, front a objektov autentizačných informácií. O ich usporiadaní rozhoduje správca systému.
- *Služby* – sú objekty určujúce spôsob definície programov tak, aby boli vykonané pri spustení alebo zastavení správcu front. Môžu byť typu server alebo príkaz.
- *Prijímače* – sú procesy, ktoré prijímajú požiadavky od iných správcov front alebo klientskych aplikácií a spúšťajú príslušné kanály
- *Definície procesov* – obsahujú definície atribútov pre spustenie aplikácií, ktoré musí správca front poznať.
- *Kanály* – sú komunikačné spojenia medzi distribuovanými správcami front. Môžu byť jednosmerné na prenos správ od jedného správcu front k inému alebo obojsmerné na prenos príkazov od klienta správcovi front a odpovedí od správcu front klientovi.
- *Objekty autentizačných informácií* – obsahujú autentizačné informácie potrebné k šifrovanému prenosu správ pomocou SSL.

[23] http://www-01.ibm.com/software/integration/wmq/

Správy

Správy sa skladajú z vlastností a aplikačných dát. Správca fronty sa zaujíma len o formát vlastností, ale aplikáciu, ktorá spracúva správu, zaujíma aj formát samotných dát. Ak je správa posielaná medzi rôznymi platformami, môže byť potrebné aplikačné dáta skonvertovať do inej kódovej sady znakov. Takáto konverzia môže prebiehať v odosielajúcom alebo prijímajúcom správcovi front.

WebSphere MQ definuje štyri typy správ, z ktorých prvé tri sú používané aplikáciami na výmenu informácií, posledný typ sa používa najmä na oznamovanie výskytu chyby. Je možné definovať si aj vlastné typy správ. Základné typy správ sú nasledujúce:

- *datagram* – správa, na ktorú nie je očakávaná odpoveď,
- *požiadavka* – správa, na ktorú je očakávaná odpoveď,
- *odpoveď* – odpoveď na požiadavku,
- *oznam* – správa, ktorá oznamuje výskyt udalosti.

Pri použití point-to-point modelu komunikácie poskytuje WebSphere MQ možnosť zoskupenia a segmentácie správ. Zoskupenie správ je definované spoločným atribútom *GroupId* a je zložené z viacerých logických správ číslovaných v rámci jednej skupiny. Logické správy sa používajú pre zaistenie poradia správ a umožňujú aplikáciám zoskupiť podobné správy. Segmentácia správ slúži na rozdelenie veľkých správ. Každá logická správa je zároveň jednou fyzickou správou, ak nie je rozdelená na segmenty. Inak pozostáva zo segmentov tvoriacich jednu fyzickú správu. Každý segment môže byť časťou práve jednej správy.

Perzistentné správy sú zapisované do záznamu, z ktorého sú obnovené po reštarte správcu front, ak sa vyskytne chyba alebo ak bol správca front ukončený. Keďže zápis na disk si vyžaduje viac času, perzistentné správy je vhodné použiť len vtedy, keď je to nutné. Naopak ne-perzistentné správy sa používajú na rýchlu komunikáciu.

Klient

WebSphere MQ klient je samostatný komponent WebSphere MQ produktu. Môže byť inštalovaný na počítači bez bežiaceho správcu front, kde umožňuje aplikácii spojiť sa so správcom front na inom systéme – serveri. Klientska aplikácia komunikuje so správcom front na serveri cez MQI kanál, ktorý musí aplikácia udržiavať v aktívnom stave po celú dobu, kedy ho chce využívať.

Použitie WebSphere MQ klienta na odlišnom stroji, ako na ktorom beží správca front, má svoje výhody. Nie je potrebná plná inštalácia WebSphere MQ na klientskom stroji, čím sa znižujú nároky na hardware aj administráciu systému. Okrem toho aplikácia, ktorá beží v klientskom prostredí, sa môže pripojiť k viacerým správcom front na rôznych systémoch a môže používať kanály s rôznymi transportnými protokolmi.

Vzájomná komunikácia

Vzájomná komunikácia predstavuje posielanie správ od jedného správcu front k inému. Lokálny správca front sa nazýva zdrojový, vzdialený sa nazýva cieľový. Cieľový správca front sa môže nachádzať na tom istom stroji, ale aj na inom stroji s inou platformou. V takom prípade sa jedná o distribuovaný systém.

Pri posielaní správy vzdialenému správcovi front, lokálny správca front správu najprv uloží do tzv. *prenosovej fronty (transmission queue)*. Prenosová fronta je špeciálny typ fronty, ktorý slúži na uloženie správ posielaných vzdialenému správcovi front, až kým nie sú pripravené na prenos. Prenos správ sa uskutočňuje cez jednosmerný kanál a je riadený agentom *Message Channel Agent (MCA)* na oboch koncoch kanála.

2.4 Aplikačný server JBoss

Aplikačný server JBoss (JBoss AS) [9] je open source Java EE aplikačný server – implementácia enterprise časti platformy Java. Keďže je napísaný v programovacom jazyku Java, je prenositeľný na všetky operačné systémy, pre ktoré existuje vhodný virtuálny stroj JVM.

Pôvodný projekt JBoss pokrýval len implementáciu EJB časti Java EE špecifikácie. JBoss AS verzie 3 je už plnohodnotný Java EE aplikačný server, to znamená, že pokrýva celú Java EE špecifikáciu. Nasledujúce verzie implementujú vždy novšiu verziu špecifikácie Java EE. V tejto práci je využitý JBoss AS verzie 5 (Java EE 5) a 7 (Java EE 6). Tieto verzie sa od seba líšia svojou architektúrou, konfiguráciou, spôsobom nasadzovania aplikácií a ďalšími vlastnosťami. V nasledujúcich častiach sú tieto rozdiely popísané.

2.4.1 JBoss AS 5

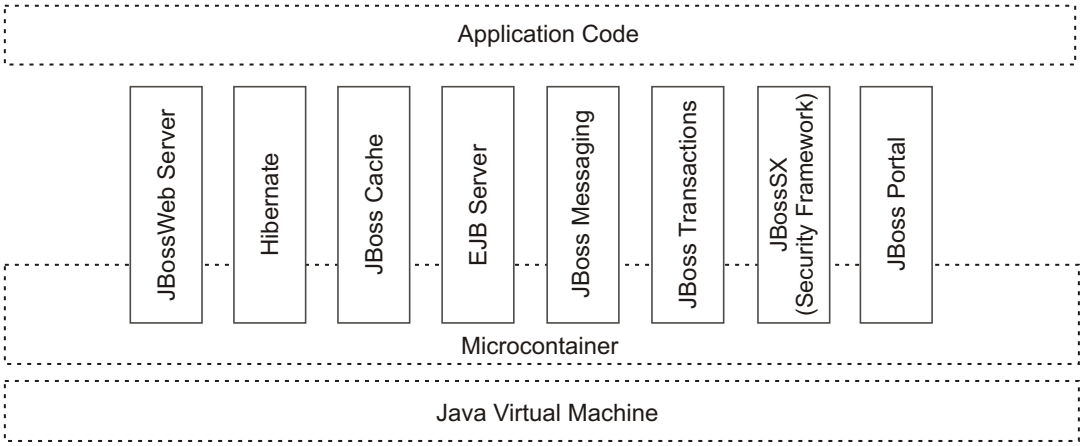
JBoss AS 5 je kolekciou nezávislých komponentov, z ktorých každý sa zameriava na konkrétnu oblasť Java EE funkcionality. Architektúra aplikačného servera je založená na mikrokontajneri a je znázornená na Obrázku 2.6, kde na najnižšej vrstve je JVM, nad ktorým sa nachádza mikrokontajner. Služby, ktoré aplikačný server poskytuje v podobe komponentov, sú implementované ako Plain Old Java Object (POJO) a pripojované do mikrokontajnera podľa potreby aplikácie. Mikrokontajner zabezpečuje komunikáciu komponentov s JVM a komunikáciu jednotlivých komponentov navzájom. Kód aplikácie nasadenej v aplikačnom serveri potom môže využívať rôzne služby pripojené do mikrokontajnera.

Nasleduje výpočet základných služieb, ktoré aplikačný server v podobe komponentov poskytuje.

- *JBoss Web Server* – je webový server založený na Apache Tomcat[24], poskytuje platformu pre webové technológie ako servlet, JavaServer Pages [12] alebo JavaServer Faces [12].
- *Hibernate* – zabezpečuje perzistenciu objektov pomocou objektovo-relačného mapovania.
- *JBoss Cache* – je transakčná, distribuovaná rýchla vyrovnávacia pamäť, ktorú využíva mnoho ďalších služieb JBoss AS.
- *EJB Server* – je implementácia EJB3 špecifikácie.
- *JBoss Messaging* – je JMS server, umožňuje synchrónnu a asynchrónnu komunikáciu pomocou zasielania správ.
- *JBoss Transactions* – je distribuovaná technológia poskytujúca transakčné spracovanie.

[24] tomcat.apache.org

- *JBossSX* – je deklaratívna bezpečnostná služba založená na roliach.
- *JBoss Portal* – je portálový server. Portál slúži na vytvorenie webových stránok pomocou portletov [12] – zložením rôznorodých kúskov zdrojového kódu, ktoré navonok pracujú ako jedna aplikácia.



Obrázok 2.6: Architektúra JBoss AS 5.

Množina služieb spolu s aplikáciami, ktoré sú pri štartovaní servera spustené, sa nazýva *konfigurácia aplikačného servera*. JBoss AS 5 poskytuje tri základné konfigurácie, ktoré sú popísané nižšie. Základné konfigurácie je možné si prispôsobovať a pridávať vlastné konfigurácie. Na druhú stranu existuje možnosť zmenšovania servera odstránením nadbytočných služieb. Ide o tzv. *slimming* a umožňuje spustenie len nevyhnutných služieb. Výhodou sú nižšie pamäťové nároky, vyššia rýchlosť, ale aj menšia náchylnosť k poruchám a vyššia miera bezpečnosti. Základné konfigurácie JBoss AS 5 sú nasledujúce:

- *default* – je implicitná konfigurácia, zahŕňa všetky služby nevyhnutné pre spustenie servera bez podpory clusteringu.
- *minimal* – obsahuje minimálnu množinu služieb vrátane mikrokontajnera a adresárovej služby JNDI.
- *all* – spúšťa všetky služby poskytované JBoss AS 5.

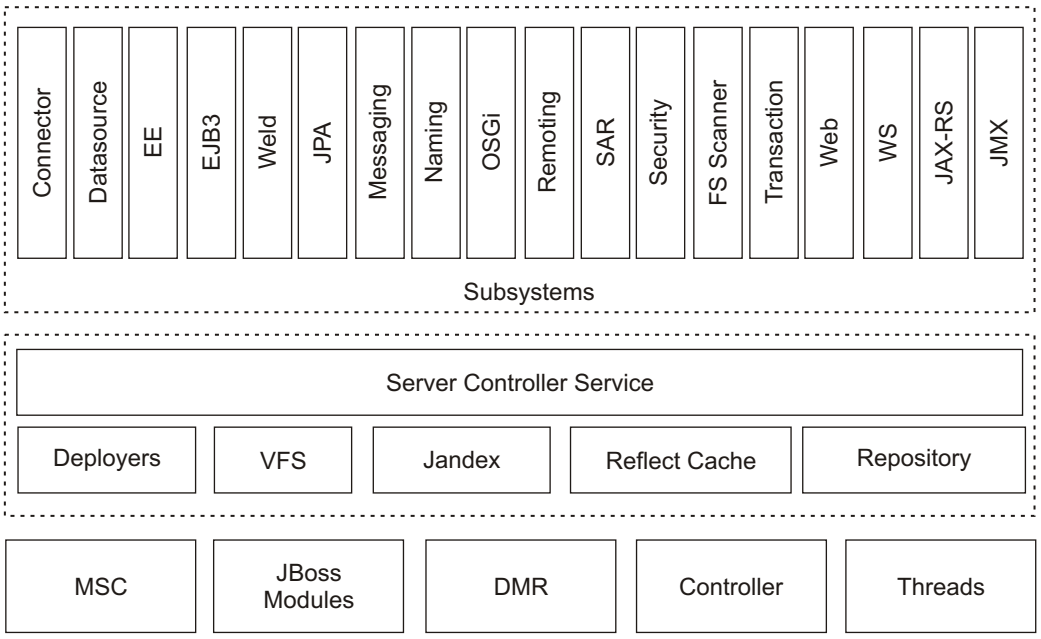
Všetky dostupné konfigurácie sú umiestnené v adresári `server/`, každá konfigurácia má svoj podadresár `deploy/`, kde sú nasadzované aplikácie a služby. Nasadenie aplikácie prebieha v dvoch fázach. Najprv je aplikačný server priamo či nepriamo upozornený na aplikáciu, ktorá má byť nasadená. V druhej fáze aplikačný server vykoná potrebné kroky na to, aby bola aplikácia pripravená na použitie. Najjednoduchším spôsobom nasadenia aplikácie na aplikačný server je jej nakopírovanie do adresára `deploy/` príslušnej konfigurácie. Ďalšou možnosťou nasadenia aplikácie je použitie administračnej konzoly JBoss AS. K administračnej konzole sa pristupuje na URL `http://localhost:8080/admin-console/`. Existujú dva typy nasadzovania aplikácií. V prípade, kedy je potrebné aplikačný server najprv vypnúť, aplikáciu nasadiť a až potom server znova spustiť, sa jedná o *cold deployment*. Pri tzv. *hot deployment* je možné aplikáciu nasadiť za behu aplikačného servera.

2.4.2 JBoss AS 7

Jadro aplikačného servera, hoci stavia na rovnakých základných projektoch, bolo od základu prepísané za účelom zjednodušenia konfigurácie, zvýšenia spravovateľnosti a použiteľnosti.

JBoss AS 7 môže byť spustený v dvoch režimoch – *standalone* a *domain*. JBoss AS v režime standalone je jediný proces v jednom JVM. Tento režim je podobný ako v predchádzajúcich verziách, umožňuje automatické nasadenie aplikácií skopírovaním do príslušného adresára a je vhodný pre vývoj. Režim domain umožňuje spúšťať a spravovať na jednom hostiteľskom počítači viac JBoss AS inštancií, každú vo vlastnom JVM, a spravovať ich z jedného miesta. Okrem procesu inštancie aplikačného servera obsahuje tento režim ďalšie dva procesy, a to *process controller* a *host controller*. Process controller je jednoduchý proces, ktorý spravuje životný cyklus procesov – ich spúšťanie a zastavovanie. Jeden host controller sa nazýva *domain controller* a je zodpovedný za spravovanie a konfiguráciu všetkých inštancií aplikačného servera.

JBoss AS 7 má oproti predchádzajúcim verziám novú architektúru. Obrázok 2.7 ukazuje tri vrstvy aplikačného servera. Subsystémy tvoriace vrchnú vrstvu sú rôzne služby, ktoré môžu aplikácie využívať, ako napríklad komunikáciu pomocou zasielania správ, transakcie, zabezpečenie apod. Subsystémy využívajú služby strednej vrstvy, ku ktorým patrí virtuálny súborový systém, služba na indexovanie anotácií alebo repozitár. Tieto služby využívajú služby najnižšej vrstvy. Architektúra JBoss AS 7 je postavená na *Modular Service Container (MSC)* a umožňuje spúšťať služby na požiadanie, to znamená, len keď ich aplikácia potrebuje. *JBoss Modules* je projekt, na ktorom je založené modulárne zavádzanie tried. *Dynamic Model Representation (DMR)* poskytuje centrálne API pre správu aplikačného servera. Všetky informácie sa nachádzajú v jednom konfiguračnom súbore, do ktorého sú ukladané vykonané zmeny. Pre režim standalone je to `standalone.xml` a pre režim domain zas `domain.xml`.



Obrázok 2.7: Architektúra JBoss AS 7.

2.5 Nástroj na správu projektov – Maven

Aplikácia Maven [8] je nástroj pre automatizovanú správu projektov. Pomáha v základných oblastiach, ktorými je potrebné zaoberať sa pri vývoji softwarových projektov okrem samotnej implementácie. Sú to podporné činnosti ako písanie dokumentácie, priebežné spúšťanie a vyhodnocovanie testov, aktualizácia webovej stránky a podobne. V tejto práci je aplikácia Maven využitá na preklad testovacej aplikácie a jej zabalenie do archívu EAR a ďalej na spustenie testov.

Konfigurácia celého projektu je založená na jazyku XML. Jadrom každého projektu, ktorý využíva Maven je tzv. *objektový model projektu* zapísaný v súbore `pom.xml`. Koreňový element modelu sa nazýva *project* a obsahuje vnorené všetky ostatné elementy obsahujúce informácie o projekte. Tieto elementy sa logicky delia do dvoch skupín, sú to základné charakteristiky a zostavovacie informácie. Medzi základné charakteristiky patria elementy definujúce názov a verziu projektu, informácie o organizácii, názov balíčku, ktorý obsahuje projekt, údaje o domovskej stránke projektu, spôsob pripojenia k systému správy zdrojových súborov, záznamy o vývojových vetvách projektu, informácie o vývojároch a licenciách týkajúcich sa projektu. Zostavovacie informácie zahŕňajú popis závislostí, parametre potrebné pre zostavenie projektu, určenie súborov s testami a zoznam zdrojov spojených s projektom a jeho testami. Operácie definované v objektovom modeli projektu nie sú realizované priamo aplikáciou Maven, ale pomocou zásuvných modulov. Maven spolupracuje s veľkým množstvom zásuvných modulov, ktoré sú špecializované na konkrétnu operáciu.

Knižnice, ktoré tvoria závislosti projektu, nemusia byť súčasťou každého projektu, ale sú ukladané v repozitároch. Závislosti majú definovaný vzdialený repozitár, z ktorého Maven danú knižnicu stiahne. Pri inštalácii aplikácie Maven je vytvorený lokálny repozitár, do ktorého sú sťahované knižnice ukladané pre budúce použitie. Pri hľadaní knižnice sa najprv prehľadá lokálny repozitár, až ked sa v ňom hľadaná knižnica nenájde, je stiahnutá zo vzdialeného repozitára.

Aplikácia Maven je spúšťaná z príkazového riadka. Pri spustení sú zadané ciele, ktoré majú byť vykonané. V tejto práci sú využité ciele:

- `clean` – vymaže výstup predchádzajúceho zostavenia projektu.
- `package` – preloží a zabalí projekt do definovaného balíčka (napríklad JAR).
- `install` – skopíruje artefakty projektu do lokálneho repozitára.
- `deploy` – skopíruje artefakty projektu do vzdialeného repozitára.
- `test` – spustí jednotkové testy.
- `site` – generuje dokumentáciu alebo správu s výsledkami testov.

Okrem preddefinovaných cieľov umožňuje Maven použiť aj užívateľské ciele alebo preddefinované ciele upraviť.

Kapitola 3

Postup integrácie JMS poskytovateľov do aplikačného servera JBoss

Integrácia JMS poskytovateľov do aplikačného servera je dôležitá v prípade, kedy sa namiesto použitia predvoleného JMS poskytovateľa pre daný aplikačný server vyžaduje použitie JMS poskytovateľov tretích strán. V tejto kapitole sú popísané konkrétne postupy integrácie vybraných JMS poskytovateľov tretích strán do open source aplikačného servera JBoss (JBoss AS). Pre túto úlohu je použitý JBoss AS verzie 5 a 7 a adaptéry zdrojov od jednotlivých JMS poskytovateľov. JBoss ponúka dvoch JMS poskytovateľov. Vo verzii 5 je predvoleným JMS poskytovateľom JBoss Messaging, vo verzii 7 je to HornetQ. HornetQ je možné použiť aj v JBoss AS 5 po nainštalovaní pomocou predpripraveného inštalačného balíčka. Kapitola popisuje integráciu troch vybraných JMS poskytovateľov tretích strán. V prvej podkapitole je popísaná integrácia WebSphere MQ od IBM, druhá podkapitola je venovaná ActiveMQ od Apache a posledným JMS poskytovateľom je Red Hat MRG Messaging. Ku každému poskytovateľovi je uvedený postup od inštalácie potrebného softwaru až po jednoduché otestovanie funkčnosti pomocou komponentu Message Driven Bean (MDB).

3.1 Postup integrácie WebSphere MQ

Prvým krokom integrácie WebSphere MQ je inštalácia a konfigurácia WebSphere MQ podľa konkrétnych požiadaviek používajúcej aplikácie. Sem patrí vytvorenie WebSphere MQ objektov. Ďalej je potrebné nasadiť WebSphere MQ adaptér zdrojov do JBoss AS. Posledným krokom je konfigurácia odchodzej a príchodzej komunikácie. V nasledujúcich častiach sú tieto kroky podrobne popísané a vysvetlené pre JBoss AS 5. Nízkoúrovňové kroky závislé od operačného systému a ďalších vlastností prostredia nie sú súčasťou textu, všetky použité skripty písané pre operačný systém Linux sú uvedené v prílohe.

Inštalácia a konfigurácia WebSphere MQ

V tejto práci je použitý WebSphere MQ verzie 7.5, ktorého skúšobnú verziu je možné stiahnuť zo stránok IBM[1]. Administráciu WebSphere MQ môže vykonávať len užívateľ *mqm* alebo užívateľ zo skupiny *mqm*, preto je potrebné najprv vytvoriť tohto užívateľa, prípadne

[1] http://www14.software.ibm.com/webapp/download/search.jsp?pn=WebSphere+MQ

skupinu. Inštalácia podľa dokumentácie v [1] vytvorí v adresároch `/opt/mqm` a `/var/mqm` potrebné súbory. Prvý z adresárov obsahuje samotnú inštaláciu s dokumentáciou, binárnymi súbormi, ukážkovými programami a podobne. Do druhého adresára budú ukladané dáta z behu WebSphere MQ ako použité objekty alebo logovacie súbory s prípadnými chybovými hláškami.

Po nainštalovaní WebSphere MQ je možné vytvoriť objekty, ktoré sú potrebné pre vlastnú aplikáciu. Všetky objekty je možné vytvárať rôznymi spôsobmi. V práci sú použité skriptovacie príkazy MQSC, ktoré sú výhodné pre automatizovanú konfiguráciu. Prvým objektom, ktorý je nevyhnutný pre každú aplikáciu, je správca front. Správca front sa vytvára príkazom `crtmqm <názov správcu front>`, ďalej je potrebné ho spustiť pomocou príkazu `strmqm <názov správcu front>`, aby bolo možné pripájať sa k nemu a používať ho. Definícia objektov patriacich danému správcovi front sa vykonáva v MQSC konzole. Táto konzola sa spúšťa pre konkrétneho správcu front príkazom `runmqsc <názov správcu front>`. K týmto objektom patria fronty, topiky, kanály, prijímače a autentizačné informácie. Z MQSC konzole sa vychádza príkazom `end`. Definícia základných objektov v MQSC konzole môže vyzerať nasledovne:

- `define qlocal(Q1)` – vytvorí lokálnu frontu `Q1`.
- `define topic (T1) topicstr (T1)` – vytvorí topik `T1` s reťazcom topiku `T1`.
- `define listener (QML) trptype(TCP) ipaddr(10.16.88.122) port(1414) control(QMGR)` – vytvorí prijímač `QML`, ktorý bude čakať na požiadavky na IP adrese 10.16.88.122 a TCP porte 1414 a bude typu `QMGR`.
- `start listener (QML)` – spustí vytvorený prijímač `QML`.
- `define channel (CH) chltype(SVRCONN) trptype(TCP) sslcauth(OPTIONAL) mcauser('mqm')` – vytvorí TCP kanál s názvom `CH` typu `SVRCONN` s možnosťou použiť zabezpečený transport cez SSL pre užívateľa mqm.
- `set chlauth(*) type(BLOCKUSER) userlist(*MQADMIN) action(REMOVE)` – odstráni autentizačné informácie typu `BLOCKUSER` zo všetkých kanálov. Tento príkaz spôsobí odstránenie zabezpečenia prenosu dát.

Automatizáciu vytvárania a správy WebSphere MQ objektov umožňujú MQSC skripty, ktoré môžu obsahovať príkazy MQSC konzoly.

Nasadenie adaptéra zdrojov

Adaptér zdrojov pre WebSphere MQ sa nachádza priamo v inštalácii WebSphere MQ, v adresári `/opt/mqm/java/lib/jca` a nazýva sa `wmq.jmsra.rar`. Tento súbor je kľúčovým komponentom, ktorý obsahuje kód integrácie medzi aplikačným serverom a systémom posielania správ WebSphere MQ. Adaptér zdrojov je možné nasadiť na aplikačný server ľubovoľným zo spôsobov popísaných v kapitole 2.4.1.

Po nasadení adaptéra zdrojov je samotná integrácia hotová, ostáva správne nakonfigurovať aplikáciu nasadenú v JBoss AS tak, aby bola schopná komunikovať s WebSphere MQ serverom cez nasadený adaptér zdrojov. Konfigurácia príchodzej komunikácie je popísaná v nasledujúcej časti.

Konfigurácia príchodzej komunikácie

Pre príchodziu komunikáciu od WebSphere MQ k aplikácii v aplikačnom serveri je potrebná konfigurácia nasadeného adaptéra zdrojov. Jedným zo spôsobov je použitie komponentu MDB, ktorý bude prijímať správy od WebSphere MQ. Ďalšími komponentmi, ktoré je možné použiť, sú napríklad servlet alebo MBean. Táto časť je zameraná na MDB.

Na konfiguráciu adaptéra zdrojov v zdrojovom kóde MDB slúžia anotácie. Minimálna konfigurácia pre prijímanie správ z fronty `Q1` použitím správcu front `QM` je popísaná na Obrázku 3.1.

```
@MessageDriven(name = "test/mdb", activationConfig = {
    @ActivationConfigProperty(propertyName="messagingType",
                              propertyValue="javax.jms.MessageListener"),
    @ActivationConfigProperty(propertyName="destinationType",
                              propertyValue="javax.jms.Queue"),
    @ActivationConfigProperty(propertyName="destination",
                              propertyValue="Q1"),
    @ActivationConfigProperty(propertyName="queueManager",
                              propertyValue="QM"),
    @ActivationConfigProperty(propertyName="transportType",
                              propertyValue="CLIENT") })
@ResourceAdapter("wmq.jmsra.rar")
```

Obrázok 3.1: Anotácie pre konfiguráciu adaptéra zdrojov v MDB pre WebSphere MQ.

Anotácie `@ActivationConfigProperty` okrem iného definujú typ (`destinationType`) a názov (`destination`) destinácií, kanály (`channel`), správcov front (`queueManager`), autentizačné informácie (`user` a `password`), spôsob transportu dát (`transportType`), adresu (`hostName`) a port (`port`), na ktorých čaká prijímač. Anotácia `@ResourceAdapter` určuje použitý adaptér zdrojov, v tomto prípade `wmq.jmsra.rar`. Rovnaký výsledok má použitie XML súborov nazývaných deployment descriptor.

Hotovú skompilovanú aplikáciu je možné nasadiť na aplikačný server. Z WebSphere MQ sa dá správa poslať napríklad pomocou nástroja MQ Explorer alebo použitím programu `/opt/mqm/samp/bin/amqsput`. Po odoslaní správy do fronty `Q1` MDB správu prijme a spracuje podľa kódu v metóde `onMessage()`. V testovacej MDB je v tele tejto metódy výpis správy do konzoly.

3.2 Postup integrácie ActiveMQ

Ďalším JMS poskytovateľom je ActiveMQ. Spolupráca ActiveMQ s JBoss AS je možná dvomi spôsobmi. Prvým spôsobom je použitie ActiveMQ servera nasadeného vovnútri aplikačného servera a druhým je externé použitie ActiveMQ servera. Obidva spôsoby vyžadujú JCA adaptér zdrojov, druhý navyše vyžaduje inštaláciu ActiveMQ servera. V tejto časti je popísaný prvý spôsob, a to použitie ActiveMQ vovnútri JBoss AS bez potreby inštalácie externého ActiveMQ servera. V práci je použitý ActiveMQ verzie 5.6.0, ktorý je voľne dostupný na stránkach Apache[2].

[2] http://activemq.apache.org/activemq-560-release.html

Postup integrácie je obdobný, ako to bolo pri WebSphere MQ. Adaptér zdrojov je najprv potrebné nainštalovať do aplikačného servera. Ďalším krokom je vytvorenie a konfigurácia JMS objektov. Nakoniec je nakonfigurovaná príchodzia a odchodzia komunikácia pomocou MDB.

Nasadenie a konfigurácia adaptéra zdrojov

Adaptér zdrojov pre ActiveMQ je možné stiahnuť zo stránok Apache[3]. Na tento súbor je odkazované zo zdrojového kódu MDB. Z tohto dôvodu je vhodné ho premenovať na `activemq-ra.rar`.

Adaptér zdrojov je nakonfigurovaný pre transport pomocou protokolu virtuálneho stroja (VM protokol), kedy komunikácia prebieha v rámci jedného JVM. Táto konfigurácia sa nachádza v súbore `activemq-ra.rar/META-INF/ra.xml`, konkrétne ide o vlastnosť `ServerUrl`, ktorá musí byť nastavená na hodnotu `vm://localhost`. Ďalej je potrebné nastaviť cestu k súboru s konfiguráciou pre vnútorné použitie servera ActiveMQ, teda vlastnosť `BrokerXmlConfig` na hodnotu `broker-config.xml`. V tomto súbore je element <`kahaDB`>, ktorého atribút `directory` je nutné upraviť tak, aby ukazoval na existujúci adresár, napríklad `data/activemq` v adresári aplikačného servera.

Definícia JMS objektov

Pre otestovanie príchodzej a odchodzej komunikácie je potrebná jedna fronta a objekt `ConnectionFactory`. Tieto objekty je možné definovať v konfiguračnom súbore aplikačného servera `standalone/configuration/standalone-full.xml` ako je ukázané na Obrázku 3.2. Prvý element `<connection-definition>` definuje objekt `ConnectionFactory` dostupný cez JNDI pod záznamom `/activemq/ConnectionFactory` s veľkosťou bloku pripravených spojení minimálne 1 a maximálne 20 spojení. Druhý element `<admin-object>` definuje frontu so skutočným názvom `queue.queue_in` a s JNDI záznamom `/activemq/queue_in`.

Definovaním JMS objektov v konfiguračnom súbore je integrácia ActiveMQ do JBoss AS hotová. Po spustení aplikačného servera je možné prijímať správy odoslané serveru ActiveMQ do fronty `queue.queue_in`, prípadne do nej správy odosielať. Konfigurácia príchodzej komunikácie je popísaná v nasledujúcej časti kapitoly.

Konfigurácia príchodzej komunikácie

Podobne ako pri WebSphere MQ je pre otestovanie príchodzej komunikácie použitý komponent MDB. Znova sú použité anotácie, ActiveMQ vyžaduje minimálnu konfiguráciu, ktorá je na Obrázku 3.3. Jedná sa o definovanie spôsobu potvrdzovania správ (`acknowledgeMode`) a typu (`destinationType`) a názvu (`destination`) destinácie.

Takýto MDB čaká na správy vo fronte `queue.queue_in` a spracuje ich v závislosti od implementácie metódy `onMessage()`. Inštalácia servera ActiveMQ obsahuje príklady klientov, ktoré je možné použiť na zasielanie správ do tejto fronty.

3.3 Postup integrácie MRG Messaging

Posledným JMS poskytovateľom spomenutým v tejto kapitole je Red Hat MRG Messaging. Integrácia MRG s JBoss AS vyžaduje bežiaci MRG server. Na aplikačný server je potom

[3] http://repo1.maven.org/maven2/org/apache/activemq/activemq-rar/5.6.0/activemq-rar-5.6.0.rar

```
<connection-definition>
  <class-name "org.apache.activemq.ra.ActiveMQManagedConnectionFactory"/>
  <jndi-name"java:/activemq/ConnectionFactory"/>
  <enabled "true"/>
  <pool-name="ActiveMQConnectionFactoryPool"/>
  <use-ccm="true"/>
  <xa-pool>
    <min-pool-size>1</min-pool-size>
    <max-pool-size>20</max-pool-size>
  </xa-pool>
</connection-definition>
<admin-object class-name="org.apache.activemq.command.ActiveMQQueue">
  <jndi-name "java:/activemq/queue_in"/>
  <enabled "true"/>
  <pool-name "ActiveMQQueue.queue_in"/>
  <config-property name="PhysicalName">
    queue.queue_in
  </config-property>
</admin-object>
```

Obrázok 3.2: Definícia JMS objektov pre ActiveMQ.

```
@MessageDriven(name = "test/mdb", activationConfig = {
    @ActivationConfigProperty(propertyName = "acknowledgeMode",
                              propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName = "destinationType",
                              propertyValue = "javax.jms.Queue"),
    @ActivationConfigProperty(propertyName = "destination",
                              propertyValue = "queue.queue_in") })
@ResourceAdapter("activemq-ra.rar")
```

Obrázok 3.3: Anotácie pre konfiguráciu adaptéra zdrojov v MDB pre ActiveMQ.

potrebné nasadiť adaptér zdrojov a nakonfigurovať ho pre použitie s týmto MRG serverom. Príchodziu komunikáciu je možné zachytiť pomocou MDB, rovnako ako tomu bolo pri predchádzajúcich JMS poskytovateľoch.

Inštalácia a konfigurácia MRG Messaging

MRG nie je voľne dostupný systém, z toho dôvodu je pre testovanie použitá inštalácia MRG vo firme Red Hat. Tento MRG server beží na operačnom systéme Red Hat Enterprise Linux. Je súčasťou skupiny balíčkov *MRG Messaging*, je teda možné ho nainštalovať pomocou správcu softwarových balíčkov *yum*[4]. Pri inštalácii MRG servera je dôležité správne nastaviť bránu firewall tak, aby bola povolená príchodzia komunikácia na porte, na ktorom MRG čaká spojenie. Predvoleným portom pre komunikáciu pomocou AMQP je 5672.

[4] http://fedoraproject.org/wiki/Yum

Nasadenie a konfigurácia adaptéra zdrojov

Hlavným krokom v integrácii MRG s JBoss AS je nasadenie a konfigurácia adaptéra zdrojov. V súčasnosti MRG poskytuje adaptér zdrojov len pre C++ MRG server. Nachádza sa v balíčku `qpid-jca` štandardnej inštalácie MRG a je možné ho nasadiť na aplikačný server štandardným spôsobom. Rovnako je potrebné nasadiť JMS objekty s konfiguráciou v súbore `qpid-jca-ds.xml`, ako je ukázané na Obrázku 3.4. Prvý element `<mbean>` definuje objekt `ConnectionFactory` s JNDI menom `MRGConnectionFactory`. Atribút `ConnectionURL` špecifikuje vlastnosti spojenia s MRG serverom. Hodnoty `MRG_HOST` a `MRG_PORT` musia byť nastavené tak, aby ukazovali na spustený MRG server, v tomto prípade je to `mrg01.mw.lab.eng.bos.redhat.com:5672`. Druhá časť kódu ukazuje definíciu fronty `QueueIn` s JNDI záznamom `queue/queue_in`.

```
<connection-factories>
  <mbean code="org.jboss.resource.deployment.AdminObject"
      name="qpid.jca:name=MRGConnectionFactory">
    <attribute name="JNDIName">MRGConnectionFactory</attribute>
    <depends optional-attribute-name="RARName">
      jboss.jca:service=RARDeployment,name='qpid-ra.rar'
    </depends>
    <attribute name="Type">javax.jms.ConnectionFactory</attribute>
    <attribute name="Properties">
      ConnectionURL=
      amqp://guest:guest@/test?brokerlist='tcp://MRG_HOST:MRG_PORT'
    </attribute>
  </mbean>
  <mbean code="org.jboss.resource.deployment.AdminObject"
      name="qpid.jca:name=QueueIn">
      <attribute name="JNDIName">queue/queue_in</attribute>
      <depends optional-attribute-name="RARName">
        jboss.jca:service=RARDeployment,name='qpid-ra.rar'
      </depends>
      <attribute name="Type">
        org.apache.qpid.ra.admin.QpidQueue
      </attribute>
      <attribute name="Properties">
        DestinationAddress=queue_in;{create:always,node:{type:queue}}
      </attribute>
  </mbean>
</connection-factories>
```

Obrázok 3.4: Definícia JMS objektov pre MRG.

Konfigurácia príchodzej komunikácie

Obrázok 3.5 znázorňuje anotácie komponentu MDB pre komunikáciu so serverom MRG. Atribúty `acknowledgeMode`, `destinationType` a `destination` majú rovnaký význam ako pri predchádzajúci JMS poskytovateľoch. Navyše je tu atribút `ConnectionURL`, ktorý sa

musí zhodovať s nastavením na Obrázku 3.4.

```
@MessageDriven(name = "jms/QpidListener", activationConfig = {
    @ActivationConfigProperty(propertyName = "acknowledgeMode",
                              propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName = "destinationType",
                              propertyValue = "javax.jms.Queue"),
    @ActivationConfigProperty(propertyName = "destination",
                              propertyValue = "queue/queue_in"),
    @ActivationConfigProperty(propertyName = "ConnectionURL",
                              propertyValue =
    "amqp://guest:guest@/test?brokerlist='tcp://MRG_HOST:MRG_PORT'") })
@ResourceAdapter("qpid-ra.rar")
```

Obrázok 3.5: Anotácie pre konfiguráciu adaptéra zdrojov v MDB pre MRG.

Kapitola 4

Testovacia sada

Táto kapitola popisuje návrh a implementáciu testovacej sady, ktorej účelom je otestovať integráciu JMS poskytovateľov tretích strán do aplikačného servera JBoss popísanú v kapitole 3. Cieľom testov je zistiť, či sú títo JMS poskytovatelia schopní pracovať integrovaní v aplikačnom serveri. Testy sú zamerané na transakcie, výkonnosť, prácu v clusteri a splnenie podmienok vysokej dostupnosti. Transakčné testy overujú schopnosť JMS poskytovateľov spracovávať správy v transakciách. Účelom výkonnostných testov je možnosť porovnať jednotlivých JMS poskytovateľov z hľadiska rýchlosti spracovania veľkého množstva správ. Clusterové testy využívajú cluster aplikačných serverov a testujú schopnosť rozdeľovať záťaž medzi jednotlivé uzly v clusteri. Testy vysokej dostupnosti pridávajú k jednoduchému clusterovému testu zastavenie jedného z uzlov a testovanie schopnosti druhého uzla prevziať prácu po zastavenom uzle. Testovacia sada je podľa uvedeného zamerania rozdelená na dve hlavné časti. Prvú časť tvoria transakčné testy, ktoré sú použité aj na meranie výkonnosti a v druhej časti sú clusterové testy, ku ktorým patria aj testy vysokej dostupnosti.

Kapitola sa skladá z troch podkapitol. Prvá podkapitola vysvetľuje návrh testov a architektúru testovacej sady, popisuje formát posielaných správ a návrh databázy použitej na ukladanie správ, popisuje testovacie scenáre jednotlivých testov a očakávaný výsledok každého testu. Ďalšia časť tejto kapitoly je venovaná samotnej implementácii. Je tu uvedený popis modulov, tried, významných operácií testovacej aplikácie a potrebné úpravy pri prechode z JBoss AS 5 na JBoss AS 7. V poslednej podkapitole je popis troch možných spôsobov spustenia testov od plne manuálneho po plne automatizovaný.

4.1 Návrh

Na posielanie správ medzi aplikačným serverom a JMS serverom sú využité komponenty Java servlet a Message Driven Bean (MDB). Servlet slúži ako komponent, ktorý v požiadavke dostane názov testu, ktorý je spustený. Na základe konkrétneho testu posiela správy do vstupnej fronty na JMS serveri a po prijatí odpovede z výstupnej fronty vypíše výsledok testu. MDB nasadený v aplikačnom serveri asynchrónne prijíma správy, ktoré posiela servlet do vstupnej fronty a spracuje ich podľa požiadaviek daného testu. Naspäť posiela potvrdenia o prijatí a ďalšie informácie do výstupnej fronty.

Uvedená funkcionalita je spoločná bez rozdielu pre všetkých JMS poskytovateľov a je obsiahnutá v triedach komponentov MDB a servlet a v triedach obsahujúcich logiku testov. Triedy potrebné pre každého JMS poskytovateľa zvlášť sú potomkovia generických tried obsahujúcich spoločnú funkcionalitu. Špecifické triedy sú potrebné pri vytváraní JMS ob-

jektov `ConnectionFactory` a `Destination`, ktoré musia byť získané od JMS poskytovateľa. Okrem toho má každý JMS poskytovateľ iný URL, na ktorom servlet čaká požiadavky.

4.1.1 Formát posielaných správ

V testoch sú použité textové JMS správy a skladajú sa z niekoľkých častí. Všetky testy pre jednoduchosť spracovania používajú jednotný formát správ. Použitý formát posielaných správ spolu s príkladmi je uvedený v Tabuľke 4.1 a má päť častí oddelených medzerami. Prvý riadok tabuľky ukazuje všeobecný formát s tromi povinnými časťami a s dvomi časťami pre špecifické použitie. Prvá časť obsahuje názov testu, v rámci ktorého je správa poslaná. Druhá časť je použitá len v prvej a poslednej správe v každom teste, prvá správa obsahuje reťazec `First Message`, posledná správa obsahuje reťazec `Last Message` a ostatné správy túto časť neobsahujú vôbec. Ďalšia časť je uvedená znakom `#`, za ktorým nasleduje identifikátor skupiny správ, ktorý musí byť rovnaký pre všetky správy v rámci jedného testu. Štvrtá časť začínajúca znakom `$` obsahuje index správy – poradie, v ktorom bola správa odoslaná číslované od nuly. Poslednú časť správy pridáva MDB pri posielaní odpovede. Ostatné riadky tabuľky ukazujú konkrétne príklady správ. Napríklad správa s textom `transactions01 #1366743556040 $1` je druhá správa posielaná servletom v teste `transactions01` a správa `transactions01 Last Message #1366743556040 $99 ACK` je odpoveď od MDB na poslednú správu v teste `transactions01`, v ktorom bolo spolu poslaných 100 správ.

všeobecný formát	`test< [First|Last] Message> #correlation $index< ACK>`
prvá správa	`transactions01 First Message #1366743556040 $0`
	`transactions01 First Message #1366743556040 $0 ACK`
druhá správa	`transactions01 #1366743556040 $1`
	`transactions01 #1366743556040 $1 ACK`
posledná správa	`transactions01 Last Message #1366743556040 $99`
	`transactions01 Last Message #1366743556040 $99 ACK`

Tabuľka 4.1: Ukážka formátu posielaných textových správ.

4.1.2 Databáza

Testovacia sada obsahuje testy, ktorých súčasťou testovacieho scenára je ukladanie správ do relačnej databázy. MDB ukladá do databázy prijaté správy a servlet kontroluje ich počet a prípadné duplicity. Databáza obsahuje jednu tabuľku s tromi stĺpcami – jednoznačný identifikátor správy, identifikátor skupiny správ a celý text správy. Ako identifikátor skupiny správ je použitá tretia časť správy začínajúca znakom `#`. Tento stĺpec tabuľky slúži na kontrolu správ na strane servletu. Nasleduje rozbor jednotlivých testov a testované scenáre rozdelené do už spomínaných dvoch častí na transakčné a clusterové testy.

4.1.3 Transakčné testy

V prvej časti testovacej sady je použitý jeden JMS server a jeden aplikačný server. Tieto testy využívajú transakčné spracovanie a okrem toho sú tu aj testy zamerané na meranie výkonnosti.

Test `mdbMessagingTransactions`

Test `mdbMessagingTransactions` je jednoduchý transakčný test. Servlet pošle do vstupnej fronty `MESSAGE_COUNT` správ. MDB tieto správy prijme a na každú pošle naspäť potvrdenie o prijatí. Servlet skontroluje, či boli doručené a potvrdené všetky správy a skontroluje, či sa každá správa doručila a potvrdila práve jedenkrát.

Tabuľka 4.2 obsahuje ukážky výstupov tohto testu. Na prvom riadku je reťazec, ktorý predstavuje výsledok testu. Začína znakmi `##` a obsahuje štyri položky oddelené znakmi `@@`. Prvá položka (`sprav`) obsahuje počet prijatých potvrdení od MDB a mala by byť zhodná s počtom odoslaných správ `MESSAGE_COUNT`. Druhá položka (`chyb`) predstavuje počet chýb, ktoré sa pri prijímaní vyskytli a mala by byť rovná nule. Tretia položka hovorí, či sa počet prijatých a odoslaných správ rovná. Štvrtá položka je výsledok kontroly výskytov. Kontrola výskytov je úspešná, pokiaľ bola každá správa doručená a potvrdená práve jedenkrát, v opačnom prípade sú okrem výsledného reťazca vypísané aj všetky správy s nesprávnym počtom výskytov. Správna hodnota tretej a štvrtej položky je `true`.

Na druhom riadku je ukážka jediného správneho výstupu pre `MESSAGE_COUNT=100`. Znamená to, že bolo prijatých 100 správ, nevyskytla sa žiadna chyba a každá správa bola potvrdená práve jedenkrát. Príklady nesprávnych výstupov sú na treťom, štvrtom a piatom riadku. V prvom prípade sa vyskytla nejaká chyba pri doručení správy. V druhom prípade neboli prijaté všetky správy a kvôli tomu majú posledné dve položky hodnotu `false`. V poslednom prípade bolo prijatých 100 správ a tretia položka má hodnotu `true`. Avšak môže sa stať, že niektorá správa nebola doručená a naopak, niektorá bola doručená viacnásobne, vďaka čomu sa počty prijatých a odoslaných správ rovnajú, ale kontrola výskytov skončí s výsledkom `false`.

výsledok	`##sprav@@chyb@@prijate=odoslane@@kontrola_vyskytov`
správne	`##100@@0@@true@@true`
nesprávne	`##100@@1@@true@@true`
nesprávne	`##99@@0@@false@@false`
nesprávne	`##100@@0@@true@@false`

Tabuľka 4.2: Tabuľka výsledkov testu `mdbMessagingTransactions`

Test `mdbEjbTransactions`

Tento test oproti predchádzajúcemu využíva navyše relačnú databázu na uloženie správ. MDB uloží všetky prijaté správy do databázovej tabuľky a na poslednú pošle potvrdenie do výstupnej fronty. Servlet prijme potvrdenie z výstupnej fronty, prečíta správy z databázy a skontroluje počet uložených správ.

Výsledkom (viď Tabuľka 4.3) je opäť reťazec začínajúci znakmi `##` a obsahuje tri položky. Prvá udáva počet správ uložených v databáze, druhá hovorí, či bolo prijaté potvrdenie poslednej správy a tretia položka znamená výsledok kontroly počtu prijatých správ. Posledné dve položky by mali mať hodnotu `true`, ako je uvedené v druhom riadku tabuľky. Tretí a štvrtý riadok ukazujú nesprávne výsledky testu. V prvom prípade nebolo doručené potvrdenie prijatia poslednej správy, v druhom prípade sa počet prijatých správ nerovnal počtu odoslaných správ.

výsledok	##sprav@@potvrdena@@prijate=odoslane
správne	##100@@true@@true
nesprávne	##100@@true@@false
nesprávne	##98@@false@@true

Tabuľka 4.3: Tabuľka výsledkov testu `mdbEjbTransactions`

Test `mdbMessagingPerformance`

Prvý z dvoch výkonnostných testov je jednoduchý test merajúci čas medzi prijatím prvej správy zo vstupnej fronty a prijatím poslednej správy zo vstupnej fronty. MDB pošle tento čas v správe do výstupnej fronty, servlet správu prijme a vypíše ako výsledok. Tento test nemá správne a nesprávne výstupy, slúži len na porovnanie doby spracovania v rôznych podmienkach, napríklad pre rôznych JMS poskytovateľov. Výsledkom je čas v milisekundách. Aby boli rozdiely zjavné, je potrebné pracovať s väčším množstvom správ ako pri bežných testoch – `PERFORMANCE_MESSAGE_COUNT`.

Test `mdbEjbPerformance`

Druhý výkonnostný test pracuje rovnako ako test `mdbEjbTransactions`, ale posiela väčší počet správ a meria čas potrebný na ich odoslanie a spracovanie.

Tabuľka 4.4 opäť ukazuje výsledky testu. Prvý riadok tabuľky ukazuje výsledný reťazec zložený z troch položiek oddelených znakmi `@@`. Prvé dve položky sú zhodné s testom `mdbEjbTransactions`, to znamená potvrdenie poslednej správy a kontrola počtu prijatých správ. Posledná položka je počet milisekúnd trvania testu. Druhý riadok tabuľky ukazuje príklad správneho výsledku pre počet odoslaných správ `PERFORMANCE_MESSAGE_COUNT=10000`. Zvyšné riadky predstavujú rôzne nesprávne výsledky, ich popis je uvedený pri transakčnom teste `mdbEjbTransactions`. Posledná položka výsledku sa líši pre rôznych JMS poskytovateľov a aj pre rôzne behy testu pre jedného JMS poskytovateľa.

výsledok	##potvrdena@@prijate=odoslane@@cas
správne	##true@@true@@45000
nesprávne	##true@@false@@45000
nesprávne	##false@@true@@45000

Tabuľka 4.4: Tabuľka výsledkov testu `mdbEjbPerformance`

4.1.4 Clusterové testy

Druhá časť testovacej sady je zameraná na clusterové testy. Clusterové testy overujú schopnosť JMS poskytovateľov pracovať s výpočtovým clusterom aplikačných serverov a rozdeľovať záťaž medzi jednotlivé uzly v clusteri. Keďže vysoká dostupnosť je vlastnosť clusteringu, okrem samotných clusterových testov obsahuje táto časť testovacej sady aj testy vysokej dostupnosti, čo znamená, že správy musia byť správne spracované aj pri zastavení jedného uzla v clusteri.

Test `testClusteredEap`

Test `testClusteredEap` je jednoduchý clusterový test s dvomi aplikačnými servermi tvoriacimi cluster. Servlet pošle správy do vstupnej fronty prvého uzla. MDB nasadený na tomto uzle správy prepošle do výstupnej fronty toho istého uzla. Servlet ich následne vyčíta z výstupnej fronty druhého uzla a skontroluje počet prijatých správ a počet ich výskytov. Výsledný reťazec aj správne a nesprávne výsledky sú totožné s transakčným testom `mdbMessagingTransactions`, viď Tabuľka 4.2.

Test `testKillEap`

Tento test je prvý z dvoch testov vysokej dostupnosti. Podobne ako v predchádzajúcom teste je použitý cluster dvoch aplikačných serverov a servlet posiela správy na prvý z nich. Rozdiel je však v tom, že MDB po prijatí správy uloží túto správu do databázy a uprostred prijímania správ je druhý uzol zastavený a zvyšné správy prijme a uloží do databázy MDB v prvom uzle. Rovnako ako pri transakčných testoch využívajúcich databázu MDB posiela potvrdenie poslednej správy do výstupnej fronty. Nakoniec sú z databázy vyňaté všetky správy uložené v tomto teste a je skontrolovaný ich počet. Výsledný reťazec a výsledky sú zhodné s transakčným testom `mdbEjbTransactions` a sú ukázané v Tabuľke 4.3.

Test `testKillEapStartEap`

Druhý test vysokej dostupnosti oproti predchádzajúcemu pridáva znovu spustenie zastaveného aplikačného servera. V tomto prípade by mal znovu spustený aplikačný server opäť prevziať časť prichádzajúcich správ. Výsledný reťazec a správne aj nesprávne výsledky sú totožné s predchádzajúcim testom (viď Tabuľka 4.3).

4.2 Implementácia

V predchádzajúcej podkapitole bol uvedený návrh testov. Navrhnutá testovacia sada je implementovaná v programovacom jazyku Java. Tento jazyk bol prirodzeným výberom, keďže JBoss AS je Java EE aplikačný server a platforma Java EE je postavená na jazyku Java. Okrem toho testovacia sada využíva rozhranie JMS API, ktoré je rozhraním pre programovanie aplikácií v programovacom jazyku Java. Na správu projektu je použitá aplikácia Maven predstavená v kapitole 2.5, pomocou ktorej je možné skompilovať testovaciu aplikáciu, zabaliť do balíčka nasaditeľného na aplikačný server a spustiť testy. Výsledná aplikácia je zabalená do EAR archívu `mdb-test.ear` a obsahuje všetky preložené triedy, potrebné knižnice a konfiguračné súbory.

V testovacej sade sú zahrnutí traja JMS poskytovatelia, a to WebSphere MQ, MRG a predvolený JMS poskytovateľ v danej verzii JBoss AS, to znamená JBoss Messaging pre JBoss AS 5 a HornetQ pre JBoss AS 7.

4.2.1 Popis modulov

Testovacia aplikácia sa skladá z troch modulov: `mdb-ejb`, `mdb-servlet` a `mdb-test`. V module `mdb-ejb` sa nachádzajú EJB komponenty spolu so súborom `persistence.xml` obsahujúcim konfiguráciu databázy a sú zabalené do JAR archívu `mdb-ejb.jar`. Obrázok 4.1 znázorňuje diagram tried pre tento modul, ktorý sa skladá z generickej triedy MDB a jej potomkov, triedy reprezentujúcej správu uloženú do relačnej databázy a triedy používanej

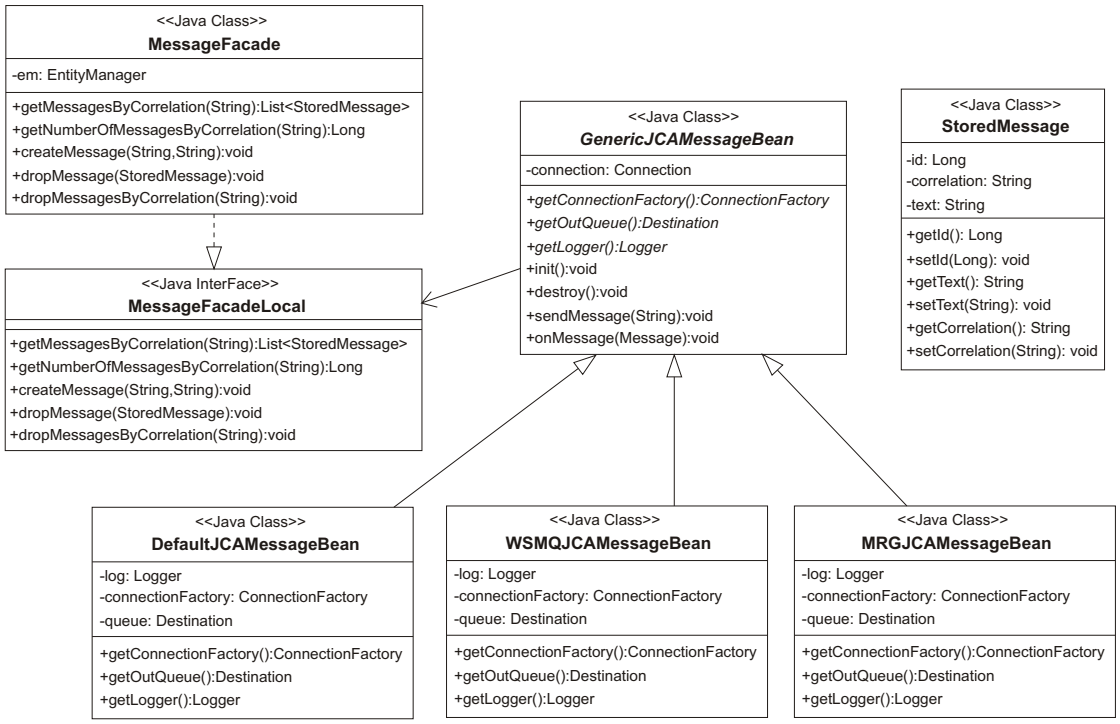
pre prístup k databáze spolu s rozhraním, ktoré implementuje. Všetky triedy a ich dôležité metódy sú popísané nižšie.

Modul `mdb-servlet` obsahuje servlety, testovacie triedy a konfiguračný súbor `web.xml`, ktorý definuje mapovanie tried servletu na URL. Tento modul je zabalený do WAR archívu `mdb-servlet.war`. Diagram tried pre tento modul je na Obrázku 4.2, pre zjednodušenie neobsahuje všetkých potomkov generických tried obsiahnutých v testovacej aplikácii, iba potomkov pre predvoleného JMS poskytovateľa.

Modul `mdb-test` predstavuje výslednú aplikáciu a celý EAR archív `mdb-test.ear`. Obsahuje definíciu JMS objektov pre jednotlivých JMS poskytovateľov a konfiguračný súbor `application.xml`, ktorý deklaruje WAR a JAR archívy a adresár s knižnicami vnútri archívu EAR.

4.2.2 Popis tried

Nasleduje popis tried, ktoré sú nezávislé od konkrétneho JMS poskytovateľa. Generické triedy, ktorých názov začína reťazcom `Generic`, obsahujú abstraktné metódy, ktoré je potrebné implementovať v ich potomkoch. Pre každého JMS poskytovateľa musí byť implementovaný potomok každej z týchto generických tried. Jedná sa o komponent MDB, servlet a dve triedy so samotnými testami. Takto je možné v budúcnosti testovaciu sadu jednoducho doplniť o ďalších JMS poskytovateľov po vykonaní kroku integrácie do aplikačného servera.



Obrázok 4.1: Diagram tried modulu `mdb-ejb`.

`GenericJCAMessageBean`

Táto abstraktná trieda implementuje rozhranie `MessageListener` a jeho jedinú metódu `onMessage()`, ktorú musí implementovať každý MDB. MDB čaká správy vo vstupnej fronte. Pri príchode správy MDB z jej obsahu zistí, ktorý test spôsobil jej odoslanie. MDB reaguje na prijatie správy v závislosti od zdrojového testu, a to jedným z troch spôsobov. Prvým z nich je odoslanie správy s potvrdením do výstupnej fronty, druhým je odoslanie správy do databázy a posledným spôsobom je meranie času medzi prvou a poslednou prijatou správou.

Pre prijímanie aj odosielanie správ potrebuje MDB vytvoriť spojenie. Spojenie je vytvorené a spustené v metóde `init()`, ktorá je anotovaná anotáciou `@PostConstruct`, čo spôsobí jej zavolanie pri vytvorení novej inštancie MDB. Podobne je spojenie zastavené a zrušené v metóde `destroy()` s anotáciou `@PreDestroy`.

Spojenie sa vytvára z objektu `ConnectionFactory`, ktorý je špecifický pre každého JMS poskytovateľa. Pre prijímanie správ navyše potrebuje MDB vedieť názov vstupnej fronty a pre odosielanie názov výstupnej fronty. Všetky tieto informácie obsahujú potomkovia triedy `GenericJCAMessageBean`. Cez metódy `getConnectionFactory()` a `getQueue()` poskytujú `ConnectionFactory` a výstupnú frontu, vstupná fronta je definovaná anotáciami. Testovacia sada obsahuje troch potomkov generickej triedy reprezentujúcej MDB, a to `DefaultJCAMessageBean` pre predvoleného JMS poskytovateľa, `WSMQJCAMessageBean` pre WebSphere MQ a `MRGJCAMessageBean` pre MRG.

`StoredMessage`

Entity Bean `StoredMessage` reprezentuje správy ukladané do relačnej databázy. Má atribúty `id`, `correlation` a `text`, ktoré korešpondujú so stĺpcami tabuľky v databáze. Trieda definuje aj niekoľko pomenovaných dotazov – zistenie počtu všetkých správ v databáze, nájdenie všetkých správ v databáze alebo nájdenie správ so zadaným atribútom `correlation`. Dotazy sú definované pomocou anotácie `@NamedQueries`, v ktorej je zoznam všetkých pomenovaných dotazov. Jedna položka z tohto zoznamu je napríklad dotaz, ktorý vráti počet uložených správ v databáze s hodnotou v stĺpci `correlation` zhodnou so zadanou hodnotou: `@NamedQuery(name = "findNumberOfMessagesByCorrelation", query = "SELECT COUNT(m) FROM StoredMessage m WHERE m.correlation=:correlation")`.

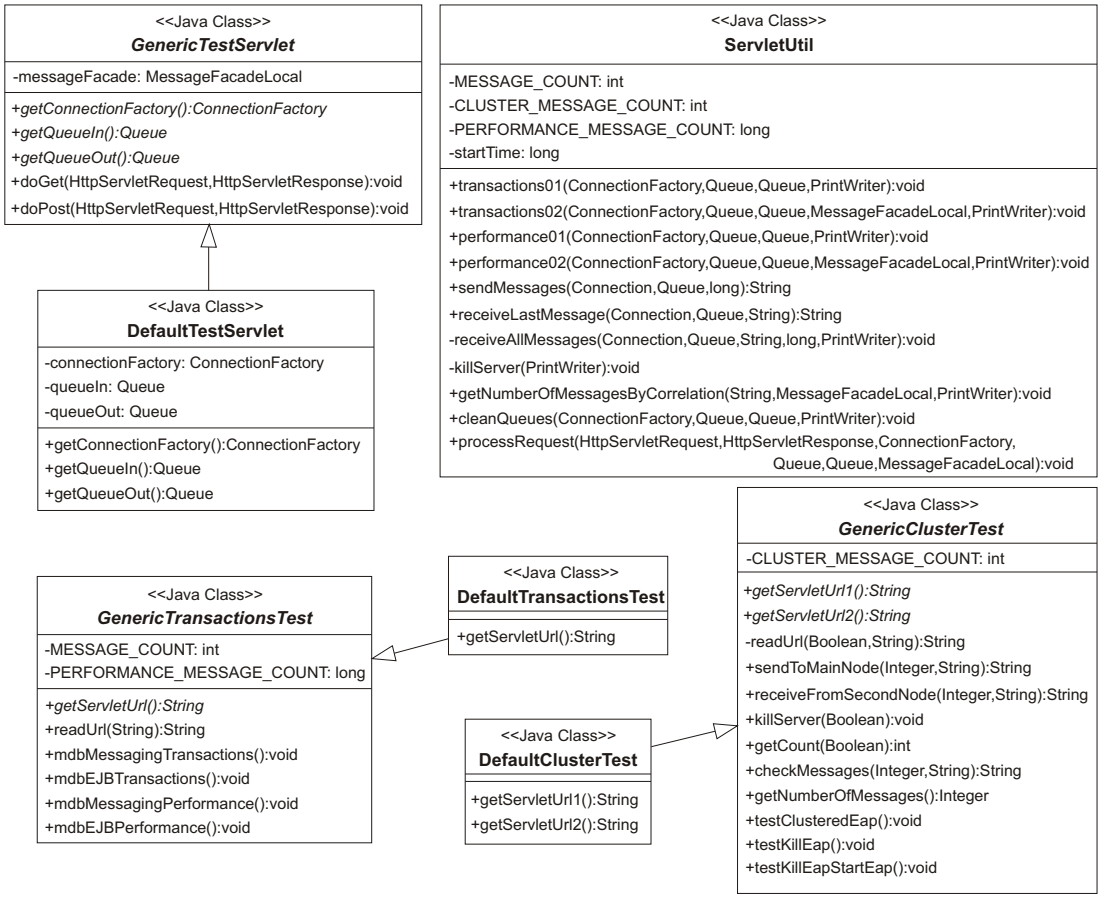
`MessageFacade`

Trieda implementuje navrhnuté rozhranie `MessageFacadeLocal` a slúži na komunikáciu s relačnou databázou pomocou správcu entít `EntityManager`. Sú v nej implementované metódy na uloženie správy do tabuľky v databáze, zmazanie správy z tabuľky, zmazanie všetkých správ v tabuľke so zadanou hodnotou atribútu `correlation` a metódy využívajúce pomenované dotazy z triedy `StoredMessage`.

`GenericTestServlet`

Táto abstraktná trieda je zdedená z triedy `HttpServlet`, ktorá poskytuje metódy na prácu s HTML formulármi. Trieda `GenericTestServlet` implementuje zo svojej rodičovskej triedy metódy `doGet()` a `doPost()`, ktoré reagujú na HTTP požiadavky GET a POST. Spracovanie oboch typov požiadaviek je implementované v triede `ServletUtil`.

Podobne ako pri triede `GenericJCAMessageBean`, aj potomkovia tejto triedy musia implementovať metódy `getConnectionFactory()` a `getOutQueue()` a okrem nich navyše



Obrázok 4.2: Diagram tried modulu `mdb-servlet`.

metódu `getInQueue()` na zistenie vstupnej fronty. Testovacia sada opäť obsahuje troch potomkov, a to `DefaultTestServlet` pre predvoleného JMS poskytovateľa, `WSMQTestServlet` pre WebSphere MQ a `MRGTestServlet` pre MRG.

`ServletUtil`

Táto trieda obsahuje metódu `processRequest()`, ktorá spracuje HTTP požiadavky GET a POST prijaté servletom. V každej požiadavke je očakávaný povinný parameter `op`, ktorého hodnota udáva operáciu, ktorá má byť vykonaná. Pri transakčných testoch je touto operáciou názov testu. Pri clusterových testoch však rôzne operácie v rámci jedného testu môžu pracovať s rôznymi servermi a rôznymi URL servletu. Preto vyžadujú spustenie každej operácie zvlášť. Ako hodnoty tohto parametra sa postupne použijú `send` a `receive`, prípadne `check`. Ďalej je možné použiť tri nepovinné parametre. Prvý z nich, `count`, určuje počet posielaných správ a ak je zadaný, prepíše implicitné nastavenia `MESSAGE_COUNT`, `PERFORMANCE_MESSAGE_COUNT` a `CLUSTER_MESSAGE_COUNT`. Transakčné testy posielajú implicitne 100 správ, výkonnostné 10000 a clusterové 5000. Druhý nepovinný parameter, `test`, je použitý len pri clusterových testoch a udáva meno volajúceho testu. Posledný z nepovinných parametrov `correlation` udáva identifikátor skupiny správ pri operácii `check`. V kapitole 4.2.3 je popis jednotlivých operácií, ktoré servlet vykoná ako reakciu na danú

požiadavku. Na každú požiadavku servlet vráti ako odpoveď obsah HTML stránky, ktorý volajúca strana môže prečítať a spracovať.

`GenericTransactionsTest`

Prvá z dvoch tried obsahujúcich testovacie metódy je zameraná na transakčné testy. Obsahuje štyri metódy s anotáciou `@Test`. Ukážka implementácie testovacej metódy je na Obrázku 4.3. Pre výkonnostné testy a testy ukladajúce správy do databázy bol ako časový limit zvolený 120 minút, pre ostatné testy 60 minút. Oba limity sú nadhodnotené a nemali by sa vôbec uplatniť, pokiaľ nenastane neočakávaná udalosť. Všetky testy majú na prvom riadku volanie metódy `readUrl()` s parametrom udávajúcim názov volajúceho testu. Metóda `readUrl()` zistí URL servletu podľa konkrétneho JMS poskytovateľa (viď Tabuľka 4.5), doplní parameter `op` názvom volajúceho testu a pošle požiadavku GET. Požiadavka vyvolá spustenie samotného testu, ktorého výsledok je na poslednom riadku odpovede od servletu. Nakoniec sa výsledný reťazec spracuje – rozdelí sa na očakávaný počet častí podľa dvojíc znakov `@@` a každá časť sa porovná s očakávaným výsledkom, ako je pre každý test uvedené pri návrhu transakčných testov v kapitole 4.1.3.

```
@Test(timeOut = 60 * 60000, enabled = true)
public void mdbMessagingTransactions() throws Exception {
    String result = readUrl("transactions01");
    Reporter.log(result);
    String results[] = result.substring(2).split("@@");

    assertEquals(results.length, 4,
        "Cannot parse result string - unexpected number of parts.");
    assertEquals(results[0], MESSAGE_COUNT,
        "Wrong number of messages received.");
    assertEquals(results[1], "0",
        "Unexpected errors detected.");
    assertEquals(results[2], "true",
        "Invalid number of messages has been processed.");
    assertEquals(results[3], "true",
        "Duplicate messages has been detected.");
}
```

Obrázok 4.3: Ukážka implementácie testovej metódy transakčného testu.

Na zistenie URL servletu pre každého JMS poskytovateľa slúži abstraktná metóda `getServletUrl()`, ktorú implementujú potomkovia triedy `GenericTransactionsTest`. Potomkovia tejto triedy sú v testovacej sade znova traja – `DefaultTransactionsTest` pre predvoleného JMS poskytovateľa, `WSMQTransactionsTest` pre WebSphere MQ a pre MRG je to `MRGTransactionsTest`.

`GenericClusterTest`

Druhá trieda s testovacími metódami je zameraná na clusterové testy. Obsahuje tri metódy s anotáciou `@Test`, na Obrázku 4.4 je opäť uvedená ukážka implementácie jednej z nich. Na rozdiel od transakčných testov tieto metódy obsahujú viac volaní metódy `readUrl()`

s rôznymi URL servletu. V tejto triede má metóda `readUrl()` jeden parameter navyše a udáva uzol, na ktorý posiela požiadavku. Parameter je typu `Boolean` a má hodnotu `true` pre prvý uzol a hodnotu `false` pre druhý uzol. Test `testClusteredEap` podľa návrhu najprv odošle správy na prvý uzol, počká 10 sekúnd na doručenie správ a potom prijme správy z druhého uzla. Operácia `receive` vráti výsledný reťazec, ktorý je spracovaný rovnako ako pri transakčných testoch. Posledné štyri príkazy `assertEquals` sa zhodujú s príkladom na Obrázku 4.3.

```
@Test(timeOut = 120 * 60000, enabled = true)
public void testClusteredEap() throws Exception {
    readUrl(true,
        "op=send&count="+CLUSTERED_MESSAGE_COUNT+"&test=testClusteredEAP");
    Thread.sleep(10000);
    String result = readUrl(false,
        "op=receive&count="+CLUSTERED_MESSAGE_COUNT);

    Reporter.log(result);
    String results[] = result.substring(2).split("@@");

    assertEquals(results.length, 4,
        "Cannot parse result string - unexpected number of parts.");
    assertEquals(results[0], CLUSTERED_MESSAGE_COUNT,
        "Wrong number of messages received.");
    assertEquals(results[1], "0",
        "Unexpected errors detected.");
    assertEquals(results[2], "true",
        "Invalid number of messages has been processed.");
    assertEquals(results[3], "true",
        "Duplicate messages has been detected.");

}
```

Obrázok 4.4: Ukážka implementácie testovej metódy clusterového testu.

Ďalšou testovacou metódou v tejto triede je test `testKillEap`. Je to test vysokej dostupnosti, kedy je jeden z aplikačných serverov v clusteri zastavený. Najprv sú odoslané správy na prvý uzol metódou `sendToMainNode()`. Keďže sú oba aplikačné servery v clusteri, rozdelia si príchodzie správy podľa svojej aktuálnej záťaže. Z oboch uzlov prijímajúce MDB uložia správy, ktoré prijali, do databázy. Test potom postupne kontroluje počet správ v databáze pomocou metódy `getCount()` a v čase, kedy je už polovica správ uložených, zastaví druhý aplikačný server použitím metódy `killServer()`. Vtedy prvý uzol prevezme prácu oboch uzlov a dokončí ukladanie správ do databázy. Na konci testu je prevedená kontrola, či sa počet správ v databáze rovná počtu odoslaných správ – metóda `checkMessages()`.

Poslednou testovacou metódou je druhý test vysokej dostupnosti `testKillEapStartEap`. Tento test je obdobou predchádzajúceho testu, ale po zastavení druhého uzla tento uzol znova spustí. Pre spustenie servera je potrebný spúšťací skript, ktorý je zavolaný z testovacej metódy. Skript s názvom `start-server.sh` sa nachádza v hlavnom adresári testovacej

sady a predáva sa testu ako parameter.

Potomkovia tejto triedy musia implementovať dve metódy pre dva uzly v clusteri `getServletUrl1()` a `getServletUrl2()`. V testovacej sade sú traja potomkovia – pre predvoleného JMS poskytovateľa `DefaultClusterTest`, `WSMQClusterTest` pre WebSphere MQ a `MRGClusterTest` pre MRG.

4.2.3 Popis operácií servletu

Nasleduje popis všetkých operácií, ktoré poskytuje servlet a sú implementované v triede `ServletUtil`. Operácie korešpondujú s názvami metód, ktoré ich implementujú. Prvé štyri operácie sú transakčné a výkonnostné testy, nasledujú operácie využívané clusterovými testami a nakoniec je uvedená operácia pre manuálne vyčistenie používaných front.

Operácia `transactions01`

Táto operácia spúšťa prvý z dvoch transakčných testov `mdbMessagingTransactions`. Operáciu je možné zavolať s jedným nepovinným parametrom `count`, rovnako ako pri všetkých operáciách spúšťajúcich transakčný alebo výkonnostný test. Prvým krokom je vytvorenie spojenia, ktoré je využité pri odosielaní aj prijímaní správ, pre obe operácie je vytvorené samostatné sedenie z tohto spojenia. Pre odosielanie je použité transakčné sedenie, to znamená, že správy budú odosielané v jednej transakcii. Pred odosielaním správ je vytvorený producent pre vstupnú frontu a vygeneruje sa reťazec, ktorý predstavuje skupinový identifikátor správ posielaných v rámci jedného testu. Týmto identifikátorom je reťazcová reprezentácia aktuálneho času v milisekundách. Producent odosiela perzistentné správy, aby v prípade výskytu chyby nedošlo k ich strate. Správy sú vždy typu `TextMessage`. Po samotnom odoslaní všetkých správ je transakcia potvrdená zavolaním metódy `commit()` na sedenie.

Nasleduje prijatie správ z výstupnej fronty. Vytvorí sa nové sedenie, tentokrát netransakčné, a konzument na výstupnej fronte. Konzument prijíma všetky správy z výstupnej fronty a ukladá si ich do zoznamu dvojíc kľúč-hodnota, kde kľúč tvorí identifikátor správy a hodnota je počet, koľkokrát je daná správa prijatá.

Po prijatí všetkých správ je nad týmto zoznamom vykonaná kontrola, či sa v ňom každá očakávaná správa vyskytuje, či neobsahuje nadbytočné správy a či obsahuje všetky správy práve jedenkrát. Na konci je vrátená odpoveď s výstupným reťazcom, ktorý obsahuje výsledky testu podľa Tabuľky 4.2.

Operácia `transactions02`

Pri požiadavke s touto operáciou je spustený druhý transakčný test `mdbEjbTransactions`. Na strane servletu prebieha prvá časť rovnako ako pri predchádzajúcom teste, rozdiel je len na strane MDB, ktorý tentokrát správy ukladá do tabuľky v relačnej databáze. Keď servlet prijme potvrdenie poslednej správy, počká 10 sekúnd, aby sa do databázy stihli uložiť všetky správy, potom vyjme z databázy všetky správy so skupinovým identifikátorom pre tento test. Výsledkom je kontrola, či je počet vyňatých správ z databázy rovný počtu odoslaných správ (viď Tabuľka 4.3).

Operácia `performance01`

Táto operácia spúšťa prvý výkonnostný test `mdbMessagingPerformance`. Posielanie správ je opäť rovnaké ako pri transakčných testoch, servlet nakoniec prijme potvrdenie poslednej správy, ktoré vráti ako výsledok. V tomto potvrdení sa nachádza čas trvania testu v milisekundách, ktorý zistí MDB ako rozdiel aktuálneho času pri príchode prvej správy a aktuálneho času pri príchode poslednej správy.

Operácia `performance02`

Pri prijatí požiadavky s operáciou `performance02` sa spustí test `mdbEjbPerformance`. Servlet pracuje rovnako ako pri operácii `trasactions02` a vráti rovnaký výsledok. Jediný rozdiel je v počte odosielaných správ, výkonnostný test pracuje s niekoľkonásobne vyšším počtom správ ako pri bežných transakčných testoch. Pred odoslaním prvej správy servlet zistí aktuálny čas, rovnako aj po uložení všetkých správ do databázy. Rozdiel týchto dvoch časov vráti ako dĺžku trvania testu.

Operácia `send`

Táto operácia je prvým krokom clusterových testov, posiela správy na jeden z uzlov v clusteri aplikačných serverov. Pri volaní operácie `send` sú povinné dva parametre – `count` a `test`. Pred odoslaním prvej správy je z aktuálneho času vygenerovaný identifikátor skupiny správ. Následne je vytvorené sedenie a producent pre vstupnú frontu a je odoslaný požadovaný počet správ. Na konci je na sedenie zavolaná metóda `commit()`, producent aj sedenie sú ukončené a tým táto operácia končí. Operácia vracia ako výsledok vygenerovaný identifikátor skupiny správ.

Operácia `receive`

Operácia `receive` spôsobí prijatie správ z výstupnej fronty. Servlet prijíma správy, pokiaľ sa mu v danom čase podarí nejakú prijať. Keď v tomto čase nijakú správu neprijme, končí. Operácia má len jeden nepovinný parameter `count`, aby mohol servlet skontrolovať počet prijatých správ a vrátiť výsledok testu. Túto operáciu využíva test `testClusteredEAP` na prijatie správ z druhého uzla v clusteri.

Operácia `check`

Táto operácia je používaná v testoch `testKillEap` a `testKillEapStartEap` po odoslaní správ na overenie správneho výsledku testu. Použitie operácie `check` vyžaduje ďalšie dva parametre, a to počet správ poslaných operáciou `send` (`count`) a identifikátor skupiny správ pre tento test (`correlation`) generovaný pri operácii `send`.

Operácia `kill-server`

Túto operáciu volajú testy vysokej dostupnosti na zastavenie aplikačného servera.

Operácia `get-count-db`

Reakciou na túto operáciu je zavolanie metódy na zistenie počtu správ v tabuľke z triedy `MessageFacade`. Operácia vyžaduje nepovinný parameter `correlation`, podľa ktorého zisťuje len počet správ odoslaných v jednom konkrétnom teste.

Operácia `drop-db`

Táto operácia zavolá metódu na zmazanie všetkých správ v tabuľke so zadaným skupinovým identifikátorom z triedy `MessageFacade`. Operácia vyžaduje znova parameter `correlation`.

Operácia `clean-queues`

Táto operácia vyčistí vstupnú aj výstupnú frontu od správ, ktoré v nich mohli ostať. Najprv sa vytvorí konzument na vstupnej fronte, ktorý v cykle prijíma správy z tejto fronty, pokiaľ tam nejaké sú. Všetky nájdené a zahodené správy sú vypísané ako výsledok operácie. Následne je celý postup zopakovaný pre výstupnú frontu.

4.2.4 Migrácia z JBoss AS 5 na JBoss AS 7

Kvôli rozdielom medzi JBoss AS verzie 5 a 7 nie je možné použiť jednu testovaciu aplikáciu pre obe verzie JBoss AS. Testovacia aplikácia bola najprv implementovaná pre JBoss AS 5 a následne upravená pre JBoss AS 7. Zdrojové kódy s príponou `.java` sú pre obe verzie totožné, rozdiely sú v konfiguračných súboroch. Ďalším rozdielom je odlišný predvolený JMS poskytovateľ a adresárová štruktúra samotného aplikačného servera. Namiesto JMS objektov pre JBoss Messaging je potrebné vytvoriť tieto objekty pre HornetQ. JBoss Messaging objekty sú definované v súbore `jbm-queues-service.xml`, konfigurácia pre HornetQ sa nachádza v súbore `standalone.xml`. Postup, ako správne nasadiť testovaciu aplikáciu a potrebné súbory, je pre obe verzie JBoss AS uvedený v kapitole 2.4.

4.3 Spustenie testov

Implementovaná testovacia sada je súčasťou aplikácie, ktorú je možné nasadiť na aplikačný server. Sú dve možnosti ako pristupovať k testom, a to cez webový prehliadač na URL servera alebo pomocou cieľa `test` aplikácie Maven. Obidva spôsoby vyžadujú okrem prístupu k JMS serveru daného JMS poskytovateľa aj spustený aplikačný server s nasadenou testovacou aplikáciou a integrovanými požadovanými JMS poskytovateľmi podľa kapitoly 3. Tieto kroky je nutné v oboch prípadoch vykonať manuálne, plná automatizácia procesu testovania je popísaná v závere tejto podkapitoly.

4.3.1 Manuálne spustenie testov

Spustenie testov cez webový prehliadač je plne manuálne. Servlet spúšťajúci testy beží na porte 8080. V prípade clusterových testov je možné obidva uzly spustiť na rovnakej adrese (napríklad `localhost`), druhý uzol potom musí používať odlišné porty. Napríklad pre posunutie všetkých používaných portov o hodnotu 100 slúži pri spustení aplikačného servera parameter `-Djboss.service.binding.set=ports-01`. To znamená, že servlet na druhom uzle beží na porte 8180.

Tabuľka 4.5 obsahuje rôzne príklady URL servletu. Na prvom riadku je znázornený všeobecný tvar URL. Skladá sa z IP adresy servera, portu, na ktorom servlet beží, reťazca `mdb-servlet`, reťazca predstavujúceho konkrétneho JMS poskytovateľa a parametrov. Predvoleného JMS poskytovateľa, to znamená JBoss Messaging v JBoss AS 5 a HornetQ v JBoss AS 7 reprezentuje reťazec `test`, pre Websphere MQ je to `wsmqtest` a pre MRG

`mrgtest`. Prvý parameter `op` je povinný a predstavuje operáciu, ktorú má servlet vykonať – názov požadovaného testu alebo operácie `send`, `receive` a `check`. Všetky dostupné operácie sú uvedené pri popise implementácie v kapitole 4.2.3. Druhý parameter `count` je nepovinný a znamená počet správ, ktoré majú byť v rámci daného testu posielané. Ak tento parameter nie je zadaný, použijú sa implicitné hodnoty. Posledný parameter `correlation` je použitý len pri testoch vysokej dostupnosti pre operáciu `check` a predstavuje identifikátor skupiny správ poslaných operáciou `send` v rámci rovnakého testu. Príklad na treťom riadku spôsobí spustenie testu `performance02` pre predvoleného JMS poskytovateľa s počtom správ 20000. Príklad na štvrtom riadku odošle na prvý uzol v clusteri, ktorý používa ako JMS poskytovateľa WebSphere MQ správy a vypíše skupinový identifikátor týchto správ ako výsledok. Piaty príklad spôsobí prijatie správ z druhého uzla v clusteri a je vhodný pre jednoduchý clusterový test. Pre testy vysokej dostupnosti sa používa posledný príklad na kontrolu správnosti. Parameter `count` sa musí zhodovať s počtom odoslaných správ. Ak bola operácia `send` volaná bez tohto parametra, čo znamená, že bolo poslané predvolené množstvo správ, je vhodné aj túto operáciu volať bez parametra a použije sa rovnaké predvolené množstvo správ. Parameter `correlation` je však povinný a musí to byť číslo, ktoré vrátila operácia `send`.

`IP:port/mdb-servlet/poskytovatel?op=operacia&count=pocet&correlation=id`
`localhost:8080/mdb-servlet/mrgtest?op=transactions01`
`localhost:8080/mdb-servlet/test?op=performance02&count=20000`
`localhost:8080/mdb-servlet/wsmqtest?op=send`
`localhost:8180/mdb-servlet/wsmqtest?op=receive`
`localhost:8080/mdb-servlet/wsmqtest?op=check&count=1000&correlation=1234`

Tabuľka 4.5: URL Servletov.

Pri manuálnom testovaní je nakoniec potrebné vyhodnotiť výsledky testov. Po dokončení testu servlet vráti HTML stránku, ktorej obsah tvorí výsledok testu. Týmto výsledkom je v prípade úspešného dokončenia požadovanej operácie výsledný reťazec špecifikovaný pre každý test pri popise návrhu testovacej sady v kapitole 4.1. Inak vrátená HTML stránka obsahuje popis vzniknutej chyby.

4.3.2 Spustenie testov pomocou aplikácie Maven

Aplikácia Maven čiastočne automatizuje testovanie, a to konkrétne časť spúšťania a vyhodnocovania testov. Maven umožňuje naraz spustiť všetky testy v rámci testovacej aplikácie, všetky testy pre konkrétneho JMS poskytovateľa, jeden konkrétny test pre všetkých JMS poskytovateľov alebo jeden test pre konkrétneho JMS poskytovateľa. Testy sa spúšťajú z hlavného adresára testovacej aplikácie a je možné použiť príkazy z Tabuľky 4.6, ktorá ukazuje všetky spomínané spôsoby. Napríklad príkaz na prvom riadku spustí všetky testy, príkaz na poslednom riadku spustí test `mdbEjbTransactions` s WebSphere MQ.

Výsledkom príkazov z Tabuľky 4.6 je počet spustených testov a počet, koľko zo spustených testov skončilo neúspešne, koľko skončilo s chybou, prípadne koľko testov bolo z nejakého dôvodu vynechaných. Test môže byť vynechaný napríklad v prípade, kedy je závislý na inom teste a tento test skončí s chybou. Test skončí úspešne len v prípade, kedy všetky volania `Assert` daného testu boli úspešné. V prípade, keď test neprejde, je na výstup a do logu vypísaná hláška z neúspešného volania `Assert`.

`mvn test`
`mvn test -Dtest=*#mdbEjbTransactions`
`mvn test -Dtest=WSMQTransactionsTest`
`mvn test -Dtest=WSMQTransactionsTest#mdbEjbTransactions`

Tabuľka 4.6: Príkazy aplikácie Maven pre spustenie testov.

4.3.3 Automatizácia procesu testovania

Na úplnú automatizáciu testov je použitý nástroj Jenkins[14][1]. Je to open source nástroj pre priebežnú integráciu, ktorý umožňuje automatizovať jednotlivé kroky potrebné pre spustenie testov – stiahnuť zdrojové kódy testovacej sady, pripraviť aplikačný server a prostredie pre beh testov, spustiť testy, vygenerovať správu s výsledkami testov a celý proces parametrizovať.

Príprava prostredia pre beh testov je automatizovaná pomocou frameworku vyvíjaného a používaného interne vo firme Red Hat, ktorý nie je voľne dostupný. Tento framework je implementovaný v programovacom jazyku Groovy a obsahuje šablónu pre spúšťanie testov pomocou nástroja Jenkins. Pripravená šablóna obsahuje metódy na stiahnutie a rozbalenie aplikačného servera, alokovanie a uvoľnenie požadovanej databázy, spustenie a zastavenie aplikačného servera, prípravu aplikačného servera a spustenie testov. Všetky metódy je možné upraviť podľa požiadaviek konkrétnych testov. V tomto prípade je nutné upraviť prípravu aplikačného servera, spustenie testov a pre clusterové testy aj spustenie aplikačného servera. Ďalej sú rozobrané len tieto rozdielne metódy.

Príprava aplikačného servera obsahuje kroky integrácie JMS poskytovateľov do aplikačného servera, skompilovanie testovacej aplikácie a jej nasadenie na aplikačný server, ako ukazuje Obrázok 4.5.

```
ant.copy(overwrite: true, todir: deployDir) {
  fileset(dir: new File(basedir, "resources").absolutePath) {
    include(name: "qpid-ra.rar")
    include(name: "wmq.jmsra.rar")
    include(name: "qpid*-ds.xml")
    include(name: "jboss-jmsra-ds.xml")
  }
}
runMaven("mdb", "clean")
runMaven("mdb", "package", "-DskipTests")
ant.copy(
  file: new File(basedir, "mdb/mdb-test/target/mdb-test.ear").absolutePath,
  todir: new deployDir)
```

Obrázok 4.5: Príprava aplikačného servera v Groovy frameworku.

Prvý príkaz `copy` nasadí adaptéry zdrojov a pripravené konfiguračné súbory jednotlivých JMS poskytovateľov na aplikačný server skopírovaním z adresára `resources`. Nasledujú príkazy aplikácie Maven `runMaven` na skompilovanie testovacej aplikácie a nakoniec

[1] https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins

nasadenie zabalenej aplikácie na aplikačný server posledným príkazom `copy`.

Kvôli clusterovým testom, ktoré používajú dva aplikačné servery, je potrebné predefinovať metódu `startServer()` (Obrázok 4.6), ktorá spúšťa len jeden aplikačný server. V predefinovanej metóde sa najprv spustí prvý server a ak má byť spustený cluster aplikačných serverov, skopíruje sa konfigurácia prvého a spustí sa s rovnakým parametrom `udpGroup` a s hodnotami všetkých portov posunutých o hodnotu 100.

```
def startServer(){
  super.startServer()
  if(isEapClustered) {
    ant.copy(todir: new File(serverDir, "server/" +
                             config + "_2").absolutePath) {
      fileset(dir: new File(serverDir, "server/" + config).absolutePath)}
    secondServer = JBossServer.getInstance(jbossServer.jbossHome,
                                           config + "_2")
    secondServer.setUdpGroup(jbossServer.udpGroup)
    secondServer.setUserCmdLine("-Djboss.service.binding.set=ports-01")
    secondServer.start()
  }
}
```

Obrázok 4.6: Spustenie aplikačného servera v Groovy frameworku.

Kapitola 5

Vyhodnotenie testov

Testovanie integrácie JMS poskytovateľov tretích strán do aplikačného servera JBoss prebiehalo pomocou implementovanej testovacej sady. Boli použité dva z troch možných spôsobov spúšťania testov popísaných v kapitole 4.3, a to manuálne a plne automatizované. Spúšťanie testov pomocou aplikácie Maven je zahrnuté v plne automatizovanom testovaní, preto nie je potrebné samostatné testovanie týmto spôsobom. Pri testovaní JMS poskytovateľov vyžadujúcich externý JMS server bola použitá inštalácia vo firme Red Hat.

V tejto kapitole je popísaný priebeh testovania, zistené výsledky testov a ich analýza. Testovanie bolo rozdelené na dve časti, každá venovaná jednému spôsobu testovania. Prvá podkapitola popisuje výsledky automatizovaného testovania pre obidve testované verzie JBoss AS. Druhá podkapitola vysvetľuje manuálne testovanie a získané výsledky výkonnostných testov pre JBoss AS 5. Obsahom poslednej podkapitoly je porovnanie rôznych JMS poskytovateľov a rôznych verzií JBoss AS na základe výsledkov oboch spôsobov testovania.

5.1 Automatizované testovanie

Prvá časť testovania prebiehala plne automatizovane pomocou nástroja Jenkins. Pre testy s JMS poskytovateľmi MRG a WebSphere MQ boli použité inštalácie vo firme Red Hat. Na ukladanie správ bola použitá databáza MySQL verzie 5.1. Aplikačný server, WebSphere MQ server, MRG server aj databáza boli spustené na samostatných strojoch. Pre automatizované testovanie boli použité obidve verzie JBoss AS a všetci JMS poskytovatelia zahrnutí v testovacej aplikácii.

Priložené CD obsahuje vygenerované správy s výsledkami testov. Ide o jednoduché HTML stránky, ktoré prehľadne zobrazujú všetky spustené testy po jednotlivých triedach, zvýrazňujú testy, ktoré skončili neúspešne a obsahujú výsledné reťazce s prípadnou chybovou hláškou, ak niektorý test neprešiel bez chyby. Keďže transakčné a výkonnostné testy používajú jeden aplikačný server, môžu byť spustené naraz – pred ich spustením je spustený aplikačný server a po dokončení testov je aplikačný server zastavený. Základný clusterový test môže byť opäť spustený postupne so všetkými JMS poskytovateľmi pri jednom spustení clusteru aplikačných serverov. Testy vysokej dostupnosti však musia byť spustené oddelene, pretože pri spustení testov sú potrebné dva aplikačné servery, ale počas testov je jeden z nich zastavený.

5.1.1 JBoss AS 5

Výsledky transakčných testov pre všetkých JMS poskytovateľov testovaných s JBoss AS 5, JBoss Messaging (JBM), WebSphere MQ (WSMQ) a MRG Messaging (MRG), sú v Tabuľke 5.1. Tieto testy používali predvolený počet správ 100. V druhom stĺpci tabuľky sú výsledné reťazce testu `mdbMessagingTransactions`. Vo všetkých prípadoch sa výsledný reťazec zhoduje s očakávaným výsledkom v Tabuľke 4.2. Vždy bolo prijatých všetkých 100 správ práve jedenkrát a nevyskytla sa nijaká chyba. Podobne tretí stĺpec obsahuje výsledné reťazce pre test `mdbEjbTransactions` zhodné s očakávaným výsledkom podľa Tabuľky 4.3, čo znamená, že test skončil úspešne pre všetkých testovaných JMS poskytovateľov. Potvrdenie poslednej správy bolo servletu doručené a v databáze bolo uložených 100 správ.

JMS poskytovateľ	`mdbMessagingTransactions`	`mdbEjbTransactions`
JBM	`##100@@0@@true@@true`	`##100@@true@@true`
WSMQ	`##100@@0@@true@@true`	`##100@@true@@true`
MRG	`##100@@0@@true@@true`	`##100@@true@@true`

Tabuľka 5.1: Výsledky transakčných testov pre JMS poskytovateľov v JBoss AS 5.

Tabuľka 5.2 obsahuje výsledky clusterových testov spolu s testami vysokej dostupnosti pre JBoss AS 5. Odosielaných bolo vždy 5000 správ, čo je predvolený počet správ v týchto testoch. Očakávaný výsledok testu `testClusteredEap` je vysvetlený v Tabuľke 4.2, podľa ktorej tento test pre JBoss Messaging a MRG vrátil očakávaný výsledok, ale pre WebSphere MQ neprešiel. Bolo prijatých 5004 správ, a preto ani kontrola počtu prijatých správ, ani kontrola výskytov neskončila úspešne. Podľa dodatočnej kontroly konkrétnych prijatých správ boli prijaté všetky správy aspoň jedenkrát a nadbytočné správy boli duplicity. Oba testy vysokej dostupnosti majú očakávaný výsledok v Tabuľke 4.3, pre 5000 posielaných správ je to konkrétne reťazec `##5000@@true@@true`. Tabuľka 5.2 hovorí, že tieto testy skončili úspešne pre všetkých testovaných JMS poskytovateľov.

JMS p.	`testClusteredEap`	`testKillEap`	`testKillEapStartEap`
JBM	`##5000@@0@@true@@true`	`##5000@@true@@true`	`##5000@@true@@true`
WSMQ	`##5004@@0@@false@@false`	`##5000@@true@@true`	`##5000@@true@@true`
MRG	`##5000@@0@@true@@true`	`##5000@@true@@true`	`##5000@@true@@true`

Tabuľka 5.2: Výsledky clusterových testov pre JMS poskytovateľov v JBoss AS 5.

Ako posledné sú uvedené výsledky výkonnostných testov pre predvolenú hodnotu 10000 správ. Prvý z dvoch testov nemá definovaný očakávaný výsledok, určuje len čas medzi prijatím prvej a poslednej správy. Podľa výsledkov v Tabuľke 5.3 bol najnižší čas nameraný pri použití JBoss Messaging a najvyšší pri použití WebSphere MQ. Čas nameraný v druhom výkonnostnom teste je pre rovnaký počet správ v rámci jedného JMS poskytovateľa omnoho vyšší, pretože zahŕňa nielen samotné doručovanie správ na strane MDB, ale aj odosielanie správ na strane servletu, ukladanie správ do relačnej databázy a prijímanie potvrdenia poslednej správy od MDB. Tento čas sa približuje k celkovému času behu testu a z celkového času doň nie je započítané len vytvorenie a zastavenie spojenia a zavolanie metódy servletu z testovacej triedy. Najpomalší bol opäť WebSphere MQ, najrýchlejší tentokrát MRG. Rozdiely však už nie sú také veľké ako pri predchádzajúcom teste.

JMS poskytovateľ	mdbMessagingPerformance	mdbEjbPerformance
JBM	##1539ms	##true@@true@@102580ms
WSMQ	##140074ms	##true@@true@@300505ms
MRG	##1753ms	##true@@true@@98467ms

Tabuľka 5.3: Výsledky výkonnostných testov pre JMS poskytovateľov v JBoss AS 5.

5.1.2 JBoss AS 7

Tabuľka 5.4 obsahuje výsledky transakčných testov pre oboch použitých JMS poskytovateľov v JBoss AS 7. Opäť bolo posielaných 100 správ. Výsledné reťazce pre test `mdbMessagingTransactions` sa zhodujú s očakávaným výsledkom podľa Tabuľky 4.2. Podobne sú všetky výsledky testu `mdbEjbTransactions` zhodné so správnym výsledkom `##100@@true@@true`.

JMS poskytovateľ	mdbMessagingTransactions	mdbEjbTransactions
HornetQ	##100@@0@@true@@true	##100@@true@@true
MRG	##100@@0@@true@@true	##100@@true@@true

Tabuľka 5.4: Výsledky transakčných testov pre JMS poskytovateľov v JBoss AS 7.

Výsledky clusterových testov pre JBoss AS 7 a predvolený počet odosielaných správ sú v Tabuľke 5.5. Porovnanie výsledných reťazcov so správnymi hodnotami v Tabuľke 4.2 pre test `testClusteredEap` a v Tabuľke 4.3 pre testy vysokej dostupnosti ukazuje, že všetky tri testy prešli bez chyby pre oboch testovaných JMS poskytovateľov.

JMS p.	testClusteredEap	testKillEap	testKillEapStartEap
HornetQ	##5000@@0@@true@@true	##5000@@true@@true	##5000@@true@@true
MRG	##5000@@0@@true@@true	##5000@@true@@true	##5000@@true@@true

Tabuľka 5.5: Výsledky clusterových testov pre JMS poskytovateľov v JBoss AS 7.

Nakoniec boli spúšťané výkonnostné testy pre rôzne počty odosielaných správ – 10, 100, 1000, 10000 a 100000. Výsledky dvoch výkonnostných testov pre všetkých testovaných JMS poskytovateľov obsahuje Tabuľka 5.6. Opäť sa dá ukázať porovnaním s očakávanými výsledkami testu `mdbEjbPerformance` v Tabuľke 4.3, že testy vždy skončili úspešne. Úlohou výkonnostných testov je však ukázať rozdiely medzi JMS poskytovateľmi z hľadiska rýchlosti spracovania a nie z hľadiska funkčnosti. Podľa testu `mdbMessagingPerformance` je posielanie menšieho počtu správ rýchlejšie pri HornetQ ako pri MRG, avšak pri vyšších počtoch sú časy oboch JMS poskytovateľov vyrovnané. Čas rastie najmä pri vyššom počte odosielaných správ priamo úmerne s počtom správ. Test `mdbEjbPerformance` bol vo všetkých prípadoch rýchlejší s MRG ako s HornetQ. Čas ale nerastie priamo úmerne s počtom posielaných správ, pretože obsahuje vykonávanie akcií, ktoré sa neopakujú pre každú správu.

počet správ	JMS poskytovateľ	mdbMessaging-Performance	mdbEjbPerformance
10	HornetQ	##6ms	##10@@0@@true@@true@@70312ms
	MRG	##41ms	##10@@0@@true@@true@@70100ms
100	HornetQ	##24ms	##100@@0@@true@@true@@70523ms
	MRG	##98ms	##100@@0@@true@@true@@70264ms
1000	HornetQ	##264ms	##1000@@0@@true@@true@@73804ms
	MRG	##270ms	##1000@@0@@true@@true@@72559ms
10000	HornetQ	##2623ms	##10000@@0@@true@@true@@116803ms
	MRG	##2290ms	##10000@@0@@true@@true@@108788ms
100000	HornetQ	##27812ms	##100000@@0@@true@@true@@689281ms
	MRG	##27248ms	##100000@@0@@true@@true@@562806ms

Tabuľka 5.6: Výsledky výkonnostných testov pre JMS poskytovateľov v JBoss AS 7.

5.2 Manuálne testovanie

Druhá časť testovania prebiehala manuálne s JBoss AS 5. Použitá bola databáza H2, ktorá je súčasťou aplikačného servera a predvolený JMS poskytovateľ JBoss Messaging. Tabuľka 5.7 ukazuje výsledky oboch výkonnostných testov pre rôzne počty zasielaných správ, podobne ako to bolo pri JBoss AS 7. Oproti výsledkom z JBoss AS 7 pre HornetQ sú časy v teste `mdbMessagingPerformance` pre JBoss Messaging nižšie, pri počte správ 10000 a 100000 sú už rozdiely výrazné. Naopak v teste `mdbEjbPerformance` sú časy oproti výsledkom z JBoss AS 7 vyššie.

počet správ	mdbMessagingPerformance	mdbEjbPerformance
10	##4ms	##10@@0@@true@@true@@70303ms
100	##19ms	##100@@0@@true@@true@@71608ms
1000	##217ms	##1000@@0@@true@@true@@85374ms
10000	##965ms	##10000@@0@@true@@true@@193769ms
100000	##6832ms	##100000@@0@@true@@true@@896107ms

Tabuľka 5.7: Výsledky výkonnostných testov pre JMS poskytovateľov v JBoss AS 5.

5.3 Zhrnutie výsledkov

Na základe výsledkov v predchádzajúcich podkapitolách 5.1 a 5.2 je možné vysloviť záver, že integrácia JMS poskytovateľov tretích strán do JBoss AS bola úspešná. Integrovaní JMS poskytovatelia dokážu posielať správy v transakciách, vedia rozdeľovať záťaž pri práci v clusteri a pracovať aj po výpadku jedného uzla v clusteri aplikačných serverov. Pre predvolených JMS poskytovateľov JBoss Messaging a HornetQ skončili všetky testy podľa očakávaní bez chyby v oboch testovaných verziách JBoss AS.

Pri použití MRG skončili všetky transakčné aj clusterové testy úspešne a výsledný reťazec sa zhodoval s očakávaným výsledkom testu. Výkonnostné testy ukázali, že rýchlosť pri použití MRG je porovnateľná s rýchlosťou predvolených JMS poskytovateľov, pri vyšších

počtoch správ (rádovo desaťtisíce) je dokonca mierne vyššia.

Pri použití WebSphere MQ skončili transakčné testy opäť úspešne podľa očakávaných výsledkov. Pri clusterových testoch však jeden test vrátil nesprávny výsledok, kedy sa počet prijatých a odoslaných správ nerovnal a prijatých bolo o 4 správy viac. To znamená, že hoci WebSphere MQ dokáže pracovať s clusterom aplikačných serverov, nezaručuje doručenie správ práve jedenkrát. Pri viacnásobnom spustení tohto testu sa opakovala rovnaká situácia, vždy bolo prijatých viac alebo rovnaký počet správ, nijaká správa sa však nestratila. Obidva testy vysokej dostupnosti skončili úspešne. Čo sa rýchlosti spracovania týka, WebSphere MQ je výrazne pomalší ako MRG alebo predvolení JMS poskytovatelia.

Rozdiely vo výkonnosti nie sú len medzi jednotlivými JMS poskytovateľmi, ale aj medzi rôznymi verziami JBoss AS. Systém MRG dosiahol s JBoss AS 5 lepšie výsledky ako s JBoss AS 7, rovnako JBoss Messaging v JBoss AS 5 bol rýchlejší ako HornetQ v JBoss AS 7. Pri menších počtoch správ sú rozdiely minimálne, pri počtoch v rádoch tisícov sú výsledky vyrovnané, ale pri rádoch desaťtisícov a vyšších sú už rozdiely výrazné. Napríklad na prijatie 100000 správ potrebuje JBoss Messaging 6832 milisekúnd, HornetQ až 27812 milisekúnd.

Kapitola 6

Záver

Cieľom práce bolo navrhnúť a popísať integráciu dvoch JMS poskytovateľov tretích strán do aplikačného servera JBoss a ďalej navrhnúť a implementovať testovaciu sadu transakčných a clusterových testov na overenie funkčnosti navrhnutej integrácie. Zadanie bolo splnené v plnom rozsahu, naviac bol vypracovaný postup integrácie tretieho JMS poskytovateľa a do testovacej sady boli zahrnuté aj výkonnostné testy pre porovnanie rýchlosti jednotlivých JMS poskytovateľov. Implementovaná testovacia sada umožňuje spustenie testov nielen s integrovanými JMS poskytovateľmi, ale aj s predvolenými poskytovateľmi jednotlivých verzií JBoss AS. Celé testovanie bolo navyše plne automatizované.

Najprv bolo nutné zoznámiť sa s potrebnými technológiami, najmä s komunikáciou pomocou zasielania správ a štandardmi Java Message Service a Java EE Connector Architecture, ktoré sú kľúčové pre integráciu. Java Message Service poskytuje rozhranie pre asynchrónnu komunikáciu zasielaním správ a Java EE Connector Architecture umožňuje komunikáciu aplikačného servera s externými systémami, ktorými sú v tomto prípade jednotliví JMS poskytovatelia. Ďalšie významné technológie tvoria samotní JMS poskytovatelia ako od JBoss, tak aj poskytovatelia tretích strán.

Na základe preštudovaných technológií boli navrhnuté konkrétne postupy integrácie JMS poskytovateľov tretích strán do aplikačného servera JBoss. Pre túto úlohu boli vybrané tri systémy posielania správ. Prvým bol komerčný systém WebSphere MQ od IBM, druhým open source ActiveMQ od Apache a tretím open source Red Hat MRG Messaging. Všetky tri systémy poskytujú Java EE Connector Architecture adaptér zdrojov, ktorý je hlavným komponentom pri integrácii systému posielania správ do aplikačného servera. Základná funkčnosť integrácie bola pri všetkých vybraných systémoch overená pomocou komponentu Message Driven Bean. Message Driven Bean nasadený v aplikačnom serveri je schopný prijať správy zasielané do fronty na JMS serveri externého poskytovateľa.

Na overenie integrácie JMS poskytovateľov do JBoss AS bola navrhnutá a implementovaná univerzálna testovacia sada. Testovacia sada využíva na posielanie a prijímanie správ komponenty Message Driven Bean a servlet a je podľa zamerania rozdelená na dve časti. Prvá časť obsahuje transakčné testy, ktorých obmena je použitá aj na testovanie výkonnosti. Výkonnostné testy merajú čas potrebný na spracovanie správ a slúžia na porovnanie jednotlivých JMS poskytovateľov a rôzne verzie JBoss AS. Druhá časť testovacej sady je zameraná na clusterové testy, kedy je použitý cluster dvoch aplikačných serverov. Táto časť obsahuje aj testy vysokej dostupnosti, pri ktorých je jeden z uzlov v clusteri zastavený a klient sa musí pripojiť na pracujúci uzol.

Vlastná implementácia testovacej sady bola realizovaná v programovacom jazyku Java. Kvôli odlišnostiam JBoss AS 5 a JBoss AS 7 vznikli dve samostatné aplikácie, každá pre

jednu z verzií aplikačného servera. Testovacia aplikácia pre JBoss AS 5 obsahuje implementáciu pre troch JMS poskytovateľov, a to WebSphere MQ, MRG Messaging a predvolený JMS poskytovateľ JBoss Messaging. V JBoss AS 7 je predvolený JMS poskytovateľ HornetQ a ďalším použitým JMS poskytovateľom v testovacej aplikácii pre túto verziu aplikačného servera je MRG Messaging. Spustenie testov je možné tromi spôsobmi od úplne manuálneho prístupom na URL servletu vo webovom prehliadači, až po plne automatizované pomocou nástroja pre priebežnú integráciu Jenkins. Pre preklad, vytvorenie archívu s testovacou aplikáciou a spustenie testov je použitý nástroj na správu projektov Maven.

Výsledná testovacia bola použitá na overenie funkčnosti integrácie JMS poskytovateľov do JBoss AS. Testovanie prebiehalo aj automatizovane s použitím inštalácií JMS serverov vo firme Red Hat, aj manuálne. Z dosiahnutých výsledkov vyplýva, že integrácia bola úspešná. Pre MRG skončili všetky transakčné aj clusterové testy bez chyby. Pre WebSphere MQ skončili transakčné testy úspešne, jeden z clusterových testov však skončil s chybou, kedy bolo viac správ prijatých ako odoslaných. Výsledky výkonnostných testov ukázali, že najrýchlejší z testovaných JMS poskytovateľov boli predvolení poskytovatelia JBoss Messaging a HornetQ, avšak MRG Messaging dosiahol porovnateľné výsledky. WebSphere MQ bol vo všetkých testoch výrazne pomalší. Rozdiely vo výkonnosti sa ukázali aj medzi rôznymi verziami JBoss AS. MRG aj predvolený JMS poskytovateľ dosiahli v JBoss AS 5 lepšie výsledky ako v JBoss AS 7. Rozdiely sa začali prejavovať pri počte správ 10000 a vyššom.

Práca ponecháva priestor pre možné rozšírenia. Prvým rozšírením je použitie viacerých JMS poskytovateľov. Testovacia sada je navrhnutá tak, aby bol tento spôsob rozšírenia čo najjednoduchší. Najnáročnejšou časťou pri pridaní ďalšieho JMS poskytovateľa je návrh jeho integrácie do JBoss AS. Keďže testovacia aplikácia je samostatná pre každú verziu aplikačného servera, nie je nutná integrácia do oboch použitých verzií. Po úspešnej integrácii ľubovoľného množstva JMS poskytovateľov postačí pre každého poskytovateľa podediť každú generickú triedu a implementovať jej abstraktné metódy, ako bolo popísané v kapitole 4.2. Ďalšou možnosťou rozšírenia práce je testovanie integrácie do väčšej hĺbky, čím sa otestuje väčšia časť funkcionality JMS serverov. Môže sa pridať zameranie na ďalšie oblasti okrem štyroch implementovaných alebo sa môžu navrhnuté zamerania rozšíriť o ďalšie testovacie scenáre. Dodatočnými oblasťami funkcionality JMS serverov sú napríklad topiky, filtrovanie správ, mosty alebo rôzne typy JMS správ.

Literatúra

[1] IBM WebSphere MQ information center [online]. 2011 [cit. 2012-10-11].
URL http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/in%dex.jsp

[2] JSR 220: Enterprise JavaBeansTM,Version 3.0 : EJB 3.0 Simplified API [online]. Máj 2006 [cit. 2012-10-12].
URL http://download.oracle.com/otn-pub/jcp/ejb-3_0-fr-eva%l-oth-JSpec/ejb-3_0-fr-spec-simplified.pdf

[3] Java 2 EE Connector Architecture Specification : Version 1.5 [online]. November 2003 [cit. 2012-10-22].
URL http://download.oracle.com/otn-pub/java/j2ee_connecto%r/1.5-fr/j2ee_connector-1_5-fr-spec.pdf

[4] JDBC Overview [online]. November 2003 [cit. 2012-10-22].
URL http://www.oracle.com/technetwork/java/overview-141217%.html

[5] Binildas, C. A.: *Service Oriented Java Business Integration*. Packt Publishing, 2008, iSBN 9780-1-847194-40-4.

[6] Curry, E.: *Message-Oriented Middleware*. John Wiley & Sons, Ltd, 2004, iSBN 9780470862087.

[7] Fox, T.; e.a.: JBoss Messaging 2.0 User Manual : Setting the Standard for High Performance Mesaging [online]. 2009-07-27 [cit. 2012-10-08].
URL http://labs.jboss.com/file-access/default/members/jbos%smessaging/freezone/docs/usermanual-2.0.0.beta4/pdf/JBossMessaging_UserManual%.pdf

[8] Hynar, M.: *Java – nástroje*. Praha, Neocortex, 2004, iSBN 80-86330-16-8.

[9] Jamae, J.; Johnson, P.: *JBoss in Action*. Greenwitch, Manning Publications, 2009, iSBN 978-1-933988-02-3.

[10] Jendrock, E.; e.a.: Java EE 6 Tutorial : Basic Concepts [online]. Júl 2012 [cit. 2012-10-01].
URL http://docs.oracle.com/javaee/6/tutorial/doc/index.htm%l

[11] Matthews, J.: *Computer Networking : Internet Protocols in Action*. Hoboken: John Wiley & Sons, 2005, iSBN 978-0-471-66186-3.

[12] Mukhar, K.; Zelnak, C.: *Beginning Java EE 5 : From novice to professional*. Apress, 2006, iSBN 15-905-9470-3.

[13] Richards, M.; Monson-Haefel, R.; Chappel, D. A.: *Java Message Service*. O'Reilly, 2009, iSBN 978-0-596-52204-9.

[14] Smart, J. F.: *Jenkins : The Definitive Guide*. O'Reilly, 2010, iSBN 978-1449305352.

[15] Snyder, B.; e.a.: *Active MQ in Action*. Greenwitch, Manning Publications, 2011, iSBN 978-1933988948.

[16] Suconic, C.; e.a.: HornetQ User Manual : Putting the buzz in messaging [online]. 2012-04-19 [cit. 2012-10-08].
URL http://docs.jboss.org/hornetq/2.2.14.Final/user-manual%/en/pdf/HornetQ_UserManual.pdf

[17] Vinoski, S.: Advanced Message Queuing Protocol [online]. 2006 [cit. 2013-03-02].
URL http://steve.vinoski.net/pdf/IEEE-Advanced_Message_Q%ueuing_Protocol.pdf

[18] Wulf, J.: Red Hat Enterprise MRG 2 : Messaging Installation and Configuration Guide [online]. 2012-11-14 [cit. 2013-03-02].
URL https://access.redhat.com/knowledge/docs/en-US/Red_Ha%t_Enterprise_MRG/2/html-single/Messaging_Installation_and_Configuration_%Guide/index.html

Dodatok A

Zoznam skratiek

AMQ Active Message Queueing

CCI Common Client Interface

EAR Enterprise Archive

EIS Enterprise Information System

EJB Enterprise Java Beans

ESB Enterprise Service Bus

JAAS Java Authentication and Authorization Service

JAXM Java API for XML Messaging

JCA Java Connector Architecture

JDBC Java Database Connectivity

JMS Java Message Service

JMX Java Management Extensions

JNDI Java Naming and Directory Interface

JVM Java Virtual Machine

MDB Message Driven Bean

MOM Message Oriented Middleware

MRG Messaging Realtime Grid

POJO Plain Old Java Object

SSL Secure Sockets Layer

SOA Service Oriented Architecture

Dodatok B

Obsah CD

Priložené CD obsahuje:

- LaTeXa PDF verziu tejto práce
- zdrojové kódy testovacej sady a konfiguračné súbory
- vygenerované správy s výsledkami testov
- binárnu spustiteľnú verziu testovacej aplikácie
- súbor README s popisom použitia testovacej aplikácie