IDU0080

Seotud infosüsteemid -> infosüsteemide integratsioon -> veebteenused -> SOAP -> harjutustunni ülesanded.

1. Seotud infosüsteemid. Jagatud ülesanded, jagatud töö.

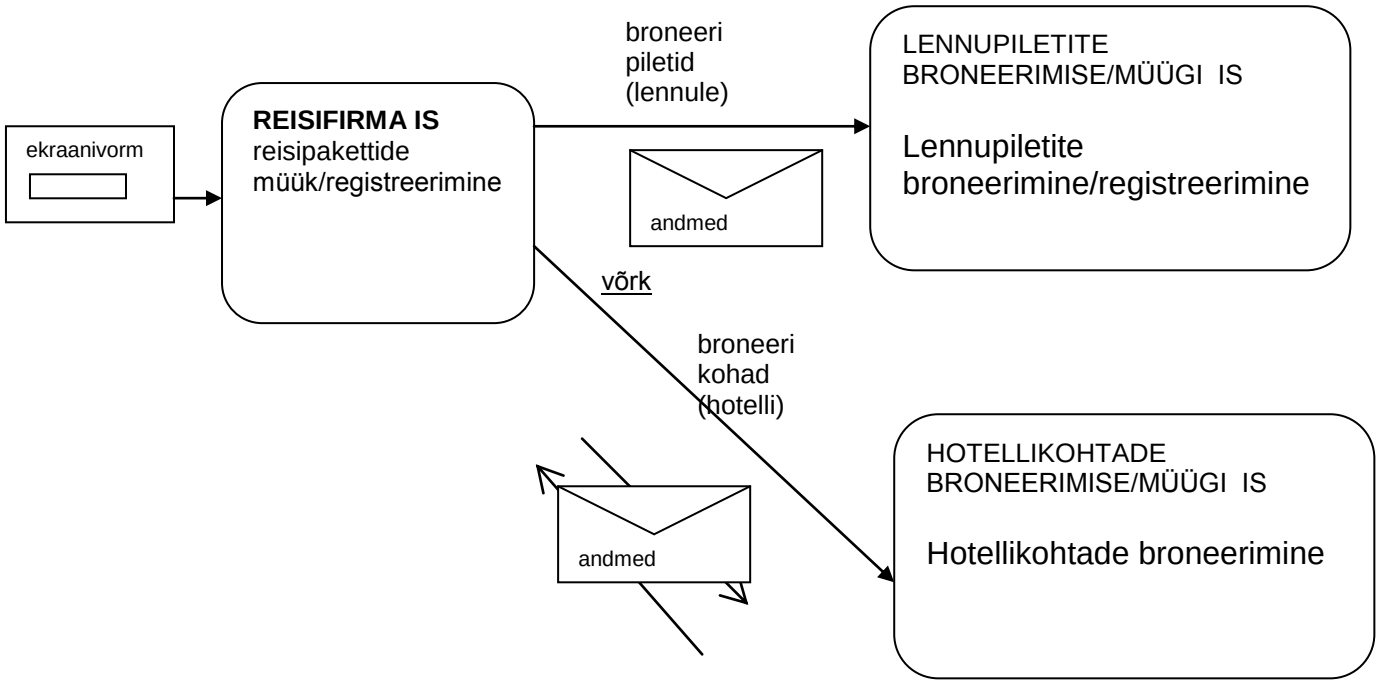
Tänapäeval ei toimi ettevõtte infosüsteem enamasti üksi, vaid kasutab oma ülesannete täitmiseks teiste infosüsteemide teenuseid.

See tähendab, et osa mingi infosüsteemi/rakenduse eesmärgi täitmiseks tehtavaid töid tehakse väljaspool selle infosüsteemi piire – teises infosüsteemis.

Antud kontekstis eesmärk = hulk eesmärgi täitmiseks tehtavaid tegevusi ehk täidetavaid ülesandeid.

Eesmärgi täitmiseks tehtav omavahel seotud tegevuste/tööde/ülesannete hulk on äriprotsess.

Reisipaketi müügi registreerimine on äriprotsess (võib omakorda olla mõne suurema äriprotsessi osa).



Selleks et reisifirma saaks kliendile reisi müüa peab tema infosüsteem pöörduma lennupiletite broneerimise infosüsteemi poole ülesandega/palvega/väljakutsega broneerida piletid mingile reisile.

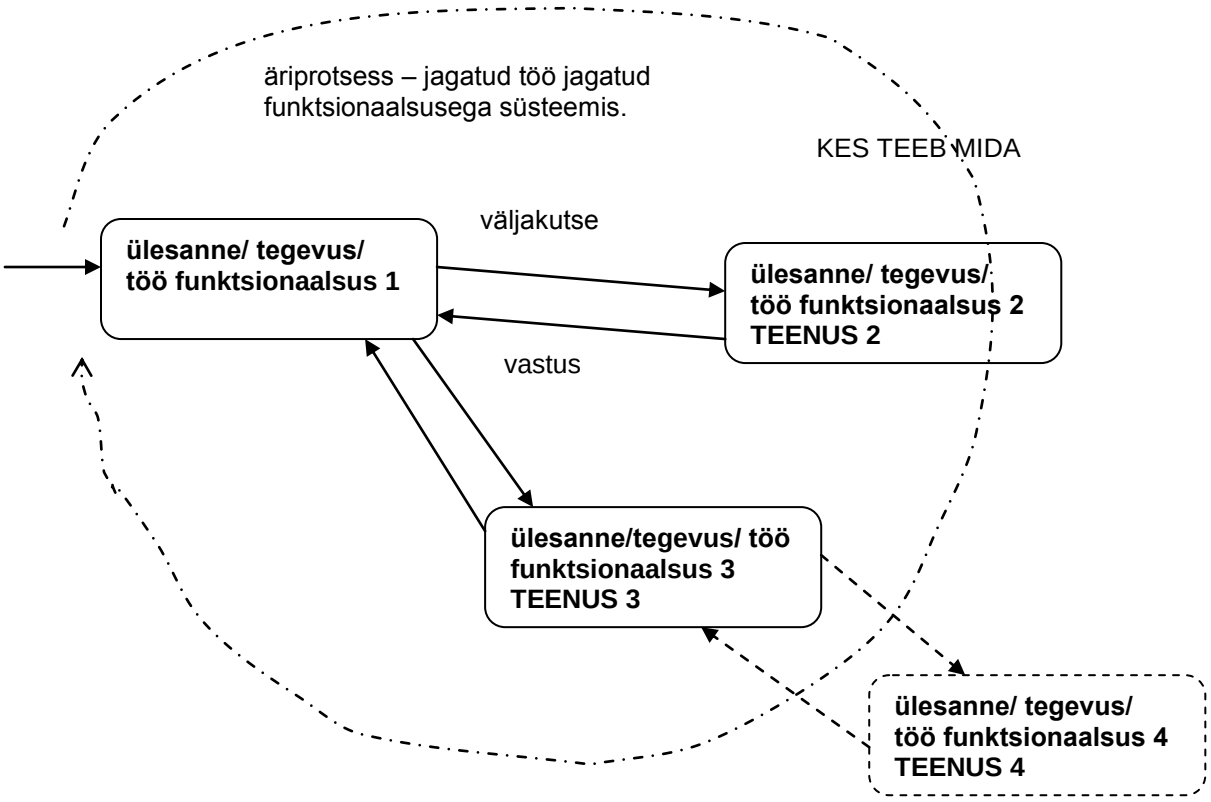
Selleks et reisifirma saaks kliendile reisi müüa peab tema infosüsteem pöörduma hotellikohtade broneerimise süsteemi poole ülesandega broneerida hotellikohad.

Reisipaketi müümisel/registreerimisel tuleb teha tööd kolmes infosüsteemis.

Hajussüsteem on mitmest infosüsteemist koosnev süsteem mille osad suhtlevad väljakutsete abil – üks süsteem kutsub teist välja ja annab **täitmiseks ülesande** - ehk **kutsub välja teenust.**

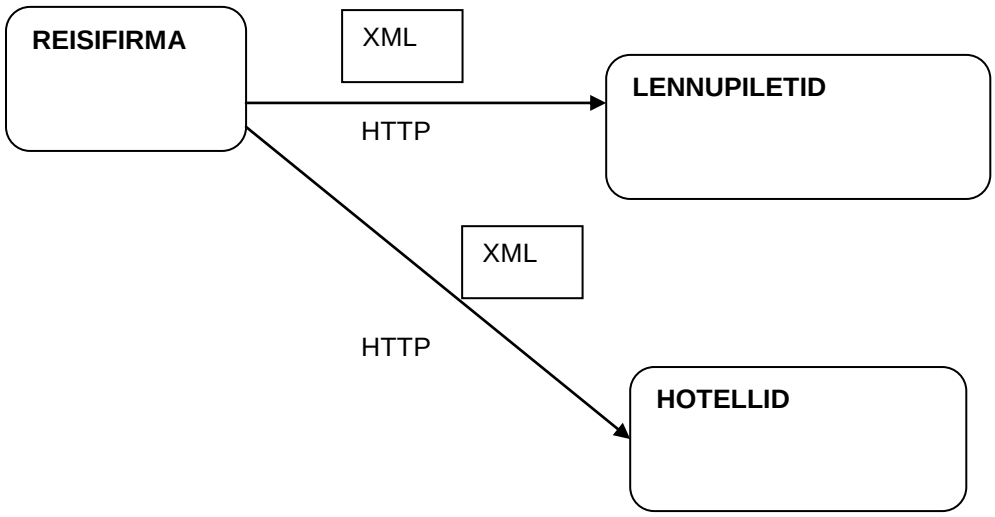
Teenuse väljakutsega kaasneb **andmevahetus** – väljakutses sisalduvad sisendandmed (millisele lennureisile kohad broneerida) ja väljakutsutav infosüsteem tagastab väljundandmed (broneeringu numbri, info toimingu õnnestumise kohta jms.).

Seotud infosüsteemides on mingi eesmärgi täitmiseks (näiteks reisipaketi müümiseks) tehtav töö jagatud infosüsteemide vahel, need infosüsteemid suhtlevad üle võrgu omavahel, kutsuvad teineteist välja, väljakutsetes sisalduvad sisendandmed ülesannete täitmiseks ja vastused sisaldavad ülesannete täitmisel loodud andmeid (tellimuse numbrid, broneeritud hotellikohtade numbrid, lepingu maksegraafikud, lepingu intressid jne.)



Selliseid teiste infosüsteemide poolt üle võrgu välja kutsutavaid tegevusi (õigemini nende tegevuste realisatsioone serveril) nimetatakse **teenusteks.**

2. veebteenused – tehnoloogia süsteemide suhtlemiseks üle võrgu. XML üle HTTP.



Infosüsteemid kasutavad teiste süsteemid väljakutsumiseks mingit **protokolli** – põhimõtteliselt **kokkulepitud reegleid** selle kohta kuidas andmeid vahetada.

Infosüsteemid kasutavad andmevahetuseks (üle protokolli) mingit **andmeformaati** – ehk siis mingit kokkulepitud **kuju andmete esitamiseks** millest mõlemad pooled saavad aru.

Tänapäeval kasutatakse infosüsteemide vaheliseks andmevahetuseks enamasti HTTP protokolli ja andmeformaadiks XML-i – seega on infosüsteemide vaheline infovahetus tehnoloogiliselt väga sarnane sellele kuidas brauser vahetab infot veebserveriga – mõlemad saadavad üle HTTP teksti. (XML ja HTML on tektsiformaadi erijuhud).

Miks **HTTP** ? Läheb tulemüüridest läbi.

Miks **XML** ? XML-i suudavad lugeda ja moodustada suvalistel platvormidel realiseeritud infosüsteemid (rakendused).

3. SOAP – veebteenuste kõige levinum erijuht.

Kõige levinum XML/HTTP-d kasutav veebteenuste tehnoloogia on SOAP-tehnoloogia.

Mis on SOAP.

Simple **O**bject **A**ccess **P**rotocol

* **SOAP on andmeformaat**, täpsemalt SOAP-standardi reeglitele vastavalt vormindatud XML

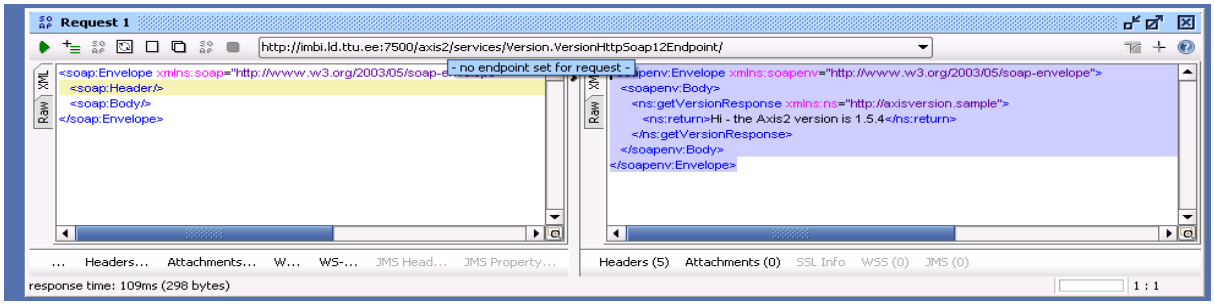
Kui süsteemid kasutavad andmevahetuseks SOAP-i siis on nii pöördumine kui pöördumise vastus selles kokkulepitud XML-formaadis.

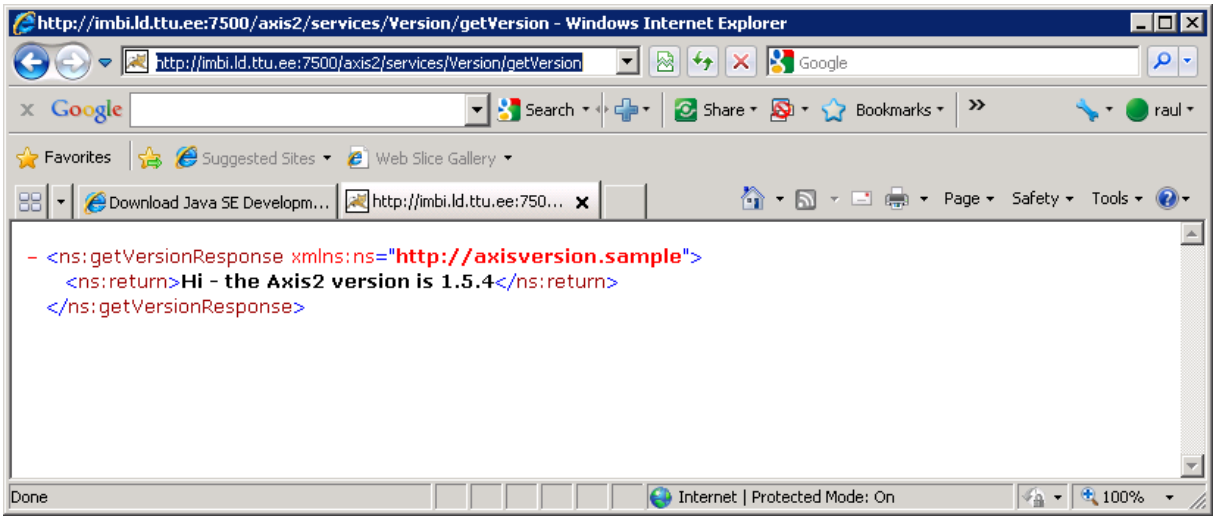
SOAP-vormingus pöördumise ja vastuse andmeid nimetatakse SOAP-sõnumiks (**SOAP-message**).

Aadressil http://imbi.ld.ttu.ee:7500/axis2/services/Version/getVersion

saadetud SOAP pöördumine ja selle vastus:

(teenus on vähemalt loengu päeval ja ilmselt hiljemgi üleval. Teenus oskab vastata ka brauserist tehtud pöördumisele.)





Teenuse aadressiks

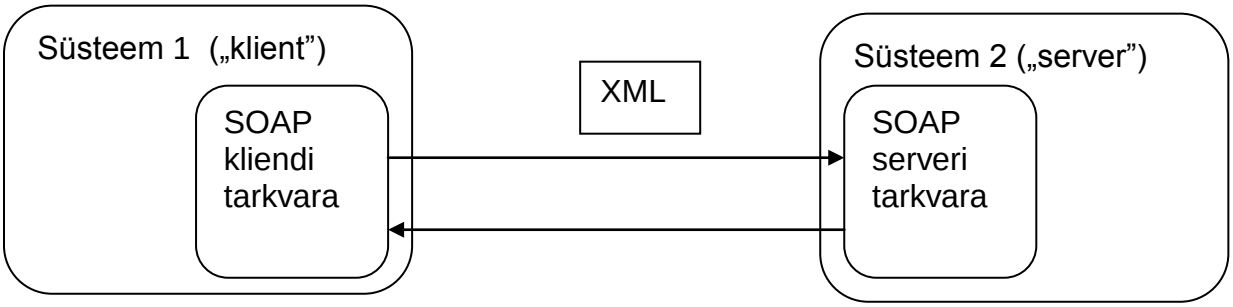
http://imbi.ld.ttu.ee:7500/axis2/services/Version

Sellises XML-formaadis SOAP-sõnumit nimetatakse ka SOAP-ümbrikuks (SOAP-envelope).

* **SOAP on standard** – kokkulepitud spetsifikatsioon selle kohta kuidas SOAP teenuse väljakutsuja ja SOAP-teenuse pakkuja peavad omavahel suhtlema, millised on sellise suhtlemise reeglid . SOAP-sõnumi formaat on standardi üks osa.

* **SOAP-protokolli kasutamiseks on vaja tarkvara** – sellist tarkvara mis oskab SOAP sõnumeid formaatida, saata, vastu võtta, sealt andmeid välja lugeda. Seega on tegelikult vaja kaks tarkvara-„tükki" – teenuse välja kutsuja poolel üks „tükk" ja teenuse osutaja (serveerija) poolel teine „tükk".

Väljakutsujat nimetame edaspidi „**kliendiks**", süsteemi mida välja kutsutakse aga „**serveriks**".



SOAP kliendi ja SOAP serveri tarkvara peavad järgima SOAP-standardit (standard – kokkulepe, **leping**). Kui nad seda järgivad siis on võimalik ühendada kokku SOAP –protokolli „rääkivaid" infosüsteeme sõltumata sellest millises tarkvarakeskkonnas need erinevad infosüsteemid töötavad.

4. SOAP-server kui veebteenuste server. Rakendusserver.Serveri teenuse kirjutamine on lihtne.

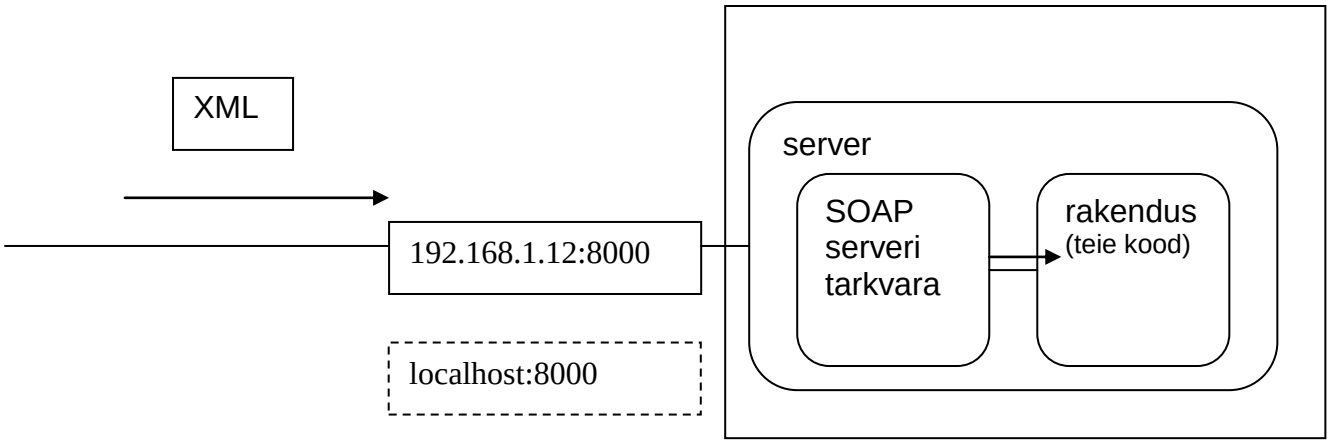
Selleks et väljakutsutav infosüsteem meil igal hetkel vastaks kui meil teda (tema teenust) vaja on peab väljakutsutavas infosüsteemis töötama tarkvara mida nimetatakse serveriks – konkreetsemalt SOAP-serveriks.

See on rakendus mis on väljakutsutavas süsteemis 24/7 üleval ja ootab pöördumisi.

Põhimõtteliselt on tegemist nagu avatud MS Wordi rakendusega (näiteks, või siis avatud OpenOffice'i või avatud Notepad-iga – kellele mis rohkem meeldib) mis on arvutis käivitatud ja ootab klaviatuurivajutusi. Ainult et serveri puhul tulevad need „klaviatuurivajutused" ehk pöördumised üle võrgu ja sisaldavad, nagu juba teame – „SOAP-ümbrikke".

Sellisel serveril on 2 olulist omadust:

- ta on kogu aeg arvutisüsteemis töötavas olekus (laetud, käivitatud). Selle kohta öeldakse – „**on arvutis protsessina üleval**".
- ta **kuulab** mingi serveri **võrguaadressi** mingit **porti** et võtta vastu üle võrgu saabuvaid sõnumeid.

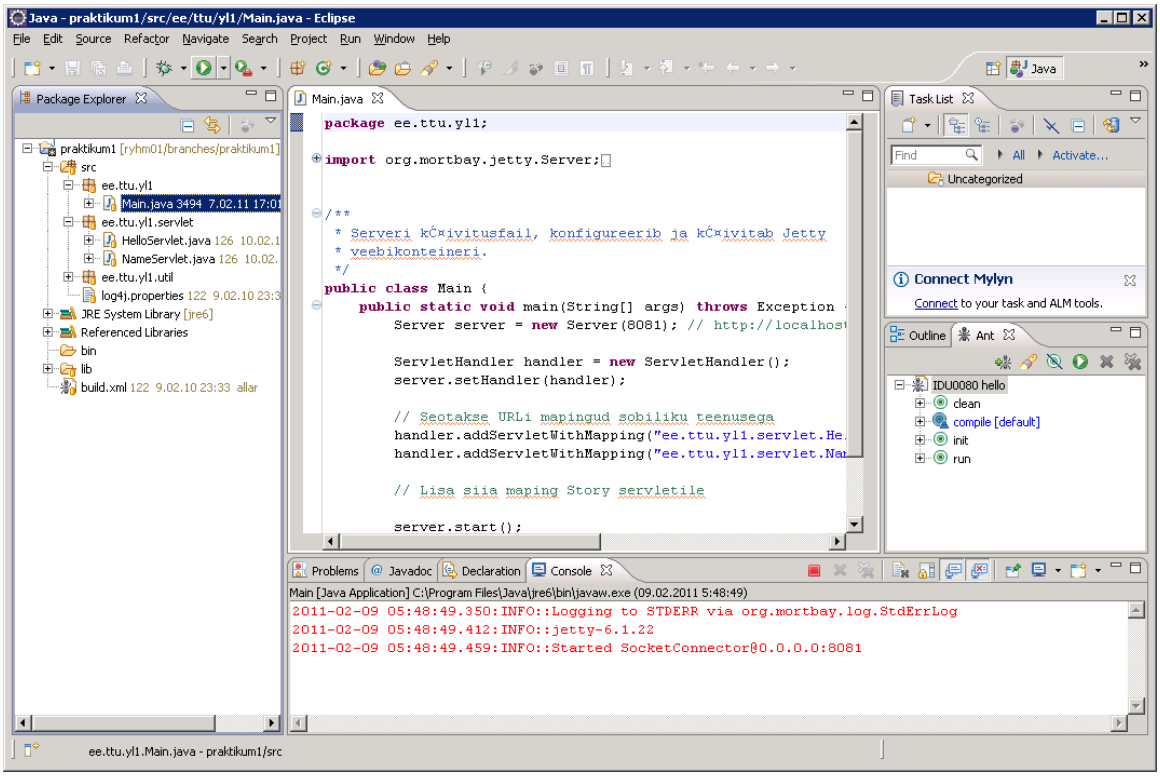


Koduarvutis käivitatud server on enamasti kättesaadav aadressilt **localhost:<mingi port>**

Kui pöördumine tuleb siis võtab serveri tarkvara selle vastu, serveri tarkvara koosseisus on ka SOAP serveri tarkvara („SOAP sofware stack") mis oskab laekunud XML sõnumist andmed serveri mällu lugeda ja annab siis tööjärje koos mällu laetud andmetega üle programmeerija kirjutatud SOAP teenusele. (rakendusele mis on SOAP teenuse „taga", mis realiseerib SOAP teenust).

Üheks SOAP serveri tarkvara näiteks on Apache CXF mida kasutate ka 2013 aasta kevadsemestril harjutusülesannetes.

Harjutustes te käivitate serverit küll Eclipse'st kuid tegelikult on tegemist eraldi serveritarkvaraga.



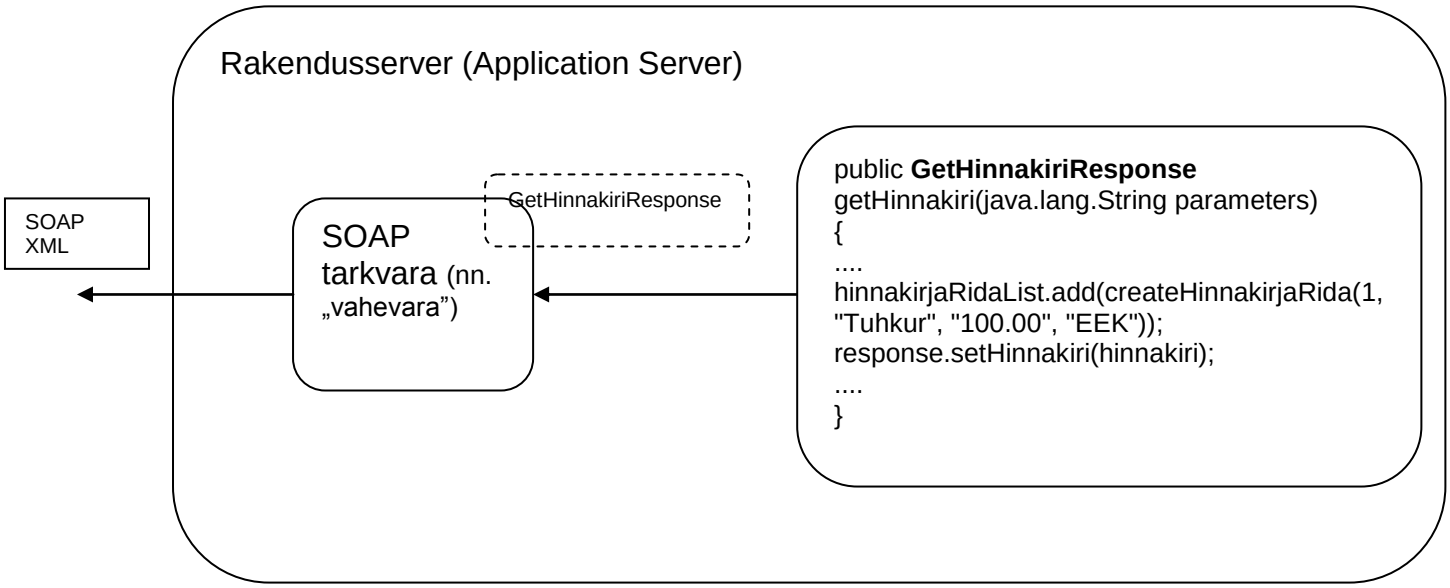
Ülemisel ekraanivormil on server pordil :8081 käivitatud.

Harjutusülesandes 1 käivitatakse ka server. See server ei ole küll SOAP server vaid server mis kõlbab veebirakenduste jooksutamiseks, esimeses ülesandes käivitatav kood ongi veebirakendus. Kuid nagu juba eespool öeldud ei erine veebirakenduse server tehnoloogiliselt oluliselt SOAP-teenuse serverist, põhimõte on („kliendi" poolt vaadates) üsna sama – tekst üle HTTP. (Kuigi üle HTTP saab SOAP-protokolli abil saata ka binaarkujul andmeid, faile, siis seda me selles aines ei vaata).

Selliseid servereid mis teenindavad veebirakendusi ja/või teisi infosüsteeme nimetatakse rakendusserveriteks (***application server***).

Korraga sama võrguaadressi samal pordil mitut serverit käivitada ei saa.

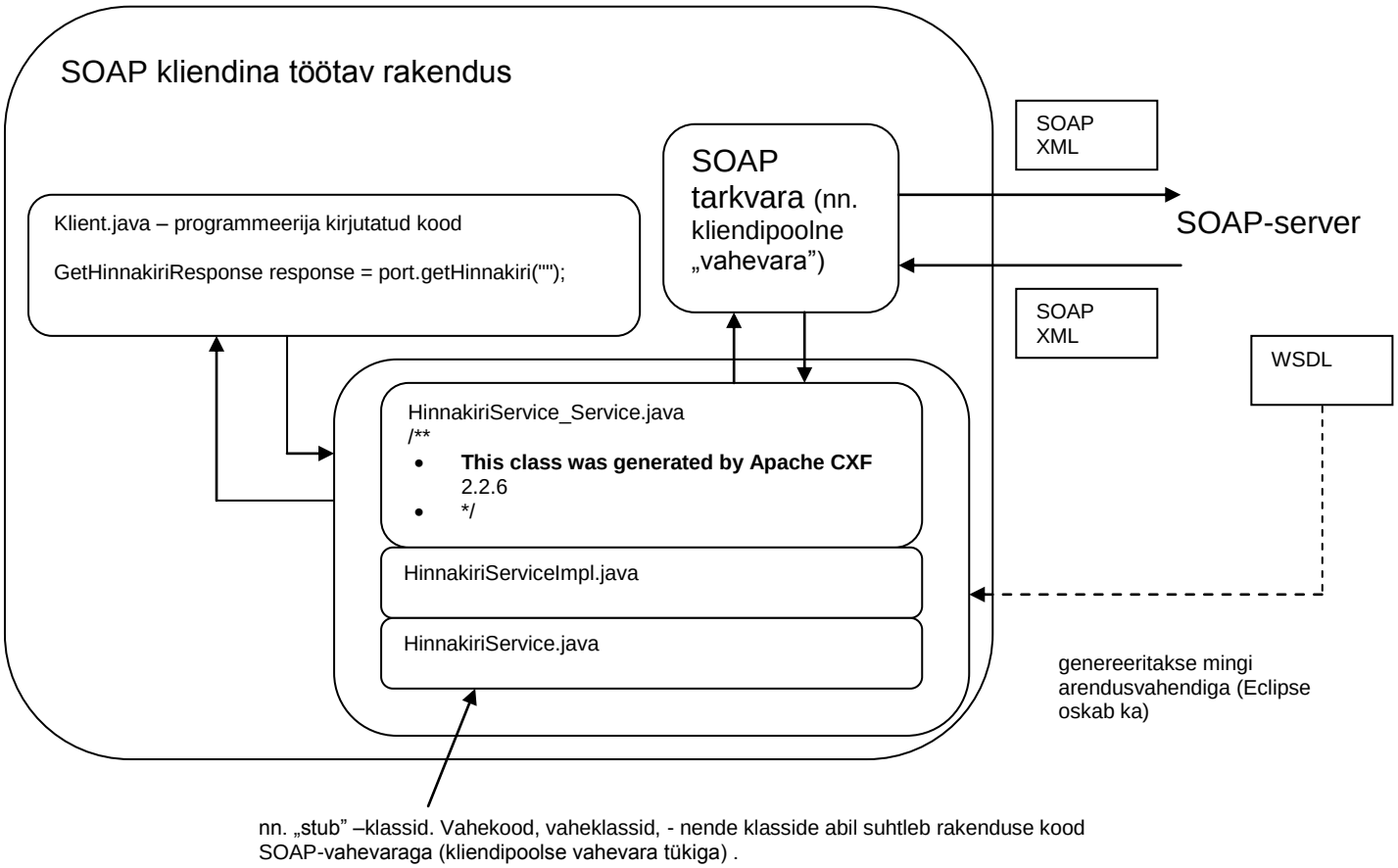
Vastuse mida saadab programmeerija kirjutud veebteenus formaaditakse serveri SOAP-tarkvara poolt SOAP-teenusele ja saadetakse SOAP-protokolli reegleid järgides kliendile (väljakutusjale).



Pöörame tähelepanu sellele et programmeerija tagastab veebteenuse kliendile java objekti „GetHinnakiriResponse" – selle Java objekti andmete XML-kujule teisendamisega tegeleb juba SOAP serveri tarkvara (meie puhul enamasti Apache CXF). Programmeerija serveri poolel koodi kirjutades SOAP-protokolli spetsiifikaga praktiliselt kokku puutuma ei pea – selle töö teeb serveri tarkvara tema eest ära.

5. SOAP-klient. WSDL

Infosüsteemis (õigem oleks nüüd rääkida juba konkreetsemalt rakendusest) mis peab pöörduma SOAP-teenuse poole on samuti vaja kasutada SOAP-teenuse tarkvara – selleks et kliendi-rakenduse väljakutsed teisendada SOAP-sõnumiteks ja veebteenuselt laekunud SOAP-sõnumid **teisendada** mingiteks selles programmeerimiskeeles kasutatavateks **objektideks kliendipoolse rakenduse mällu**.



SOAP-kliendi tarkvaraks sobib ka kasutada Apache CXF-i (tavaliselt nii ongi – SOAP-teenuse tarkvara tootjalt on saada nii SOAP-serveri kui SOAP-kliendi jaoks tarkvara.).

Kui seadistate Eclipse nii nagu on juhendatud **harjutustunni 2** juhendis siis sellega installeerite endale Apache CXF-i kliendipoolse SOAP-vahevara ka.

Kliendiga on asi siiski mõnevõrra keerulisem kui serveriga.

Kuidas klient teab kuidas serveri poole pöörduda? Milliseid andmeid saata? Milliseid andmeid on vastu oodata? Selleks on WSDL – Web Service Definition Languge spetsifikatsioonile vastav XML andmestruktuur. WSDL defineerib teenuse (liidese) ehk annab kliendile teada kuidas tuleb teenusega suhelda – millised on meetodid, argumendid ja andmestruktuurid selles.

WSDL võib olla failina salvestatud kliendi süsteemi (ta peab kliendirakendusele kättesaadav olema) kuid võib olla kättesaadav kliendile ka üle võrgu – samast kohast kus veebteenus ise paikneb. Näiteks:

http://imbi.ld.ttu.ee:7500/axis2/services/Version?wsdl

```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:ns1="http://org.apache.axis2/xsd" xmlns:ns="http://axisversion.sample"
    xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    targetNamespace="http://axisversion.sample">
  <wsdl:documentation>Version</wsdl:documentation>
- <wsdl:types>
- <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
    targetNamespace="http://axisversion.sample">
- <xs:complexType name="Exception">
- <xs:sequence>
  <xs:element minOccurs="0" name="Exception" nillable="true" type="xs:anyType" />
    </xs:sequence>
    </xs:complexType>
- <xs:element name="Exception">
- <xs:complexType>
- <xs:sequence>
  <xs:element minOccurs="0" name="Exception" nillable="true" type="ns:Exception" />
    </xs:sequence>
    </xs:complexType>
    </xs:element>
- <xs:element name="getVersionResponse">
- <xs:complexType>
- <xs:sequence>
  <xs:element minOccurs="0" name="return" nillable="true" type="xs:string" />
    </xs:sequence>
    </xs:complexType>
    </xs:element>
    </xs:schema>
    </wsdl:types>
  <wsdl:message name="getVersionRequest" />
- <wsdl:message name="getVersionResponse">
  <wsdl:part name="parameters" element="ns:getVersionResponse" />
    </wsdl:message>
- <wsdl:message name="Exception">
  <wsdl:part name="parameters" element="ns:Exception" />
    </wsdl:message>
- <wsdl:portType name="VersionPortType">
```

```
- <wsdl:operation name="getVersion">
  <wsdl:input message="ns:getVersionRequest" wsaw:Action="urn:getVersion" />
  <wsdl:output message="ns:getVersionResponse"
     wsaw:Action="urn:getVersionResponse" />
  <wsdl:fault message="ns:Exception" name="Exception"
     wsaw:Action="urn:getVersionException" />
    </wsdl:operation>
    </wsdl:portType>
- <wsdl:binding name="VersionSoap11Binding" type="ns:VersionPortType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"
     />
- <wsdl:operation name="getVersion">
  <soap:operation soapAction="urn:getVersion" style="document" />
- <wsdl:input>
  <soap:body use="literal" />
    </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
    </wsdl:output>
- <wsdl:fault name="Exception">
  <soap:fault use="literal" name="Exception" />
    </wsdl:fault>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:binding name="VersionSoap12Binding" type="ns:VersionPortType">
  <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
     style="document" />
- <wsdl:operation name="getVersion">
  <soap12:operation soapAction="urn:getVersion" style="document" />
- <wsdl:input>
  <soap12:body use="literal" />
    </wsdl:input>
- <wsdl:output>
  <soap12:body use="literal" />
    </wsdl:output>
- <wsdl:fault name="Exception">
  <soap12:fault use="literal" name="Exception" />
    </wsdl:fault>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:binding name="VersionHttpBinding" type="ns:VersionPortType">
  <http:binding verb="POST" />
- <wsdl:operation name="getVersion">
  <http:operation location="Version/getVersion" />
- <wsdl:input>
  <mime:content type="text/xml" part="getVersion" />
    </wsdl:input>
- <wsdl:output>
  <mime:content type="text/xml" part="getVersion" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:service name="Version">
```

```
- <wsdl:port name="VersionHttpSoap11Endpoint"
    binding="ns:VersionSoap11Binding">
  <soap:address
    location="http://imbi.ld.ttu.ee:7500/axis2/services/Version.VersionHttpSoap11
    Endpoint/" />
    </wsdl:port>
- <wsdl:port name="VersionHttpSoap12Endpoint"
    binding="ns:VersionSoap12Binding">
  <soap12:address
    location="http://imbi.ld.ttu.ee:7500/axis2/services/Version.VersionHttpSoap12
    Endpoint/" />
    </wsdl:port>
- <wsdl:port name="VersionHttpEndpoint" binding="ns:VersionHttpBinding">
  <http:address
    location="http://imbi.ld.ttu.ee:7500/axis2/services/Version.VersionHttpEndpoi
    nt/" />
    </wsdl:port>
    </wsdl:service>
    </wsdl:definitions>
```

Kas seda wsdl-faili peab nüüd programmeerija uurima selleks et veebteenusega ühendust võtta (oma koodis)?

Üldiselt mitte väga, kuigi mõnikord on vaja sinna wsdl-i sisse vaadata ka.

Staatiliste keelte puhul (Java on staatiline – andmetüübid on kompileerimise ajal teada ja paigas) tehakse nii:

1. kliendirakenduse kirjutamisel **genereeritakse** spetsiaalse töövahendiga (Java puhul on selle nimeks tihti WSDL2Java) niinimetatud vaheklassid. Need vaheklassid sisaldavad meetodeid veebteenuse (kliendipoolse) vahevaraga suhtlemiseks.

2. veebteenuse kliendipoolne vahevara omakorda suhtleb veebteenusega.

3. Kui programmeerija tahab veebteenust välja kutsuda siis ta kirjutab koodi mis kutsub välja nende genereeritud vaheklasside teatud meetodeid (vt. eelnevat joonist).

Selliseid genereeritud vahe-klasse nimetatakse ka „stub”-klassideks.

Samasugune loogika kehtib ka teiste staatiliste keelte puhul – näiteks C#. Sellisel juhul on genereeritavad klassid muidugi C#-i klassid.

Kaasaegset IDE-t kasutades toimub genereerimine peaaegu automaatselt (wsdl on vaja ette anda), küll aga peab programmeerija nendes genereeritud klassides nüüd niipalju orenteeruma et ta saaks aru milliseid meetodeid sealt välja kutsuda.

Kokkuvõtteks võib öelda – kliendipoole tegemine on mõnevõrra keerukam staatiliste keelte puhul ja sisaldab juba valmis vahevara ja kliendi tegemisel wsdl-i alusel genereeritud vahevara.

Dünaamilistes keeltes (PHP, Python ,..) ehk niinimetatud skriptikeeltes „stub”-ide genereerimisega tegeleda ei ole vaja ja veebteenuse klienti kirjutada on lihtsam.

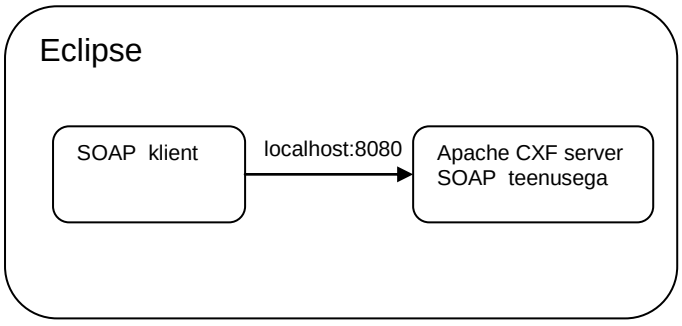
6. Veebteenuse klient ja server ühes masinas, ühe Eclipse „sees”.

Mõnevõrra segadust võib tekitada see et nii server kui klient käivitatakse ühes arvutis ja ühest arenduskeskkonnast. Tegelikult vastuolu pole – käivitakse tegelikult ikkagi 2 rakendust:

- esimesena käivitatkse SOAP server (mis jääb siis mingit porti „kuulama”)
- siis käivitataks SOAP klient mis samas masinas juba käivitatud SOAP-serveriga ühendust võtab (küll samas masinas aga siiski üle võrgu).

Tegelikult võivad harjutuses 2 tehtud server ja klient asuda vabalt arvutivõrgus erinevates masinates.

Tavaolukorras SOAP-teenuse serverit ei käivitada muidugi Eclipse abil vaid eraldi antud serveri tarkvara sisalduvate startup-skriptidega.



7. Teistest harjutusülesannetest.

* Asünkroonne päring.Sõnumiteenus.

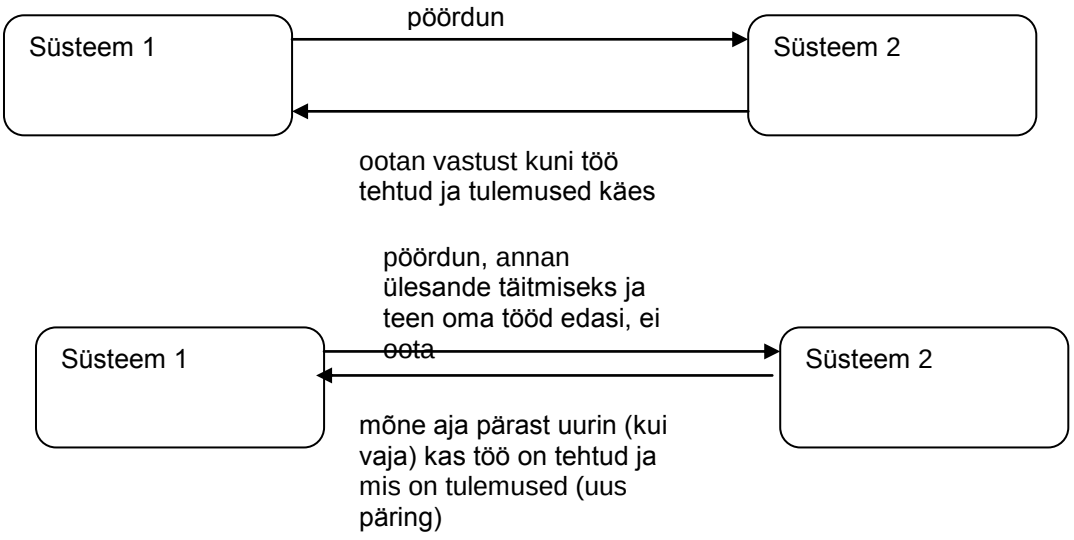
Üldjuhul käib tänapäeva integratsioon tihti asünkroonselt, selle asünkroonsuse saavutamiseks saab kasutada sõnumiteenuse serverit (Java platvomil – JMS server)

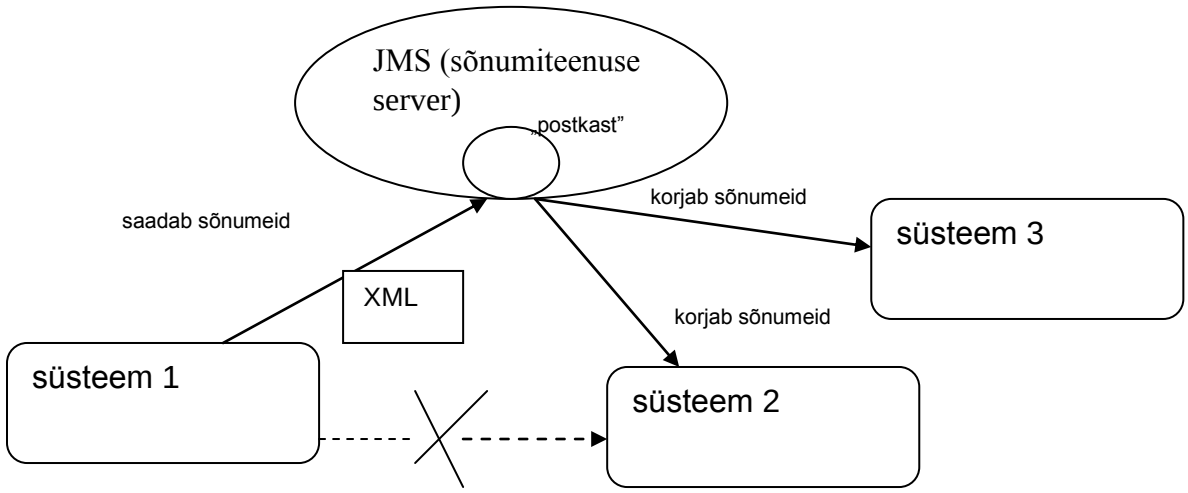
Põhimõtteid 2:

- saadame ülesande/väljakutse – aga vastust ei oota. Vastuse järele läheme serverile uue päringuga. Mingi aja pärast.

- Meie (st. kliendi) poolt saadetud väljakutse saadetakse sõnumiserverisse konkreetsesse sõnumite järjekorda ja meie (st. „klient") ei tea kes või mis (süsteem, rakendus) selle sõnumi sealt ära korjab ja töötleb. Maksimaalne lahtisidestus kahe süsteemi vahel. Sõnumite saatja ei tea kes korjab, sõnumite korjaja (ja töötleja) ei tea kes saatis. Asünkroonsuse moment siin – sõnumeid võib üles korjata siis kui tahad , siis kui jõuad. Üleskorjamata sõnumid säilivad sõnumite serveris järjekorras.

Sünkroonse seose puhul on süsteemid nii aeglased kui on kõige aeglasem protsessis osalev osapool. Kui sünkroonne sõnumivahetus jookseb kinni, siis jäävad kõik ootama. Põhjusi palju – jõudlus, vajalik inimese sekkumine protsessi,võrgu probleemid jne. Sünkroonse seose korral kahe süsteemi vahel – ootab kohe vastust, nii kaua ootab (on põhimõtteliselt ajutiselt „kinni jooksnud" , ei reageeri) kuni teine osapool (server) saab pöördumise töödeldud ja andmed tagasi saadetud.





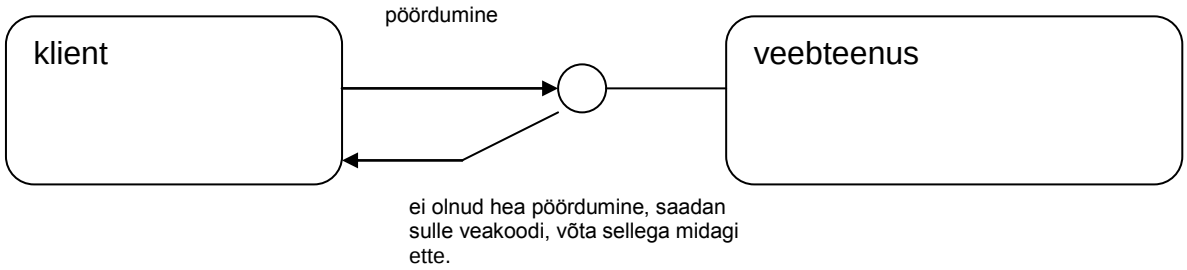
otseside, otsepöördused süsteemide vahel puuduvad

Sõnumiteenust kasutavad süsteemid „ei tea” teineteisest, ei suhtle omavahel.

Sõnumiteenuse server (broker) kindlustab et ka vahepeal mitte-ühenduses-olnud süsteemid saavad sõnumid postkastist kätte (need sõnumid mis on saadetud ajal kui nad ei olnud ühenduses sõnumiteenuse serveriga).

* Vigade haldus. (ülesanne „Hajuspäringu veahaldus”)

Veebteenusele ei pruugi „meeldida” kliendilt saadud sisendandmed (äriloogilistel või formaadi põhjustel). Tuleb kliendile teada anda et tema väljakutset ei saanud teenindada. Las klient siis võtab selle infoga midagi ette.



Vigade töötlemine (eriti äriloogiliste, sisendandmete kontroll) on äriprotsessi realisatsiooni üks osa, kuulub protsessi juurde. Ei saa loota et ainult positiivsed stsenaariumid reaalses elus toimuvad.

* Ülesanne „Transaktsioonid , kompenseerivad tegevused.” (kui see 2013.a. tuleb, siis kompleksülesande 6 koosseisus)

Üle võrgu töötav hajussüsteem on oma loomult ebatöökindel. Sellest üle ei saa. Peab sellega arvestama. Mida teha kui poole pealt jääb katki – lend on broneeritud aga hotellikohtade broneerimise süsteem annab vea või on üldse kättesaamatu.

Peame kuidagi elegantselt välja tulema situatsioonist. Tuleks taastada katkenud äriprotsessi käivitamisele eelenud seis (rollback) – lennu broneerimine tuleks ka tühistada.