## 

**J-road teegi kasutusjuhend**

AS Nortal

**Muudatuste haldus**

|  |  |
| --- | --- |
| **Kuupäev/autor** | **Muudatus** |
| Dmitri Danilkin | Algne versioon |
| 28.04.2016 / Lauri Lättemäe | Juhendi ajakohastamine vastavalt X-tee versioonile 6 |

**Sisukord**

Sissejuhatus 4

1. Teegi struktuur ja moodulite kirjeldus 5

1.1. Teegi ehitamine lähtekoodist 5

2. Teenuste tarbimine 6

2.1. Java klasside genereerimine 7

2.1.1. Eelistatud meetod 7

2.1.2. Käsitsi 7

2.2. Teenuste implementeerimine 8

2.3. Konfiguratsioon 9

2.3.1. Manuste kasutamine 11

2.3.2. MIME manused 11

2.3.3. swaRef manused 11

3. Teenuste pakkumine 13

3.1. Teenuse XSD kirjeldamine 14

3.2. Java klasside genereerimine 14

3.3. Teenuste Endpoint-de kirjutamine 16

3.4. Konfiguratsioon 17

3.5. Manuste kasutamine 19

# Sissejuhatus

**J-road teek on *Java 1.6+,* *Spring*'i, X-tee 6+veebiteenustega ühilduv abivahend**, mis võimaldab tarbida ja pakkuda X-tee teenuseid.

Antud teek on mõeldud nii olemasolevast andmekogust teenuste kasutamiseks, kui ka uute teenuste loomiseks. Kuna need kasutuslood on suhteliselt erinevad, jaguneb käesolev juhend kaheks.

Teek vastab võimalikult täpselt standardile, mille sätestab X-tee spetsifikatsioon.

Teek on üles ehitatud kasutades *XMLBeans*, *Spring-WS* ja *JAXB* tehnoloogiaid.

Seoses X-tee 6-ga kaob tugi tagasiulatuvalt eelnevatele X-tee sõnumiprotokollidele.

Seni oli teenuste pakkuja osa implementeeritud X-tee protokolli 2.0 jaoks. Seega teenuste migreerimise perioodiks toetab j-road teek teenuste pakkuja osa tagasiulatuvalt seni toetatud X-tee protokolli 2.0 sõnumeid ja käitub vaikimisi nagu eeldatakse X-tee 6 uue protokoll 4.0 järgi.

Teenuste klientide osas kaob vaikimisi tugi eelnevatele protokollidele ehk teenuste kliendid käituvad X-tee protokoll 4.0 järgi.

# Teegi struktuur ja moodulite kirjeldus

J-road teek on *maven*-i projekt järgmiste moodulitega:

1. *common* – Funktsionaalsus, mida kasutatakse mitme mooduli poolt.
2. *client-transport* – Õhuke X-tee klient (teenuste tarbimiseks).
3. *client-service* – X-tee teenuste java liidesed (paks klient), mis jaguneb omakorda mooduliteks iga andmekogu jaoks.
4. *typegen* – WSDL-ist Java objektide generaator.
5. *server* – adapterserver (andmekogu/teenuste loomiseks).
6. *example* – näidisrakendus nii kliendi kui ka serveri kohta.

Märkused:

* Serveriosa vajab jooksmiseks *common* ja *server* mooduleid.
* Kliendiosa vähemalt *common* ja *client-transport* mooduleid.
* Juhul, kui j-road teeki kasutatakse paksu kliendina, siis ka *client-service* moodulit.
* Näidisrakenduse ehitamisel tehakse *war*, millest piisab rakenduse käivitamiseks.

## Teegi ehitamine lähtekoodist

Teegi ehitamiseks tuleb käivitada käsk:

|  |
| --- |
| mvn install |

Kui ehitate teeki väljaspool AS Nortal, siis suure tõenäosusega ei ole teil kõikidele j-road teegis realiseeritud teenustele ligipääsu, mis on vajalikud testide läbimiseks ning build ei õnnestu. Selle vältimiseks lülitage testid välja või uuendage testide konfiguratsioon (xroad.properties) vastavalt oma asutusele.

Testide väljalülitamiseks ehitusprotsessi käigus kasutage käsku:

|  |
| --- |
| mvn install -DskipTests=true |

# Teenuste tarbimine

Teenuste tarbimine j-road teegi vahendusel koosneb järgmistest etappidest:

1. Teenuse *WSDL*-ist *XMLBeans* artifaktide (java klasside) genereerimine.
2. Teenusele Java liidese implementeerimine.
3. Andmete edastamine rakenduses kasutades liidest.

Kõik seni tehtud teenuste implementatsioonid on olemas X-tee teegis.

Kui te implementeerite arenduse käigus mõne uue teenuse, siis on väga teretulnud selle teenuse implementatsiooni panek j-road teeki.

Enne implementeerimise alustamist vaadake üle teegis juba olemas olevate teenuste implementatsioonid veendumaks, et teile vajalikku teenust teegis juba ei leidu. Kui soovite kasutada teegis juba realiseeritud teenust, siis on projektis vaja teha vaid *Spring* konfiguratsioon ning asetada *classpath-*i olemasolev teek.

## Java klasside genereerimine

### Eelistatud meetod

Reeglina tuleb teostada Java tüüpide genereerimiseks järgnevad sammud:

1. Luua uus moodul *client-service* mooduli alla mõne olemasoleva mooduli näitel. Mooduli nimi peaks võrduma andmekogu nimega.
2. Paigaldada loodud moodulisse andmekogu WSDL (kataloogi *src/main/resources*).
3. Käivitada *maven*i ehitusprotsess.
4. Edasi genereeritakse ja kompileeritakse vastava mooduli alla kõik vajalikud tüübid, mis WSDL-s sisalduvad.

### Käsitsi

Tüübid võib genereerida ka käsitsi kasutades teegis sisalduvat *TypeGen* programmi. *TypeGen* programm vajab järgmiseid parameetreid:

1. *wsdldir* – kaust, kust loetakse WSDL failid
2. *sourcedir* – kaust, kuhu genereeritakse Java lähtekood
3. *xsbdir* – kaust, kuhu genereeritakse XMLBeans-i ja X-tee metainfo. Peab olema classpathis.
4. *basepackage* – pakett, milles asuvad kõik genereeritud tüübid.
5. dbclassespackage – pakett, kuhu genereeritakse võimalike teenuste minimaalne implementatsioon

Näide programmi korrektsest väljakutsest (eeldatakse, et classpath on eelnevalt korrektselt seadistatud):

|  |
| --- |
| java -jar xtee-typegen-xxx.jar wsdldir=x:\wsdls sourcedir=x:\sources xsbdir=x:\classes basepackage= com.nortal.jroad.example.client.types dbclassespackage=com.nortal.jroad.example.client.database |

Nüüd on olemas lähtekood, mis on vaja kompileerida ning pakendada koos xsb failidega.

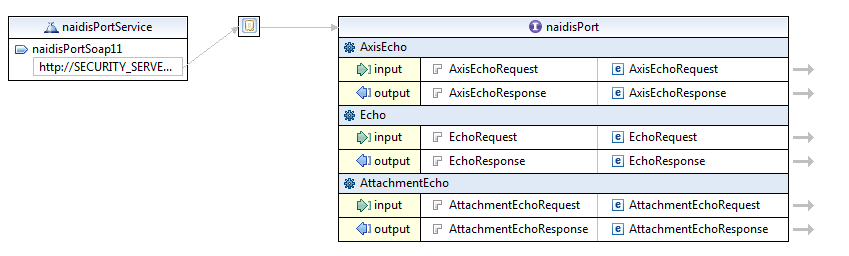
## Teenuste implementeerimine

Teenuste implementeerimise protsess reeglina välja järgmisena:

1. Luuakse X-tee päringuobjekt
2. Päringuobjekt täidetakse parameetritega
3. Teostatakse päring
4. Tagastatakse päringust saadud vastuseobjekt (ja/või manused).

Kindlasti tuleks vältida rakendusespetsiifilise äriloogika implementeerimist, j-road liidese implementatsioon on vaid X-teega suhtlemiseks.

Selleks, et teada saada, milline objekt tuleb saata ja vastu võtta, peab vaatama *WSDL*-i.

Vaatame näitena näidisandmekogu *näidis.Echo.v1* meetodit. Järgneval pildil on osa *naidis* andmekogu *WSDL*-i graafilisest vaatest *Eclipse*-s.

Kuna operatsiooni nimi on alati sama, mis meetodi nimi, siis näeme *WSDL*-st, et sisendtüüp meid huvitavale meetodile on *EchoRequest* ning väljundiks on *EchoResponse*.

Allpool on toodud antud meetodi genereeritud implementatsioon.

|  |
| --- |
| **public** com.nortal.jroad.example.client.types.ee.riik.xtee.naidis.producers.producer.naidis.EchoResponse echoV1(com.nortal.jroad.example.client.types.ee.riik.xtee.naidis.producers.producer.naidis.EchoRequest input) **throws** XRoadServiceConsumptionException {  XRoadMessage<com.nortal.jroad.example.client.types.ee.riik.xtee.naidis.producers.producer.naidis.EchoResponse> response = send(**new** XmlBeansXRoadMessage<com.nortal.jroad.example.client.types.ee.riik.xtee.naidis.producers.producer.naidis.EchoRequest>(input), "Echo", "v1");  **return** response.getContent();  } |

Teenuse mugavamaks kasutamiseks oleks mõistlik genereeritud liidesele nö. *wrapper* liides tekitada, milles koostatakse genereeritud liidese sisendobjektid ja väljundist tagastatakse vajalik osa.

Allpool on toodud antud meetodi *wrapper* liidese implementatsioon.

|  |
| --- |
| @Override  **public** String sendEcho(String text) **throws** XRoadServiceConsumptionException {  EchoRequest req = EchoRequest.Factory.*newInstance*();  req.setText(text);  **return** naidisXRoadDatabase.echoV1(req).getText();  } |

Andmekogu spetsiifiline konfiguratsioon tuleb lisada vastava mooduli *classpathi* xroad-<andmekogu>.properties manusesse ja seadistatavad parameetrid peavad olema kirjeldatud <andmekogu>-parameeter kujul. Implementeeritud andmekogu seadistus võiks vastata toodangkeskkonnale, et lõppkasutajal oleks teenuste kliente kasutada.

|  |
| --- |
| naidis-protocol-version=4.0  naidis-xroad-instance=SERVICE\_TARGET  naidis-member-class=ORG  naidis-member-code=77777777777  naidis-subsystem-code=naidis |

## Konfiguratsioon

Teenuste tarbimiseks tuleb defineerida minimaalselt:

1. Spring-i WebServiceTemplate
2. XRoadConsumer – j-road teenuste õhuke klient

Juhul, kui kasutatakse ka j-road teegi paksu klienti või selle konventsiooni järgi tehtud teenuseid, siis tuleks defineerida:

1. XRoadServiceConfigurationProvider
2. Teenused

Juhul, kui on palju teenuseid, siis on soovitatav kasutada teenuste nimetuste konvenning defineerida XRoadDatabaseService abstraktse klassina.

Näidiskonfiguratsioon:

|  |
| --- |
| <bean id=*"webServiceTemplate"* class=*"org.springframework.ws.client.core.WebServiceTemplate"* />  <bean id=*"naidisXRoadDatabase"* class=*"com.nortal.jroad.example.client.database.NaidisXRoadDatabaseImpl"* />  <bean id=*"naidisXRoadService"* class=*"com.nortal.jroad.example.client.NaidisXRoadServiceImpl"* />  <bean id=*"xRoadConsumer"* class=*"com.nortal.jroad.client.service.consumer.StandardXRoadConsumer"*>  <property name=*"webServiceTemplate"* ref=*"webServiceTemplate"* />  </bean>  <bean id=*"xRoadServiceConfigurationProvider"* class=*"com.nortal.jroad.client.service.configuration.provider.PropertiesBasedXRoadServiceConfigurationProvider"*>  <property name=*"resource"* value=*"/WEB-INF/xroad.properties"* />  </bean> |

### Manuste kasutamine

Klient toetab kahte liiki manuseid:

1. *MIME* manused (vastavalt X-tee spetsifikatsioonile).
2. *swaRef* manused (*WS-I attachments* profiil).

### MIME manused

*MIME* manuste saatmiseks piisab manuse lisamisest välja saadetavale XRoadMessage dokumendile.

Manuse abstraktsioon on XRoadAttachment, seega tuleb luua uus XRoadAttachment ning lisada see XRoadMessage-le.

Näide:

|  |
| --- |
| @Override  **public** String sendEchoMime(String text) **throws** XRoadServiceConsumptionException {  EchoRequest echoReq = EchoRequest.Factory.*newInstance*();  echoReq.setText(text);  BeanXRoadMessage<EchoRequest> req =  **new** BeanXRoadMessage<EchoRequest>(**null**,  echoReq,  Arrays.*asList*(**new** XRoadAttachment("cid", "text/plain", text.getBytes())));  XRoadMessage<EchoResponse> rsp = send(req, "Echo", "v1");  **return** rsp.getContent().getText();  } |

Vastuvõtmisel saab samamoodi XRoadMessage-st kätte kõik sissetulevad MIME manused.

NB! MIME manuseid *Base64*-ja teegi poolt ei kodeerita. Kui vastuvõttev infosüsteem töötleb manuseid *Base64*-s, siis peate kodeerimise ja/või dekodeerimise ise implementeerima.

### swaRef manused

*swaRef* manuste kasutamiseks lisatakse andmetüüpide genereerimisel vastavatele tüüpidele spetsiaalsed väljad, mis kasutavad javax.activation.DataHandler-it.

Siinkohal XRoadAttachment-i tüüpi ei kasutata, vaid manus antakse edasi otse päringuobjektile.

Näide:

|  |
| --- |
| @Override  **public** AttachmentEchoResponse sendAttachment(String contentType, **byte**[] content)  **throws** XRoadServiceConsumptionException {  DataHandler reqHandler = **new** DataHandler(**new** ByteArrayDataSource(contentType, content));  AttachmentEchoRequest request = AttachmentEchoRequest.Factory.*newInstance*();  request.addNewNest().setAttachmentHandler(reqHandler);  **return** naidisXRoadDatabase.attachmentEchoV1(request);  } |

Antud näites on manusega välja nimi „attachment“. Manuse käitlemiseks loodavad piilumeetodid on alati Handler suffiksiga.

# Teenuste pakkumine

Teenuste arendamine ja pakkumine käib „*Contract First*“ stiilis. See tähendab, et esimese sammuna tuleb kirjeldada *XML Schema (XSD)*,mida seejärel töödeldakse mingi XSD skeemi kompilaatoriga, näit. *xjc*. *XSD* skeemi võib mugavalt koostada näiteks *Eclipse*-i *WTP* graafiliste vahenditega.

**NB!** *ListMethods* ja muid X-tee spetsiifilisi metateenuseid *XSD-*s kirjeldama ei pea, need on teegi poolt juba toetatud ning on kirjeldatud X-tee nimeruumis.

X-tee spetsifikatsioonile vastav *WSDL* genereeritakse skeemis automaatselt rakenduse käivitamisel, kuid soovi korral võib selle ka ise kirjutada.

Teenuse pakkumiseks on töökäik järgmine:

1. Andmekogu loomine – dokumenteeritud turvaserveri kasutusjuhendis.
2. Päringute sisendi ja väljundi kirjeldamine *XSD* skeemi abil.
3. Java objektide genereerimine *XSD* skeemist (pole kohustuslik).
4. Äriloogika kirjeldamine.

Antud teek aitab X-tee standarditele vastava päringu vastuvõtmise, vastuse väljastamise ja *WSDL* moodustamisega. See tähendab, et arendaja poolt jääb teha vaid teenuse sisendi ja väljundi disain ning äriloogika. X-tee spetsiifilist osa *WSDL*ist ega *WSDL*i ennast muuta ei ole vajadust.

## Teenuse XSD kirjeldamine

Teenuse *XSD* skeemi kirjeldamiselt tuleb täita järgmisi nõudeid:

1. Kõik päringu juurelemendid peavad omama suffiksit „*Request“*.
2. Kõik vastuse juurelemendid peavad omama suffiksit „*Response*“.
3. Ükski element peale ülaltoodud elementide ei tohi omada suffiksit „*Request*“ ega „*Response*“. Päringute/vastuste sufiksit on võimalik ka konfigureerida.
4. „*targetNamespace*“ nimeruum võiks olla kujul: „http://<andmekogu>.x-road.eu. Teadaolevalt kaob X-tee versioon 6ga kitsendused nimeruumile ära ehk kasutada võib eelnevate protokollide nimeruume näit. juhul kui käsil on andmekogu teenuste migreerimisega uuele versioonile.
5. [*elementFormDefault* peab olema kas „*unqualified*“ või defineerimata, sest X-tee spetsifikatsiooni järgi kehas nimeruume ei tohi olla.](http://producers.andmekogu.xtee.riik.ee/producer/andmekogul)

Näidis:

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <schema xmlns=*"http://www.w3.org/2001/XMLSchema"* targetNamespace=*"http://naidis.x-road.eu"* xmlns:tns=*"http://naidis.x-road.eu"* elementFormDefault=*"unqualified"*>    <element name=*"EchoRequest"* type=*"tns:EchoRequest"*></element>  <element name=*"EchoResponse"* type=*"tns:EchoResponse"*></element>  <complexType name=*"EchoRequest"*>  <sequence>  <element name=*"Text"* type=*"string"*></element>  </sequence>  </complexType>    <complexType name=*"EchoResponse"*>  <sequence>  <element name=*"Text"* type=*"string"*></element>  </sequence>  </complexType>  </schema> |

## Java klasside genereerimine

Java klassid genereeritakse teenust kirjeldavatest XSD skeemidest Java standarddistributsiooni kuuluva *JAXB* implementatsiooni *xjc* utiliidi abil. Viimane on saadav käsurealt iga *JEE 1.5*+ distributsiooniga ning ka *Eclipse*-i pluginina. *Schema*-st on võimalik genereerida Java klassid järgmise käsuga:

|  |
| --- |
| xjc -p com.project.model -d myproject/src myschema.xsd |

Seejärel saab genereeritud klasse kasutada edasises äriloogikas.

Kuna X-tee teenuste definitsioonid on staatilised ja versioneeritud, ei ole klasside genereerimine ehitusprotsessi ajal otstarbekas ning seda tuleks teha üks kord teenuse arendamisel.

Objektide genereerimine *JAXB* abil ei ole kohustuslik. Võib kasutada ka muid tehnoloogiaid (*DOM*, *XMLBeans*, *Axis* jne.). Vajadusel saab *marshalling* funktsionaalsuse ise implementeerida laiendades AbstractXRoadBaseEndpoint-i.

AbstractXRoadBaseEndpoint-i abstraktse meetodi signatuur on kujul:

|  |
| --- |
| **protected** **abstract** **void** invokeInternal(XTeeMessage<Document> request, XRoadMessage<Element> response) **throws** Exception; |

XTeeMessage on ühe sissetuleva päringu abstraktsioon, mis koosneb päisest, kehast ja manustest.

## Teenuste Endpoint-de kirjutamine

Iga teenuse versiooni jaoks tuleb kirjutada *Endpoint*. MVC struktuuri järgi on teenusekiht *Model*, *Endpoint Controller*-i ja XML formaadis vastus *View* rollis.

Seega määrab *Endpoint*, kuidas töödeldakse sissetulevat päringut. *Endpoint*-i jõudmise hetkeks on päring esitatud Java objektina mille tüüp (klass) vastab teenust kirjeldavas skeemis kirjeldatud tüübile.

Näide:

|  |
| --- |
| @Component  @XTeeService(name = "Echo", version = "v1", title = "Echo service")  **public** **class** EchoEndpoint **extends** AbstractXTeeJAXBEndpoint<EchoRequest> {  @Override  **protected** Class<EchoRequest> getParingKehaClass() {  **return** EchoRequest.**class**;  }  @Override  **protected** EchoResponse invokeBean(EchoRequest requestBean) {  EchoResponse response = **new** EchoResponse();  response.setText(requestBean.getText());  **return** response;  }  } |

Juhend:

1. Kõikide *Endpoint*-de klassihierarhias peab sisalduma AbstractXTeeBaseEndpoint. Antud näite puhul on *AbstractXTeeJAXBEndpoint* AbstractXTeeBaseEndpoint-i alamklass.
2. Vaikimisi meetodi nimi andmekogus genereeritakse *Endpoint*-i klassi nimest. *HelloWorldEndpoint*-i puhul oleks meetodi nimi kujul: andmekogu.helloWorld.v1
3. Meetodi nimi ja versioon on muudetavad annotatsiooni @XTeeService atribuutide *name* ja *version* abil.
4. *EchoRequest* ja *EchoResponse* on siin näites XSD skeemist *xjc* abil genereeritud Java klassid.

## Konfiguratsioon

Esmalt tuleks *web.xml*-s defineerida X-tee päringuid vastu võttev *servlet*, mida võib teha allpool toodud näite põhjal:

|  |
| --- |
| *<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd" version="2.4">*  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>/WEB-INF/xroad-servlet.xml</param-value>  </context-param>  <servlet>  <servlet-name>springLoaderServlet</servlet-name>  <servlet-class>  org.springframework.web.context.ContextLoaderServlet  </servlet-class>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet>  <servlet-name>xroad</servlet-name>  <servlet-class> org.springframework.ws.transport.http.MessageDispatcherServlet  </servlet-class>  <load-on-startup>2</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>xroad</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  </web-app> |

Sellele järgnevalt on vaja luua *servlet*ile vastav konfiguratsioonifail. Valdav enamus konfigureeritakse annotatsioonide abil, seega on konfiguratsioonifaili kirjutamine reeglina ühekordne.

Siinkohal on olulised:

1. *context-path –* pakett, kus asuvad skeemadest genereeritud klassid.
2. *xteeDatabase* – andmekogu nimi millesse teenus kuulub
3. *naidis –* WSDL definitsioon. WSDL dokumendi nimi sõltub bean-i nimest. Jooksvas aplikatsioonis on WSDL antud näite puhul kättesaadav aadressil [*http://server/xroad/naidis.wsdl*](http://server/xroad/naidis.wsdl).
4. Annotatsioonide konfiguratsioon paketile *com.nortal.jroad* (tagamaks et toimuks *Spring'i* @Component annotatsioonidega klasside skaneerimine)

Näide:

|  |
| --- |
| <beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:context=*"http://www.springframework.org/schema/context"* xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd*  *http://www.springframework.org/schema/context* [*http://www.springframework.org/*](http://www.springframework.org/) *schema/context/spring-context-2.5.xsd"*>  <context:component-scan base-package=*"com.nortal.jroad"* />  <bean id=*"contextPath"* class=*"java.lang.String"*>  <constructor-arg type=*"java.lang.String"* value=*"com.nortal.jroad.example.model"* />  </bean>  <bean id=*"schema"* class=*"org.springframework.xml.xsd.SimpleXsdSchema"*>  <property name=*"xsd"* value=*"/WEB-INF/schema/naidis.xsd"* />  </bean>  <bean id=*"naidis"* class=*"com.nortal.jroad.wsdl.XTeeWsdlDefinition"*>  <property name=*"schema"* ref=*"schema"* />  <property name=*"portTypeName"* value=*"naidisPort"* />  </bean>  <bean id=*"xteeDatabase"* class=*"java.lang.String"*>  <constructor-arg value=*"naidis"* />  </bean>  </beans> |

## Manuste kasutamine

Suuremate andmehulkade (näiteks piltide, digidoci failide, jms) edastamiseks on soovitatav kasutada manuseid.

Manused omavad mitmeid eeliseid binaarsete andmete otse *XML*-s saatmise ees:

1. Välditakse manuse parsimist *XML* parseriga.
2. Ei kulutata ressursse *Base64*-ks kodeerimiseks ja dekodeerimiseks.
3. Manuseid saadetakse binaarkujul seega väheneb sõnumite maht.

Mingi elemendi sisu saatmiseks eraldi manusena, tuleb toimida järgnevalt:

*XSD* skeemi peab importima swaref nimeruumi:

|  |
| --- |
| <schema xmlns=*"http://www.w3.org/2001/XMLSchema"* targetNamespace=*"http://www.example.org/test"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:tns=*"http://www.example.org/test"* elementFormDefault=*"qualified"*  xmlns:ref=*"http://ws-i.org/profiles/basic/1.1/xsd"*>  <import namespace=*"http://ws-i.org/profiles/basic/1.1/xsd"* schemaLocation=*"*<http://ws-i.org/profiles/basic/1.1/swaref.xsd>*"*/> |

Seejärel peaks defineerima mõne elemendi tüübiga swaRef.

|  |
| --- |
| <complexType name=*"AttachmentEchoRequest"*>  <sequence>  <element name=*"Attachment"* type=*"ref:swaRef"*></element>  </sequence>  </complexType> |

Nüüd on võimalik genereerida Java klassid ning võtta nad kasutusele *Endpoint*-s.

## Näidisrakendus

J-road teekiga tuleb kaasa näidisrakendus, milles on implementeeritud nii serveri kui ka kliendi osa vajaliku konfiguratsiooni ja kirjeldustega. Näidisrakenduse käivitamiseks ehita j-road teek kokku ja navigeeri moodulisse *example* ja käivita *Jetty* server.

|  |
| --- |
| mvn jetty:run |

Näidisrakenduse kontekstid:

* Teenus kuulab päringuid kontekstil /xroad ja teenuse *WSDL*-i saab kätte kontekstilt /xroad/näidis.wsdl
* Teenuse klient on kättesaadav kontekstilt /client/test