TARTU ÜLIKOOL
Matemaatika-informaatikateaduskond
Arvutiteaduse instituut
Informaatika õppekava

Raivo Eriksoo

Java Message Service objektide hoidla realisatsioon .NET platvormil

Bakalaureusetöö (6 EAP)

Juhendaja: Vambola Leping

Tartu 2014

Java Message Service objektide hoidla realisatsioon .NET platvormil

Lühikokkuvõte:

Antud töö tutvustab sõnumivahetust ja Java Message Service valdkonda. Praktilise osana realiseeritakse JMS objektide hoidla .NET platvormil. Valminud JMS hoidlale tehakse testid hindamaks tarkvara jõudlust.

Võtmesõnad:

Sõnumivahetus, Java Message Service, hoidla

The Pool of Java Message Service Objects Implementation on .NET Platform

Abstract:

The goal of this thesis is to give base knowledge of messaging and Java Message Service. In the practical part of the thesis pool of JMS objects on .NET platform will be implemented. JMS pool will be tested in order to assess the performance of the software.

Keywords:

Messaging, Java Message Service, pool

Sisukord

1 Sissejuhatus

Infotehnoloogia tähtsust tänapäeval ei alahinda arvatavasti enam mitte keegi. On loomulik, et inimesed soovivad laitmatult, vigadeta ning kiirelt töötavaid arvutisüsteeme. Meid ümbritseb ühe rohkem süsteeme, mille korrasolekust sõltub igaühe elu päevast päeva. Keegi meist ei soovi, et vigaste arvutiprogrammide tõttu kaob näiteks elekter või jääb palk õigel ajal saamata.

Arvutisüsteemid võib tinglikult jaotada kaheks. Esiteks süsteemid, mis on täiesti autonoomsed. Need ei sõltu teistest süsteemidest. Ja teiseks süsteemid, mis suhtlevad ka teiste süsteemidega. Töö autor väidab oma kogemuse põhjal erasektorist, et autonoomsed süsteemid on selges vähemuses. Suurem enamus kasutusel olevatest süsteemidest on seotud ehk liidestatud teiste süsteemidega.

Antud bakalaureusetöö esimene eesmärk on anda lugejale baasteadmised sõnumivahetusest ja JMS valdkonnast. Tutvustatakse peamisi JMS objekte, nende eesmärke ja kasutamist. Õige JMS objektide kasutamise viis tagab süsteemi jõudluse. JMS objektide loomine on aega- ja ressurssinõudev tegevus. Et arvutisüsteem oleks käideldav ka suurel koormusel, tuleb loodud objektid alles hoida taaskasutuseks JMS objektide hoidlas (JMS *pool*'is).

Töö teine eesmärk on realiseerida JMS objektide hoidla Microsoft .NET [6] platvormil. Töö autor valis .NET platvormi, sest erinevalt Java platvormist, pole .NET platvormil olemas üldlevinud JMS objektide hoidlat. Antud töö tulemusi saavad kasutada tarkvaraarendajad, keda huvitab, kuidas JMS objektide hoidlat realiseerida või kes soovivad võtta kasutusele töö käigus valmiva hoidla.

1.1 Töö struktuur ja eelteadmised

Töö on jaotatud kolme ossa. Töö esimeses peatükis tutvustakse lugejale JMS valdkonda. Selles peatükis antavad teadmised loovad eelduse järgnevate peatükkide mõistmiseks. Töö teises osas esitatakse lahendus JMS objektide hoidla realisatsioonile. Töö viimases osas hinnatakse hoidla realisatsiooni jõudlustestidega.

Töö nõuab lugejalt programmeerimise kogemust või oskust koodi lugeda. JMS koodinäited on esitatud programmeerimiskeeles Java ja hoidla on programmeeritud keeles C#.

1.2 Eestikeelsed terminid

Töö autor ei leidnud avalikest interneti allikatest Java Message Service valdkonna kohta eestikeelseid materjale, millest oleks saanud eeskuju võtta seonduva terminoloogia tõlkimisel. Autor tõlkis enda subjektiivse nägemuse järgi järgnevad valdkonna terminid, mida kasutatakse kogu töös:

message – sõnum

messaging - sõnumivahetus

broker – vahenduskomponent

point-to-point messaging – punktist-punkti sõnumivahetus

publish-subscribe messaging – publitseeri-telli sõnumivahetus

destination – sihtpunkt. Objekt vahenduskomponendis, kuhu saadetakse ja kust loetakse sõnumeid.

queue – järjekord. Sihtpunkt punkist punkti sõnumivahetuses.

topic – teema. Sihtpunkt publitseeri-telli sõnumivahetuses.

JMS pool – JMS objektide hoidla. Tarkvaraline komponent, mis hoiab taaskasutavaid JMS objekte arvuti mälus.

JMS provider – JMS toode. Sõnumivahetuse funktsionaalsust realiseeriv toode, mis pakub välja JMS liidest. Näiteks: ActiveMQ [8], SonicMQ [7], WebShpereMQ [9].

JMS sender – JMS sõnumite saatja. JMS standardkomponent sõnumite saatmiseks.

JMS receiver – JMS sõnumite saaja. JMS standardkomponent sõnumite vastuvõtmiseks.

JMS selector – JMS sõnumite selekteerija. JMS tehnoloogia sõnumite filtreerimiseks.

2 JMS valdkonna tutvustus

2.1 Süsteemiintegratsiooni stiilid

Sõnumivahetus on süsteemide vahelise integratsiooni üks stiilidest. Ettevõtete integratsiooni mustrid (*enterprise integration patterns*) loetleb neli erinevat integratsiooni stiili [10]:

1) failiedastus (*file transfer*)
2) jagatud andmebaas (*shared database*)
3) kaugprotseduuri käivitus (*Remote Procedure Invocation*)
4) sõnumivahetus (*messaging*)

Failiedastuse korral suhtlevad süsteemid omavahel läbi failide. Süsteemid vahetavad läbi FTP protokolli faile või omavad ühist failiserverit, kuhu loetakse ja kirjutatakse faile.

Jagatud andmebaasi korral on erinevatel süsteemidel ligipääs ühisele andmebaasile. Näiteks üks süsteem kirjutab andmebaasi, teine süsteem loeb andmebaasist.

Kaugprotseduuride käivituse (*RPC*) korral süsteemid pakuvad välja protseduure/funktsioone, mida teised süsteemid saavad välja kutsuda. Tüüpiliseks näiteks on siin veebiteenused [11]. Näiteks üks süsteem pakub veebiteenust ja mingid teised süsteemid tarbivad seda veebiteenust.

Sõnumivahetuse korral on süsteemid ühendatud sõnumivahetust pakkuva tootega, mis vahendab sõnumeid. Süsteemid vahetavad omavahel andmeid sõnumitena ja käituvad vastavalt sõnumitele.

2.2 Sõnumivahetuse alused

Sõnumivahetus on kommunikatsiooni meetod arvutisüsteemide vahel. Sõnumivahetus süsteemid (*messaging systems)* koosnevad sõnumivahetuse klientidest ja sõnumivahetust pakkuvast tootest.

Sõnumivahetuse toode ehk vahenduskomponent (*broker*) pakub klientidele sihtpunkte. Sihtpunktid on objektid vahetuskomponendis. Sõnumivahetuse kliendid saadavad ja

loevad sõnumeid sihtpunktist. Ehk siis, süsteemide vaheliseks kokkupuutepunktiks on ainult sihtpunktid vahenduskomponendis.

Sõnumivahetuse tootel on olemas liidesed (*API*), millega klient saab ühenduda vahenduskomponendiga ja millega saab koostada, saata ja vastu võtta sõnumeid. Antud töö käsitleb JMS liidest, kuid on olemas ka alternatiive. Näiteks AMQP (*Advanced Message Queuing Protocol*) ja XMPP (eXtensible Messaging and Presence Protocol) [12]. Joonis 1 kujutab sõnumivahetuse süsteemi.

Joonis 1. Sõnumivahetus süsteem

Sõnumivahetuse klient saab saata ja vastu võtta sõnumeid teistelt sõnumivahetuse klientidelt. Iga klient on ühendatud sõnumivahetust pakkuva tootega ehk vahenduskomponendiga. Sõnumi saatja saadab sõnumi sihtpunkt ja sõnumi vastuvõtja saab sihtpunktist sõnumit lugeda. Sõnumi saatja ja vastuvõtja ei pea olema samaaegselt kättesaadavad, et omavahel kommunikeeruda – nad ei tea üksteisest midagi. Sõnumi saatja teab ainult seda, kuhu sõnumit saata ja sõnumi vastuvõtja teab ainult seda, kust sõnumit lugeda [12].

Sõnumivahetuse kliendid saadavad sõnumeid üle võrgu. Vastavalt sõnumivahetust pakkuvale tootele, võivad võrguprotokollideks olla TCP/IP, HTTP, SSL, IP *multicast* [1].

Üks tähtsamatest sõnumivahetuse kontseptsioonidest on see, et sõnumeid süsteemide vahel saadetakse ja loetakse asünkroonselt. See tähendab, et sõnumi saatja ei pea ootama selle järel, et kas sõnumi saaja sai sõnumi kätte või sõnumi saaja protsessis sõnumi ära. [1].

2.3 Sõnumivahetuse eelised

Sõnumivahetuse eelised on: võimalus integreerida heteroognseid platvorme, vähendada süsteemi pudelikaelasid, suurendada süsteemi skaleeritavust [1].

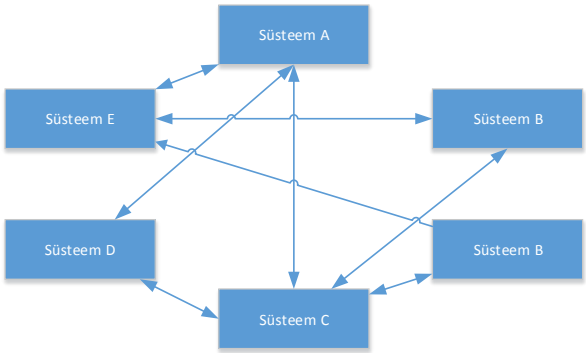
Tihti on integreemist vajavad süsteemid üles ehitatud erinevatel platvormidel ehk platvormid on heteroognsed. Näiteks üks süsteemidest baseerub .NET platvormil ja teine Java platvormil. Sõnumivahetuse tooted pakuvad liideseid erinevatele platvormidele. Näiteks toode SonicMQ pakub JMS liidese teeke platvormidel .NET, C/C++ ja Java. [13]

Süsteemi pudelikaelad ilmnevad siis, kui süsteem ei jõua protsessida päringuid enam nii kiiresti kui uusi päringuid peale tuleb. Klassikaline näide on siin halvasti disainitud andmebaasiga süsteem. Sellise süsteemi tarbijad peavad ootama näiteks kuni süsteemi andmebaasi ühendused on saadaval või kuni andmebaaside lukud vabanevad. Taoline süsteem suudab teenindada limiteeritud arvu päringuid ja ilmselt muutub see süsteem tarbijatele pudelikaelaks. Sõnumivahetust saab kasutada pudelikaelade vähendamiseks. Selle asemel, et oodata päringute vastuseid seni kuni sünkroonne süsteem töötleb neid, saab sõnumid saata sõnumivahetuse süsteemi, mis jagab sõnumid laiali mitme sõnumikuulaja vahel.

Suht samamoodi nagu saab vähenda pudelikaelu, saab tõsta ka süsteemi skaleeritavust ja läbilaske võimet. Sõnumisüsteemis saavutatakse skaleeritavus mitut sõnumi vastuvõtjat sisse tuues, mis suudavad sõnumeid töödelda paralleelselt. Teine viis tõsta skaleeritavust on teha süsteem võimalikult palju asünkroonseks. Selliselt tarkvara komponente lahutades saab süsteeme skaleerida horisontaalselt [1].

2.4 Kaugprotseduuri käivituse ja sõnumivahetuse võrldus

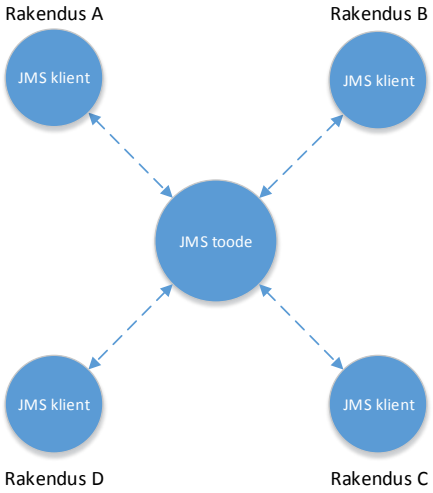
Peamine erinevus kaugprotseduuri käivituse (*RPC*) süstmeemide ja sõnumivahetus süsteemide vahel on see, et *RPC* süsteemid on sünkroonsed ja suhtlevad omavahel otse. Joonisel 2 on tüüpilise *RPC* süsteemi arhitektuur.



Joonis 2. Kaugprotseduuri käivitussüsteemi arhitektuur

Selline arhitektuur tekitab süsteemide vahel tugeva sõltuvuse – ühe süsteemi seisakul on otsene mõju teistele süsteemidele. Lisaks on selliseid süsteeme raske hallata ja monitoorida.

Joonisel 3 on toodud tüüpilise sõnumivahetussüsteemi arhitektuur.



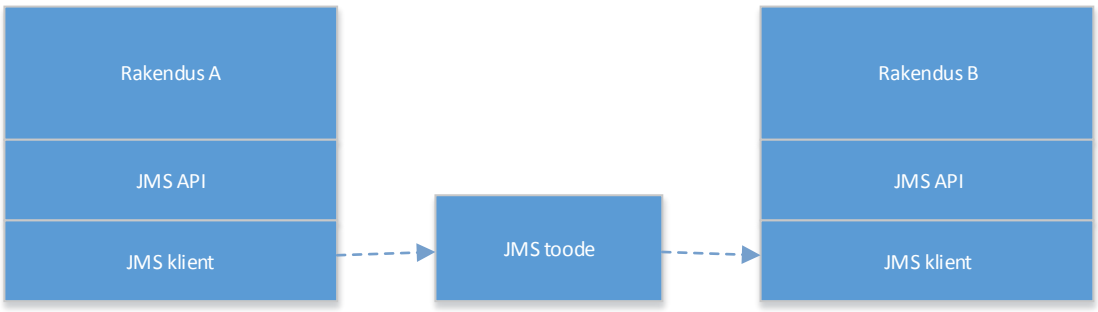
Joonis 3. Sõnumivahetus süsteemi arhitektuur

Süsteemid sõnumivahetuses ei ole otse seotud ehk süsteemid on nõrgalt seotud. Süsteemide vahel on kokkupuutepunktiks ainult sõnumivahetuse toode. Süsteemid käituvad asünkroonselt – kui üks süsteem saadab teisele süsteemile sõnumi, siis sõnumit saatev süsteem ei jää ootama vastust. See ongi peamine erinevus *RPC* ja sõnumivahetuse vahel. Sellest tulenevalt ühe süsteemi seisak ei põhjusta teiste süsteemide seismajäämist. Kuna kõik sõnumid käivad läbi vahenduskomponendi, siis sellised süsteemid on paremini monitooritavad ja hallatavad.

2.5 JMS tutvustus

JMS on üks Java Enterprise Edition standarditest, mis spetsifitseerib ära liidesed, klassid ning meetodid sõnumite saatmiseks klientide vahel [2]. JMS standardist on aja jooksul tulnud välja kolm versiooni: JMS 1.0.2b (25. juuni 2001), JMS 1.1 (18. märts 2002), JMS 2.0 (21. mai 2013) [4]. Peamiseks erinevuseks JMS 1.1 ja JMS 2.0 vahel peab töö autor eelkõige kasutusmugavust. JMS 2.0 lihtsustab teatud määral JMS liideste kasutamist [2]. Kuna JMS 2.0 standard on suhteliselt uus standard ja töö autori hinnangul on enamus JMS süsteeme implementeeritud JMS 1.1 liidesega ning ka hoidla realisatsioon kasutab ainult JMS 1.1 võimalusi, siis autor ei käsitle JMS 2.0 versiooni.

JMS sõnumivahetus süsteemi kirjeldab Joonis 4.



Joonis 4. JMS sõnumivahetussüsteem

Jooniselt näeme, et JMS sõnumivahetussüsteem põhimõtteliselt ei erine üldisest sõnumivahetussüsteemist - ka JMS süsteem koosneb klientidest ja sõnumivahetust pakkuvast tootest (antud juhul JMS toode). JMS klient saab olla kas sõnumi saatja või sõnumi vastuvõtja või ka mõlemat korraga.

JMS toote peamised omadused on:

1) pakub JMS liidest (JMS API),
2) pakub JMS klientidele sihtpunkte – järjekordasid ja teemasid,
3) pakub administreerimisliidest.

JMS toetab kahte sõnumivahetuse mudelit:

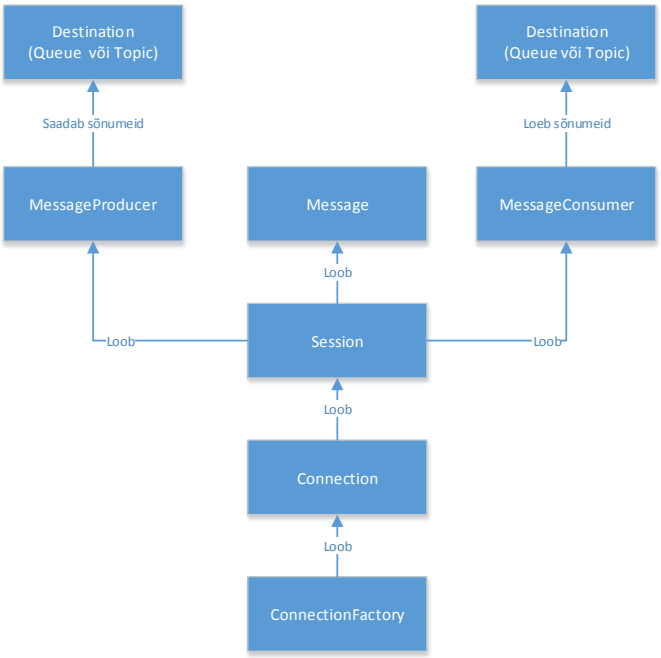
1) punkist punkti sõnumivahetus,
2) publitseeri-telli sõnumivahetus.

Neid mudeleid tutvustakse peatükkides 1.4.3 ja 1.4.4.

Võrreldes AMQP ja XMMP sõnumivahetusstandarditega ei määra JMS ära sõnumi formaati. Erinevatel JMS toodetel on küll üks JMS liides, kuid see, milline on sõnumiformaat, sõltub juba konkreetsest JMS tootest. Näiteks ei saa saata SonicMQ JMS kliendiga sõnumit WebSphereMQ JMS tootesse.

JMS objektimudel

Joonis 5 kirjeldab JMS objektide mudelit ja objektide omavahelised seosed.



Joonis 5. JMS objekti mudel

ConnectionFactory on liides, mis loob Connection tüüpi objekti. Iga JMS toode pakub oma ConnectionFactory implementatsiooni. Toodete kaupa on need implementatsioonid erinevad. Näiteks SonicMQ ConnectionFactory implemenatsioonis saab määrata loodava Connection'ile parameetreid nagu: [13]

1) ühenduse tõrkekindlus (*fault tolerant*) – kui ühendus katkeb, siis SonicMQ JMS teek suubab ühenduse ise taastada,
2) ühenduse loomise aegumine (*connection timeout*) – määrab ära maksimaalse aja kaua ühendust üritatakse luua.

Connection ehk JMS ühenduse objekt esitab reaalset TCP/IP ühendust JMS tootesse. Connection liides määrab ära meetodid ühenduse alustamiseks, lõpetamiseks ning sulgemiseks [2]. Connection liidest kasutakse Session objekti ehk JMS sessiooni loomiseks.

Session ehk JMS sessiooni objekt esitab konteksti, millega saab luua sõnumeid, sõnumi saatjaid ja sõnumi vastuvõtjaid.

MessageProducer ehk sõnumi saatja on liides JMS standardis sõnumite saatmiseks sihtpunkti. Sõnumi saatmisel saab saatjale ette öelda näiteks sõnumi eluea (*timetolive*). Sõnumi eluiga määrab ära maksimaalse aja vahenduskomponendis, kus sõnum on loetav. Kui eluea jooksul sõnumit kliendi poolt ei loeta (juhtub siis, kui ühtegi JMS saajat pole sihtpunkti ühendatud), siis vahenduskomponendis sõnum kustutakse. Sõnumi eluaja

kasutamine on soovitatav, kui sõnumite kohale jõudmine saajale pole kriitiline – vahenduskomponendis on sihtpunktil maksimaalne mahutavus ja kui see saab täis, siis sellesse sihtpunkti sõnumeid enam saata ei saa.

MessageConsumer ehk sõnumi saaja on liides JMS standardis sõnumite lugemiseks sihtpunktist. Sünkroonsel lugemisel on kaks varianti: aegumisega ja aegumiseta. Aegumisega lugemisel antakse sõnumi saaja objektile ette maksimaalne ooteaeg (*timeout*). Kui ooteaeg saab täis, siis tagastatakse *null* sõnum kliendile. Asünkroonsel lugemisel registreeritakse sõnumi saajas MessageListener tagasikutsung (*callback*). Programmeerija ülesandeks jääb implementeerida MessageListner liideses *onMessage* meetod. Kui sõnum saabub, siis MessageConsumer kutsub MessageListener implementatsioonist välja *onMessage* meetodi.

Message ehk sõnum on informatsioon, mida kantakse edasi süsteemide vahel.

Destination ehk sihtpunkt on kanal, mille kaudu sõnumeid vahendatakse.

JMS objektid ja lõimed

JMS objekt	Toetab samaaegset kasutust
Destination	Jah
ConnectionFactory	Jah
Connection	Jah
Session	Ei
MessageProducer	Ei
MessageConsumer	Ei

Tabel 1. JMS objektide samaaegne kasutus

Tabelist 1 on näha, et Session, MessageProducer, MessageConsumer on samaaegselt kasutatavad ainult ühe lõime poolt. Objektide hoidla realisatsioonil tuleb sellega arvestada. Jagatud ressurss peab olema lukustatud, et teised lõimed seda ei kasutaks.

JMS sõnumi struktuur

JMS sõnum koosneb kolmest osast [2]:

1) päised (*headers*),

2) omadused (*properties*),

3) sisu (*payload*).

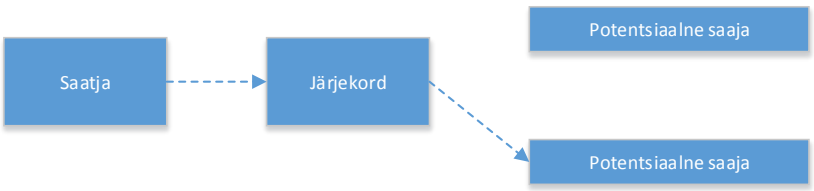
Ainuke kohustuslik element JMS sõnumis on sõnumi päis. Sõnumi päist kasutavad JMS kliendid ja vahenduskompnent sõnumite identifitseerimiseks ja ruutimiseks [12]. Sõnumi päis koosneb JMS standardiga eeldefineeritud võti-väärtus paaridest. Näiteks JMSMessageID, mille väärtus tekib automaatselt sõnumi saatmisel. Või JMSReplyTo päis, millega JMS saatja määrab ära, mis sihtpunkti oodatakse sõnumile vastust.

Sõnumi omadused on samuti võti-väärtus paarid. Erinevus päistest on see, et päise nimed ja otstarve on ette määratud, sõnumi omaduste nimed ja ka otstarbe saab ise valida. Sõnumi omadusi kasutatakse juhul, kui on vaja lisaks päistele veel lisada midagi sõnumisse ja seda ei saa lisada sõnumi sisusse. Selline olukord võib tekkida, kui sõnumi sisu on kindla struktuuriga, näiteks XML [14], mida muuta ei saa. Lisaks kasutatakse sõnumi omadusi, kui on vaja teha sõnumite filtreerimist (vt. peatükk 1.4.6).

Sõnumi sisu kannab edasi informatsiooni süsteemide vahel. Sõnumi sisuks võib olla näiteks XML, tavaline tekst või binaarsed andmed.

Punktist-punkti sõnumivahetus

Punktist-punkti sõnumivahetust kasutakse siis, kui on vaja saata sõnumit ainult ühele sõnumi vastuvõtjale. Isegi kui mitu erinevat sõnumikuulajat on järjekorda kuulamas, saab alati ainult üks sõnumi saaja sõnumi kätte. Sellist sõnumivahetust illustreerib Joonis 6.



Joonis 6. Punktist-punkti sõnumivahetus

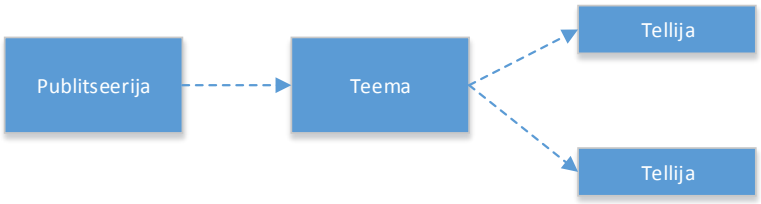
Sõnumeid vahendatakse läbi kanali, mida nimetatakse järjekorraks. Järjekord on sihtpunkt, kust JMS saatja saadab sõnumeid ja JMS saaja loeb sõnumeid. JMS sõnumite saatmine on alati sünkroonne ja JMS sõnumite lugemine on alati asünkroonne [2].

On kahte tüüpi punktist-punkti sõnumivahetust [1]. Esiteks on saada-ja-unusta sõnumi töötlemine. Sellisel juhul sõnumi saatja saadab sõnumi järjekorda ja saatja ei jää ootama vastust. Sellist tüüpi sõnumivahetust saab kasutada siis, kui vastust pole koheselt vaja. Töö autori arvates võiks selliselt käia näiteks monitoorimise süsteemi raportite saatmine.

Teiseks on päring-vastus tüüpi sõnumivahetus. Sellisel juhul saadab sõnumi saatja sõnumi järjekorda ja jääb blokeeruvalt ootama vastust sõnumi vastuvõtjalt. Näide sellist tüüpi sõnumivahetusest võib olla autentimissüsteemi kasutamine. Rakendus saadab päringu autentimissüsteemi ja ennem äriloogika käivitamisega edasi ei lähe, kui tuleb positiivne vastus.

Publitseeri-telli sõnumivahetus

Publitseeri-telli sõnumivahetust esitab Joonis 7.



Joonis 7. Publitseeri-telli sõnumivahetus

Publitseeri-telli sõnumivahetuses vahetatakse sõnumeid läbi kanali, mida nimetatakse teemaks. Sõnumi saatjat nimetatakse publitseerijaks ja sõnumi saajat tellijaks. Publitseerija saab saata ühte sõnumit korraga mitmele tellijale. See on ka peamine erinevus punktist-punkti sõnumivahetusest, kus sõnumi saajaid on maksimaalselt üks. [1].

JMS sõnumite filtreerimine

Mõnikord sõnumi saajat huvitab ainult teatud omadustega sõnumid järjekorrast või teemast. Ilma sõnumit filtreerimata saab sõnumi saaja kõik sõnumid oma sihtpunktist, millest sõnumeid loetakse. Sellega on kaks probleemi. Esiteks, tuleb koodis huvipakkuvad sõnumid programmis filtreerida, mis on lisakoormus sõnumi saajale. Teiseks, kõikide sõnumite saatmine sõnumi saajasse tekitab liigset võrgukoormust.

JMS API pakub selle probleemi lahendamiseks JMS sõnumi selekteerijaid. JMS selekteerija on SQL-iga sarnane lause konstruktsioon, mis antakse ette JMS sõnumi saaja loomisel. Kõik selekteerijad registreeritakse vahenduskomponendis. Vahenduskomponent filtreerib ise sõnumeid ja toimetab sõnumi saajale ainult filtreerimise tingimustele vastavad sõnumid. Sõnumite selekteerimist rakendatakse sõnumi päisele ja omadustele. Sõnumi sisule selekteerijaid rakendada ei saa.

Järgnev on näide JMS selekteerijast, mis valib välja sõnumid, mille omadustes PaymentType on võrdne stringiga Quick või ExtraQuick.

```
PaymentType = 'Quick' or PaymentType='ExtraQuick''
```

3 JMS objektide hoidla realisatioon .NET platvormil

JMS objektide hoidla kriitiline vajadus tekib siis, kui JMS liidest kasutav süsteem peab olema kiire ka suurel koormusel. Ilma hoidlata tuleb iga kord JMS liidese kasutamisel luua JMS objektid käigupealt – luuakse JMS ühendus, ühendus loob sessiooni, sessioon loob sõnumi saatja ja saaja. Selline loomine on ressurssi- ja aeganõudev tegevus, mille probleemiks on see, et nende objektide loomine eeldab suhtlust vahenduskomponendiga. See tekitab võrguliiklust ning koormab liigselt vahenduskomponenti. Suure koormuse korral mõjutab nimetatu oluliselt rakenduse töötamist – vasteajad suurenevad, halvimal juhul terve rakendus lõpetab töötamise. JMS objektide hoidlast on objektid taaskasutatavad, mis lahendab eelnevalt loetletud probleemid.

Antud töö vaatleb olukorda, kus on olemas üks .NET platvormi süsteem ja üks JAVA platvormi süsteem. .NET platvormi süsteem on veebiteenus ja JAVA süsteem JMS teenus. On vaja need kaks süsteemi ühendada - .NET veebiteenus peab hakkama saatma päringuid JAVA süsteemi ja lugema vastuseid päringutele JAVA süsteemist. Veebiteenus baseerub HTTP protokollil, mis on oma olemuselt sünkroonne. Seega hoidlas peab implementeerima päring-vastus tüüpi punkist-puntki sõnumivahetuse.

Funktsionaalsed nõuded:

1) hoidla peab olema tarkvaraline teek,
2) hoidla peab taaskasutuseks hoidma mälus JMS objekte,
3) konfiguratsioonis peab saama määrata JMS ühenduste arvu vahenduskomponenti,
4) konfiguratsioonis peab saama määrata JMS sessioonide arvu ühes ühenduses,
5) rakendusele peab JMS objektide hoidla pakkuma meetodit:

```
public string SendReceive (string senderName,  string message, long timeout)
```

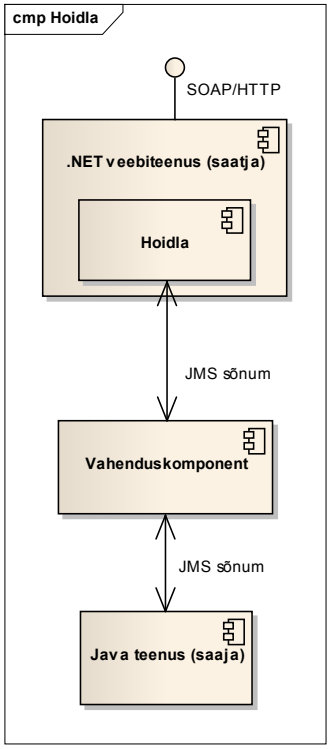
See meetod saadab sõnumi ära ning blokeerub (jääb ootele) kuni vastuse saabumiseni või kuni saabub aegumine (*timeout*).

Mittefunktsionaalsed nõuded:

1) hoidla peab suutma töötada klasterdatud keskkonnas,
2) hoidlast saadetavad ja vastuvõetavad sõnumid peavad olema monitooritavad.

3.1 Arhitektuur

Hoidla kasutust illustreerib järgnev joonis:



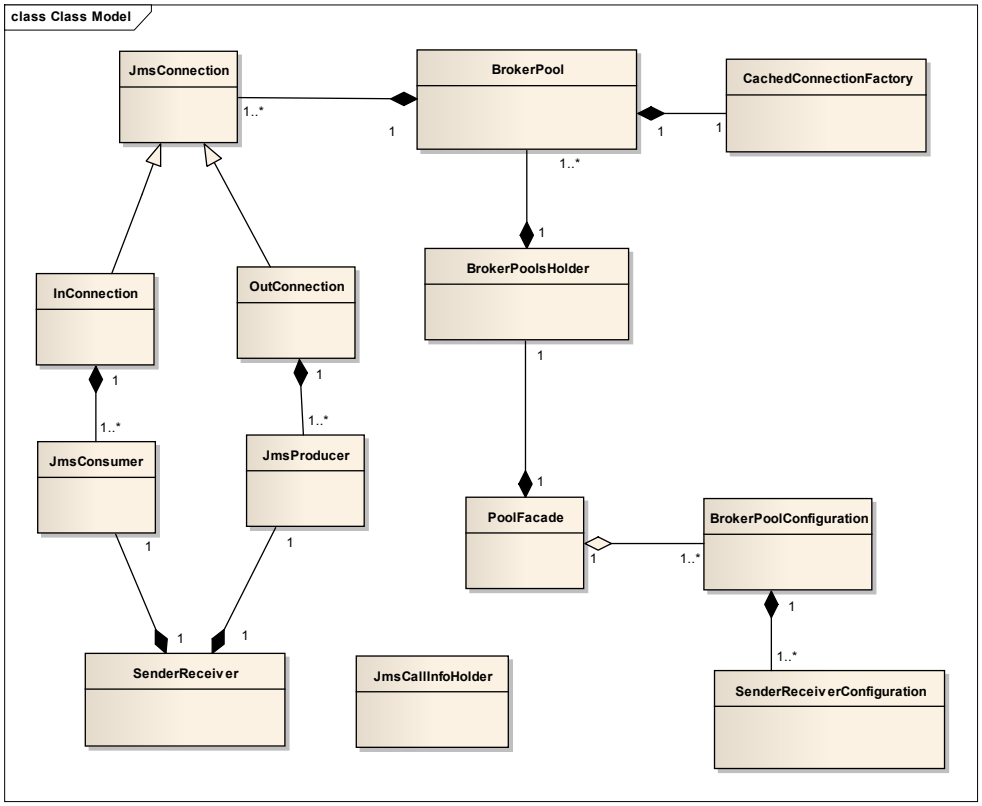
Joonis 8. Hoidla kasutuse arhitektuur

Antud töö käigus valmib hoidla ja lihtne .NET veebiteenus, mis töötab Microsoft Windows veebiserveris (IIS). Hoidla on tarkvaraline teek, mida rakendused saavad kasutada.

Optimaalne on hoidlas arvuti mällu salvestada kõik JMS objektid: ühendused, sessioonid, sõnumi saatjad ja sõnumi vastuvõtjad. Rakendus küsib hoidla käest ainult sõnumi saatja või saaja ning paneb need kasutamise ajaks lukku. Peale saatja ja saaja kasutamist rakendus ei kustuta neid, vaid paneb hoidlasse tagasi.

Lahenduse klassidiagramm

Järgnev klassidiagramm kirjeldab hoidla klasside omavahelised seosed.

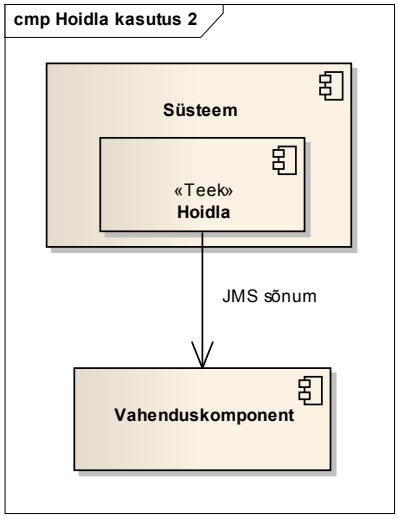


Joonis 9. Klassidiagramm

Klasside kirjeldused leiab tarkvara dokumentatsioonst. Vaata Lisa 1.

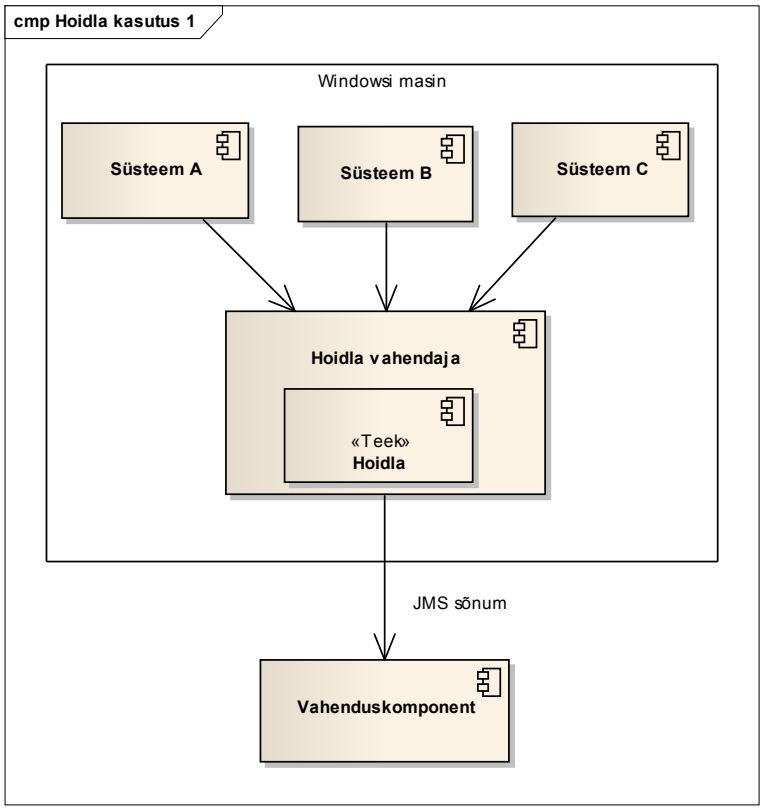
Hoidla kasutamise võimalused

Kui ühes Windowsi masinas on ainult üks süsteem, mis soovib hoidla funktsionaalsust kasutada, siis see süsteem võib hoidlat kasutada otse:



Joonis 10. Hoidla otsekasutus

Kui ühes Windowsi masinas töötab palju teenuseid, mis soovivad teha JMS sõnumivahetust, siis tasub mõedla hoidla vahendaja kasutamisele.



Joonis 11. Hoidla kasutamine läbi vahendaja

Ilma hoidla vahendajata teeks iga süsteem vahenduskomponenti omad JMS ühendused. Arvestades seda, et iga süsteem teeb mitu JMS ühendust, tekitab see olukorra, kus JMS ühendusi on soovimatult palju. Liiga palju JMS ühendusi hakkab vahenduskomponenti koormama. Selle vältimiseks saab süsteemide ja vahenduskomponendi vahele ehitada hoidla vahendamise rakenduse, mis teeb piiratud arv ühendusi vahenduskomponenti. Süsteemide ja hoidla vahendaja ühendamiseks võib kasutada näiteks TCP/IP protokolli.

Hoidla initsialiseerimine

Hoidla initsialiseerimine seisneb järgmistes sammudes:

1) laetakse sisse konfiguratsiooni fail (Web.config),
2) vahenduskomponendi ühenduste massiiv (*BrokerPool*) initsialiseeritakse ja luuakse reaalsed ühendused.

   Iga *BrokerPool* omab kahte ühenduste massiivi:

   a) väljuvate JMS ühenduste ja sessioonide massiiv,

b) sissetulevate JMS ühenduste ja sessioonide massiiv.

3) JMS saatjate-vastuvõtjate massiivide (SenderReceiverPool) initsialiseerimine
   a) luuakse esimene SenderReceiverPool,
   b) SenderReceiverPool'is on nii palju Sender-Receiver paare, kui on näidatud parameetriga "sender-receiver-pool.n.amount":
      i. Iga JMS saatja küsib hoidla käest ühe välja mineva JMS sessiooni,
      ii. Iga JMS vastuvõtja küsib hoidla käest ühe sissetuleva JMS sessiooni.
   c) Kui esimene SenderReceiverPool edukalt loodud, siis minnakse järgmise juurde. Tsüklis luuakse niipalju SenderRecieverPool'e, kui konfiguratsiooni failis on näidatud.

3.2 Võrldus Spring raamistiku JMS objektide hoidlaga

Spring raamistik [15] on populaarne raamistik rakenduste programmeerimisel. Spring raamistik pakub JMS standardile abstraktsiooni, mis tunduvalt lihtsustab programmeerijal JMS kasutamist. Samas on see nagu "must kast" – kuidas Spring JMS-i kasutab, üldiselt programmeerijaid autori hinnangul ei huvita (või pole aega huvituda). Töö autor uuris Springi JMS hoidla [16] lähtekoodi ja tuvastas Spring JMS hoidlas järgmised puudused:

1) ei saa määrata JMS ühenduste arvu vahenduskomponenti,
2) punktist-punkti päring-vastus tüüpi sõnumi implementatsioon on puudulik.

JMS ühenduste arvu määramise võimalus on rakenduse skaleerumise seisukohast oluline. Suure koormuse korral suureneb ka andmehulk ühenduses, mis hakkab ühendust koormama. Mitme ühenduse korral saab ühes ühendses liigutavate andmete hulka vähendada.

Spring hoiab mälus sõnumi saajaid (MessageConsumer objekte), kuid mällu salvestamine tehakse JMS selekteerija alusel. Springis tuleb sõnumi saatamisel sõnumi selekteerija ette öelda. Kuna igal sõnumil on oma selekteerija, siis iga kord luuakse uus sõnumi saaja ehk reaalselt neid ei taaskasututa. Põhimõtteliselt on võimalik, et Springi rakendusse ehitada

ise JMS selekteerijate hoidla ja sõnumi saatmisel võtta sellest hoidlast vaba selekteerija, kuid arhitektuuriliselt tähendab see seda, et rakenduses on kaks eraldi seisvat JMS valdkonda puudutavad hoidlat.

4 Testid valminud hoidlale

4.1 Testide eesmärk

Valminud hoidla realisatsioonile viiakse läbi testid. Testidel on kaks eesmärki. Kõigepealt soovitakse teada, millised on valminud lahenduse jõudlusnäitajad – sõnumite läbilaske võime ja päringute vasteajad (*response time*). Seejärel uuritakse, millise hoidla konfiguratsiooniga on jõudlusnäitajad kõige paremad. Proovitakse leida vastused järgnevatele küsimustele:

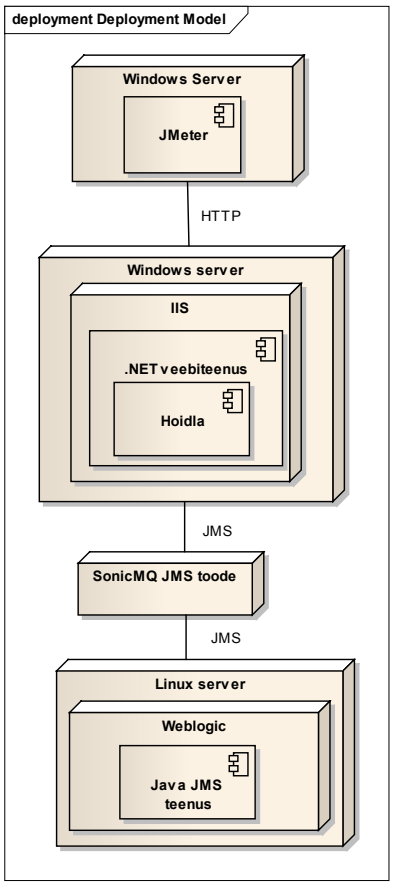
Kui palju JMS ühendusi on vaja luua vahenduskomponenti?

Mitu JMS sessiooni peaks ühes ühenduses olema?

4.2 Läbiviidud testid

Testide läbiviimiseks tehti koormustestid kasutades JMeter[17] tarkvara. JMeter võimaldab teha teste erinevate koormustega ja tulemused salvestada.

Testide arhitektuuri kirjeldab järgnev joonis:



Joonis 12. Testi arhitektuur

Testides saatis JMeter ühe kilobaidiseid XML sõnumeid .NET veebiteenusesse, mis hoidlat kasutades saatis need SonicMQ JMS tootesse. Sõnumi saajaks oli lihtne JAVA platvormil töötav SonicMQ JMS teenus, mis sissetulnud sõnumi peegeldas tagasi sõnumi saatjale.

Testide tulemused on näha järgnevates tabelites. Konfiguratsiooni veerus C=n/S=m tähendab, et antud test viidi läbi konfiguratsiooniga, kus:

a) JMS ühenduste arv oli *2n* (n ühendust sisse ja n ühendust välja),
b) igas JMS ühenduses oli *m* JMS sessiooni. Väljaminevas JMS ühenduses on *m* JMS saatjat ja väljaminevas JMS ühenduses on *m* JMS saajat.

Seega, saatja-saaja paaride arv testis on lihtsalt leitav valemiga *nm*.

Lõimesid	25	Päringute arv	5000	
Konfigu-ratsioon	Keskmine vasteaeg (ms)	Miinumum vasteaeg (ms)	Maksimum vasteaeg (ms)	Läbilaske võime (sõnumit sekundis)
C=1/S=10	10	6	36	851
C=1/S=25	9	6	31	860
C=1/S=50	8	7	18	914
C=1/S=100	8	6	13	909
C=2/S=10	7	6	17	911
C=2/S=25	8	6	15	940
C=2/S=50	9	6	14	989
C=2/S=100	7	6	17	909
C=5/S=10	7	6	19	933
C=5/S=25	8	6	18	914
C=5/S=50	8	6	19	984
C=5/S=100	7	6	16	956

Tabel 2. Testide tulemused 25 lõimega

Testidest on näha, et kõige halvemad vasteajad on saadud konfiguratsiooniga, kus on kaks JMS ühendust ja JMS sessioone ühenduses on 10 või 25. Kõige parem läbilaskevõime on 989 sõnumit sekundis, mis saavutati konfiguratsiooniuga C=2/S=50. See on seletatav sellega, et sellise konfiguratsiooniga loodi hoidlas piisavalt ette saatja-saaja paare. Kõige halvemad tulemused on seletatavad sellega, et hoidlas loodi nendel juhtumitel käigupealt

juurde JMS ühendusi, sessiooni, saatjaid, saajaid, sest lõimede arv oli suurem kui JMS saatja-saaja paaride arv.

Lõimesid	100	Päringute arv	5000	
Konfigu-ratsioon	Keskmine vasteaeg (ms)	Miinumum vasteaeg (ms)	Maksimum vasteaeg (ms)	Läbilaske võime (sõnumit sekundis)
C=1/S=10	50	36	85	660
C=1/S=25	57	32	83	695
C=1/S=50	49	39	79	657
C=1/S=100	51	43	73	725
C=2/S=10	46	41	84	720
C=2/S=25	49	37	90	637
C=2/S=50	48	39	93	708
C=2/S=100	44	39	64	710
C=5/S=10	59	38	73	714
C=5/S=25	50	42	78	689
C=5/S=50	56	35	67	677
C=5/S=100	42	38	71	643

Tabel 3. Testide tulemused 100 lõimega

Test 100 lõimega näitas meile samuti, et kõige halvemad tulemused saadakse kui saatja-saaja paaride arv on suurem kui lõimede arv ja kõige paremad tulemused saadakse siis, kui hoidlas on alati saadaval saatja-saaja paarid.

Lõimesid	250	Päringute arv	5000	
Konfigu-ratsioon	Keskmine vasteaeg (ms)	Miinumum vasteaeg (ms)	Maksimum vasteaeg (ms)	Läbilaske võime (sõnumit sekundis)
C=1/S=10	206	175	219	519
C=1/S=25	200	181	241	526
C=1/S=50	205	193	235	547
C=1/S=100	196	178	228	519
C=2/S=10	204	183	215	586
C=2/S=25	201	172	230	583
C=2/S=50	197	184	217	541
C=2/S=100	202	181	244	580
C=5/S=10	209	190	238	578

C=5/S=25	201	174	235	532
C=5/S=50	188	168	221	567
C=5/S=100	193	164	204	584

Tabel 4. Testide tulemused 250 lõimega

Ja ka test 250 lõimega kinnitab meile eelnevat- et saada kõige paremaid tulemusi, peab olema saatja-saaja paare piisavalt ette loodud.

Seega, kui on tegemist ühe kilobaidiste sõnumitega, siis pole JMS sessioonide ja ühenduste arv olulised. Oluline on aga see, et hoidlas oleks alati saadaval vaba JMS saatja-saaja paar.

5 Kokkuvõte

Antud töö esimene eesmärk oli anda baasteadmised sõnumivahetusest ja JMS valdkonnast. Tutvustati sõnumivahetuse põhimõtteid ja eeliseid. Anti ülevaade JMS objektidest ning nende omavahelistest seostest ja kasutamisest.

Teine eesmärk oli realiseerida JMS objektide hoidla .NET platvormil. Töö käigus arendati C# keeles tarkvaraline teek, mida programmeerijad saavad vajadusel kasutada oma rakendustes. Hoidlas realiseeriti punktist-punkti sõnumivahetuse üks liikidest - päring-vastus tüüpi sõnumivahetus.

Hindamaks hoidla jõudlust tehti vastavad testid. Testid viidi läbi erinevate konfiguratsioonide ja lõimede arvuga. Testide tulemused näitasid, et hoidla skaleerub ja et läbilaskevõime on hea.

Valminud tööl mitmeid edasiarendamise võimalusi. Esiteks võiks realiseerida saada ja unusta tüüpi asünkroonse sõnumivahetuse. Sellist sõnumivahetust saab kasutada asünkroonsetes süsteemides. Teiseks võiks täiustada jõudlusteste – esiteks võib uurida, et kuidas muutub hoidla jõudlus kui sõnumite suurused kasvavad. Ja teiseks võib uurida, et milline hoidla konfiguratsioon on suurte sõnumitega parim.

6 Viited ja kasutatud kirjandus

[1] Mark Richards, Richard Monson-Haefel, David A Chappell, “Java Message Service”, O'Reilly Media; 2 edition (19.05.2009)

[2] Mark Hapner, Rich Burridge, Rahul Sharma, Joseph Fialli, Kate Stout, “JSR-343 Java Message Service (JMS) 2.0”, Sun Microsystems; (20.05.2013)

https://jcp.org/aboutJava/communityprocess/final/jsr343/index.html (viimati vaadatud 26.04.2014)

[3] Introduction to Integration Styles

http://www.eaipatterns.com/IntegrationStylesIntro.html (viimati vaadatud 27.04.2014)

[4] Wikipedia – Java Message Service

http://en.wikipedia.org/wiki/Java_Message_Service (viimati vaadatud 27.04.2014)

[5] Wikipedia - Java Platform, Enterprise Edition

http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition (viimati vaadatud 27.04.2014)

[6] .NET raamistik

http://www.microsoft.com/net (viimati vaadatud 27.04.2014)

[7] SonicMQ

http://www.progress.com (viimati vaadatud 26.04.2014)

[8] ActiveMQ

http://activemq.apache.org/ (viimati vaadatud 26.04.2014)

[9] IBM WebSphere MQ

http://www-03.ibm.com/software/products/en/wmq (viimati vaadatud 26.04.2014)

[10] Introduction to Integration Styles

http://www.eaipatterns.com/IntegrationStylesIntro.html (viimati vaadatud 30.04.2014)

[11] Wikipedia – Web service

http://en.wikipedia.org/wiki/Web_service (viimati vaadatud 03.05.2014)

[12] Kim Haase, Java™ Message Service API Tutorial

http://docs.oracle.com/javaee/1.3/jms/tutorial/jms_tutorial-1_3_1.pdf (viimati vaadatud 03.05.2014)

[13] SonicMQ Application Programming Guide

http://documentation.progress.com/output/Sonic/8.0.0/Docs8.0/books/mq_application_program.pdf (viimati vaadatud 26.04.2014)

[14] Extensible Markup Language (XML) 1.0 (Fifth Edition)

http://www.w3.org/TR/REC-xml/ (viimati vaadatud 26.04.2014)

[15] Spring raamistik

http://docs.spring.io/spring-framework/docs/3.2.8.RELEASE/spring-framework-reference/html/ (viimati vaadatud 26.04.2014)

[16] Springi JMS lähtekood

https://github.com/spring-projects/spring-framework/tree/master/spring-jms (viimati vaadatud 26.04.2014)

[17] JMeter

http://jmeter.apache.org/ (viimati vaadatud 07.05.2014)

The Pool of Java Message Service Objects Implementation on .NET Platform

Bachelor's thesis (6 ECTS)

Raivo Eriksoo

Summary

The first purpose of this thesis was to provide the basic knowledge of messaging and the field of JMS. The principles and advantages of messaging were introduced. A review on JMS objects and the connection between them as well as on using them was given.

The second purpose was to realize the pool of JMS objects on the .NET platform. A software library on C# language which can be used by programmers in their applications if necessary was developed along with the progress of this thesis. One of the types of point-to-point messaging – query-response messaging – was realized in the pool.

In order to assess the performance of the pool, performance tests were carried out. These tests were carried out with different numbers of configurations and threads. The results showed that the pool scales and that the throughput is good.

This thesis has several possibilities for expansion. Firstly, a send and forget type of asynchronous messaging could be realized. This type of messaging can be used in asynchronous systems. Secondly, performance testing could be improved: (A) the changes in the performance of the pool in case of the growth in the size of the messages can be researched and (B) the best configuration for the pool in case of big messages can be researched.

Lisad

I. Hoidla lähtekood

Hoidla lähtekood on avalikus SVN repositooriumis ja on leitav internetist aaddresilt https://code.google.com/p/dotnetjmspoollib/.

II. Koodinäited JMS liidese kasutamisest

Ühenduse loomine

```
// loome ConnectionFactory
ActiveMQConnectionFactory connFactory = new ActiveMQConnectionFactory();
// loome ühenduse
Connection connection = connFactory.createConnection();
// peale start meetodi väljakutset on connection valmis saatja
// ja vastu võtma sõnumeid
connection.start();
```

Sessiooni loomine

```
// loome sessiooni. Parameetrid createSession meetodis
// 1) false – sessioon ei transaktsiooniline. Töö piiratud mahu tõttu JMS
//    transaktsioone ei käsitleta.
// 2) Session.AUTO_ACKNOWLEDGE – sõnumi saabumisel JMS klient annab automaatselt
//    teada sõnumi saatjale, et sõnum on loetud.
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

Sõnumi saatmine järjekorda

```
// Küsime sessioonilt viite järjekorrale nimega MYQUEUE
Queue queue = session.createQueue("MYQUEUE");
// Loome sõnumi saatja
MessageProducer producer = session.createProducer(queue);
// Loome sõnumi
String text = "Ping message”;
TextMessage message = session.createTextMessage(text);
// Saadame sõnumi
producer.send(message);
// Vabastame JMS objektid
producer.close();
session.close();
connection.close();
```

Sõnumi saatmine teemasse

```
// Küsime sessioonilt viite teemale nimega MYTOPIC
Topic topic = session.createQueue("MYTOPIC");
// Loome sõnumi saatja
MessageProducer producer = session.createProducer(topic);
// Loome sõnumi
String text = "Ping message”;
TextMessage message = session.createTextMessage(text);
// Saadame sõnumi
producer.send(message);
// Vabastame JMS objektid
producer.close();
session.close();
connection.close();
```

Sõnumi lugemine järjekorrast

```
// küsime sessioonilt viite järjekorrale MYQUEUE
Queue queue = session.createQueue("MYQUEUE");
```

```
// Loome sõnumi saaja
MessageConsumer consumer = session.createConsumer(queue);
// Loeme järgmise sõnumi järjekorrast ooteajaga 5 sekundit.
Message message = consumer.receive(5000);
// Prindime sõnumi
if (message instanceof TextMessage) {
   TextMessage textMessage = (TextMessage) message;
    String text = textMessage.getText();
   System.out.println("Received: " + text);
}
consumer.close();
session.close();
connection.close();
```

Sõnumi lugemine teemast

```
// küsime sessioonilt viite järjekorrale nimega MYTOPIC
Topic topic = session.createQueue("MYTOPIC");
// Loome sõnumi saaja
MessageConsumer consumer = session.createConsumer(topic);
// Loeme järgmise sõnumi teemast ooteajaga 5 sekundit.
Message message = consumer.receive(5000);
// Prindime sõnumi
if (message instanceof TextMessage) {
   TextMessage textMessage = (TextMessage) message;
    String text = textMessage.getText();
   System.out.println("Received: " + text);
}
consumer.close();
session.close();
connection.close();
```

III. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina **Raivo Eriksoo** (sünnikuupäev: 26.07.1978)
(*autori nimi*)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
Java Message Service objektide hoidla realisatsioon .NET platvormil,
(*lõputöö pealkiri*)

mille juhendaja on Vambola Leping,
(*juhendaja nimi*)

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **14.05.2014**