Tuomas Toivonen

Monimutkaisen järjestelmän Java-version nosto

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

3.6.2018

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Tuomas Toivonen Monimutkaisen järjestelmän Java-version nosto 53 sivua + 0 liitettä 3.6.2018
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Ohjelmistokehittäjä Okko Welin Lehtori Simo Silander

Tämä raportti kuvaa monimutkaisen liiketoimintajärjestelmän Java-version noston toteutuksen. Kohdejärjestelmä on vuonna 2006 maanlaajuiseen viranomaiskäyttöön otettu asiainhallintajärjestelmä, joka on ollut jatkuvan kehityksen alaisuudessa ensimmäisestä julkaisuversiosta lähtien. Järjestelmä on toteutettu pääsääntöisesti Java-teknologioilla. Kohdejärjestelmän Java-versio nostettiin versiosta 6 (ohjelmakoodi) ja 7 (ajoympäristö) versioon 8 niin ohjelmakoodin kuin ajoympäristön osalta. Versionnoston toteutusvaiheessa ilmeni lukuisia virhetilanteita, joiden korjaaminen vaati runsaasti selvitystyötä ja asiantuntemusta. Lopputuloksena järjestelmän versionnosto Java 8 -alustalle onnistui, ja tähän raporttiin on dokumentoitu yksityiskohtaisesti versionnoston toteutuksen vaiheet profiloiden ilmenneet virhetilanteet ratkaisuineen. Toteutus on vain osa koko versionnoston prosessia, johon laajempana kokonaisuutena kuuluvat edeltävä suunnitteluvaihe ja seuraava testausvaihe sekä viimeisenä tuotantoon vienti. Tässä raportissa kuvataan vain versionnoston toteutusvaihe sekä toteutukseen liittyvät esivalmistelut ja kehitysympäristön perustaminen.

Avainsanat	Java, Java EE, Oracle WebLogic, versioyhteensopivuus, JDK, luokkalataus, luokkapolku, kirjasto, konfiguraatio

Author(s) Title Number of Pages Date	Tuomas Toivonen Java Version Migration of a Complex System 53 pages + 0 appendices 03 June 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Okko Welin, Software Developer Simo Silander, Senior Lecturer

This report describes the Java version migration of a complex enterprise system. The system in question is national document management system used by the authoritative. The system saw light in 2006, being under continuous development and maintenance since the initial release. Implementation of the system is based on the Java technology for the most part. On the migration, the Java version was upgraded from major version 6 (program code) and 7 (runtime environment) to major version 8 for both the program code and the runtime environment. During the migration various runtime errors emerged, which required considerable amount of effort and expertise to trace and resolve. Ultimately, the Java version migration succeeded. In this report is documented the whole implementation phase of the migration process profiling emerged problems and their effective solutions in detail. Implementation is just a fraction of the whole migration process, others being the preceding planning stage and the subsequent testing phase, and ultimately the release. However, this report describes only the implementation phase and the necessary preparing. Also, setting up the proper development environment is described in this document.

Keywords	Java, Java EE, Oracle WebLogic, version compatibility, JDK, class loading, classpath, library, configuration

Sisällys

Lyhenteet

1 Johdanto 1

2 Kohdejärjestelmä 3

2.1 Järjestelmäarkkitehtuuri 3
2.2 Ohjelmistoarkkitehtuuri 4
2.2.1 Sovelluskerrokset 4
2.2.2 Tietokanta 6
2.2.3 Java EE 8
2.2.4 OpenFrame-ohjelmistokehys 14
2.2.5 Valmiskomponentit 16
2.3 Java-projektit 16
2.4 Riippuvuudet 18

3 Java-version nosto 20

3.1 Tavoite 20
3.2 Kehitysympäristön asennus 21
3.2.1 Ubuntu Linux -käyttöjärjestelmä 21
3.2.2 Java SE Development Kit 8 22
3.2.3 Ant ja Maven koontityökalut 25
3.2.4 Eclipse 27
3.2.5 Oracle WebLogic Server 12.2.1.2 28
3.2.6 Apuohjelmat 29
3.2.7 Ympäristön perustamisen automatisointi 30
3.3 Lähdekoodien tasonnosto 31
3.3.1 Käyttöliittymä 31
3.3.2 OpenFrame 38
3.3.3 Integraatio 44
3.3.4 Avaintenhallinta 47
3.3.5 Avaintenhallinnan käyttöliittymä 48
3.4 Versionnoston jälkeen 50
3.4.1 Regressiotestaus 50
3.4.2 Kehityshaarjojen yhdistäminen pääkehityshaaraan 51

3.4.3 Ympäristöjen ja sovellusten asennukset 52

4 Yhteenveto 52

Lähteet 53

Lyhenteet

CI	Continuous Integration (jatkuva integraatio). Ohjelmistotuotannon menetelmä tuotantolinjan automatisoinniksi ja muutosten integroimiseksi. CI-palvelimen automatiikka koostaa uuden sovelluspaketin välittömästi projektin muutoksen yhteydessä suorittaen järjestelmälle vaadittavat hyväksymistestit toiminnallisuuden varmistamiseksi.
DD	Deployment Descriptor. Java EE -komponentin käyttöönottokuvaus. Sovelluspalvelimen vaatima konfiguraatio komponentin käyttöönottamiseksi.
EAR	Enterprise Application Archive. Java EE -spesifikaation määrittämä sovelluspaketin tiedostoformaatti ja määrätty hakemistorakenne.
EJB	Enterprise JavaBeans. Java EE:n määrittämä teknologia liiketoimintakomponenttien toteuttamiseksi. EJB tarjoaa liiketoimintakomponenteille muun muassa transaktionhallintaan ja tietoturvaan liittyvää ajonaikaista tukea.
J2EE	Java EE -alustan vanhempi nimitys 1.5 versioon asti.
JAR	Java Archive. Java-luokkia ja resursseja sisältävän arkiston tiedostoformaatti ja määrätty hakemistorakenne. Kirjastot ja EJB-komponentit paketoidaan JAR-arkistoon.
Java EE	Java Platform, Enterprise Edition. Spesifikaatio ja joukko ohjelmointirajapintoja hajautettujen liiketoiminnallisten palvelukokonaisuuksien toteuttamiseksi. Java SE:n laajennus.
Java SE	Java Platform, Standard Edition. Javan perustoiminnallisuuden tarjoama alusta. Käsittää esimerkiksi JVM-virtuaalikoneen ja Java-luokkakirjaston.
JAXB	Java Architecture for XML Binding. Teknologia XML-muotoisen tiedon muuttamiseksi Java-olioiksi ja toisin päin.

JAX-WS	Java API for XML Web Services. Java EE -alustan määrittämä teknologia ensisijaisesti SOAP-pohjaisten WS-palveluiden toteuttamiseksi.
JCP	Java Community Process. Vapaaehtoistyöhön perustuva toimintamalli, jossa kiinnostuneet yksityishenkilöt tai organisaatiot osallistuvat Java-alustan teknisten spesifikaatioiden määrittely- ja kehitysprosesseihin muodostamalla JSR-dokumentteja JCP-työvaliokunnan käsiteltäväksi.
JDBC	Java Database Connectivity. Java-alustan tarjoama ohjelmointirajapinta tietokantaoperaatioihin.
JDK	Java Developement Kit (kehittäjän työkalut). Käsittää muun muassa JVM-virtuaalikoneen, luokkakirjaston, Java-kääntäjän ja muita työkaluja.
JMS	Java Message Service. Java EE:n määrittämä teknologia sanomajono-pohjaiseen viestintään.
JNDI	Java Naming and Directory Interface. Java EE:n määrittämä teknologia hakemistopalvelujen käyttöön.
JSR	Java Specification Requests. JCP-yhteisön kehittämä määrittelydokumentti, joka kuvaa uutta ehdotettua Java-alustan toiminnallisuutta. JCP-työvaliokunnan hyväksyessä ehdotus toiminnallisuus lisätään osaksi Java-alustan tulevaa julkaisuversiota. Lopulliseen JSR-dokumenttiin liittyy myös spesifikaation referenssitoteutus.
JVM	Java Virtual Machine. Java-sovellusten suoritusympäristö, alustariippumaton virtuaalikone.
LOC	Lines of code. Koodirivien määrän ilmaiseva suure.
MDB	Message-Driven Bean. EJB-komponentti, jonka herätteenä toimivat JMS-sanomajonoon saapuvat sanomat.
MVC	Model-view-controller. Malli-näkymä-käsittelijä-arkkitehtuuri, suosittu suunnittelumalli graafisten käyttöliittymien toteuttamiseksi.

OF	OpenFrame. J2EE-alustaan perustuva sovelluskehys.
ORM	Object-relational mapping. Yleisnimitys tekniikoille, joilla tietokannan tauluihin varastoitu relaatiomuotoinen tieto muutetaan Java-olioiksi ja toisin päin.
POJO	Plain Old Java Object. Tavallinen Java-luokka, jonka toteutukseen ei liity infrastruktuurin asettamia rajoitteita tai vaatimuksia.
QA	Quality Assurance (laadunvarmistus). Kehitettävän tuotteen tai palvelun laadun säilyvyyden takaava toimintamalli. QA-ympäristössä esimerkiksi suoritetaan järjestelmän hyväksymistestit.
RAC	Oracle Real Application Clusters. Oraclen kehittämä tuote tietokantainstanssien ryvästämiseksi (monistamiseksi).
SEI	Service Endpoint Interface. WS-palvelurajapintaa (WSDL) vastaava Java EE:n määrittelemä luokkarajapinta. Käytetään SOAP-palveluiden toteutukseen.
Servlet	Java EE:n määrittämä teknologia ja ohjelmointirajapinta web-toiminnallisuuden toteuttamiseksi. Matalimman tason rajapinta Java EE -sovelluksen ja web-palvelintoteutuksen välillä.
SOA	Service-oriented architecture (palvelukeskeinen arkkitehtuuri). Järjestelmäarkkitehtuuri, jossa järjestelmäkokonaisuus koostuu erillisistä, keskenään viestivistä palveluista.
SOAP	Simple Object Access Protocol. XML-pohjainen tietoliikenneprotokolla alustariippumattomien WS-palvelujen ja etäproseduurikutsujen toteuttamiseksi. Eräs ratkaisu SOA-arkkitehtuurin käyttämäksi viestintäformaatiksi.
WAR	Web Application Archive. Java EE:n määrittämä web-sovelluksen tiedostoformaatti ja määrätty hakemistorakenne.
WLS	Oracle WebLogic Server. Oraclen kehittämä sovelluspalvelin, eräs Java EE -spesifikaation toteuttava teknologia.

WSDL	Web Service Description Language. W3C-organisaation määrittämä standardi ja XML-dokumentti teknologiariippumattomien verkkopalvelurajapintojen kuvaukseen.
WS-palvelu	Web service. Verkossa tarjottava palvelurajapinta muille verkon järjestelmille, esimerkiksi SOAP-palvelu.
XML	Extensible Markup Language. Yleiskäyttöinen merkintäkieli ja standardi rakenteellisen tiedon kuvaukseen, varastointiin, siirtämiseen ja validointiin.

1 Johdanto

Versioyhteensopivuus on monimutkainen ja toisinaan harmaita hiuksia aiheuttava aihealue, joka ennemmin tai myöhemmin tulee vastaan jokaiselle Java-alustalla työskentelevälle ohjelmistokehittäjälle.

Tämä raportti kuvaa työnkulun, kuinka monimutkaisen järjestelmäkokonaisuuden Java-version tasonnosto toteutettiin ja millaisia haasteita versionnostoon liittyi. Versionnosto suoritettiin osana kohdejärjestelmän migraatiota AIX-ympäristöstä Linux-ympäristöön. Linux-migraatio käsitti myös sovelluspalvelimen ja tietokannan versiopäivitykset, uuden QA-ympäristön perustamisen ja lukuisia muita oheistoimia. Tämä työ käsittelee kohdejärjestelmän Java-version nostoa ja sovelluspalvelimen päivitystä. Muita Linux-migraation osa-alueita käsitellään vain siinä määrin, kuin ne ovat versionnoston kannalta oleellisia.

Versionnosto ei monitutkaisten järjestelmien tapauksessa ole suoraviivainen yhden napin painalluksen vaativa toiminto, vaan huolellista suunnittelua, tutkimustyötä ja virheenselvitystä vaativa iteratiivinen prosessi. Versionnostossa piilee riski, sillä sen yhteydessä mahdollisesti ilmaantuvat ohjelmavirheet voivat olla todellinen uhka järjestelmän vakaudelle ja sen ohjaamalle liiketoiminnalle. Versionnosto on suunniteltava huolellisesti, ja toteutuksen päätteeksi on suoritettava huolellinen regressiotestaus kattaen kaikki järjestelmän käyttötapaukset.

Versionnoston toteutuksen yhteydessä ilmenevien ongelmatilanteiden johdonmukaiseksi ratkaisemiseksi on ymmärrettävä Java-alustan perustoiminnallisuus matalalla tasolla, tunnettava valitun sovelluspalvelintuotteen toimintamekanismit, sekä tietysti tunnettava kohdejärjestelmä niin toiminnallisella kuin toteutuksellisella tasolla.

Motiivi versionnostolle on järjestelmän ylläpidettävyys ja jatkuvuuden takaaminen. Vaikuttavia voimia ovat niin tuotteen kehityksessä ilmenevät tekniset vaatimukset kuin tuotteen tilaajan asettamat yleiset linjaukset. Koodikanta ja käytössä olevat teknologiat on pyrittävä pitämään tuoreina, sillä kehityksestä pois putoaminen on teknisen velan ottamista, joka voi tehdä tuotteen jatkokehityksestä tulevaisuudessa vaikeaa, kannattama-

tonta, tai pahimmassa tapauksessa mahdotonta. Versionnosto tuo uuden alustan tarjoaman toiminnallisuuden osaksi tuotteen kehitystä. Tuotteen jatkokehityksestä tulee todennäköisesti liiketoiminnallisesti kannattavampaa molemmille osapuolille.

Kohdejärjestelmässä käytettävä Java-versio nostettiin versiosta 6 versioon 8. Ennen versionnostoa lähdekoodi käännettiin yhteensopivuussyistä Java 6 -kääntäjällä, mutta ajettiin Java 7 -ympäristössä. Versionnoston yhteydessä sekä lähdekoodi että ajoympäristö yhdenmukaistettiin Java 8 -tasolle. Ajoympäristön Java-version nosto käsitti Oracle WebLogic -sovelluspalvelimen (jatkossa WLS) päivityksen versiosta 11g versioon 12c. WLS 12c -palvelin on täyden pinon Java EE 7 -toteutus, jolle Oracle takaa saumattoman Java 8 -yhteensopivuuden.

Versionnoston suurimmat haasteet liittyivät uuden WLS 12c -palvelimen ennalta-arvaamattomaan, aiemmasta 11g-versiosta poikkeavaan käyttäytymiseen. Suurin tekijä ongelmatilanteiden syntyyn oli Java EE -spesifikaation suppeus tietyissä määreissä, erityisesti ohjelmistokomponenttien tunnistamiseen liittyvissä sovelluspalvelintuotteen toteuttamissa mekanismeissa. Näihin ongelmatilanteisiin palataan 3. pääluvussa, jossa kuvataan kohdejärjestelmän versionnosto yksityiskohtaisesti.

Järjestelmän versionnoston ensimmäinen vaihe, eli lähdekoodin tasonnosto, valmistui noin kolmessakymmenessä henkilötyöpäivässä, mutta seuraavaa tärkeää vaihetta, eli regressiotestausta ei ole aloitettu tämän raportin valmistumiseen mennessä. Raportti käsittelee versionnostoprosessia toteuttajan näkökulmasta tuoden esiin suurimmat toteutuksen aikana ilmenneet ongelmatilanteet ja niiden ratkaisut. Virhetilanteiden juurisyyt pyritään selittämään teknisen seikkaperäisesti.

Koska testausvaihetta ei ole vielä aloitettu, ei tässä raportissa esitettyjä ratkaisuja voida pitää täydellisen varmoina - korkeintaan todennäköisinä, tai ainakin suuntaa antavina. Raportin tarkoituksena on viitoittaa tietä niille, jotka kohtaavat vastaavia haasteita Java-versionnoston yhteydessä tai WLS 12c -palvelimen käyttöönotossa.

2 Kohdejärjestelmä

2.1 Järjestelmäarkkitehtuuri

Kohdejärjestelmä, jolle versionnosto toteutettiin, on julkishallinnon maanlaajuisessa, ympärivuorokautisessa käytössä oleva asiainhallintajärjestelmä, joka tukee organisaation harjoittaman liiketoiminnan sähköistämistä. Järjestelmän toiminnallisiin kokonaisuuksiin kuuluvat muun muassa asiakkuuksien hallinta, asiainhallinta, raportit ja tilastot, sähköinen asiointi ja työjonot. Järjestelmäarkkitehtuuri perustuu hajautettuun SOA-malliin, jossa eri toimintoja tarjoavat palvelut viestivät keskenään verkon välityksellä WS-palvelurajapintojen läpi.

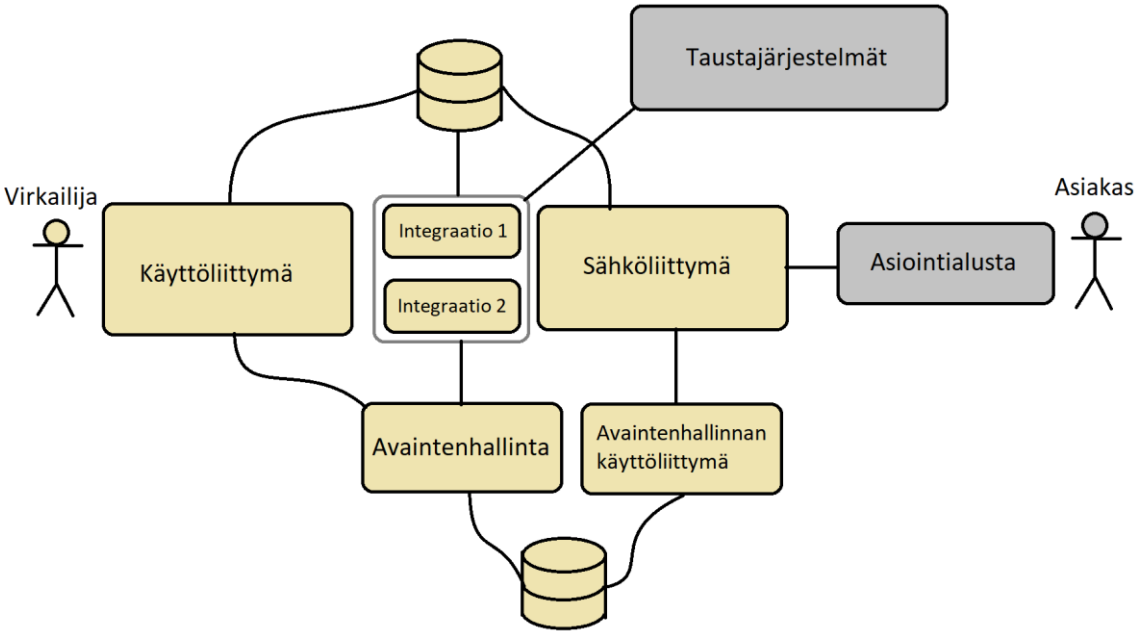
Järjestelmäkokonaisuus koostuu kuudesta järjestelmäkomponentista, joita ovat

- käyttöliittymä
  - web-sovellus
  - tarjoaa selainpohjaisen käyttöliittymän virkailijalle
  - keskustelee erinäisten työasemasovellusten kanssa.
- sähköliittymä
  - tarjoaa verkkopalveluita sähköiselle asiointialustalle.
- integraatio
  - viestii taustajärjestelmien kanssa sanomapohjaisesti
  - toteutus pilkottu kahteen erilliseen moduuliin (sovelluspakettiin).
- avaintenhallinta
  - tarjoaa järjestelmille varmenne- ja salauspalveluita.
- avaintenhallinnan käyttöliittymä.
  - selainpohjainen käyttöliittymä varmenneketjujen ylläpitoon.

Keskenään järjestelmät kommunikoivat verkon yli käyttäen SOAP-protokollaa, sanomajonoa tai operoiden yhteisessä tietokannassa. Järjestelmiä ajetaan **WLS 11g** -sovelluspalvelimella ja tietovarastona toimii **Oracle DB 11g R2** -relaatiotietokanta. Tuotanto- ja testiympäristöissä järjestelmät ovat korkean saatavuuden takaamiseksi asianmukaisesti

monistettu. Verkkotopologiaan liittyvät ratkaisut ovat kuitenkin tämän raportin aihealueen ulkopuolella.

Järjestelmään kytkeytyy erinäisiä virkailijan työasemalla suoritettavia työasemasovelluksia. Nämä sovellukset on toteutettu muilla teknologioilla kuin Javalla, eivätkä kuulu versionnoston vaikutuksen alaisuuteen. Kuvassa 1 on esitetty järjestelmän keskeisimmät komponentit ja niiden väliset yhteydet.



Kuva 1. Järjestelmäkomponentit.

2.2 Ohjelmistoarkkitehtuuri

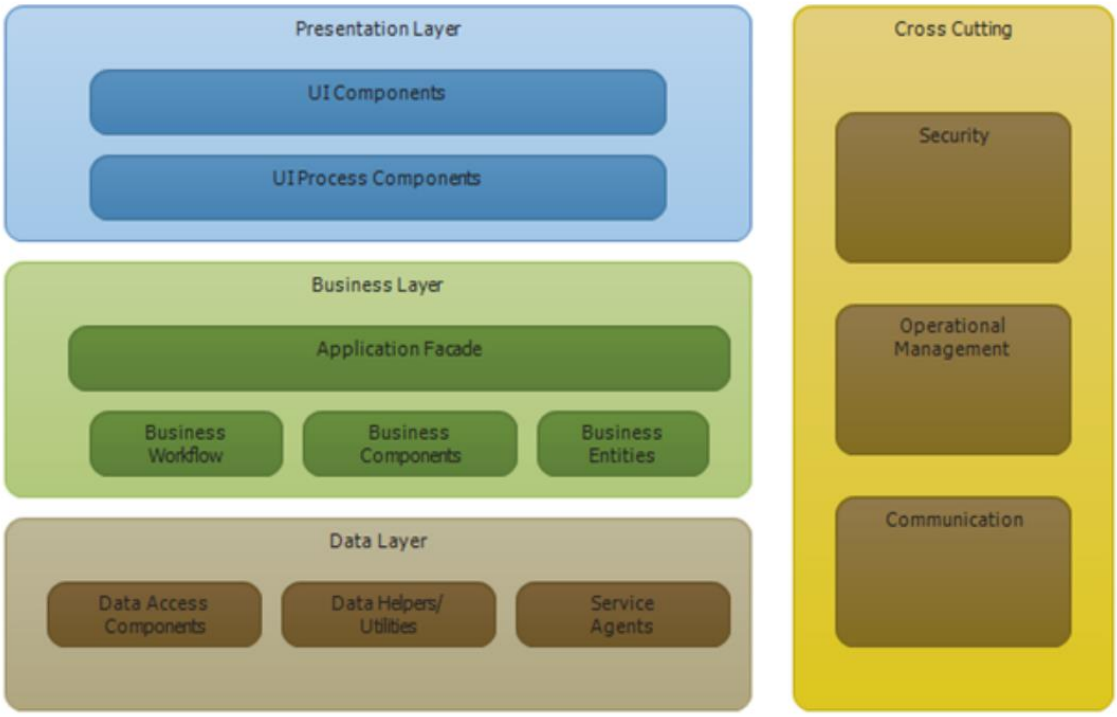
2.2.1 Sovelluskerrokset

Järjestelmien tekninen toteutus perustuu Java EE -alustalle rakennettuun monikerrosarkkitehtuuriin, jossa järjestelmän sovelluskoodi on jaettu vastuualueensa mukaan rajattuihin sovelluskerroksiin. Pääsääntöisesti käytössä ovat seuraavat yleisesti hyväksytyt sovelluskerrokset:

- käyttöliittymäkerros (esitystapakerros)
    - Tuottaa virkailijalle näytettävän käyttöliittymän (verkkosivun) ja ohjaa käyttöliittymältä saapuvat pyynnöt ohjauskerrokselle.

- ohjauskerros
    - Saa käyttöliittymäkerrokselta herätteitä ja kutsuu liiketoimintakerroksen palveluita. Ohjaa käyttäjän etenemistä sovelluksen toiminnoissa.
- liiketoimintakerros
    - Tarjoaa liiketoimintaspesifisiä, uudelleenkäytettäviä palveluita.
    - Hallitsee liiketoimintamalliin kuuluvien entiteettien (olioiden) tilaa.
    - Java EE:n EJB -teknologia huolehtii mm. transaktioiden eheydestä ja muista läpileikkaavista seikoista.
- tiedonhallintakerros
    - Huolehtii tiedon haku- ja tallennusmekanismeista abstrahoiden todellisen tallennusmekanismin muilta sovelluskerroksilta.
    - Tietovarastona voi toimia esimerkiksi relaatiotietokanta tai välimuisti kontekstista riippuen.
    - Tehokkuuden optimoinnin kannalta kriittisin sovelluskerros.
- liittymäkerros
    - Tarjoaa verkkopalvelurajapintoja sisään- ja ulospäin lähtevälle liikenteelle järjestelmien väliseen viestintään.
    - Palvelut ovat toteutettu SOAP-protokollan mukaisina WS-palveluina.
    - Toteuttaa myös tekstiviestiliittymän (SMS).

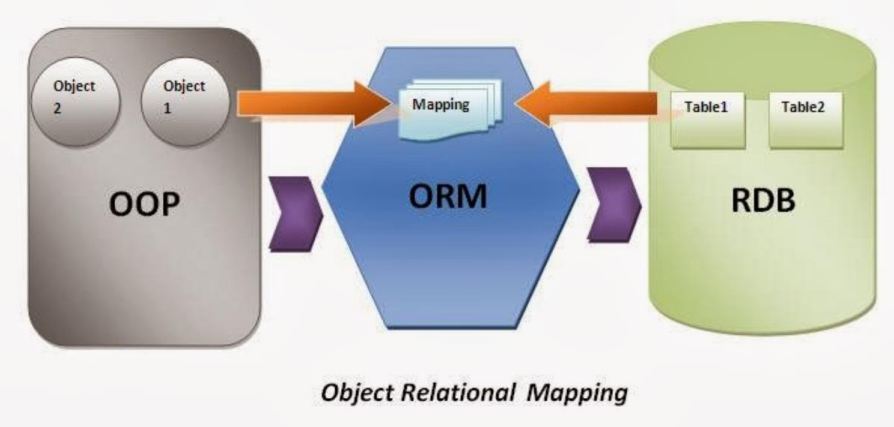
Kuva 2 havainnollistaa monikerrosarkkitehtuuria yleisesti.



Kuva 2. Eräs monikerrosarkkitehtuuriin perustuva ratkaisu (kuvituskuva).

2.2.2 Tietokanta

Järjestelmän hallinnoimat liiketoimintaentiteetit varastoidaan pysyväissäilytykseen relaatiotietokannan tauluihin. Relaatiomuotoisen tiedon ja Java-olioiden väliseen muunnokseen käytetään muun muassa ORM-teknologiaa sekä muita JDBC-rajapintaa hyödyntäviä tekniikoita. Kuva 3 havainnollistaa, kuinka ORM-teknologia toimii sovittimena relaatiotietokannan ja olioperusteisen sovelluksen välillä. Relaatio- ja oliomallin perustuvanlaatuista epäyhteensopivuutta kutsutaan alan kirjallisuudessa nimellä *object-relational impedance mismatch*. Relaatiotietokannat ja ORM-teknologiat ovat laajoja ja monimutkaisia aihealueita, joita ei tässä raportissa käsitellä tarkemmin.



Kuva 3. ORM-teknologia toimii sovittimena relaatio- ja oliomallin välillä.

Suuri osa varsinaisesta liiketoimintalogiikasta suoritetaan tietokantaproseduureissa ja liipaisimissa. Kantaan ajastetut eräajot huolehtivat säännöllisistä liiketoimintatransaktioista. Kaikki tietokannan sisältämä ohjelmalogiikka on toteutettu Oraclen kehittämällä PL/SQL-kielellä. Kuvassa 4 nähdään esimerkki PL/SQL-kielellä toteutetusta funktiosta. Koska PL/SQL-teknologialla ei ole sidoksia Javaan, ei versionnosto vaikuta kantaan toteutettuihin ohjelmistokomponentteihin.

substring from start until end position

```
CREATE OR REPLACE FUNCTION betwnstr (
   string_in   IN   VARCHAR2,
   start_in    IN   INTEGER,
   end_in      IN   INTEGER
)
   RETURN VARCHAR2
IS
   l_start PLS_INTEGER := start_in;
BEGIN
   IF l_start = 0
   THEN
      l_start := 1;
   END IF;

   RETURN (SUBSTR (string_in, l_start, end_in - l_start + 1));
END;
```

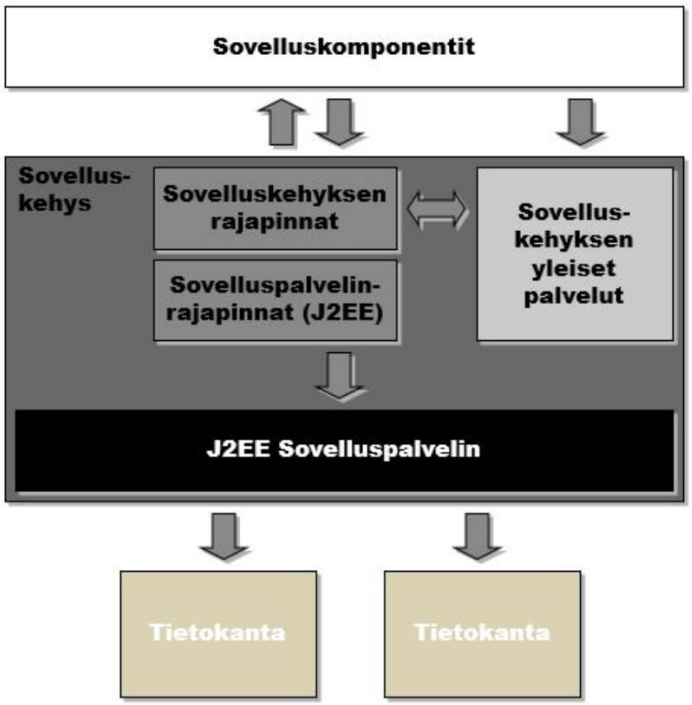
Kuva 4. Esimerkki PL/SQL-kielestä (kuvituskuva).

2.2.3 Java EE

Sovellukset hyödyntävät runsaasti Java EE -alustan tarjoamia teknologioita. Java EE on spesifikaatio ja joukko ohjelmointirajapintoja hajautettujen palvelukokonaisuuksien toteuttamiseksi. Se on liiketoimintasovellusten kehitystä tukeva alusta, joka on käytössä kaikissa tämän järjestelmän projekteissa. Spesifikaatio määrittää kaikille liiketoiminnoille tyypillisiä, valmiita infrastruktuuripalveluita ja valmiskomponentteja, joita ovat muun muassa

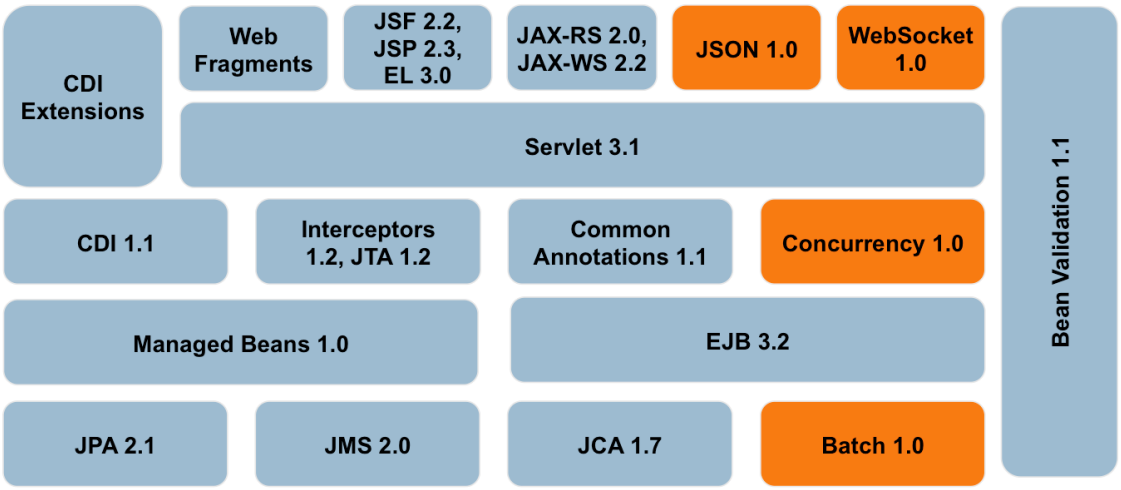
- transaktionhallinta
- web-sisällön tuottaminen
- autentikointi- ja valtuutus
- WS-palvelut
- tiedon validointi
- viestitys
- hakemistopalelut
- tiedonhallinta.

Yllä oleva lista sisältää vain murto-osan alustan tarjoamista palveluista. Liiketoimintasovellukset ohjelmoidaan Java EE:n määrittelemiä ohjelmointirajapintoja vasten. Sovelluspalvelin (alustan tuotekohtainen toteutus) tarjoaa rajapinnoille ajonaikaisen tuen toteuttaen spesifikaation määrittämät palvelut. Spesifikaatio ei kuitenkaan määrää, *kuinka* sovelluspalvelimen palvelut pitää toteuttaa. Järjestelmässä käytössä oleva WLS-palvelin on eräs Java EE -alustan kaupallinen toteutus kilpailijoiden joukossa. Java EE:n kantavana perustuksena ovat tarkkaan määritellyt rajapinnat sekä sopimus palveluiden ajonaikaisesta saatavuudesta ja toiminnallisuudesta. Kuva 5 havainnollistaa Java EE -sovelluksen rajapintakytköksiä kohdejärjestelmän toteutuksessa.



Kuva 5. Java EE perustuu rajapintoihin.

Java EE -alusta käsittää lukuisia teknologioita. Teknologiakohtaisten ohjelmointirajapintojen ja käyttöönottokuvausten (DD) avulla mistä tahansa Java-oliosta (POJO) voidaan tehdä kyseiseen teknologiaan kytkeytyvä, sovelluspalvelimen hallinnoima *Java EE -komponentti*. Kuvassa 6 on esitetty erinäisiä Java EE 7 -alustan tarjoamia teknologioita.



Kuva 6. Suppea otos Java EE 7 -alustan tarjoamista teknologioista.

Spesifikaatio määrittää myös itse sovelluspaketin formaatin ja rakenteen. Määritelmä kuvaa erinäisiä *moduuleja*, jotka voivat toimia sovelluspalvelimella itsenäisinä sovelluksina, tai osana niitä yhteen sitovaa EAR-sovelluspakettia. Moduulien rakenteen ja sisällön on täytettävä spesifikaation asettamat vaatimukset sovelluspaketin asentumiseksi ja toimimiseksi. Moduuli paketoi liiketoimintalogiikan sisältävät, teknologiakohtaiset Java EE -komponentit, taustatukea tarjoavat Java-luokat, kirjastot, resurssitiedostot sekä komponenttien käyttöönottokuvaukset ja muun vaadittavan konfiguraation yhteen pakattuun arkistoon.

Jotain teknologioita voidaan toteuttaa vain tietyn tyyppisissä moduuleissa spesifikaation määräämänä. Sovelluspalvelimen toteutukselliset kerrokset, niin kutsutut *säiliöt*, hallinnoivat moduulien sisältämien komponenttien elinkaarta tarjoten komponenttien käyttämille tai toteuttamille palveluille ajonaikaisen tuen. Taulukossa 1 on kuvattu yhteenveto erityyppisistä Java EE -moduuleista.

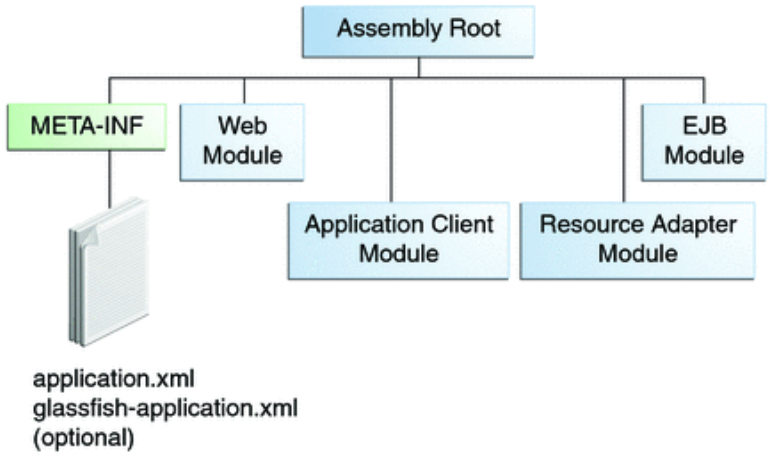
Taulukko 1. Lyhyt yhteenveto Java EE -moduuleista.

Moduuli	**Pakettointi**	**Käyttötapauksia**	**Teknologioita**	**Hallinnoiva säiliö (WLS)**
Web-moduuli	WAR	Web-sisältö, WS-palvelut	Servlet, JAX-WS, JAX-RS	Web-säiliö
EJB-moduuli	JAR	Liiketoimintakomponentit, WS-palvelut	EJB, JAX-WS, JAX-RS JMS	EJB-säiliö
Resurssiadapteri	RAR	Järjestelmien väliset integraatiot	JCA	Connector-säiliö
Asiakas-moduuli	JAR	Liiketoimintasovelluksen asiakaspääte. Kutsuu liiketoimintasovelluksen	-	Asiakas-säiliö (suoritetaan asiakaslaitteella)

		palveluita etäproseduuri-kutsuilla.		

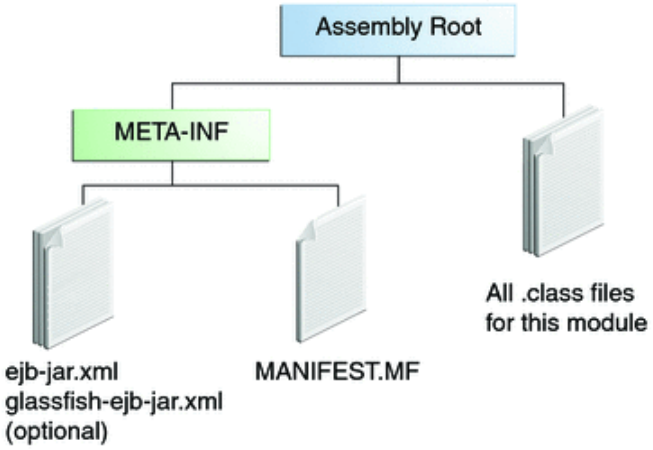
EAR-sovelluspaketti paketoi joukon moduuleja ja moduulien vaatimia kirjastoriippuvuuksia kokonaiseksi liiketoimintasovellukseksi. Moduulikohtaiset säiliöt vastaavat EAR-paketin sisältämien moduulien käyttöönotosta sovelluspaketin asennusvaiheen yhteydessä.

Kuvassa 7 on esitetty EAR-sovelluspaketin rakenne. META-INF-hakemiston on sisällettävä *application.xml*-käyttöönottokuvaus sekä mahdollisesti sovelluspalvelintuotekohtaisia käyttöönottokuvauksia. Itse moduulien tulee sijaita paketin juuressa.



Kuva 7. EAR-sovelluspaketin rakenne.

EAR-paketin sisältämät moduulit ovat pakattuja arkistoja, joilla on vastaavasti oma spesifikaation vaatima hakemistorakenteensa. Kuva 8 havainnollistaa EJB-moduulin rakennetta. META-INF-hakemisto sisältää spesifikaatio- ja sovelluspalvelintuotekohtaiset käyttöönottokuvaukset, sekä muut pakettiin liittyvät metatiedot. Java-luokat sijaitsevat paketin juuressa (package-kohtaisissa alihakemistoissa). Paketti voi myös sisältää mielivaltaisia resurssitiedostoja.



Kuva 8. EJB-moduulin rakenne.

Kuvissa 9 ja 10 on esitetty kaksi vaihtoehtoista tapaa määritellä komponenttien käyttöönottokuvakset: perinteinen XML-konfiguraatiotiedostoon perustuva ratkaisu ja nykyaikaisempi Java-annotaatioihin perustuva ratkaisu. Esimerkeissä on kuvattu tilattoman EJB-istuntopavun käyttöönoton vaatima vähimmäiskonfiguraatio (esimerkeissä käytetään erinimistä luokkaa).

```
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd"
         version="3.0">
  <enterprise-beans>
    <session>
      <ejb-name>CalculatorImpl</ejb-name>
      <business-local>org.superbiz.calculator.CalculatorLocal</business-local>
      <business-remote>org.superbiz.calculator.CalculatorRemote</business-remote>
      <ejb-class>org.superbiz.calculator.CalculatorImpl</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

Kuva 9. XML-konfiguraatiotiedostoon perustuvan käyttöönottokuvaus (kuvituskuva).

```
@Stateless
public class CalculatorService implements CalculatorServiceLocal, CalculatorServiceRemote {

    public CalculatorService() {
    }

    @Override
    public long add(long i, long j) {

  return (i + j);
    }

    @Override
    public double divide(long i, long j) {

  return ((double)i / j);
    }

    @Override
    public long multiply(long i, long j) {

  return (i * j);
    }

    @Override
    public long subtract(long i, long j) {

  return (i - j);
    }

}
```

Kuva 10. Annotaatioihin perustuva käyttöönottokuvaus (kuvituskuva).

Taulukossa 2 on lueteltu merkittävimpiä projekteissa käytettyjä Java EE -teknologioita käytössä olevine versioineen. Versiotiedot on poimittu järjestelmän vanhasta arkkitehtuurikuvauksesta, ja ovat suuntaa antavia.

Taulukko 2. Järjestelmissä käytettyjä Java EE -teknologioita.

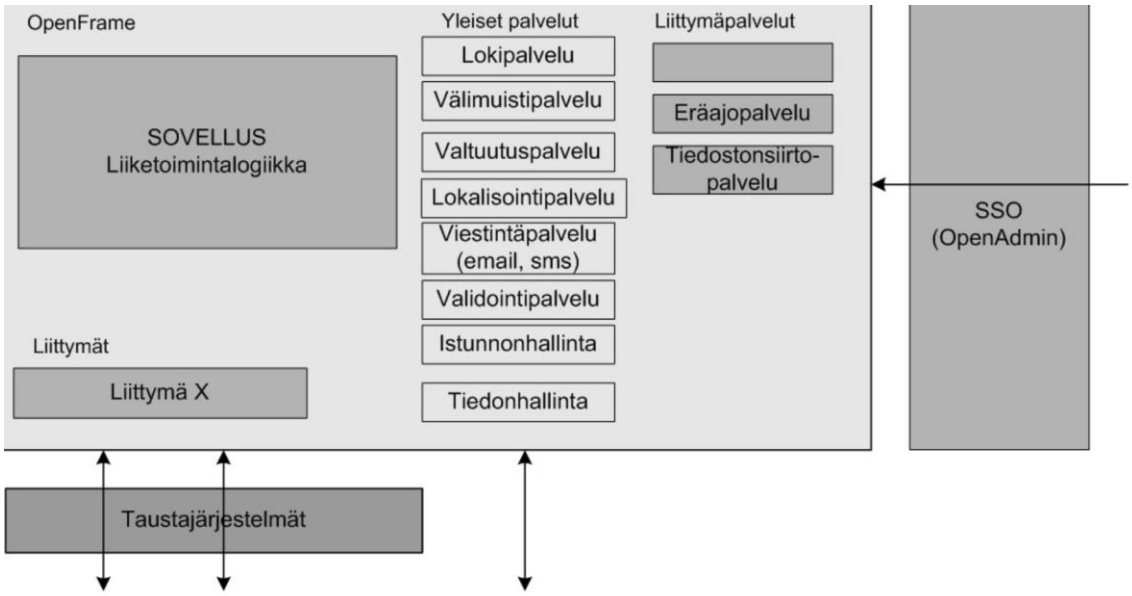
API	**Versio**
Servlet	2.3
EJB	2.0
JDBC	2.0
JMS	1.02b
JNDI	1.2
JavaMail	1.1

2.2.4 OpenFrame-ohjelmistokehys

Käyttöliittymän toteutus perustuu yhtiön vuosituhannen vaihteessa kehittämään, J2EE alustaa laajentavaan OpenFrame-ohjelmistokehykseen (jatkossa OF), joka tarjoaa joukon liiketoimintaan räätälöityjä palveluita ja valmiskomponentteja. OF-teknologian tarjoamia palveluita ja toimintoja ovat muun muassa

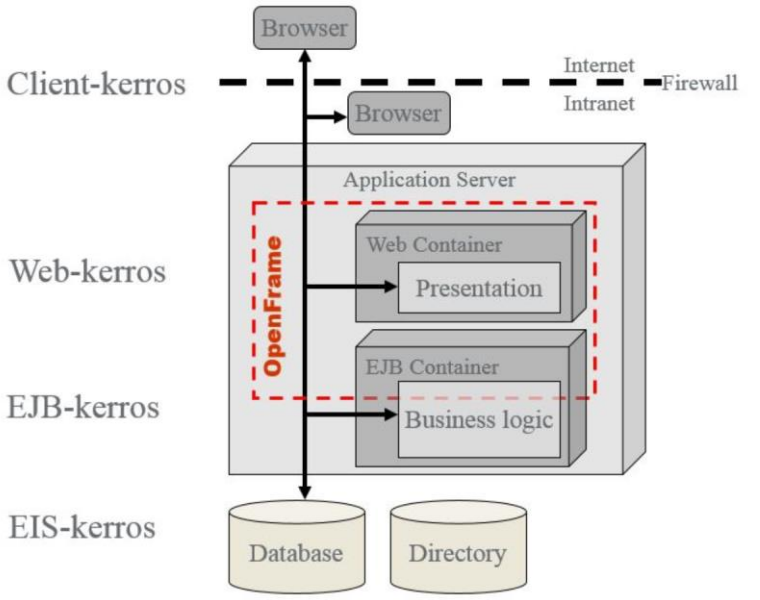
- autentikointi- ja valtuutuspalvelu
- lokipalvelu
- välimuistipalvelu
- syötteiden validointipalvelu
- istunnonhallinta
- viestintäpalvelu
- lokalisointipalvelu (monikielisyys)
- tiedonhallintapalvelu
- tietomuunnokset (XML-binäärimuunnokset)
- poikkeuskäsittely
- MVC-kehys
- sovellusmetriikka ja asetusten hallinta.

Kuva 11 havainnollistaa OF-kehyksen tarjoamia palveluita.



Kuva 11. OpenFrame-sovelluskehyksen tarjoamia yleisiä palveluita.

OF kehitettiin aikaan, jolloin J2EE-alusta ei vielä tarjonnut monia edellä mainittuja palveluita valmistuotteina. J2EE-sovelluskehitys oli tällöin äärimmäisen vaikeaa ja riskialtista projektin kannattavuuden näkökulmasta. [1.] Kuva 12 havainnollistaa, kuinka OF integroituu osaksi Java EE -teknologiapinoa.



Kuva 12. OF osana teknologiapinoa

Java EE on kuitenkin ajan saatossa kehittynyt, ja nyt 2017-luvulla alusta tarjoaa monet OF-kehyksen toteuttamista palveluista. OF on nykyään lähinnä projektin teknologista kehitystä jarruttava taakka. Vahvoista ohjelmallisista sidoksista johtuen OF-kehyksen purkaminen käyttöliittymän toteutuksesta on kuitenkin epärealistinen tavoite. Toisaalta tietyt liiketoimintaa varten räätälöidyt ominaisuudet ovat osoittautuneet erityisen hyödyllisiksi projektin kehityksen yhteydessä. Versionnoston kannalta käyttöliittämän OF-sidonnaisuuteen liittyi yksi varteenotettava uhka, joka kuvataan luvussa 3.3.2.

2.2.5 Valmiskomponentit

Järjestelmät tukeutuvat lukuisiin kolmannen osapuolen sovelluskirjastoihin erinäisistä tehtävistä suoriutuakseen. Taulukossa 3 on listattu suppeasti sovellusten käyttämiä valmiskomponentteja.

Taulukko 3. Sovellusten käyttämiä valmiskomponentteja.

Komponentti	**Käyttötarkoitus**
Log4J	Lokikirjoitus
Apache Axis	SOAP-palvelut
iBatis	ORM
Hibernate	ORM
Spring Framework	Useita käyttötarkoituksia

Projekteilla on tusinoittain kirjastoriippuvuuksia eikä kaikkia riippuvuuksia luetella tämän raportin yhteydessä.

2.3 Java-projektit

Järjestelmiä kehitetään versionhallinnan alaisuudessa erillisissä Java-projekteissa. Projekteista koostetaan koontityökalulla useita erinäisiä artefakteja, ensisijaisesti **sovelluspaketit** kehitys-, testaus- ja tuotantoympäristöihin sekä **kirjastoja** muiden projektien käyttöön. Kirjastot ovat yleinen tapa jakaa projektin sisältämää toiminnallisuutta helposti

siirrettävässä yksikössä (JAR-arkistossa). Esimerkiksi Integraatio-projektista koostetaan sähköliittymän ja integraatiomoduulien sovelluspaketit eri ympäristöihin sekä integraation toiminnallisuutta sisältävä sovelluskirjasto käyttöliittymä-projektin käyttöön. Artefaktit ovat standardinmukaisia JAR-, WAR- tai EAR-muotoisia arkistoja. Taulukko 4 sisältää yhteenvedon projekteista.

Taulukko 4. Yhteenveto projekteista.

Projekti	**Selite**	**Koontityökalu**	**Sovelluskehys**	**Koko (LOC)**
Käyttöliittymä	Selainpohjainen käyttöliittymä virkailijalle	Ant	Java EE, OpenFrame	160.000
Integraatio	Liittymät muihin järjestelmiin	Ant	Java EE	80.000
Avaintenhallinta	Varmenne- ja salauspalvelu	Ant	Java EE, Spring Framework	13.000
Avaintenhallinnan käyttöliittymä	Selainpohjainen käyttöliittymä avaintenhallintaan	Maven	Java EE, Apache Wicket	6000
OpenFrame	J2EE:tä laajentava	Ant	J2EE	75.000

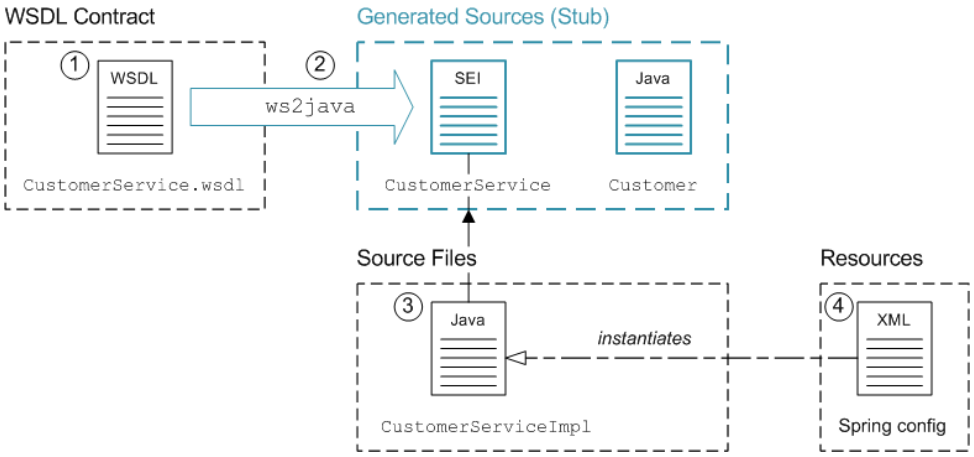
	ohjel- misto- kehys			

Projektit on luotu eri aikoina eri tarpeisiin järjestelmän ja liiketoiminnan kehittyessä hyödyntäen aikansa eri teknologioita. Kaikille projekteille on kuitenkin yhteistä Java EE -alustaan perustuva toteutus.

2.4 Riippuvuudet

Järjestelmillä on tusinoittain riippuvuuksia kolmannen osapuolen sovelluskirjastoihin. Kirjastoriippuvuuksia ei tuoda tässä raportissa esiin kuin niiden kirjastojen osalta, jotka versionnoston yhteydessä osoittautuivat ongelmallisiksi. Järjestelmät ovat SOA-arkkitehtuurin mukaisesti riippuvaisia myös toisistaan, ja nämä järjestelmien väliset riippuvuudet kuvataan tässä luvussa tarkemmin toteutuksellisella tasolla.

Järjestelmien tarjoamat WS-palvelut on toteutettu SOAP-palveluina JAX-WS -spesifikaation mukaisia ohjelmointirajapintoja hyödyntäen. Spesifikaation mukaan järjestelmä kutsuu toista järjestelmää SEI-rajapinnan toteuttavan asiakasluokan läpi. Asiakasluokka suorittaa etäproseduurikutsun, joka on täysin läpinäkyvä asiakkaana toimivalle järjestelmälle. Projekteissa on käytössä *contract first* -tyylinen kehitysmalli, jossa SEI-palvelurajapinnat ja niitä vastaavat asiakasluokat generoidaan palvelun määrittävästä WSDL-dokumentista osana palvelun tarjoavan järjestelmän koontia. Asiakasluokat pakataan JAR-muotoiseen kirjastoon, joka viedään käsin osaksi asiakasjärjestelmän projektia. Kuva 13 havainnollistaa contract first -periaatetta WS-palveluiden kehityksessä.



Kuva 13. Esimerkki contract first -tyylisestä kehitysmallista (kuvituskuva).

Järjestelmien väliset kytkökset ovat suhteellisen moniulotteisia ja vaikeasti hallittavia. Projektien rinnakkainen versionnosto vaati huolellista koordinaatiota eri projektien koonnin ja kirjastoriippuvuuksien päivittämisen vaiheistamiseksi, sillä käytössä ei ole automatiikkaa projektien välisten riippuvuuksien yhtenäiseksi hallitsemiseksi.

Taulukossa 5 on kuvattu projektien väliset riippuvuudet suuntaa-antavasti. Projekteista koostettavia kirjastoja, joihin riippuvuus lähdekooditasolla kohdistuu, on useita, eikä niitä eritellä tässä taulukossa.

Taulukko 5. Projektien väliset riippuvuudet.

Projekti	**Riippuvuus**
Käyttöliittymä	Integraatio, Avaintenhallinta, OpenFrame
Integraatio	Avaintenhallinta
Avaintenhallinta	-
Avaintenhallinta käyttöliittymä	Avaintenhallinta
OpenFrame	-

Jotkin projektit ovat riippuvaisia liiketoimintakumppaneiden toimittamista WSDL-dokumenteista, jotka kuvaavat kyseisten osapuolten hallinnoimien järjestelmien tarjoamia verkkopalvelurajapintoja.

3 Java-version nosto

3.1 Tavoite

Kohdejärjestelmän Java-version tasonnosto on kokonaisuudessaan jaoteltu kolmeen loogiseen vaiheeseen, jotka ovat

- koodikantojen Java-version tasonnosto
- järjestelmän regressiotestaus
- ympäristöjen ja sovellusten asennukset.

Tämä raportti käsittelee versionnoston enimmäistä vaihetta, eli sovelluksien koodikantojen Java-version tasonnostoa. Tavoitteena on nostaa koodikantojen Java-versio tasolle 1.8 ja varmistaa järjestelmien toimivuus WLS 12c -sovelluspalvelimella, Java SE 8 -alustalla suoritettuna. Versionnoston yhteydessä tehdään vaadittavat sovellus- ja konfiguraatiomuutokset toimivuuden takaamiseksi. Oheistuotteena syntyy dokumentaatio, jota tullaan käyttämään ympäristöjen pystykseen ja sovelluspakettien asennukseen. Koodikantojen Java-version tasonnosto käsittää kuusi esivalmistelu- tai toteutusvaihetta vaihetta, jotka ovat

- Java SE 8 -suoritusympäristön ja WLS 12c -sovelluspalvelimen asennukset ja konfiguraatiot
- koodikantojen kääntäminen, koonti ja asennus JDK 1.8 -versiolla sisältäen tarvittavat lähdekoodin muutokset
- kolmansien osapuolten kirjastoriippuvuuksien tutkinta ja päivitykset
- OpenFrame-sovelluskehyksen toiminnallisuuden tutkinta ja korjaus
- asennusten ja konfiguraatioiden dokumentointi
- virhetapausten dokumentointi.

Versionnoston toteutusta varten toteuttaja luo uutta kohdeympäristöä vastaavan henkilökohtaisen kehitysympäristön. Kehitysympäristö sisältää myös vaadittavat työkalut projektien koodikantojen tasonnostoon.

Versionnosto toteutetaan omassa versionhallinnan kehityshaarassa, jottei se sotke muuta rinnalla tapahtuvaa järjestelmän kehitystä. Toteutuksen päätteeksi projektien muutokset tullaan yhdistämään osaksi pääkehityshaaroja. Erillinen kehityshaara tarjoaa joustavamman muutostenvientipolitiikan, sillä muutoshistoria voidaan tarvittaessa kirjoittaa uudelleen sen siistimiseksi ennen lopullista yhdistämistä, kuten meneteltiin tämän versionnoston yhteydessä. Historian uudelleenkirjoittaminen on erityisen hyödyllistä versionnostoon liittyvän kehitystyön ”kokeellisesta” luonteesta johtuen.

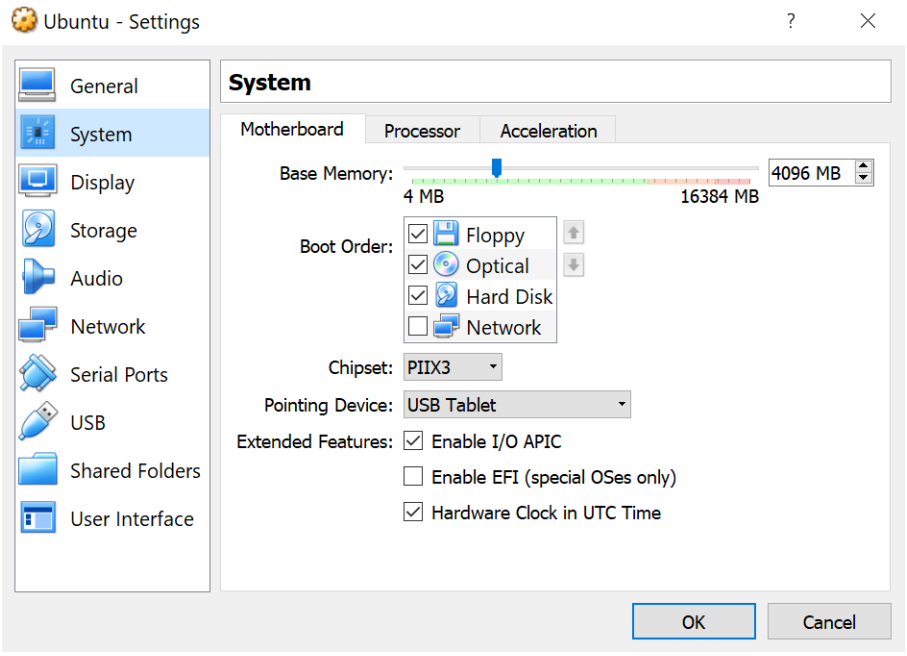
3.2 Kehitysympäristön asennus

Kehitysympäristö perustettiin virtuaalikoneessa suoritettavaan Ubuntu Linuxiin, sillä osana Linux-migraatiota kohdeympäristö tulee olemaan Linux-pohjainen. Virtualisoitu kehitysympäristö tarjoaa perustamisen vaivattomuutta ja joustavuutta tehokkuuden kustannuksella.

Seuraavissa alaluvuissa kuvataan tarkemmin vaiheet, kuinka virtualisoitu kehitysympäristö perustettiin tämän versionnoston yhteydessä. Lähtökohtaisesti työkalujen erillisiä asennusvaiheita ei kuvata, jos työkaluun kuuluu valmiiksi ohjeistettu asennusvelho.

3.2.1 Ubuntu Linux -käyttöjärjestelmä

Ubuntu Linux -jakelun uusin versio on ladattavissa Ubuntun viralliselta verkkosivulta ISO-muotoisena levykuvana. Käyttöjärjestelmä voidaan asentaa esimerkiksi **VirtualBox** -alustalla suoritettavaan virtuaalikoneeseen. Virtuaalikoneelle kannattaa varata tarpeeksi muistia ja ytimiä, jotta ohjelmat suoriutuvat vakaasti. Kuvassa 14 näkyy toteutuksen yhteydessä käytetyt virtuaalikoneen järjestelmäasetukset.

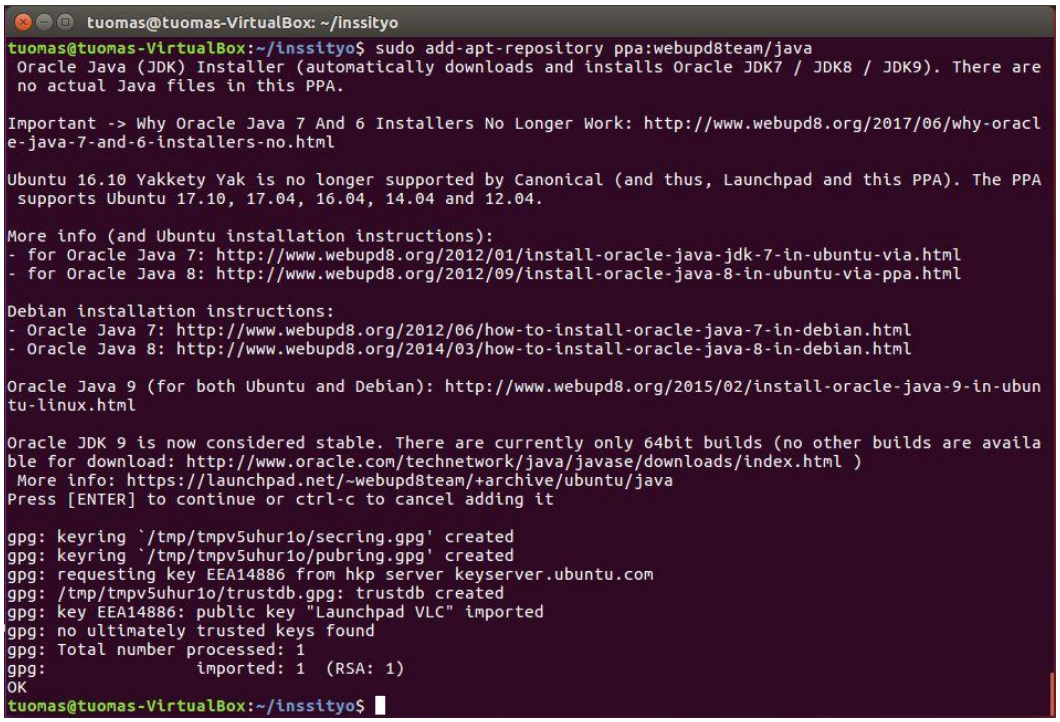


Kuva 14. Virtuaalikoneelle on varattu 4096 megatavua RAM-muistia.

3.2.2 Java SE Development Kit 8

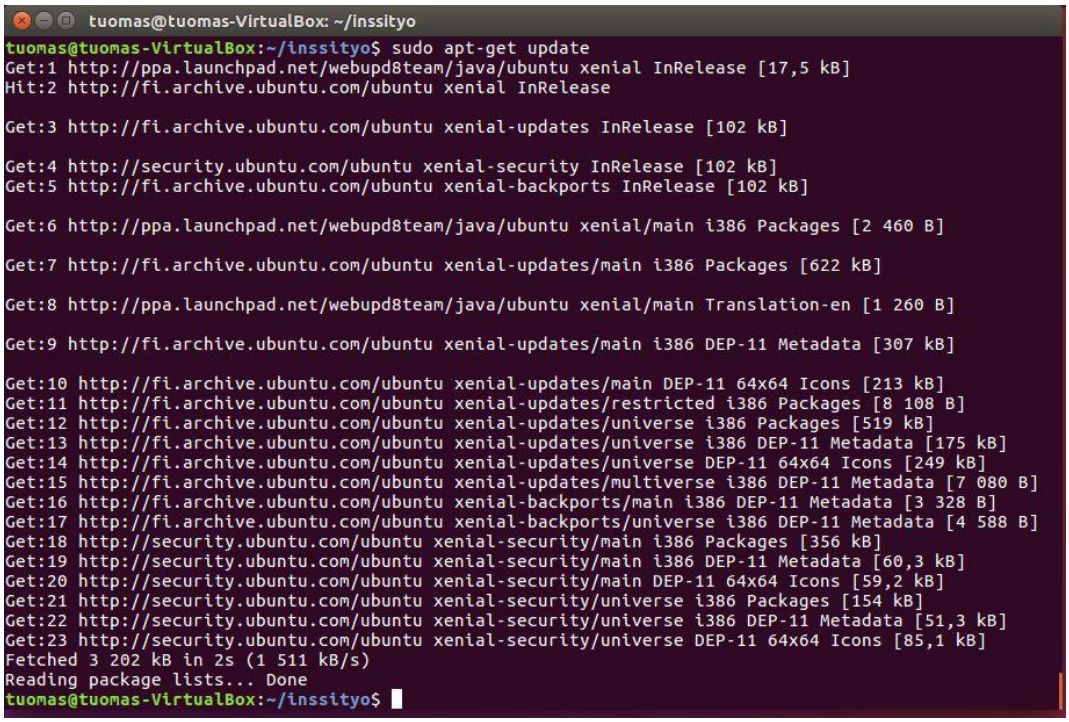
Kehitysympäristöön asennetaan Java 8 -kehitykseen vaadittavat kehittäjän työkalut. **JDK 1.8** voidaan asentaa Ubuntu-käyttöjärjestelmään käyttäen jakeluun sisältyvää komentorivipohjaista APT-paketinhallintaohjelmaa.

Lisätään Kuvan 15 mukaisesti APT-ohjelmalle ensin erityinen pakettivarasto, josta Oraclen JDK 1.8 -toteutus on tämän raportin kirjoittamisen hetkellä saatavilla.



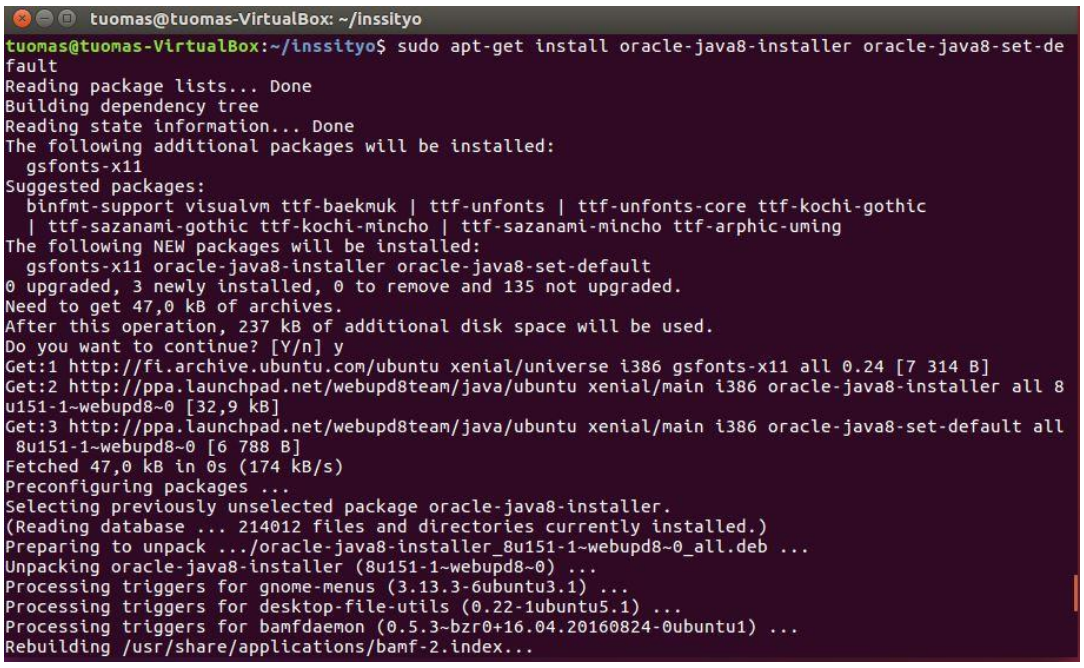
Kuva 15. Uuden pakettivaraston lisääminen.

Päivitetään ennen JDK-työkalujen asennusta APT-ohjelman käyttämät pakettivarastot ajantasaisiksi kuvan 16 mukaisesti.



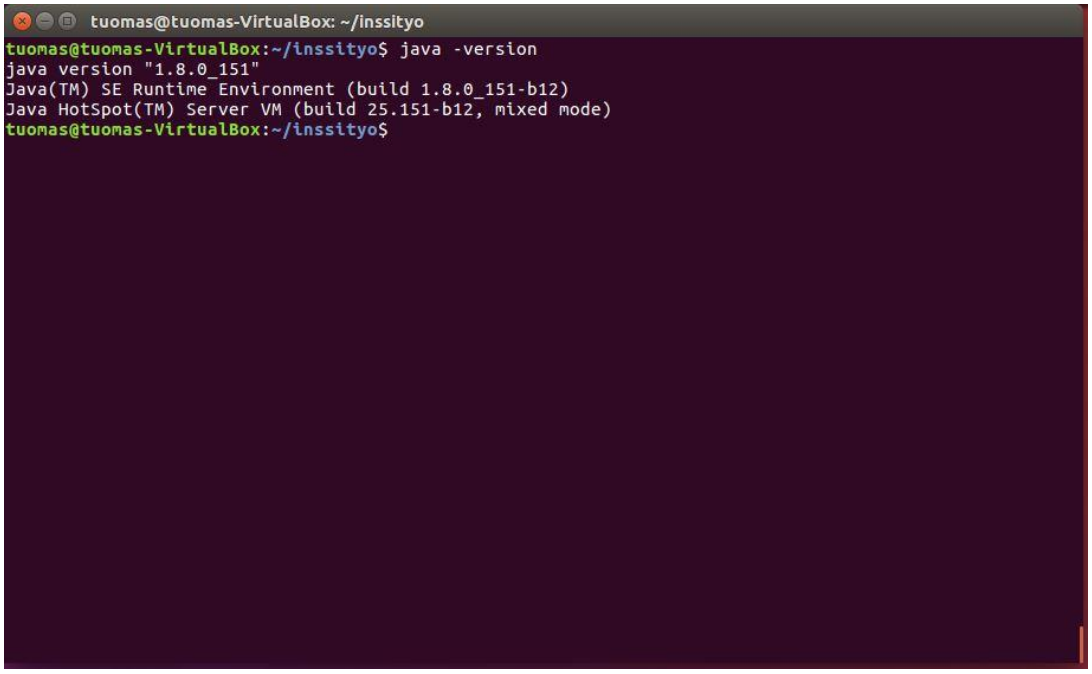
Kuva 16. Pakettivarastojen päivitys.

Asennetaan Oraclen JDK 1.8 -kehitystyökalut kuvan 17 mukaisesti.



Kuva 17. JDK 1.8 -työkalujen asennus.

Varmistetaan vielä asennuksen onnistuminen tulostamalla käytössä oleva Java-versio komentoriville kuvan 18 mukaisesti. Käytössä tulee olla versio 1.8.

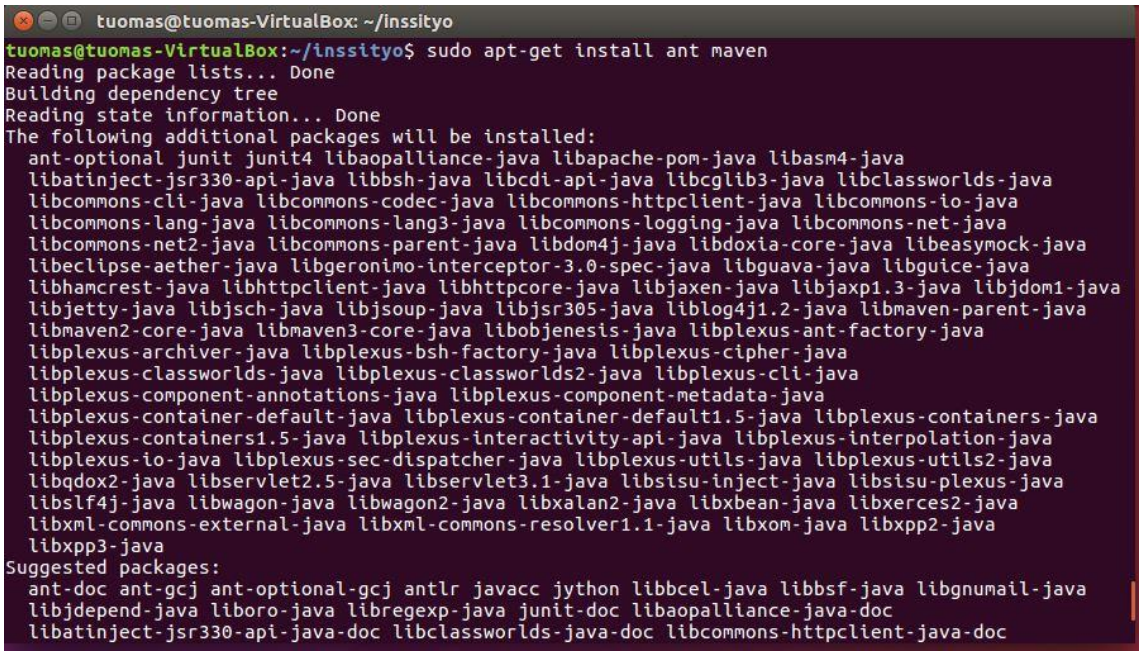


Kuva 18. Java-version tarkastaminen.

3.2.3 Ant ja Maven koontityökalut

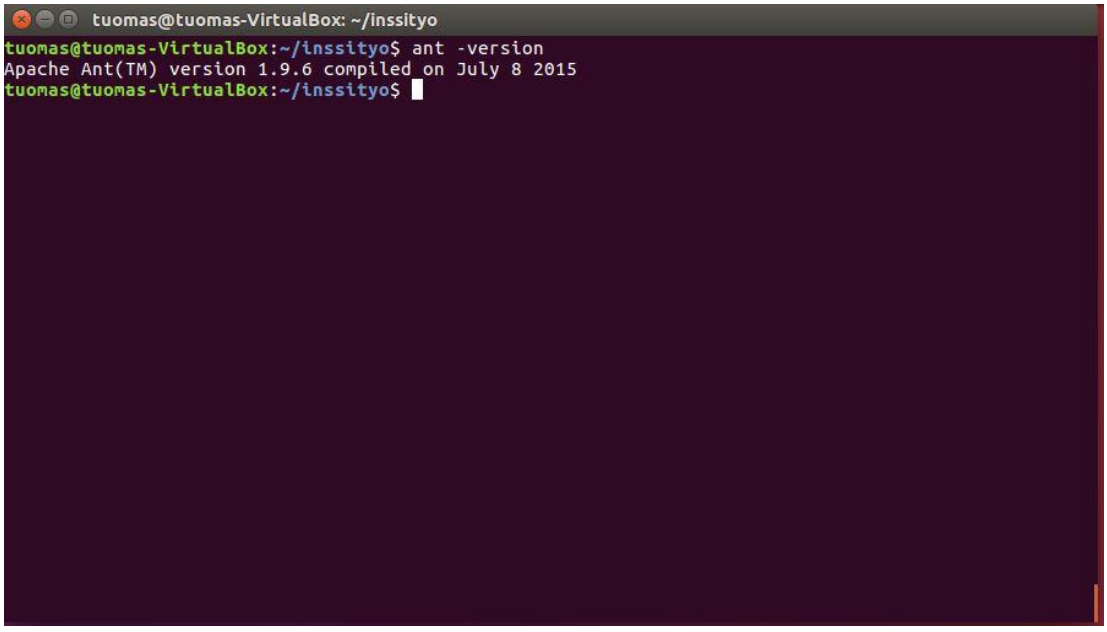
Ant ja **Maven** ovat projekteissa käytettyjä koontityökaluja, eli työkaluja, joilla projektin lähdekoodista ja metatiedoista koostetaan lopullinen sovelluspaketti (EAR, WAR, JAR), kirjastopaketti (JAR) tai muu artefakti. Molemmat työkalut ovat Apache-organisaation kehittämiä. Maven on työkaluista uudempi sisältäen myös riippuvuuksien automaattisen hallinnan ja muita edistyksellisiä toimintoja. Tyypillisesti projektin koonti käsittää lähdekoodin kääntämisen tavukoodiksi, joka yhdessä resurssi- ja konfiguraatiotiedostojen kanssa pakataan standardinmukaiseen sovelluspakettiin. Käytännössä koonti voi kuitenkin käsittää mitä tahansa ohjelmallisia toimintoja. Koonti kuvataan projektin sisältämällä koontityökalun tulkitsemalla koontiskriptillä.

Asennetaan Ant- ja Maven-koontityökalut kehitysympäristöön kuvan 19 mukaisesti.



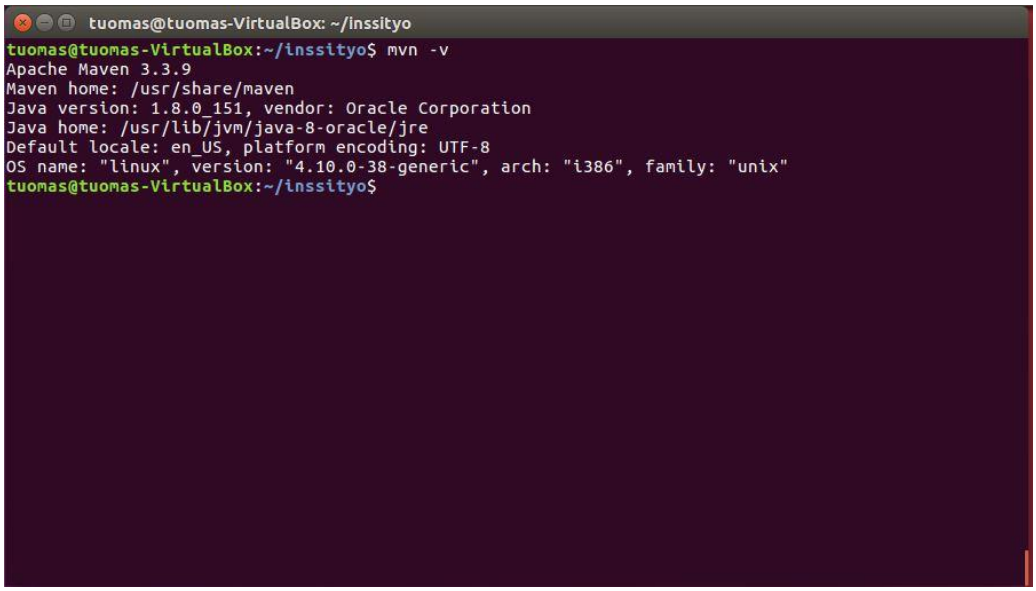
Kuva 19. Ant- ja Maven-työkalujen asennus

Varmistetaan asennuksien onnistuminen tulostamalla työkalujen versiot komentoriville kuvan 20 mukaisesti. Versioiden ei tarvitse täysin vastata kuvassa esitettyjä versioita. Uusin versio on riittävä.



Kuva 20. Ant-version tarkistaminen.

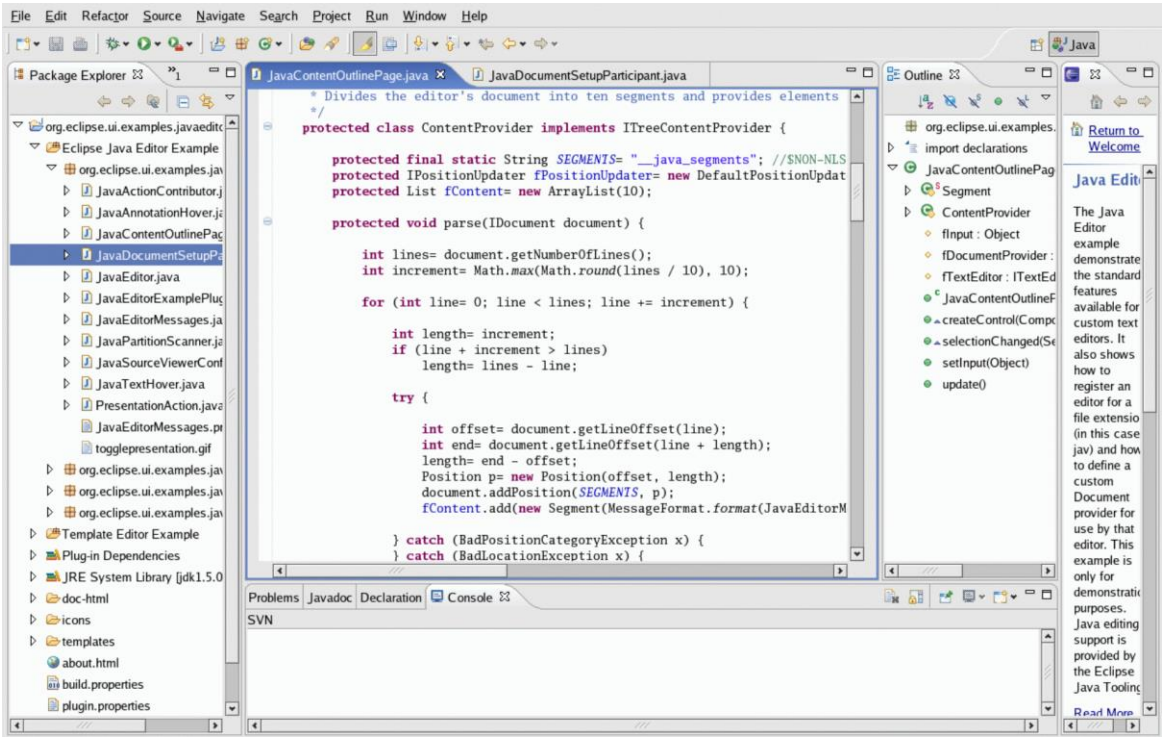
Tarkistetaan vielä käytössä oleva Maven-versio kuvan 21 mukaisesti.



Kuva 21. Maven-version tarkistaminen.

3.2.4 Eclipse

Eclipse on yleisimmin Java-kehitykseen käytetty integroitu ohjelmistokehitin (IDE). Eclipse tarjoaa muun muassa tekstieditorin, debuggerin, runsaan joukon laajennoksia sekä useita kehittäjää avustavia toimintoja lähdekoodin kanssa työskentelyyn. Tämän versionnoston yhteydessä käytettiin **Eclipse Mars 2** -versiota ja Eclipsen toiminnallisuutta laajentavaa **OEPE**-laajennuspakettia. Eclipse on ladattavissa Eclipse Foundationin virallisilta kotisivuilta. OEPE-laajennuspaketti sisältää joukon laajennuksia WebLogic-sovelluspalvelimen integroimiseksi Eclipse-ohjelmistokehittimeen. Kuva 22 esittää Eclipse-ohjelmistokehittimen editointinäkymää.



Kuva 22. Eclipse IDE (kuvituskuva).

3.2.5 Oracle WebLogic Server 12.2.1.2

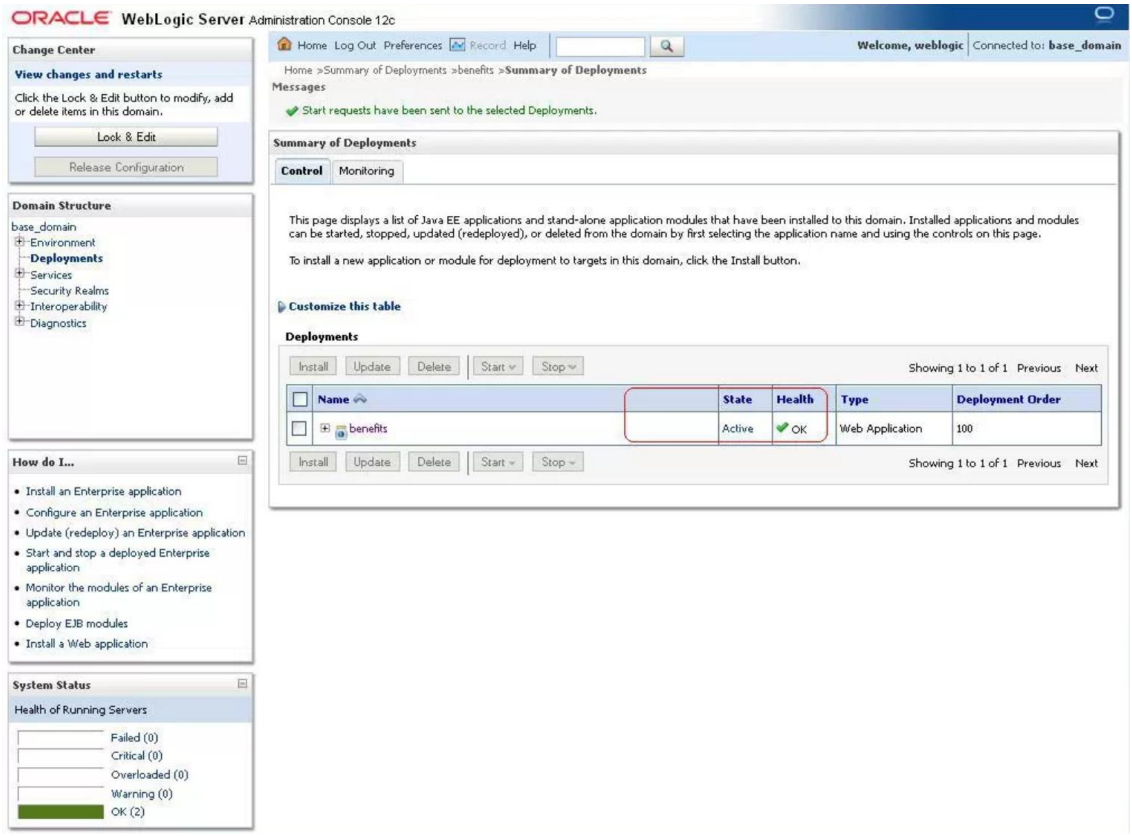
Uudessa kohdeympäristössä järjestelmiä ajetaan **WLS 12c** -sovelluspalvelimella, joten versionnostoon liittyvä kehitystyö tehdään luonnollisesti WLS 12c -palvelinta vasten.

Asennetaan WLS 12c -kehitysympäristöön. WLS-palvelimen asennuspaketti on ladattavissa Oraclen verkkosivuilta erityistä lisenssiä vastaan, sillä kyseessä on kaupallinen tuote. Asennuspaketti on JAR-muotoinen arkisto, jonka asennus käynnistetään komentoriviltä seuraavalla komennolla.

```
$ java -jar fmw_12.2.1.2.0_wls.jar
```

Komento käynnistää graafisen asennusvelhon WLS-palvelimen asentamiseksi ja vaadittavien konfiguraatioiden asettamiseksi. Asennusvelho tarjoaa yksikäsitteisen ohjeistuksen, joten asennuksen vaihteita ei tässä raportissa kuvata yksityiskohtaisemmin. Oraclen verkkosivuilta löytyy kattava dokumentaatio tuotteen käyttöön.

Palvelimen asennuksen ja käynnistämisen jälkeen ylläpitokonsolin tulisi löytyä osoitteesta http://localhost:7001/console/. Käytettävä portti määritellään asennuksen yhteydessä. Kuva 23 esittää ylläpitokonsolia toiminnassa.



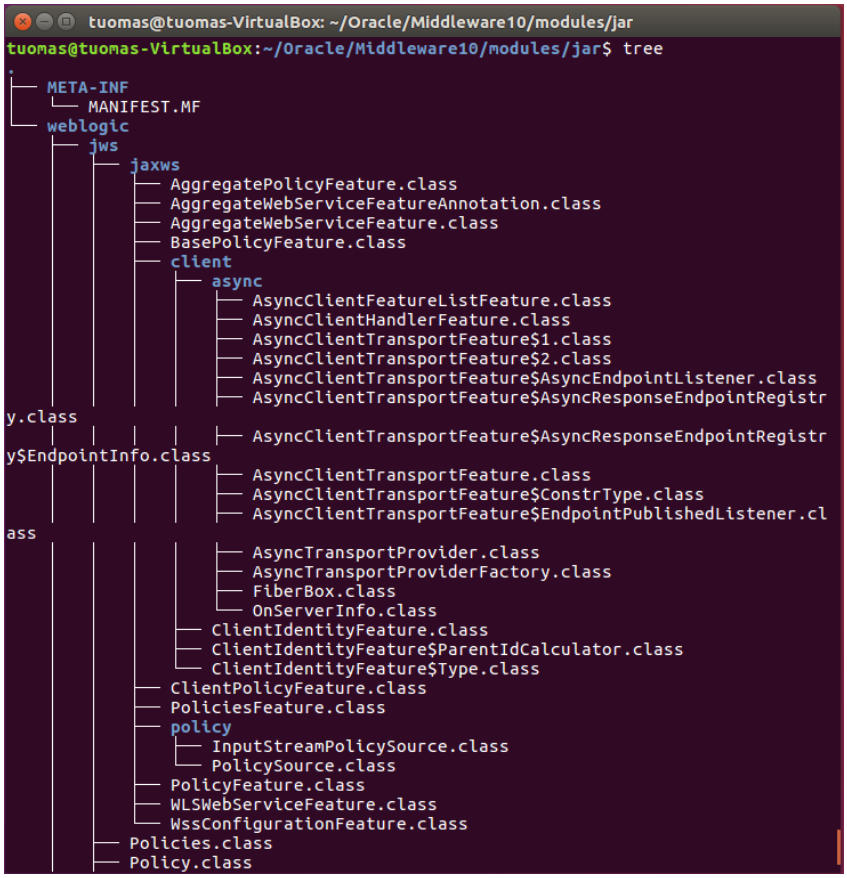
Kuva 23. WLS-palvelimen ylläpitokonsoli (kuvituskuva).

Ylläpitokonsolissa luodaan ja konfiguroidaan sovellusten käyttämät tietokantayhteydet, sanomajono, tietoturvapolitiikka ja muut infrastruktuuripalvelut. Nämä toimenpiteet suoritettiin osana kehitysympäristön pystytystä, mutta tässä raportissa ei vaiheita kuvata tarkemmin.

3.2.6 Apuohjelmat

Versionnoston toteutuksessa käytössä oli lukuisia komentorivipohjaisia UNIX-työkaluja. Jatkuvassa käytössä oli muun muassa **tree**-ohjelma hakemistorakenteiden nopeaan listaamiseen ja **vim**-tekstieditori tekstitiedostojen käsittelyyn. Apuohjelmat ovat vapaasti valittavissa kehittäjän mieltymysten mukaan. Erityisesti **find**- ja **grep**-ohjelmat todettiin

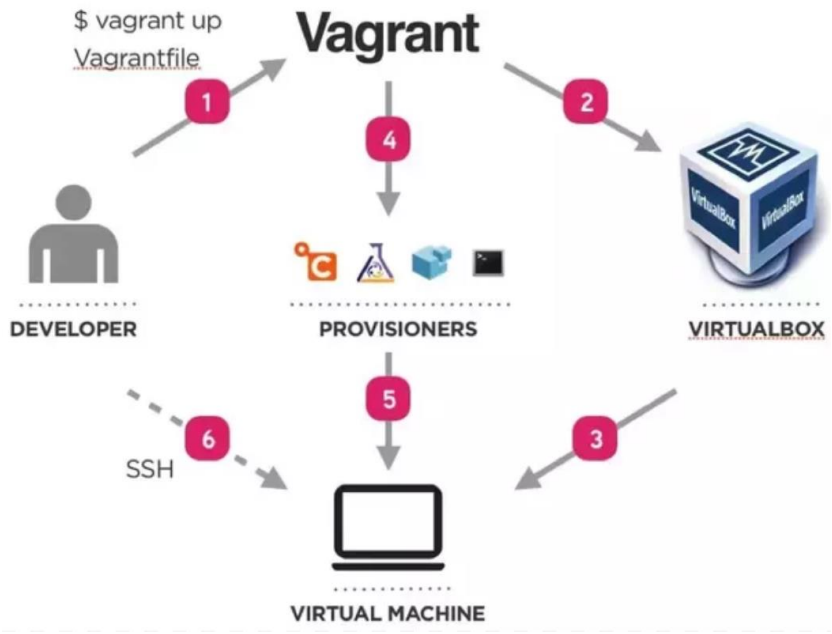
äärimmäisen hyödyllisiksi tämän versionnoston yhteydessä, erinäisten resurssien ja tietosisältöjen etsimiseksi suuresta koodikannasta. Kuva 24 havainnollistaa tree-ohjelman antamaa tulostetta.



Kuva 24. Esimerkki tree-ohjelman tulosteesta ja JAR-arkiston sisällöstä.

3.2.7 Ympäristön perustamisen automatisointi

Kun useampi henkilö on vastuussa versionnoston toteutuksesta, suositeltavaa olisi osana kehitysympäristön perustamista automatisoida kehitysympäristön perustaminen. Automatiikka mahdollistaa ympäristön virheettömän toistettavuuden. Käytännössä muut kehittäjät voivat luoda täysin vastaavan ympäristön ajasta tai paikasta riippumatta yhdellä napin painalluksella. Vagrant on suosittu työkalu virtuaalikoneen perustamisen, eli provisioinnin automatisoimiseksi. [2.] Kuva 25 havainnollistaa Vagrant-työkalun toimintaperiaatetta.



Kuva 25. Vagrant-provisiontityökalu.

Tämän versionnoston yhteydessä työskenteli kuitenkin vain yksi henkilö (allekirjoittanut), eikä ympäristön perustamista automatisoitu aikataulullisten seikkojen vuoksi.

3.3 Lähdekoodien tasonnosto

Projektien koodikantoihin tehtiin tarvittavat sovelluskoodiin ja konfiguraatioihin kohdistuvat muutokset Java 1.8 -yhteensopivuuden, sekä WLS 12c -yhteensopivuuden, saavuttamiseksi. Projektit pyrittiin käsittelemään kokonaisuuksina, mutta projektien välisistä riippuvuuksista johtuen oli toteutuksen yhteydessä toisinaan hypittävä projektien välillä edes takaisin. Seuraavissa alaluvuissa on kuvattu tasonnoston prosessi jokaiselle projektille erikseen. Kuvaus ei sisällä kaikkia toteutuksellisia yksityiskohtia, vaan tuo esiin perusteellisimmat muutokset sekä suurimmat kompastuskivet ratkaisuineen.

3.3.1 Käyttöliittymä

Ensimmäisenä toimenpiteenä asetettiin projektin koontia ohjaaviin `build.properties` ja `project.properties` -tiedostoihin käytettävän JDK-version määräävien muuttujien arvoksi 1.8.

```
compile.source = 1.8
compile.target = 1.8
```

Koostetun EAR-paketin asennus WLS-palvelimelle ei kuitenkaan onnistut, vaan asennusvaiheessa WLS heitti seuraavan ajonaikaisen poikkeuksen.

```
java.lang.ClassNotFoundException: org.apache.commons.pool2.BasePooledObjectFactory
```

Poikkeus johtuu puuttuvasta luokasta. Puuttuva luokka on *tekninen syy* asennuksen epäonnistumiselle, mutta todellisuudessa poikkeus on oire paljon perusteellisemmasta ongelmasta, joka juontaa juureensa Java EE -spesifikaation puutteellisuudesta. Spesifikaatio ei nimittäin täydellisesti määritä, kuinka sovelluspalvelintuotteen tulee sovelluspaketin asennusvaihe teknisesti toteuttaa. [3.] Uuden WLS 12c -palvelimen tapa asentaa sovelluspaketti on hieman poikkeava aiempaan 11g -versioon nähden, mikä lopulta johtaa ajonaikaisen poikkeukseen.

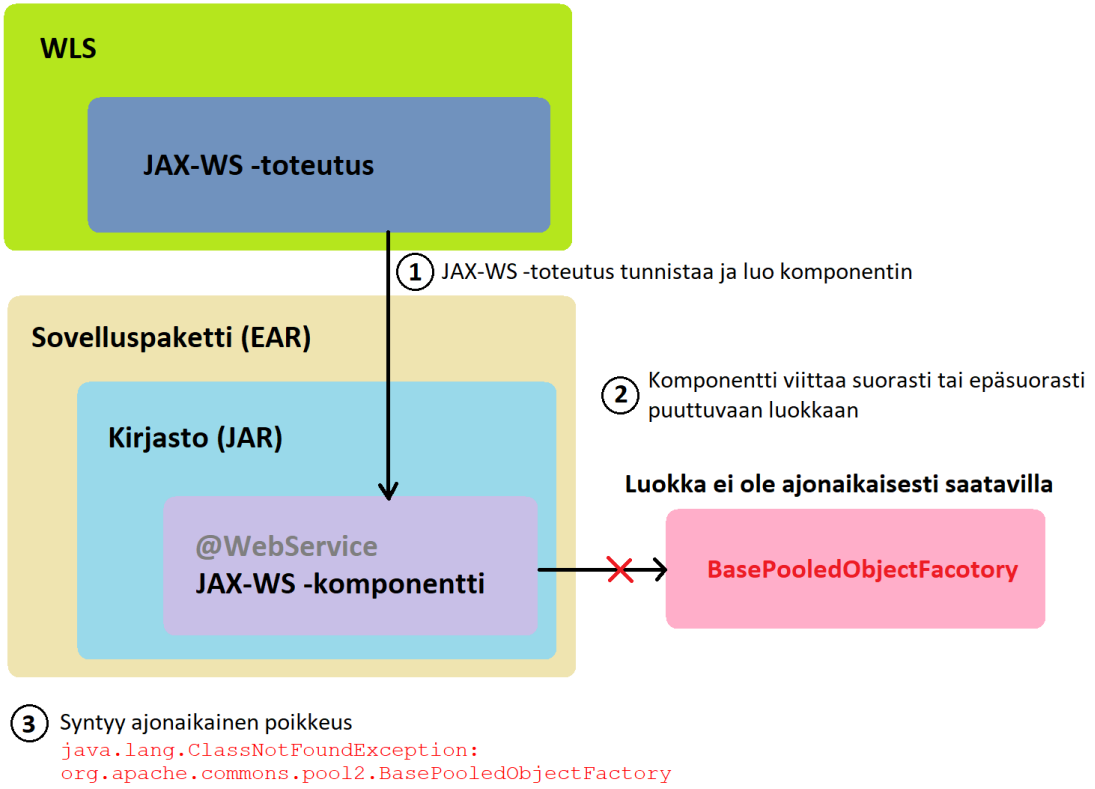
Spesifikaation rajallisuuden seurauksena voi pahimmassa tapauksessa syntyä vastaavanlaisia tilanteita, joissa ajonaikaista poikkeusta *ei* synny, vaan sovellus jatkaa toimintaansa ennalta-arvaamattomasti, jos sovelluspalvelimen toimintamekanismeja ei täysin tunneta, tai ne toimivat vastoin olettamusta. Versionnoston yhteydessä ilmenevien virheiden suurin haaste onkin asettaa virhetapaukset oikeaan kontekstiin, tunnistaa, mikä on oire ja mikä virhetilanteen todellinen lähde.

Tässä virhetilanteessa EAR-paketti sisälsi ongelmallisen integraatioprojektista tuodun kirjaston (JAR-arkiston), joka edelleen sisältää JAX-WS-komponenteiksi lukeutuvia luokkia. Kuvassa 26 on esimerkki JAX-WS-komponentiksi määritellystä luokasta.

```
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;


@WebService(serviceName = "HelloService")
public class HelloService {
    public HelloService() {
    }

    @WebMethod
    @WebResult( name = "responseMessage")
    public String sayHello(@WebParam(name = "message") String msg) {
        return "Hello back , got message: " +msg;
    }
}
```

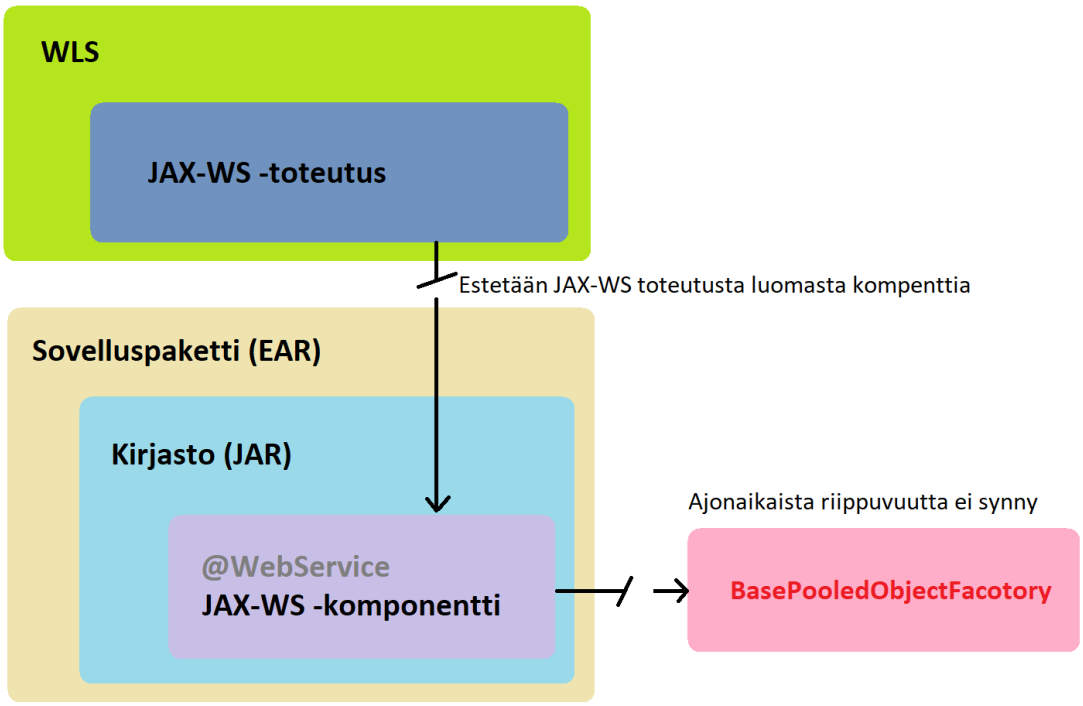
Kuva 26. JAX-WS-komponentti (kuvituskuva).

Kyseisten luokkien määrittämiä WS-palveluita ei ole tarkoitettu julkaistaviksi käyttöliittymäsovelluksessa, kuten ei tapahtunutkaan vanhemmalla WLS 11g -palvelimella sovellusta ajettaessa. Uuden WLS 12c -palvelimen hieman poikkeava tapa asentaa sovelluspaketti kuitenkin johti siihen, että asennusvaihe havaitsi nämä ongelmakirjaston sisältämät JAX-WS-komponentit, pyrkien julkaisemaan WS-palvelut WLS-palvelimen sisäistä JAX-WS-toteutusta käyttäen. Jokin näistä JAX-WS-komponenteiksi lukeutuvista luokista viittaa käännöksen aikaiseen riippuvuuteen, jota ei ole ajonaikaisesti saatavilla, josta aiheutui virheilmoituksen kuvaama ajonaikainen poikkeus puuttuvasta luokasta. Kuva 27 havainnollistaa virhetilanteen syntyä.



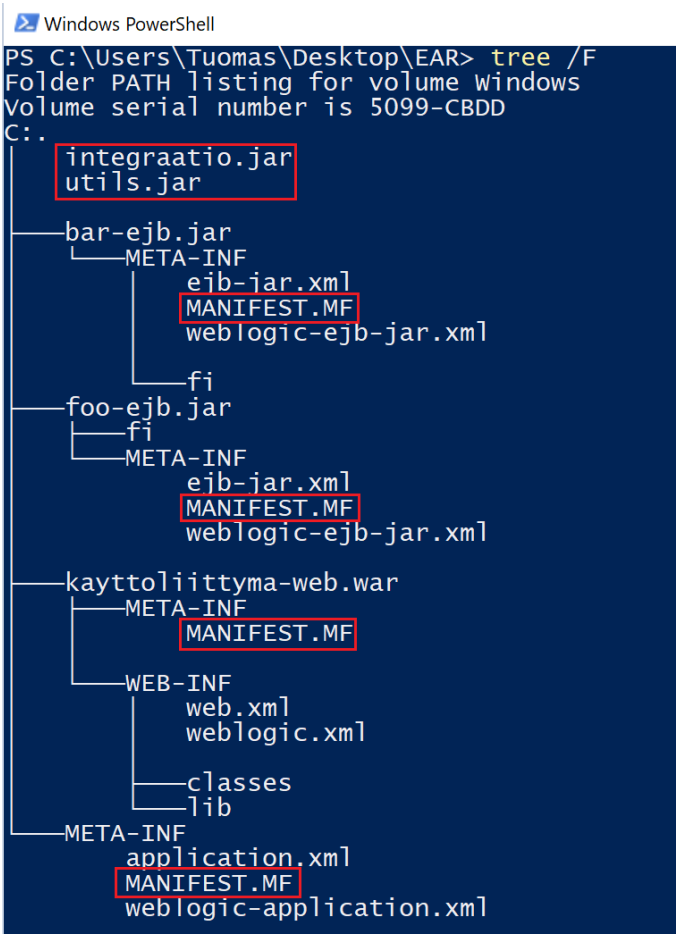
Kuva 27. Kirjasto sisältää JAX-WS-komponentin, jonka WLS-palvelin tunnistaa.

Ongelman ratkaisu ei ole lisätä poikkeuksen kuvaama puuttuva riippuvuus EAR-pakettiin, vaan korjata ongelman alkulähde, eli estää kirjaston sisältämien WS-palveluiden julkaiseminen, jolloin ajonaikainen riippuvuus luokkaan katoaa, eikä poikkeusta synny. Käytännön syy, miksi kirjasto sisältää JAX-WS-komponenteiksi lukeutuvia luokkia on se, että lähes koko integraatioprojektin koodi (sen tarjoamat WS-palvelut mukaan lukien) pakataan yhteen kirjastoon käytännöllisyyden vuoksi. Vahvoista sidoksista johtuen näitä JAX-WS-komponentteja ei voi helposti jättää pois kirjaston paketoinnista, joten ainoaksi vaihtoehdoksi jää tavalla tai toisella ohjeistaa WLS-palvelinta jättämään kyseiset komponentit kokonaan huomiotta. Kuva 28 havainnollistaa ongelman ratkaisua.



Kuva 28. Estetään JAX-WS-toteutusta luomasta kirjaston sisältämää palvelua.

JSR-109-spesifikaation mukaan sovelluspalvelin saa tulkita JAX-WS-komponenteiksi vain ne luokat, jotka muiden ehtojen täyttyessä *on paketoitu osaksi WAR- tai EJB-moduulia*. [4.] Tässä tapauksessa luokat oli paketoitu kirjastoon, mutta kirjasto tuotiin muiden sovelluspaketin moduulien käyttöön tekniikalla, jonka vaikutuksia Java EE -spesifikaatio ei määrittele lainkaan. Kirjasto sijaitsi EAR-paketin juuressa, ja jokainen kirjastoa käyttävä WAR- tai EJB-moduuli viittasi siihen moduulin manifestitiedostoon lisätyllä **Class-Path**-viittauksella. Kyseessä on JAR-spesifikaation (Java SE) määrittämä tapa laajentaa käytössä olevaa luokkapolkua. [5.] Kyseisellä metodilla kirjaston sisältämät luokat tuotiin moduulikohtaisen luokanlataajan saataville, mutta uudella WLS-palvelimella seuraukset olivat ennalta arvaamattomia. Kuvissa 29 ja 30 on havainnollistettu kirjastojen sijaintia EAR-paketissa ja manifestitiedostojen sisältämiä kirjastoviittauksia. Kirjastopakettien ja manifestitiedostojen sijainti on kehystetty kuvissa.



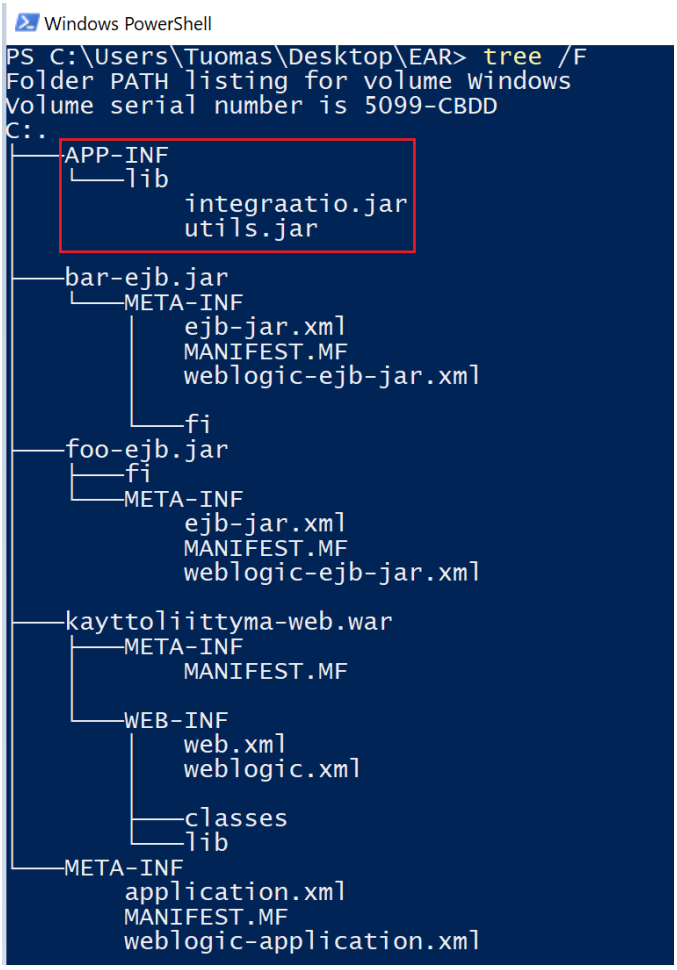
Kuva 29. EAR-sovelluspaketin rakenne. Kirjastot ja manifestitiedostot on kehystetty (kuvituskuva).

```
1 Manifest-Version: 1.0
2 Class-Path:
3  foo-ejb.jar
4  bar-ejb.jar
5  utils.jar
6  integraatio.jar
7
8
```

Kuva 30. Manifestitiedostossa lueteltuja kirjastoviittauksia (kuvituskuva).

Paketin asennusvaiheessa WLS 12c tulkitsi, että Class-Path-viittauksen kautta tuodun kirjaston luokat ovat osa kirjastoon viittaavaa moduulia. Tähän tulkintaan perustuen WLS-palvelimen tarjoama JAX-WS-toteutus pyrki luomaan palveluiden instanssit spesifikaation mukaisesti. Virhetilanteen synty oli siis seurausta muuttuneesta tuotekohtaisesta ratkaisusta.

Ongelma korjattiin muuttamalla sovelluksen paketointia siten, että kaikki kirjastoriippuvuudet paketoidaan osaksi EAR-pakettia WLS-spesifiseen *APP-INF/lib*-hakemistoon. Määritelmällisesti WLS tulkitsee kaikki kyseisen hakemiston sisältämät JAR-arkistot kirjastoiksi, tuo ne automaattisesti kaikkien moduulien saataville, ja jättää kirjastojen sisältämän Java EE -konfiguraation kokonaan huomiotta. Tämän muutoksen jälkeen ongelmallisen kirjaston sisältämiä luokkia ei enää tulkittu JAX-WS-komponenteiksi, eikä asennuksen yhteydessä aiheutunut ajonaikaista virhettä puuttuvasta luokasta, vaan sovellus asentui onnistuneesti. Muutoksen yhteydessä poistettiin kaikista manifestitiedostoista viittaukset kirjastoihin kuin myös EJB-moduuleihin (WLS-palvelimen EJB-säiliö tarjoaa kaikki EJB-komponentit web-säiliölle automaattisesti). Kuvassa 31 on kehystetty kirjastojen uusi sijainti.



Kuva 31. Kirjastot siirrettiin APP-INF/lib-hakemistoon, ja manifestiviittaukset kirjastoihin poistettiin.

Muina toimenpiteinä projektiin tuotiin toisista projekteista vaadittavat uudelleenkäännetyt kirjastoriippuvuudet sitä mukaan, kun projektien koodikantojen tasonnosto valmistui ja kirjastopaketit oli koostettu ja valmiita tuotaviksi.

Toiminnallisia virheitä ei onnistuneen asennuksen jälkeen kehitysympäristössä ilmennyt, mutta viimeistään testaajien suorittamassa regressiotestissä mahdollisesti piilevät virheet tulevat ilmi toteuttajan korjattavaksi.

3.3.2 OpenFrame

OpenFrame-sovelluskehyksen lähdekoodin tasonnosto aloitettiin asettamalla projektin `build.xml`-koontiskriptiin käytettäväksi Java-versioksi 1.8. Sovelluspaketin koonti ei kuitenkaan onnistunut, vaan syntyi seuraava käännöksenaikainen virhe:

```
[javac] /home/tutoivon/repository/ofcoresdk/tmp/build/sup-
log/fi/*****/support/tech/backend/sql/SupPreparedState-
ment.java:49: error: SupPreparedStatement is not abstract
and does not override abstract method isCloseOnCompletion()
in Statement

[javac] public class SupPreparedStatement implements Pre-
paredStatement {   [javac]        ^
```

JDK 1.8 -version myötä JDBC-rajapintoihin on lisätty uusia metodeja. OpenFrame tarjoaa omat toteutuksensa kyseisille rajapinnoille. Koska Javan syntaksi vaatii, että konkreettinen luokka toteuttaa rajapintansa kaikki metodit, nämä muuttuneet rajapinnat toteuttaviin luokkiin oli lisättävä (tyhjät) toteutukset uusille metodeille, jotta käännös JDK 1.8 -versiolla olisi lähtökohtaisesti mahdollinen. Muuttuneet JDBC-rajapinnat, jotka OF toteuttaa, olivat

- `java.sql.CallableStatement`
- `java.sql.PreparedStatement`
- `java.sql.Connection`.

Lisäämällä vaadittavat metodit yllämainitut rajapinnat toteuttaviin luokkiin, projektin koonti yksikkötesteineen onnistui, ja kootut kirjastopaketit vietiin käyttöliittymäprojektiin.

OF-sovelluskehyksen toteutus sisälsi kuitenkin erään varteenotettavan ongelman. OF tarjoaa toteutuksen Java SE:n määrittelemälle `Comparator`-rajapinnalle. Määritelmällisesti rajapinnan toteutus tutkii kahden olion keskinäistä suuruusjärjestystä palauttaen vertailutulosta vastaavan kokonaisluvun. Ohessa on suora lainaus Oraclen Java API -dokumentaatiosta (JDK 8).

> A comparison function, which imposes a total ordering on some collection of objects. Comparators can be passed to a sort method (such as Collections.sort or Arrays.sort) to allow precise control over the sort order. Comparators can also be used to control the order of certain data structures (such as sorted sets or sorted maps), or to provide an ordering for collections of objects that don't have a natural ordering. [6.]

Sopimuksen mukaan rajapinnan toteutuksen tulee palauttaa negatiivinen kokonaisluku, nolla, tai positiivinen kokonaisluku olioiden keskinäisen suurusjärjestyksen tai yhtäsuuruuden perusteella, kuten ilmenee `compare`-metodin kuvauksesta (lainaus samasta dokumentaatiosta).

> Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second. [6.]

Jää kuitenkin täysin toteutuksen päätettäväksi, *milloin* oliot tulkitaan yhdensuuruisiksi. OF-kehyksen tarjoama toteutus perustuukin varsin erikoiseen näkemykseen: toteutuksen mukaan oliot eivät nimittäin *koskaan* voi olla yhdensuuruisia. Tämä on arkkitehtuurisidonnainen ratkaisu, jonka toimintaperiaatteen kuvassa 32 esitettyyn koodiesimerkkiin liittyvä kommentti selventää. Kommentin mukaan olioiden ollessa yhdensuuruiset, sijoittuu vertailtava olio lajiteltavassa listassa yksikäsitteisesti verrokin jälkeen. OF-kehyksen puutteellisen dokumentaation vuoksi syy edellä mainitulle toiminnallisuudelle on kuitenkin ajan saatossa jäänyt hämärän peittoon. Selvyyden vuoksi mainittakoon, että `compare`-metodin kutsuma `compare`-metodi on metodin ylikuormitettu toteutus `KV`-parametrityypeille. Ylikuormitettu toteutus tutkii olioiden yhtäsuuruutta perustuen `KV`-instanssin erinäisiin attribuutteihin.

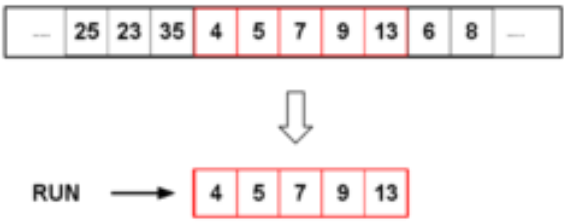
```
public int compare(Object map1, Object map2) {
    KV m1 = new KeysAndValues((Map) map1);
    KV m2 = new KeysAndValues((Map) map2);
    int comp = compare(m1, m2);
    if (comp == 0)
        return 1; // append item to collection if all orders are equal
    return comp;
}
```

Kuva 32. OpenFrame-sovelluskehyksen Comparator-toteutus.

Siirryttäessä Java 6 -tasoisesta ajoympäristöstä Java 8 -tasoiseen ajoympäristöön, OF-kehyksen erikoisesta `Comparator`-toteutuksesta johtuen syntyy riski taulukko- ja listarakenteita lajittelevien toiminnallisuuksien rikkoutumiseen. JDK 7 -version myötä taulukoiden lajitteluun käytetyn `java.util.Arrays.sort`-metodin (ja epäsuorasti `java.util.Collections.sort`-metodin) toteutus on muuttunut ratkaisevasti. Aiemman lomituslajittelualgoritmin sijasta uusi toteutus perustuu tehokkaampaan *timsort*-nimiseen hybridialgoritmiin. [7.]

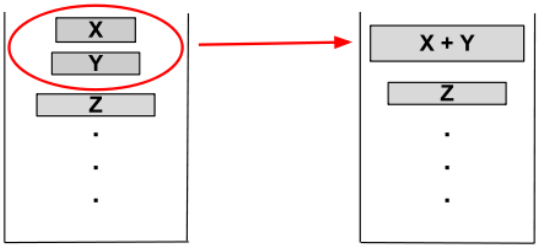
Seuraavaksi kuvataan timsort-algoritmin toimintaperiaatteet niin yksityiskohtaisella tasolla kuin OF-sovelluskehyksen `Comparator`-toteutukseen liittyvän uhan ymmärtämiseksi on tiedettävä. Kuvaus ei ole kuitenkaan kaiken kattava jättäen pois algoritmin toiminnan kannalta oleellisia toteutuksellisia yksityiskohtia.

Timsort-algoritmi on lomituslajittelualgoritmin (merge sort) ja lisäyslajittelualgoritmin (insertion sort) johdannainen ja yhdistelmä. [8.] Algoritmi etsii lajiteltavasta aineistosta valmiiksi nousevassa tai laskevassa järjestyksessä olevat, vähintään kahden alkion muodostamat sekvenssit, täydentäen niitä tarvittaessa sekvenssin vähimmäiskoon täyttymiseksi. Algoritmi pitää yllä viittauksia löytämiinsä sekvensseihin pinorakenteessa. Algoritmin toimintaperiaate hyödyntää tosimaailman aineistossa valmiiksi esiintyvää luonnollista järjestystä. Kuva 33 havainnollistaa aineistosta löytyvää sekvenssiä.



Kuva 33. Timsort-algoritmin löytämä sekvenssi.

Rinnakkain sekvenssien etsimisen ohella algoritmi suorittaa lomituslajittelun pinoon vietyjen sekvenssien kesken. Prosessia jatketaan, kunnes koko aineisto on järjestetty. Lomituslajittelu ei ole täysin suoraviivaista, vaan sisältää tiettyjä optimointeja ja erikoistoimintoja tiettyjen reunaehtojen täyttyessä säilyttäen lajittelun *vakauden* ja *tasapainon*. Sekvenssien lomituslajitteluvaiheessa hyödynnetään muun muassa binäärihakua tehokkuuden optimoimiseksi. Kuvassa 34 on kuvattu sekvenssien yhdistäminen.



Kuva 34. Sekvenssien yhdistäminen.

Eräänä tehokkuuden optimointimekanismina algoritmi voi toimia niin kutsutussa **laukkatilassa** (galloping mode). Laukkatilassa algoritmi pyrkii osittamaan sekvenssit edelleen alisekvensseihin ennen lomituslajittelua. Moodia ohjataan itseään säätelevällä kynnysarvolla (MIN_GALLOP). Kynnysarvo määräytyy yhdistettävien sekvenssien koon ja sisällön mukaan. Tilanteissa, joissa laukkatila ei tuo lisätehokkuutta, tai huonontaa sitä, tilaa ei käytetä.

JDK 7 -version `Arrays.sort`-toteutus *voi* heittää ajonaikaisen poikkeuksen, mikäli vertailuun käytetty `Comparator`-toteutus ei toteutua olioiden yhtäsuuruuden vertailua tim-

sort-algoritmin vaatimalla tavalla, toisin sanoen palauta nollaa olioiden ollessa yhtä suuret. Poikkeuksen syntyminen ei riipu pelkästään `Comparator`-rajapinnan vaillinaisesta toteutuksesta, vaan myös lajiteltavan aineistoin *koosta* ja *sisällöstä*. Lajittelualgoritmin mahdollisesti heittämä poikkeus on seuraava:

```
java.lang.IllegalArgumentException: Comparison method vio-
lates its general contract!
```

Poikkeus voi syntyä vain, jos `Comparator`-toteutus ei tutki olioiden yhtäsuuruutta oikein **ja lajittelualgoritmi päättää suorittaa lajittelun laukkatilassa**. Jälkimmäinen premissi voidaan todeta paikkaansa pitäväksi tutkimalla `Arrays.sort`-toteutuksen käyttämän `ComparableTimSort`-olion lähdekoodia. Toteutuksen `mergeLo`- ja `mergeHi` -metodit ovat ainoat metodit, joista kyseinen poikkeus voidaan heittää. Täsmällisemmin poikkeus heitetään vain sellaisissa ehdollisissa haaroissa, jossa toteutetaan laukkatilaan liittyvä toiminnallisuus. Osa toteutuksesta on kuvattu kuvassa 35.

```
            } while (count1 >= MIN_GALLOP | count2 >= MIN_GALLOP);
            if (minGallop < 0)
                minGallop = 0;
            minGallop += 2;  // Penalize for leaving gallop mode
        }  // End of "outer" loop
        this.minGallop = minGallop < 1 ? 1 : minGallop;  // Write back to field

        if (len1 == 1) {
            assert len2 > 0;
            System.arraycopy(a, cursor2, a, dest, len2);
            a[dest + len2] = tmp[cursor1]; //  Last elt of run 1 to end of merge
        } else if (len1 == 0) {
            throw new IllegalArgumentException(
                "Comparison method violates its general contract!");
        } else {
            assert len2 == 0;
            assert len1 > 1;
            System.arraycopy(tmp, cursor1, a, dest, len1);
        }
    }
```

Kuva 35. Lajittelualgoritmi voi heittää erityisen poikkeuksen vain laukkatilan yhteydessä. Koodi sijaitsee `ComparableTimSort.mergeLo`-metodissa.

Muuttuneesta lajittelualgoritmista juontuvia ajonaikaisia poikkeuksia ei versionnoston toteutuksen yhteydessä ilmennyt, mutta niitä ei voida poissulkea ongelman satunnaisesta luonteesta johtuen. Varotoimenpiteenä WLS-palvelinta suorittavalle Java-prosessille annetaan käynnistyksen yhteydessä oheinen JVM-argumentti:

```
-Djava.util.Arrays.useLegacyMergeSort=true
```

Argumentti pakottaa JVM-alustan käyttämään JDK 7 -versiota edeltäneeseen lomitusalgoritmiin perustunutta taulukkojen lajittelua kuvan 36 mukaisesti. `Comparator`-toteutuksen korjaaminen timsort-yhteensopivaksi olisi ollut vaihtoehtoinen, mutta työläämpi ja riskialttiimpi ratkaisu. Toteutukseen liittyy OF-sovelluskehyksen arkkitehtonisia sidoksia, ja sen muuttaminen voisi vastaavasti rikkoa kehyksen toiminnallisuuden.

```
public static void sort(Object[] a) {
    if (LegacyMergeSort.userRequested)
        legacyMergeSort(a);
    else
        ComparableTimSort.sort(a, 0, a.length, null, 0, 0);
}
```

Kuva 36. Arrays.sort -metodin toteutus ja timsort-algoritmin ohitus.

JDK 7 -version myötä rikkoutuneen lajittelualgoritmin ongelma on yleisesti tiedossa. Ohessa on lainaus Oraclen toimittamasta ongelman kuvauksesta. Ongelman kuvaus koskee `Comparator`-rajapinnan sijasta `Comparable`-rajapintaa, mutta ongelman juurisyy lienee sama.

> **Area**: API: Utilities
>
> **Synopsis**: Updated sort behavior for Arrays and Collections may throw an *IllegalArgumentException*
>
> **Description**: The sorting algorithm used by *java.util.Arrays.sort* and (indirectly) by *java.util.Collections.sort* has been replaced. The new sort implementation may throw an *IllegalArgumentException* if it detects a *Comparable* that violates the *Comparable* contract. The previous implementation silently ignored such a situation.
>
> If the previous behavior is desired, you can use the new system property, *java.util.Arrays.useLegacyMergeSort*, to restore previous mergesort behavior.
>
> **Nature of Incompatibility**: behavioral
>
> **RFE**: 6804124
>
> [9.]

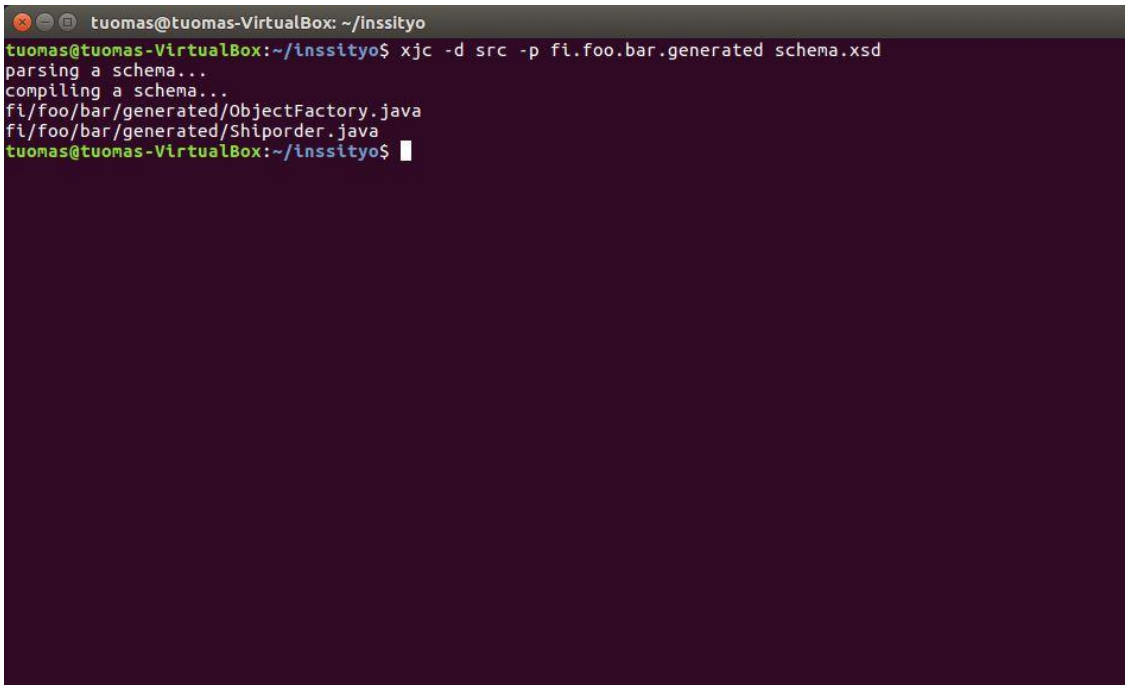
3.3.3 Integraatio

Integraatioprojektin lähdekoodin tasonnosto aloitettiin rutiininomaisesti asettamalla käytettävän JDK-version määräävät ohjaustiedot vastaamaan 1.8-versiota. Käytännössä muutettiin `compile.source` ja `compile.target`-nimiset muuttujat arvoon 1.8 projektin koontiskriptiin.

JDK 8 -version Java-kääntäjä kääntää luokat oletusarvoisesti käyttäen UTF-8 merkistöä. Jotkin WS-palveluiden skeemat (XSD-kuvaukset) sisälsivät kommentti-elementeissään UTF-8-yhteensopimatonta merkistöä (ääkkösiä). Tämän seurauksena käännös ei onnistunut, vaan syntyi seuraavanlaisia käännöksenaikaisia virheitä:

```
error: unmappable character for encoding UTF-8
```

UTF-8-epäyhteensopivat merkit skeemassa eivät itseisarvoisesti haittaa, mutta skeemasta generoidaan vastaavat JAXB-arkkitehtuurin mukaiset Java-luokat (niin kutsutut bindausoliot), ja näiden luokkien on oltava merkistöltään UTF-8-yhteensopivia. Tässä tapauksessa skeeman sisältämää UTF-8-yhteensopimatonta merkistöä päätyi generoituihin Java-luokkiin kommenteiksi. Korjauksena skeemojen dokumentaatio-osioista poistettiin UTF-8-yhteensopimattomat merkit, ja generoitiin XSD-kuvauksista skeemoja vastaavat luokat uudelleen käyttäen JDK 8:n tarjoamaa **xjc**-työkalua. Kuvassa 37 esitetään, kuinka xjc-työkalulla generoidaan Java-luokat XSD-skeemasta.



Kuva 37. xjc-työkalun käyttö (kuvituskuva).

Skeemojen korjaamisen jälkeen sähköliittymän ja integraatiomoduulien koonti ja asennus onnistui. Projektin sovelluspaketeista sähköliittymä oli suoraan toiminnallisessa kunnossa ilman muita vaadittavia muutoksia.

Kumpikaan integraatiomoduuli ei kuitenkaan onnistuneesta asennuksesta huolimatta toiminut, vaan heitti seuraavan ajonaikaisen poikkeuksen pyrkiessään lukemaan sanomajonoa:

```
wleventcontextimpl cannot be cast to java.lang.string
```

Integraation MDB-toteutus vaati yhden rivin, mutta kohtalaisen matalan tason muutoksen toteutukseensa. MDB:n toteuttavan luokan eräästä alustusmetodista oli muutettava seuraava lause

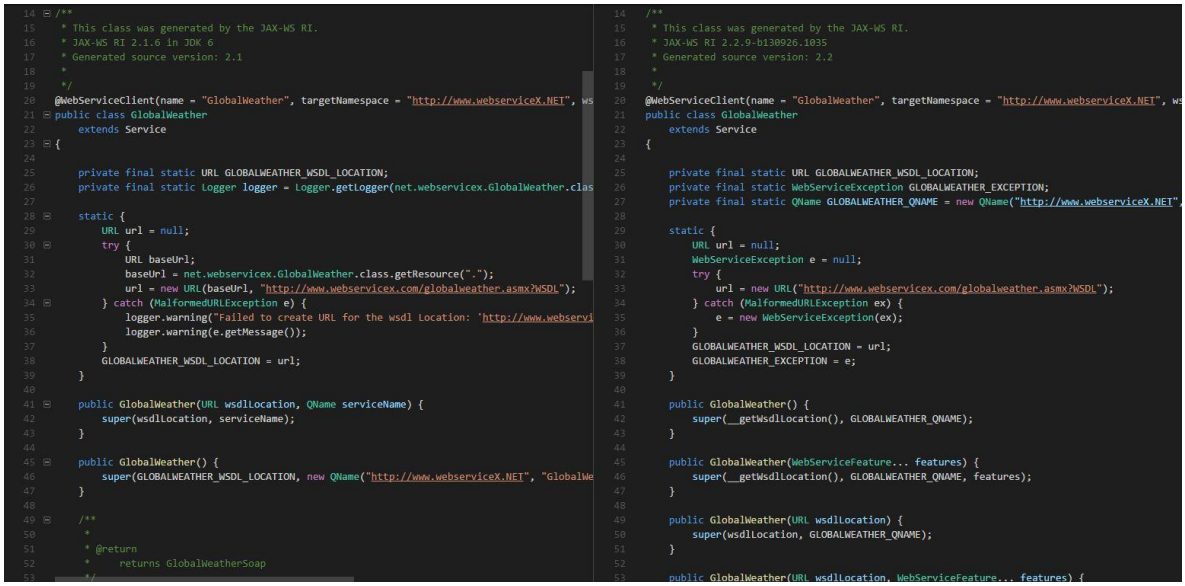
```
System.setProperty(name, (String)binding.getObject());
```

WLS 12c -ystävällisempään muotoon

```
System.setProperty(name, binding.getObject().toString());
```

Virhe johtui uuden WLS-palvelimen muuttuneesta JNDI-hakemistopalvelurajapinnan toteutuksesta. On yleisesti huono käytäntö ohjelmoida rajapinnan *toteutusta* vasten, erityisesti tukeutuen edellisen kaltaisiin matalan tason tyyppimuunnoksiin. Tällainen menettely on herkkä muuttuvalle toteutukselle johtaen vaikeasti selvitettäviin ja korjattaviin virhetilanteisiin.

JDK 1.6 asennuksen mukana tulevan **wsimport**-työkalun tapa generoida asiakasluokat WSDL-dokumentista on osittain puutteellinen. Generoitujen luokkien vaillinainen toteutus aiheutti järjestelmässä ajonaikaisia poikkeuksia WSDL-dokumentin puuttumisesta. Tämä puutteellisuus oli aikaisemmin korjattu paikkaamalla wsimport-työkalun generoimat luokat käsin. Käsin paikkaaminen teki kuitenkin asiakasluokkien automaattisesta generoinnista koonnin yhteydessä mahdotonta. JDK 1.8 -version myötä wsimport-työkalun puutteellisuus on kuitenkin korjattu, ja uuden version generoimat asiakasluokat vastaavatkin toteutuksellisesti aiemmin käsin korjattuja. Kuvassa 38 nähdään ero JDK 1.6 ja JDK 1.8-versioiden generoimien asiakasluokkien välillä.

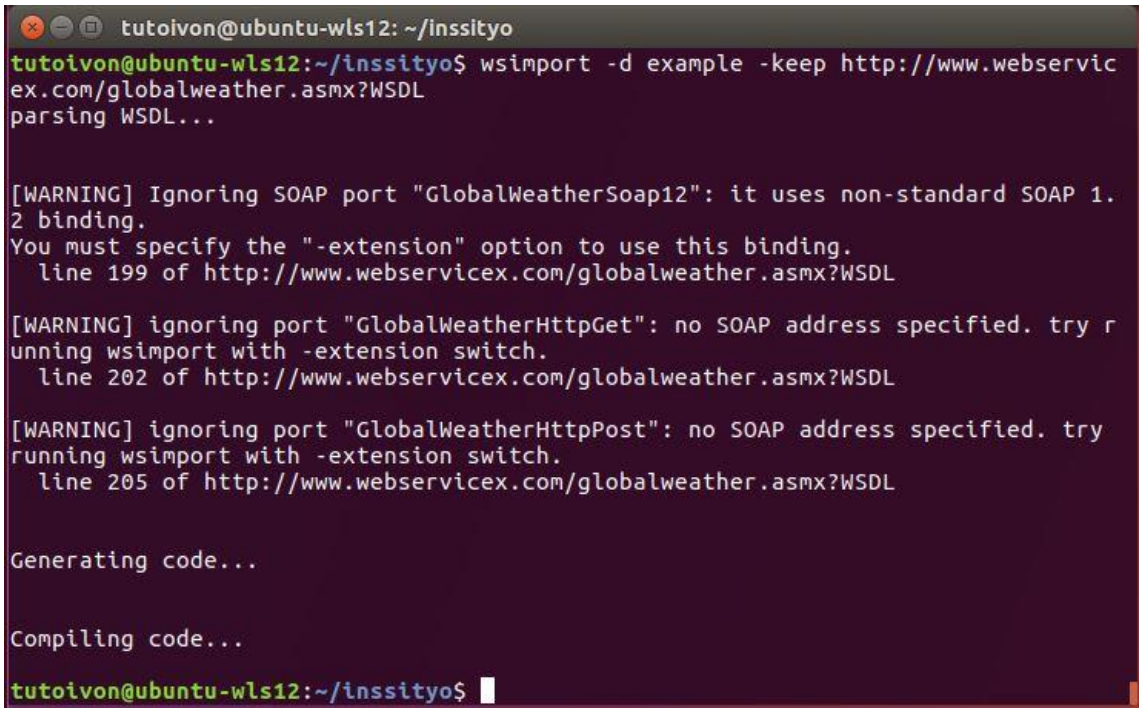


```
/**
 * This class was generated by the JAX-WS RI.
 * JAX-WS RI 2.1.6 in JDK 6
 * Generated source version: 2.1
 *
 */
@WebServiceClient(name = "GlobalWeather", targetNamespace = "http://www.webserviceX.NET", ws
public class GlobalWeather
    extends Service
{

    private final static URL GLOBALWEATHER_WSDL_LOCATION;
    private final static Logger logger = Logger.getLogger(net.webservicex.GlobalWeather.clas

    static {
        URL url = null;
        try {
            URL baseUrl;
            baseUrl = net.webservicex.GlobalWeather.class.getResource(".");
            url = new URL(baseUrl, "http://www.webservicex.com/globalweather.asmx?WSDL");
        } catch (MalformedURLException e) {
            logger.warning("Failed to create URL for the wsdl Location: 'http://www.webservi
            logger.warning(e.getMessage());
        }
        GLOBALWEATHER_WSDL_LOCATION = url;
    }

    public GlobalWeather(URL wsdlLocation, QName serviceName) {
        super(wsdlLocation, serviceName);
    }

    public GlobalWeather() {
        super(GLOBALWEATHER_WSDL_LOCATION, new QName("http://www.webserviceX.NET", "GlobalWe
    }

    /**
     *
     * @return
     *     returns GlobalWeatherSoap
     */
```

```
/**
 * This class was generated by the JAX-WS RI.
 * JAX-WS RI 2.2.9-b130926.1035
 * Generated source version: 2.2
 *
 */
@WebServiceClient(name = "GlobalWeather", targetNamespace = "http://www.webserviceX.NET", ws
public class GlobalWeather
    extends Service
{

    private final static URL GLOBALWEATHER_WSDL_LOCATION;
    private final static WebServiceException GLOBALWEATHER_EXCEPTION;
    private final static QName GLOBALWEATHER_QNAME = new QName("http://www.webserviceX.NET",

    static {
        URL url = null;
        WebServiceException e = null;
        try {
            url = new URL("http://www.webservicex.com/globalweather.asmx?WSDL");
        } catch (MalformedURLException ex) {
            e = new WebServiceException(ex);
        }
        GLOBALWEATHER_WSDL_LOCATION = url;
        GLOBALWEATHER_EXCEPTION = e;
    }

    public GlobalWeather() {
        super(__getWsdlLocation(), GLOBALWEATHER_QNAME);
    }

    public GlobalWeather(WebServiceFeature... features) {
        super(__getWsdlLocation(), GLOBALWEATHER_QNAME, features);
    }

    public GlobalWeather(URL wsdlLocation) {
        super(wsdlLocation, GLOBALWEATHER_QNAME);
    }

    public GlobalWeather(URL wsdlLocation, WebServiceFeature... features) {
```

Kuva 38. JDK 6- ja JDK 8 -versioilla generoidut SOAP-asiakasluokat (kuvituskuva).

Esimerkkikuvassa on käytetty internetistä vapaasti ladattavaa WSDL-dokumenttia [10.], eikä se liity itse projektiin. WSDL-dokumentista generoitiin JAX-WS-luokat wsimport-työkalulla seuraavassa kuvassa näkyvän komennon mukaisesti:



Kuva 39. wsimport-työkalun käyttö (kuvituskuva).

Projektiin generoitiin kaikki aiemmin käsin korjatut JAX-WS-asiakasluokat uudelleen JDK 1.8 -asennuksen tarjoamalla wsimport-työkalulla, jolloin käsin tehtävää korjausta työkalun generoimiin luokkiin ei enää tarvittu ja projekteista poistettiin aikaisemmat käsin tehdyt lisukkeet. Kuva 39 havainnollistaa wsimport-työkalun käyttöä.

Edeltävien muutosten jälkeen integraatio oli toteuttajan suorittamien testien perusteella toimintakunnossa.

3.3.4 Avaintenhallinta

Avaintenhallinta-sovelluksen koodikannan tasonnosto aloitettiin tavanomaisesti nostamalla käytettävän JDK-version määräävät konfiguraatiot 1.8-tasolle.

Lähdekoodilla on käännöksenaikaisia riippuvuuksia WLS-palvelimen asennukseen sisältyviin kirjastoihin. Kyseisten kirjastojen sijainti tiedostojärjestelmässä on kuvattu projektin koontiskriptissä. Uuden WLS 12c -palvelimen myötä asennuksen hakemistorakenne ja täten vaadittavien kirjastojen sijainti on hieman muuttunut, jonka seurauksena

kirjastojen sijainnit oli uudelleenmääriteltävä koontiskriptiin vastaamaan uutta hakemistorakennetta.

Käännös JDK 8:lla onnistui, mutta sovelluspaketin asennus ei onnistunut, vaan asennusvaiheessa syntyi seuraava ajonaikainen poikkeus:

```
weblogic.application.naming.ReferenceResolutionException:
[J2EE:160199]Error resolving ejb-ref "fi.foo.bar.Session-
facadeBean/keyManager" from module "Avaintenhallin-
taEJB.jar" of application "Avaintenhallinta". The ejb-ref
does not have an ejb-link and the JNDI name of the target
Bean has not been specified.
```

Virheilmoituksen mukaan sovelluksen vaatima EJB-komponentti on määritelty moniselitteisesti, ja riippuvuuden selvittäminen vaatii tarkempaa resoluutiota ohjelmoijan toimesta.

Virhe korjaantui poistamalla EAR-paketin sisältämän web-moduulin paketointiin sisältynyt manifestitiedosto kokonaan kyseisestä moduulista. Ongelman juurisyy oli se, että manifestitiedostossa viitattiin EJB-komponentit sisältäviin EJB-moduuleihin (JAR-arkistoihin) Class-Path-viittauksella. Näin kyseiset EJB-komponentit tulivat määritellyiksi kaksi kertaa, eikä säiliön suorittama, tyyppiin perustuva injektio onnistunut komponentin moniselitteisyyden vuoksi. Ongelman ilmenemisen syy on jälleen WLS 12c -palvelimen aggressiivisempi tapa rekisteröidä Java EE -komponentteja moduulin manifestitiedoston viittaamista arkistoista. Ongelma kuvatiin tarkemmin käyttöliittymän tasonnostoa kuvaavassa luvussa.

Seuraavana toimenpiteinä muista projekteista tuotiin uudelleenkäännetyt kirjastoriippuvuudet projektiin. Edeltävien muutosten jälkeen sovelluspaketin koonti ja asennus onnistui, ja toiminnallisuus vaikutti olevan kunnossa.

3.3.5 Avaintenhallinnan käyttöliittymä

Avaintenhallinnan käyttöliittymän lähdekoodin tasonnosto aloitettiin taas rutiininomaisesti koontia ohjaavan konfiguraation päivittämisellä JDK 1.8 -tasolle, tässä tapauksessa asettamalla Maven-koontityökalun käyttämään POM-tiedostoon `maven.compiler.source` ja `maven.compiler.target` -muuttujien arvoiksi 1.8.

Lähdekoodilla on käännöksen aikaisia riippuvuuksia WLS-palvelimen asennukseen sisältyviin kirjastoihin, kuten varsinaisella avaintenhallintaprojektillakin. Vaadittavat kirjastot on määritelty POM-tiedoston *dependencies*-osiossa *system*-tasoisiksi riippuvuuksiksi. Kirjastoriippuvuudet, joiden polut olivat uudelleenmääriteltävä projektiin, olivat

- com.oracle.ws.orawsdl_1.4.0.0.jar
- weblogic.jar.

Ensimmäisenä mainitun kirjaston vastine löytyy uuden WLS 12c -palvelimen asennuksesta nimellä **com.oracle.webservices.orawsdl-api.jar**. Jälkimmäisen kirjaston nimi on sama, mutta koska koko WLS-asennuksen hakemistorakenne on hieman muuttunut, kirjasto löytyy hieman eri sijainnista.

Projektista oli poistettava **jaxws-tools**-niminen riippuvuus, joka tarjosi sovelluksen JAX-WS-palveluille ajonaikaisen tuen. Kyseinen kirjasto aiheutti konfliktin WLS 12c -palvelimen sisäisen JAX-WS-toteutuksen kanssa aiheuttaen seuraavan ajonaikaisen poikkeuksen asennuksen yhteydessä:

```
SEVERE: MASM0001: Default configuration file [ metro-de-
fault.xml ] was not found
```

jaxws-tools-kirjaston poiston yhteydessä oli erikseen lisättävä **Apache Commons Lang** -riippuvuus, sillä kyseinen riippuvuus tuli aiemmin **jaxws-tools**-riippuvuuden yhteydessä. Vaatimus kyseiselle riippuvuudelle **jaxws-tools**-riippuvuuden poistamisen seurauksena ilmeni asennusvaiheessa syntyneen poikkeuksen muodossa:

```
java.lang.ClassNotFoundException: org.apache.com-
mons.lang.StringUtils
```

Seuraavina toimenpiteinä päivitettiin projektissa käytössä olevia teknologioita uudempiin julkaisuversioihin. Erityisesti aiemmin käytössä olleet **aspectj**- ja **cglib**-kirjastot eivät toimineet uudemman Java-version kanssa yksikkötestien osalta. Taulukossa 6 on lueteltu päivitetyt riippuvuudet.

Taulukko 6. Päivitetyt riippuvuudet.

Riippuvuus	**Vanha versio**	**Uusi versio**

spring-context	3.1.2.RELEASE	4.3.10.RELEASE
spring-core	3.1.2.RELEASE	4.3.10.RELEASE
spring-orm	3.1.2.RELEASE	4.3.10.RELEASE
spring-tx	3.1.2.RELEASE	4.3.10.RELEASE
spring-web	3.1.2.RELEASE	4.3.10.RELEASE
spring-test	3.1.2.RELEASE	4.3.10.RELEASE
spring-aspects	3.1.2.RELEASE	4.3.10.RELEASE
spring-instrument	3.1.2.RELEASE	4.3.10.RELEASE
spring-ws-core	2.1.0.RELEASE	2.4.0.RELEASE
spring-xml	2.1.0.RELEASE	2.4.0.RELEASE
aspectjrt	1.6.12	1.8.10
aspectjweaver	1.6.12	1.8.10
cglib	2.2.2	3.2.5

Edeltävien muutosten jälkeen sovelluspaketin koonti ja asennus onnistuivat, ja toiminnallisuus vaikutti olevan muilta osin kunnossa. Avaintenhallinnan käyttöliittymä oli järjestyksellisesti viimeinen projekti, jonka koodikannalle tasonnosto suoritettiin.

3.4 Versionnoston jälkeen

3.4.1 Regressiotestaus

Versionnoston ensimmäistä vaihetta, eli koodikantojen Java-version tasonnostoa seuraa huolellinen testausvaihe, jossa testaajien toimesta varmistetaan järjestelmän kaikkien käyttötapausten toimivuus, eli suoritetaan niin kutsuttu regressiotestaus. Tämän raportin kirjoittamiseen mennessä regressiotestausta ei kuitenkaan ole vielä aloitettu, eikä testauksen yhteydessä ilmenneitä virhetapauksia täten ole raportoitu tähän dokumenttiin.

3.4.2 Kehityshaarjojen yhdistäminen pääkehityshaaraan

Projektien versionnoston toteutuksen päätteeksi yhdistettiin versionnostoa varten perustetut kehityshaarat projektien pääkehityshaaroihin, eli tehtiin niin kutsuttu **merge**. Ennen historioiden yhdistämistä versionnoston kehityshaara siistittiin käyttäen **Git**-versionhallintatyökalun **rebase**-toimintoa, jolla voidaan siirtää, muokata, yhdistää tai poistaa projektin historiaan sisältyviä muutoksia. Seuraavalla komentoriviltä suoritettavalla Git-komennolla voidaan muokata historian seitsemää viimeisintä muutosta, esimerkiksi poistamalla tarpeettomia muutoksia historiasta, yhdistämällä yhteenkuuluvia muutoksia, tai tarkentamalla muutoksiin liittyviä selitteitä.

```
$ git rebase -i HEAD~7
```

Varsinainen merge-toiminto Git-versionhallintatyökalulla on tyypillisesti hyvin suoraviivainen

```
$ git checkout kayttoliittyma_8_0
$ git merge kayttoliittyma_7_3_java_8
```

Ensimmäinen komento valitsee aktiivisen kehityshaaran (branch), toinen komento yhdistää (merge) komennolle parametrina annetun kehityshaaran valittuna olevaan aktiiviseen kehityshaaraan. Esimerkissä merge-toiminto on kuvattu yksinkertaisimmillaan. Merge-toiminto on monimuotoinen työkalu, jonka toimintaa voidaan ohjata lukuisilla parametreilla. Git-versionhallinta itsessään on niin monimutkainen työkalu, että perusteisiinkin johdatus vaatisi kokonaan oman luvun (Git-versiohallintatyökalun referenssiopas käsittää yli 500 sivua). [11.] Vaikka merge-komento itsessään on yksinkertainen, voi mergen loppuunsaattaminen olla työlästä. Toisinaan mergeä suoritettaessa syntyy päivityskonflikteja, eli tilanteita, jossa toistaan erkaantuneet, yhdistettävät historiat ovat kohdistaneet eriäviä muutoksia samalle tiedoston riville. Syntyneitä konfliktitilanteiden korjausta ei luonnollisesti voida automatisoida, vaan tilanteen selvitykseen vaaditaan ihmisen arviointikykyä. Kehittäjän on itse korjattava mahdolliset päivityskonfliktit mergen loppuunsaattamiseksi. Git-työkalu tarjoaa jotain konfliktien ratkaisua helpottavia toimintoja. Konflikteilta ei vältytty tämänkään versionnoston yhteydessä, mutta niiden ratkaisu oli triviaalia.

3.4.3 Ympäristöjen ja sovellusten asennukset

Versionnoston toteutuksen yhteydessä syntyneen dokumentaation pohjalta luodaan uudet testaus-, laadunvarmistus- ja tuotantoympäristöt asennuksineen, sovelluspaketteineen ja konfiguraatioineen. CI-järjestelmään tehdään tarvittavat muutokset sovelluspakettien koostamiseksi. Ympäristöjen perustamista ei erikseen kuvata tässä raportissa, sillä ympäristön vaatimukset ovat jo hyvin pitkälti kuvattu henkilökohtaisen kehitysympäristön perustamista kuvaavassa luvussa, lukuun ottamatta ympäristö- ja sovelluskohtaisia konfiguraatioita, tietokantakantainstanssien perustamista ja kuormantasausta.

4 Yhteenveto

Versiotason nosto on monivaiheinen ja työläs prosessi, johon ei olemassa valmista onnistumisen kaavaa, sillä jokainen projekti on erilainen. Tämä raportti antaa kuvaa versionnostoon liittyvistä haasteista todellisten esimerkkien muodossa, mutta ei välttämättä tee selväksi varsinaisen *selvitystyön* haastavuutta. Suurin osa versionnostoon käytetystä ajasta kuluu kuitenkin virheiden *selvitykseen*, murto-osa niiden korjaamiseen. Syvälle ulottuva Java-tuntemus auttaa virheiden todellisen lähteen päättelyssä ja korjaamisessa, mutta toisinaan virheilmoitukset ovat hyvinkin harhaanjohtavia, standardit puutteellisia ja teknologiat niin huonosti dokumentoituja, että aiheuttavat kokeneellekin Java-ohjelmoijalle päänvaivaa.

Kohdejärjestelmän versionnostossa eniten päänvaivaa aiheutti WLS 12c -sovelluspalvelimen poikkeava käyttäytyminen aikaisempaan 11g-versioon verrattuna tietyillä osa-alueilla, sekä OpenFrame -sovelluskehyksen Java 8 -epäyhteensopivat rajapintatoteutukset. Kaikkiin ongelmiin löytyi lopulta ratkaisu, mutta ratkaisujen etsimiseen kului huomattavasti kauemmin aikaa kuin niiden toteutukseen.

Ennen tämän järjestelmän versionnostoa Oracle julkaisi uuden Java 1.9 -alustan. Versionnosto suoritettiin kuitenkin 1.8 tasolle, koska tosimaailman liiketoiminnassa, erityisesti luottamuksellista tietoa käsittelevässä kohdejärjestelmässä, alustalta vaaditaan luotettavuutta, eli sen täytyy olla kenttäolosuhteissa hyväksi todettu. Olisi liian suuri riski siirtyä suoraan uunituoreelle alustalle. Vaikka uusi alusta voi ominaisuuksiensa puolesta vaikuttaa houkuttelevalta, voi siinä piillä tietoturvaa vaarantavia haavoittuvuuksia.

Lähteet

1 J2EE project dangers! Verkkodokumentti. JavaWorld. https://www.javaworld.com/article/2075170/java-web-development/j2ee-project-dangers-.html. Päivitetty 30.3.2001. Luettu 22.11.2017.

2 Introduction to Vagrant. Verkkodokumentti. HashiCorp. https://www.vagrantup.com/intro/index.html. Päivitetty 2017. Luettu 22.11.2017.

3 Java™ EE Documentation. Verkkodokumentti. Oracle. http://www.oracle.com/technetwork/java/javaee/documentation/index.html. Päivitetty 2017. Luettu 22.11.2017.

4 JSR 109: Implementing Enterprise Web Services. Verkkodokumentti. JCP. https://jcp.org/en/jsr/detail?id=109. Päivitetty 7.6.2013. Luettu 22.11.2017.

5 Adding Classes to the JAR File's Classpath. Verkkodokumentti. Oracle. https://docs.oracle.com/javase/tutorial/deployment/jar/downman.html. Luettu 22.11.2017.

6 Interface Comparator<T>. Verkkodokumentti (Java API). Oracle. https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html. Luettu 22.11.2017.

7 JDK-6804124 : (coll) Replace "modified mergesort" in java.util.Arrays.sort with timsort. Verkkodokumentti. Oracle. http://bugs.java.com/bugdatabase/view_bug.do?bug_id=6804124. Päivitetty 16.5.2017. Luettu 22.11.2017.

8 Timsort. Wikipedia-artikkeli. Wikipedia. https://en.wikipedia.org/wiki/Timsort. Päivitetty 20.11.2017. Luettu 22.11.2017.

9 Java SE 7 and JDK 7 Compatibility. Verkkodokumentti. Oracle. http://www.oracle.com/technetwork/java/javase/compatibility-417013.html. Luettu 22.11.2017.

10 Global Weather WSDL. WSD-dokumentti. www.webservicex.com. http://www.webservicex.com/globalweather.asmx?WSDL.

11 Pro Git Book. Verkkodokumentti. Git SCM. https://git-scm.com/book/en/v2.