Tempo plugin- ja JIRA sekä PlanMill – ohjelmistoratkaisujen integrointi JBoss SOA Platformilla

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Ammattikorkeakoulun opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

Visamäki, syksy 2013

Riku Koskinen

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Visamäki
Tietojenkäsittely
Systeemityö

Tekijä Riku Koskinen **Vuosi** 2013

Työn nimi Tempo plugin- ja JIRA- sekä PlanMill –ohjelmistoratkaisujen integrointi JBoss SOA Platformilla

TIIVISTELMÄ

Työn aiheen tarjosi Ambientia Oy. Ambientia on sähköiseen liiketoimintaa ja viestintään sekä yhteisöllisiin ratkaisuihin erikoistunut asiantuntijayritys.

Opinnäytetyön tarkoituksena oli kehittää proof of concept-versio integraatiosta, jossa automatisoidaan PlanMill-sovelluksen projektitietojen kirjaus JIRA-sovellukseen ja JIRAsta tuntikirjauksien kirjaus PlanMill-sovellukseen. JIRAn asennuksessa on mukana Tempo-liitännäinen, joka on ajanseurantaan erikoistunut lisäosa. Integraatio toteutetaan JBoss SOA Platformia käyttäen.

Ohjelmistokehityksessä käytettiin työvälineinä JBoss Developer Studiota, kustomoitua Eclipse-kehitysympäristöä. Ohjelmointikielenä käytettiin Javaa. Sovellusten väliseen kommunikointiin käytettiin REST-, RPC- ja SOAP-tekniikoita.

Työn päätteeksi valmistui raportti ohjelman rakenteesta ja toteutuksesta sekä toimiva ohjelmisto, joka todistaa halutun kaltaisen integraation mahdolliseksi.

Avainsanat Java, JBoss ESB, Ohjelmointi, SOA

Sivut 26s

HAMK
UNIVERSITY OF APPLIED SCIENCES

Visamäki
Degree Programme in Business Information Technology
System Engineering

Author Riku Koskinen **Year** 2013

Subject of Bachelor's thesis Integration of Tempo plugin, JIRA and PlanMill software solutions with JBoss SOA platform

ABSTRACT

This thesis was commissioned by Ambientia Ltd. . Ambientia is a specialist company that is specialized in electronic business and communications and community-based solutions.

The purpose of the thesis was to develop a proof of concept-version of the integration, which automates PlanMill application project data entry to JIRA application and JIRA's working hour logging to PlanMill application. JIRA installation includes Tempo plugin, which is an add-on that is specialized in logging working hours. Integration was implemented by using JBoss SOA Platform.

The tool used in software development was JBoss Developer Studio, a customized Eclipse development environment. The programming language used was Java. Techniques and protocols used in communication between applications, include REST, RPC and SOAP.

As a result of the thesis, a functional software was completed, which proves that the desired type of integration is possible and the report about structure of software and the process of implementation.

Keywords Java, JBoss ESB, Programming, SOA.

Pages 26p

SISÄLLYS

IDE
Integrated Development Environment, Ohjelmointiympäristö on työkalu, jolla ohjelmistoja kehitetään. Se voi koostua yhdestä tai useammasta eri ohjelmasta.

Eclipse-IDE
Avoimeen lähdekoodin perustuva ohjelmointiympäristö. Avoimuuden takia siitä voi muokata sopivan ympäristön, erilaisille järjestelmille.

SaaS
Software as a service on palvelumalli, jossa sovellusta tarjotaan palveluna. Sitä ei tarvitse itse asentaa ja hallinnoida omalla palvelimella vaan se on käytettävissä esimerkiksi internet-selaimella.

Middleware-ohjelmisto
Middleware:lla tarkoitetaan yleensä integroitavien sovellusten välissä olevaa osiota. Yleisesti järjestelmä joka mahdollistaa viestinnän kahden eri sovelluksen välillä.

Liimakoodi
Liimakoodilla tarkoitetaan sellaista koodia, joka ei tuo uutta toiminnollisuutta ohjelmistoon vaan sitoo eri osuuksia yhteen, jotta yhteensopivuus ongelmia ei olisi.

Proof of concept
Konsepti-tyylinen sovellus, jonka tarkoitus on todentaa jonkin asian toteutus mahdollisuus. Yleensä ratkaisee jonkin ongelman uudella tavalla ja on erittäin rajattu muilta ominaisuuksiltaan.

Instantiointi
Olio-ohjelmointiin liittyvä sana, jolla kuvataan sitä prosessia, jossa luokasta tehdään olio.

XML
Extensible Markup Language. Standardi merkintäkieli, joka auttaa tiedon jäsentämisessä. Tiedon kuvaus ja itse tieto on merkittävissä XML-dokumenttiin. Käytetään monesti tiedonvälitykseen järjestelmien välillä.

1 JOHDANTO

Työn aiheen tarjosi Ambientia Oy. Ambientia on sähköiseen liiketoimintaa ja viestintään sekä yhteisöllisiin ratkaisuihin erikoistunut asiantuntijayritys. Ambientian juuret ulottuvat aina vuoteen 1996 asti, jolloin yritys tunnettiin nimellä Ambient Factor.

Ambientialla on käytetty projektien ja työajan seurantaan JIRA- ja PlanMill-sovelluksia. Molempien järjestelmien ominaisuudet menevät jonkin verran päällekkäin. JIRA on enemmän projektien hallintaan keskittyvä, kun taas PlanMill on asiakashallintaan suuntautuva järjestelmä. Ongelmana on, että projektien tiedot ja työntekijöiden ajankäyttö ilmoitukset täytyy syöttää erikseen molempiin järjestelmiin. Tämän työn tarkoituksena on toteuttaa toimiva proof of concept-versio integraatiosta, jossa automatisoidaan uusien PlanMill projektien generointi JIRAan ja työaikakirjauksien ilmoitus JIRAn Tempo-liitännäisen aika raporteista PlanMilliin. Integraatio toteutetaan käyttämällä JBoss SOA Platform -järjestelmää.

Työssä käydään läpi käytettävät tekniikat ja integroitavien sovellusten tietomalleja. Luonteeltaan työ on kehitysprojekti, jonka lopputuloksena on toimiva integraatiosovellus. Vaikka tarkoituksena olikin toteuttaa vain proof of concept-tyyppinen sovellus, niin se suoriutuu vaadituista tehtävistä ja näyttää, että täysimittainen integraatio on mahdollista toteuttaa.

Työn tärkeimpinä kysymyksinä on JIRAn ja PlanMill REST/SOAP-rajapintojen toiminta ja kuinka halutun kaltainen integraatio on mahdollista toteuttaa käyttäen JBoss SOA Platformia?

2 INTEGROITAVAT SOVELLUKSET JA TEKNIIKAT

Tässä työssä integroidaan kaksi sovellusta, JIRA ja PlanMill. Molemmat sovellukset soveltuvat projektien hallintaan ja seurantaan. PlanMill on tarkoitettu enemmän asiakassuhteiden hallintaan ja projektien hallintaa isolla mittakaavalla, kun taas JIRA on pieniinkin yksityiskohtiin ja nimenomaan projektimaiseen toimintaan kehitetty sovellus. Molempia kuitenkin yhdistää se, että niissä säilytetään samoja tietoja projekteista ja projekteihin käytetyistä työajoista. PlanMill tarvitsee projektitietoja jotta siellä säilyy yleiskuva kaikista projekteista ja asiakassuhteista ja laskutuksen takia sinne täytyy myös syöttää tuntikirjaukset. JIRAan tarvitaan projektitiedot jo pelkästään projektien toteuttamista varten, mutta tuntikirjaukset, jotka tulevat Tempo-liitännäisen avulla JIRAan, ovat varsinkin projektipäälliköille tarpeellinen työkalu projektien hallitsemisessa. Koska molemmissa säilytetään samoja tietoja, ne pitää sinne myös syöttää, joka tarkoittaa sitä, että työntekijä kirjaa kahdesti työtuntinsa näihin järjestelmiin ja projektipäälliköt luovat projektiympäristöt molempiin järjestelmiin.

Integraation tarkoituksena on poistaa tietojen kaksoissyöttäminen automatisoimalla projektin generointi JIRA-järjestelmään, kun sellainen tehdään PlanMill-järjestelmään. Myös ajankirjaus automatisoidaan siten, että kun JIRAan lisätään ajankäyttöilmoitus Tempo-liitännäisen avulla, niin se ilmoitetaan automaattisesti PlanMill-aikaraportteihin.

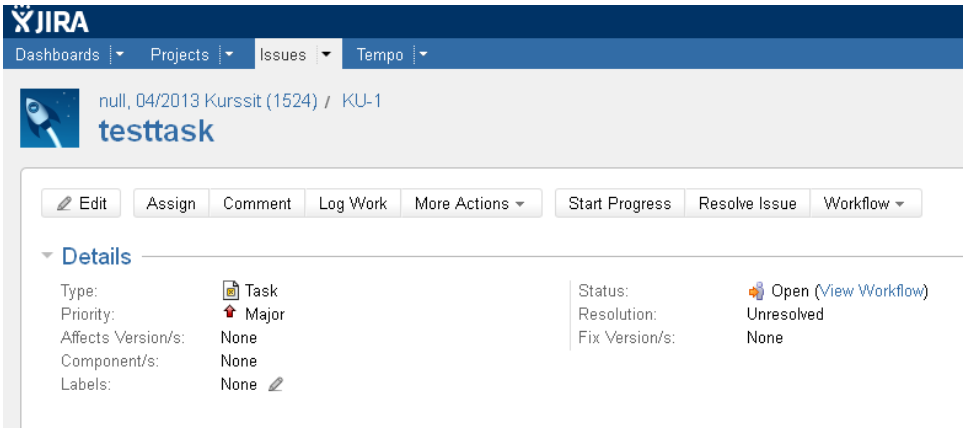
Kaiken keskelle luodaan JBoss ESB SOA Platformin avulla järjestelmä, johon luodaan palveluita ja toimintoja, jotka toteuttavat itse integraation. JBossin, Tempon ja JIRAn asentamista ei käydä tässä työssä läpi.

Koska työ on proof of concept-tyyppinen toteutus, niin koodin toimivuus kaikissa mahdollisissa ympäristöissä ja tapauksissa ei ole päätarkoitus, vaan toimintojen kehittäminen ja mahdollisten teknisten ongelmien tutkiminen ja ratkaiseminen.

2.1 JIRA

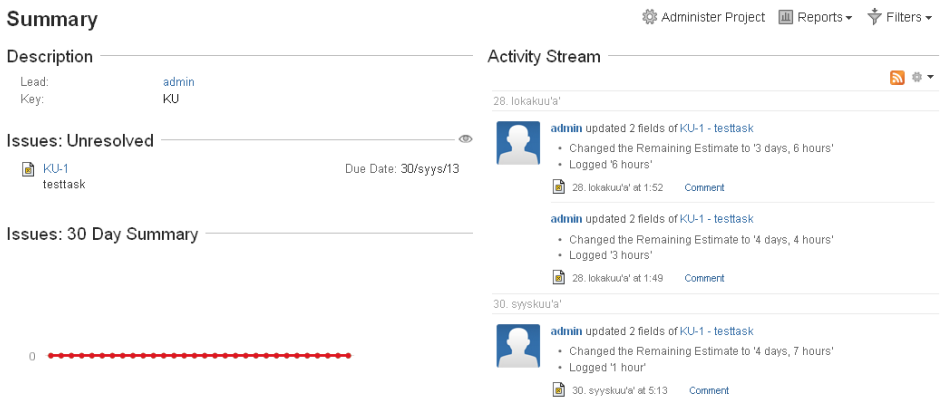
JIRA on tehtävienseurantaohjelmisto, joka auttaa asiakaspalveluun ja projekteihin liittyvien tehtävien ja pyyntöjen hallinnassa. JIRAn kehittäjä on australialainen Atlassian ohjelmistotalo, joka on perustettu vuonna 2002 ja jolla on yli 25 000 asiakasta (Atlassian. n.d. Atlassian customers). (Atlassian, n.d. Atlassian company). JIRA on ensimmäinen ja isoin ohjelmisto, jota Atlassian tarjoaa.

JIRA on maksullinen ohjelma ja sitä myydään ladattavana ja siten itse ylläpidettävänä ja OnDemand-mallilla. Eroina näissä on että ladattava on kerta maksullinen, sitä voi kustomisoida liitännäisillä joita myydään Atlassian Marketplace:ssa, sitä voi itse kustomoida ja saa täydellisen hallinnan ohjelmistoon. OnDemand-malli on pilvipalvelu, välittömästi käytössä ja tilauksen voi uusia kuukausi kerrallaan. Kirjoitushetkellä lataus-version lisenssimaksu kymmenelle henkilölle, on 10 dollaria ja OnDemand-version maksut 10 dollaria kuukaudessa. Atlassian tukee avoimen koodin kehitysprojekteja antamalla JIRAn käyttöön ilmaiseksi, jos projekti täyttää tietyt vaatimukset. Käytännössä tämä vaatii vain aidosti avointa projektia. (Atlassian. n.d. Open Source Project License Request.) (Atlassian. n.d. JIRA pricing.)



Kuva 1. JIRAn projektinäkymä

JIRAn tietokantaan voi määritellä muun muassa projekteja, käyttäjiä ja tehtäviä. JIRAn toimintaperiaatteen pohjalla on kolme pääajatusta. Ensimmäisenä ovat projektit, joilla määritellään jokin kokonaisuus (Kuva 1). Projektilla on muun muassa nimi ja avain-tunnus, jota käytetään esimerkiksi kun nimetään tehtäviä, issues. Projektin ei tarvitse olla ohjelmistokehitykseen liittyvä, vaan se voi olla esimerkiksi markkinointi kampanja tai helpdesk-järjestelmä. Toisena on työnkulku, workflow, jolla määritetään tehtävien tilaa. Tila voi olla avoin, työn alla, ratkaistu, suljettu tai uudelleen avattu. Viimeisenä on itse tehtävä, jonka avulla esitetään jokin työtä vaativa asia (Kuva 2). Kaikki tehtävät kuuluvat johonkin projektiin. Tehtävä voi kuvailla esimerkiksi jonkin vian ohjelmistossa. Jokaiseen projektiin voi tarkasti määritellä, kenellä on oikeus osallistua tehtävien antamiseen ja suorittamiseen. Tehtäviin voi lisätä liitteitä, esimerkiksi kuvan ongelmasta tai arvion suorittamiseen kuluvasta ajasta tai tiedon henkilön ongelmanratkaisuun kuluttamasta ajasta. Ongelmat voi merkitä seurattaviksi, jolloin jokainen muutos ilmoitetaan sähköpostiin. (Atlassian. 2013. Documentation for JIRA 6.0.)



Kuva 2. JIRAn tehtävä-näkymä

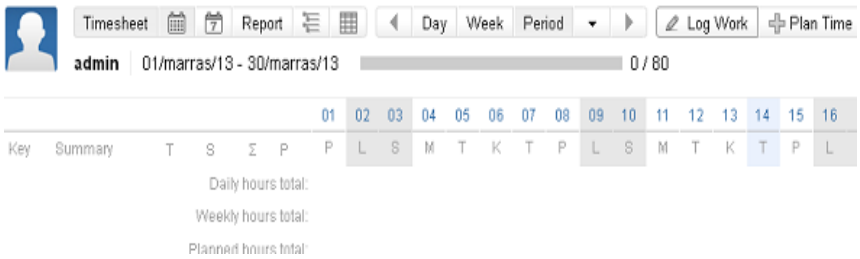
Yksi JIRAn vahvuuksista on sen liitännäisjärjestelmä. Kirjoitushetkellä JIRAan on Atlassian Marketplace-kaupassa satoja erilaisia liitännäisiä myynnissä. Kolme suosituinta on JIRA Agile, joka lisää JIRAan ketterän kehitysmallin mukaisia ominaisuuksia, kuten backlogin, työn suunnittelun ja visualisoinnin tiimin aktiivisuudesta. Toisena on Tempo, jota tässä työssä

myös käytetään, siitä kerrotaan lisää seuraavassa kappaleessa. Kolmantena on JIRA Capture, joka tarjoaa QA-tiimeille, testaajille ja kehittäjille nopeat keinot raportoida vikoja ohjelmistoista. (Atlassian. n.d. Marketplace.)

2.2 Tempo-liitännäinen

Tempo on TM Software-ohjelmistotalon tuote. Sen kehitys aloitettiin 2007 vuoden lopussa. Tarkoituksena ratkaista ajankäyttöön ja laskutukseen liittyvät ongelmat. TM Softwarella kuitenkin huomattiin pian että tällaiselle tuotteelle on kysyntää muillakin yrityksillä. Vuonna 2009 pidetyssä ensimmäisessä Atlassian Summit-konferenssissa, esiteltiin Tempo-liitännäinen. (TM Software. 2011. The Tempo times.)

Tempo on JIRA liitännäinen, jonka avulla voi suunnitella, kirjata ja seurata ajankäyttöä eri tehtävissä. Tempo-liitännäisellä voi kirjata nopeasti henkilön tiettyyn tehtävään käyttämän ajan. Myös ajankäytön suunnittelu on mahdollista. Tempo tarjoaa myös REST-APIn, josta on mahdollista hakea työaikakirjauksia erilaisilla hakusuodatuksilla (Kuva 3).



Kuva 3. Tempon ajankirjaus-näkymä

2.3 PlanMill

PlanMill on suomalainen PlanMill Oy:n valmistama ohjelmisto. Se koostuu muun muassa asiakashallinta-, toiminnanohjaus ja projektien hallintajärjestelmästä ja nämä tarjotaan SaaS-palveluina. PlanMill-järjestelmällä voi hallita projektin koko elinkaaren, sisältäen muun muassa yksittäisten tehtävien luomisen, resurssien hallinnan ja laskutuksen. Se tarjoaa välineet liiketoiminnan reaaliaikaiseen seuraamiseen ja ennustamiseen. (PlanMill Oy, 2005. Kannattavuutta ja kilpailukykyä projektiliiketoimintaan.)

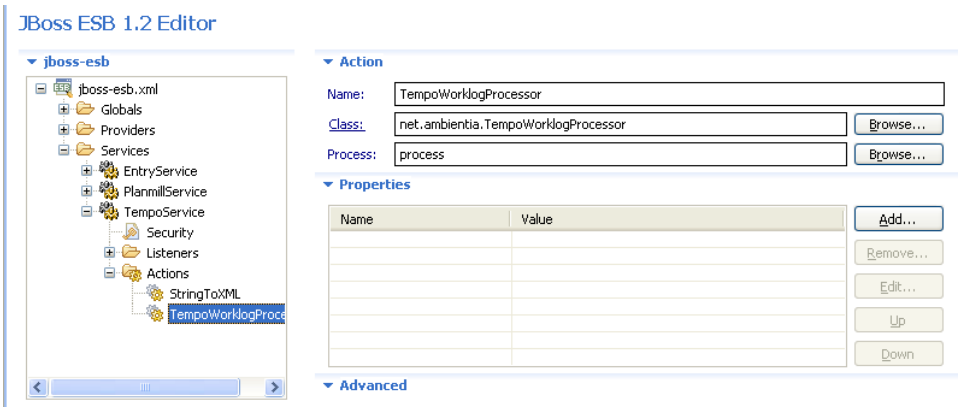
PlanMill tarjoaa käyttöön REST-APIn, jonka avulla projektien kaikki projektien tiedot on haettavissa ja muokattavissa.

3 JBOSS SOA PLATFORM

JBoss SOA Platform on avoimeen lähdekoodiin perustuva ohjelmistokomponenttikirjasto, joka on suunniteltu järjestelmäintegraatioita (EAI) ja palvelukeskeisiä (SOA) järjestelmiä varten. Sen on kehittänyt JBoss, joka on nykyään Red Hat Inc. alajaosto. Red Hat Inc. tuottaa avoimeen lähdekoodiin perustuvia sovelluksia yritysyhteisöille.

JBoss SOA Platform on kokonainen paketti, jossa on kaikki tarpeellinen järjestelmä integraation kehittelyn aloittamista varten. Se kuuluu Red Hatin middleware-ohjelmistoihin. Sen pääominaisuuksiin kuuluu JBoss Enterprise Service Bus(ESB), joka kuljettaa viestejä järjestelmien välillä. ESB-järjestelmästä kerrotaan lisää kohdassa 3.2.

JBoss SOA Platformin mukana tulee myös JBoss Developer Studio, joka on kustomoitu Eclipse-pohjainen IDE. Sen näkyvin kustomointi on näkymä, jonka avulla on helppo visuaalisesti muokata jboss-esb.xml-tiedostoa, joka on kaiken toiminnan keskipiste (Kuva 4.) (Red Hat. n.d. JBoss SOA-P Administrator guide.)



Kuva 4. Jboss-esb.xml-tiedoston visuaalinen näkymä

JBoss SOA Platformin ohjelmisto pinoon kuuluu monia ohjelmistoja. Järjestelmän alimpana on paketin mukana tuleva JBoss Enterprise Application Platform, joka on Java EE-pohjainen itsenäinen middleware-ohjelmisto. Se on kehitetty JBoss Enterprise Application -palvelinohjelmiston päälle. (Red Hat. n.d. JBoss Enterprise Application Platform.)

JBoss SOA Platformin pääominaisuuksiin kuuluu myös palvelurekisteri, java Universal Description, Discovery and integration (jUDDI), johon tallennetaan palvelun kuvaus- ja sijaintitiedot, jotta toiset palvelut voivat etsiä ja löytää sen. Se tukee löyhää riippuvuutta poistamalla palvelun käyttäjältä tarpeen tietää käyttämänsä palvelun konkreettista sijaintia.

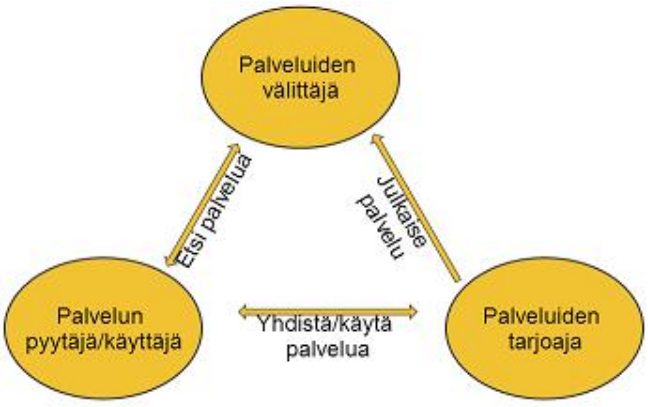
Näiden lisäksi, mukana on monia muitakin komponentteja, kuten esimerkiksi tietomuuntaja (Smooks), viestijono (HornetQ) ja BPEL järjestelmä (Riftsaw). Näitä komponentteja ei tämän tarkemmin tässä työssä käydä läpi.

3.1 SOA-arkkitehtuuri

SOA, Service Oriented Architecture, palvelukeskeinen arkkitehtuuri on sovelluskehitys-malli, joka on kehitetty järjestelmä integraatioiden toteuttamista varten. SOA-malli on tarkoitettu nimenomaan liiketoiminta-prosessien integroimiseen ja monet JBoss SOA Platformin komponenteista onkin tehty liiketoiminta-prosessit mielessä.

Tavallisessa yritysympäristössä on useita ohjelmistoja käytössä. Ajan myötä, kun esimerkiksi yritys laajenee tai muuten muuttaa toimintaansa, niin mukaan otetaan uusia ohjelmistoja. Kuitenkin vanhoille ohjelmille on yhä tarkoituksensa, ehkä joidenkin tiettyjen asiakkaiden tai tietynlaisten prosessien kanssa. Pahimmassa tapauksessa on kehitetty sisäisesti jokin lisäosa tai muuten lisätoiminnallisuutta luotu jollekin näistä vanhoista ohjelmista. Kaikki tämä aiheuttaa sen että aina kun järjestelmään tuodaan uusi versio vanhasta sovelluksesta tai kokonaan uusi sovellus, niin joudutaan kehittämään liimakoodia, joka sekavoittaa entisestään järjestelmiä. Tätä ongelmaa vastaan on kehitetty SOA-malli.

Palvelukeskeisen arkkitehtuurin sisimpänä ajatuksena on viestien vaihtelu, palveluiden tarjoajat ja pyytäjät ja yhteysväylä, joka mahdollistaa viestien liikkumisen edestakaisin palveluiden välillä. Kun SOA-järjestelmään lisätään uusi järjestelmä, sille luodaan adapteri, joka kytkee kyseisen järjestelmän kiinni SOA-järjestelmään. Tämän jälkeen mikä tahansa muu SOA-järjestelmään kytketty järjestelmä voi kutsua juuri liitettyä järjestelmää palveluna. Tämä adapteri-koodi mahdollistaa myös löyhän riippuvuuden. Löyhä riippuvuus toteutuu siten, että palvelut eivät suoraan kutsu toistensa sisäisiä funktioita tai yleensäkään käsittele toisen ohjelman lähdekoodia millään tavalla, vaan niiden välinen keskustelu toteutetaan lähettämällä viestejä. Tämä mahdollistaa niin uusien kuin vanhojenkin järjestelmien integroinnin. SOA-viestintä on XML-pohjaista, joten kaikki viestit, niin kyselyt kuin vastauksetkin, ovat XML- muodossa.



Kuva 5. SOA-roolit

SOA:ssa järjestelmillä on kolme roolia (Kuva 5). Ensimmäinen rooli on Palvelun tarjoaja, service provider, joka tarjoaa pääsyn palveluihin, luo kuvauksen palvelusta ja julkaisee sen palvelun jakelijalle. Toinen on palvelun pyytäjä, service requester, joka etsii palveluita tutkien palveluiden kuvauksia. Tätä voi myös ajatella palvelun käyttäjänä, service consumer, monissa

tapauksissa. Pyytäjä on myös vastuussa palvelun tarjoajalta saatuihin palveluihin kiinnittymisestä. Kolmantena roolina on palvelun välittäjä, service broker, joka ylläpitää rekisteriä palveluiden kuvauksista. Se on myös vastuussa pyytäjän linkityksestä tarjoajaan. (Red Hat. n.d. JBoss SOA-P Administrator guide.)

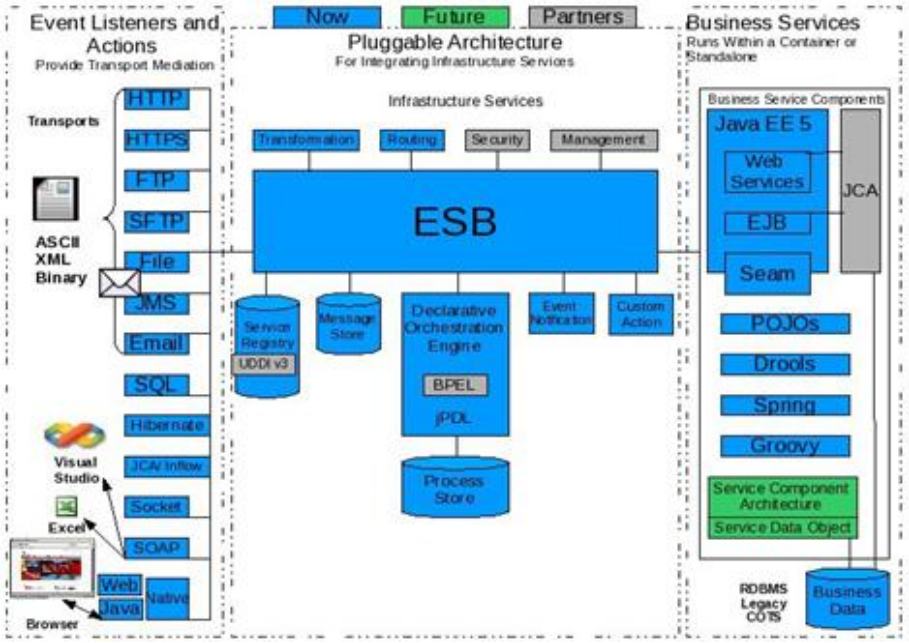
3.2 ESB

ESB, enterprise service bus, on SOA-järjestelmissä käytettävä arkkitehtuurimalli. Sillä on monia tehtäviä. Niihin lukeutuu esimerkiksi viestien reitityksen tarkkailu ja ohjaaminen, palveluiden yhteen liittäminen ja viestien puskurointi. Viestien puskurointia tarvitaan, koska palvelua ylläpitävät laitteistot voivat olla tavoittamattomissa tai viestejä tulee liikaa yhtäaikaisesti, jolloin kuormitus kasvaa. (Mulesoft. n.d. Enterprise service bus.)

SOA-periaatteiden mukaisesti, kaikkea JBoss ESB:een liitettyä kohdellaan joko palveluna tai viestinä. Palvelut kapseloivat jonkin loogisen komponentin tai vanhan järjestelmän integraatio rajapinnan. Viesteillä asiakkaat ja palvelut kommunikoivat toistensa kanssa. (Red Hat. n.d. JBossESB programmers guide.)

ESB:llä ei tarkoiteta vain JBoss ESB:tä, vaan mitä tahansa ESB-järjestelmää. ESB:n avulla järjestetään interaktiivisuus ja yleinen ohjelmien välinen kommunikointi. Se toteuttaa tämän abstraktoimalla järjestelmien väliset erot ja kohtelemalla niitä loogisina palveluina ESB-järjestelmässä. Palvelut kommunikoivat standardi mallisten viestien välityksellä, jonka ansiosta palveluiden sisäisestä arkkitehtuurista ja rakenteesta ei tarvitse välittää. Viestit ovat lähes aina XML-muotoisia. Kun palvelulle on kerran luotu kyky reagoida ESB-viestiin, niin sitä palvelua varten voi luoda viestin mistä tahansa lähteestä ESB-järjestelmään. (Red Hat. n.d. JBossESB programmers guide.)

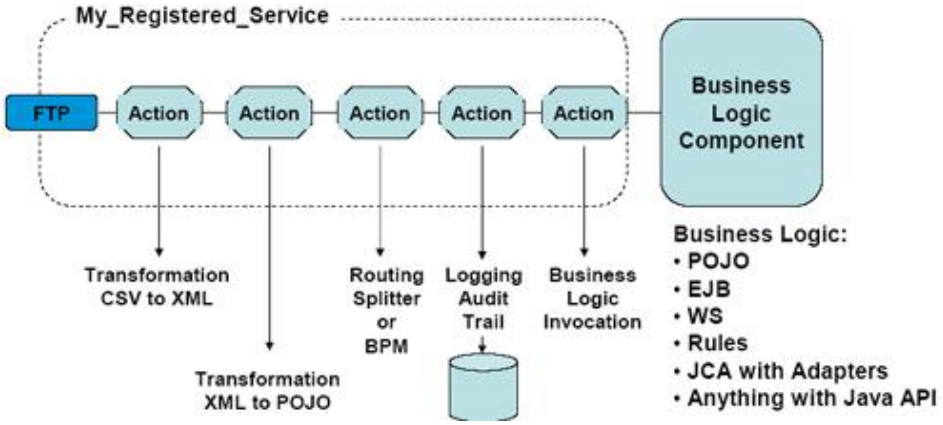
Kuvasta 6 selviää JBoss ESB-mallin toimintaidea. Vasemmalla on listattuna oletuksena tuettuja tapoja, joilla ulkomaailma eli järjestelmän ulkopuoliset sovellukset tai käyttäjät, voivat luoda tapahtumia ESB-järjestelmään. Näitä kutsutaan yhdyskäytäviksi, gateway. Yhdyskäytävien läpi voi lähettää ESB-viestin, joka on toisessa järjestelmässä jo luotu tai jotain muuta, josta luodaan ESB-viesti. (Red Hat. n.d. JBossESB programmers guide.) (JBoss ESB Beginner's guide.)



Kuva 6. JBoss ESB kokonaisuus

3.3 Toimintoketju

Toimintoketju, actionchain, on yhteen palveluun sidottujen toimintojen ketju (Kuva 7). Toisin sanoen, toimintoketju määrittää palvelun toiminnat. ESB-järjestelmä kuljettaa viestiä toimintoketjussa, alkaen palvelun ensimmäisestä jboss-esb.xml-tiedostoon määritetystä toiminnosta. Jokainen toiminto voi tehdä viestille mitä haluaa ja sen jälkeen päättää antaako viestin siirtyä seuraavalle toiminnolle vai lopetetaanko viestin kuljetus kyseiseen toimintoon.



Kuva 7. Actionchain, toimintoketju

3.4 Toiminto

Toiminto on oma luokkansa, joka periytetään abstraktista AbstractAction-PipelineProcessor-luokasta. Se tarjoaa luokalle käyttöön process-funktion, jota automaattisesti kutsutaan kun on suorituksen aika. Suorituksen aika määrittyy toiminnon sijainnista toimintoketjussa. Process-funktio voi palauttaa message-olion, jolloin seuraavat määritetyt toiminnot saavat viestin käsittelyyn tai se voi palauttaa null-arvon, jolloin toimintoketjun suoritus pysäytetään kyseiseen toimintoon (Kuva 8.)

```
1  public class MyCustomActionProcessor extends AbstractActionPipelineProcessor
2  {
3      public void initialise() throws ActionLifecycleException {
4          // Initialise resources...
5      }
6      public Message process(Message message)
7              throws ActionProcessingException {
8          // Process messages in a stateless fashion...
9          return message;
10         // Return message to chain, for other actions
11         // Return null to stop the chain
12     }
13     public void destroy() throws ActionLifecycleException {
14         // Clean-up resources...
15     }
16 }
```

Kuva 8. Kustomi ActionProcessor-luokan runko

Toiminto voi siis toteuttaa mitä tahansa toimintoja, joita normaali Java-luokkakin. Yleisen tavan mukaisesti yksi toiminto suorittaa vain yhden asian. Esimerkiksi opinnäytetyön aikana toteutetussa työssä on toiminto StringToXML. Se etsii viestistä tietyn nimistä merkkijono-muuttujaa ja yrittää muuntaa sen sisällön XML-tyyppiseksi. Tämä toiminto on käytännössä käytettävissä missä tahansa JBoss SOA Platform ympäristössä, jossa halutaan tehdä edellä mainitun kaltainen muunnos. Toiminnot on hyvä toteuttaa suorittamaan yksi asia ja tehdä se hyvin.

Kun toiminto on kerran toteutettu, niin sen uudelleen käyttö missä tahansa toisessa palvelussa vaatii vain Jboss-esb.xml-tiedoston muokkausta. Jotta uudelleen käyttö olisi mahdollista, niin se vaatii että toiminto toteutetaan uudelleen käyttö mielessä.

3.5 Käännöstyökalu Apache Ant

Apache Ant on Java-pohjainen koontityökalu. Se on suunniteltu ohjelmiston kääntämiseen vaaditun prosessin automatisointiin. Ant tulee sanoista, Another Neat Tool. Ant tarvitsee XML-pohjaisen koonti-tiedoston jossa on ohjeet ohjelmiston kääntämiselle. (Red Hat. n.d. JBoss SOA Install and Configuration.)

Ant kuului alun perin Tomcat-ohjelmiston koodikantaan ja se oli suunniteltu yksinomaan sen kääntämis- ja koontitoimintojen suorittamiseen. Sen on luonut James Duncan Davidson, joka on myös Tomcat-ohjelmiston alkuperäinen kehittäjä. Aika pian, useat projektin kehittäjät huomasivat, että

Ant voisi ratkoa ongelmia, joita heillä oli Makefile-tiedostojen kanssa. Nykyään Ant on oma osionsa ja sitä kehitetään omana projektina. (The Apache Software Foundation. n.d. Apache Ant FAQ.)

Ant ja JBoss Application Platform-serveri on konfiguroitu siten, että serveriä ei tarvitse uudelleen käynnistää aina kun koodia on päivitetty. Antamalla komennon ant deploy-esb, voi työn kääntää uudelleen ja aloittaa ohjelman toiminnan alusta. Tämä on hyvä ominaisuus, joka säästää useita tunteja. Vanhemmalla tietokoneella serverin käynnistymiseen voi mennä jopa 5-10 minuuttia.

3.6 Riippuvuuksien hallintatyökalu Apache Ivy

Ohjelmistojen ja ohjelmisto-kirjastojen välillä voi olla riippuvuuksia, dependency. Riippuvuus ohjelmien ja kirjastojen välille syntyy, kun jokin ohjelma käyttää kirjaston tarjoamia ominaisuuksia. Tällainen riippuvuus vaatii, että kirjasto täytyy kääntää ensiksi ja vasta sitten itse ohjelma, koska ohjelman kääntäminen vaatii kirjaston ominaisuuksia. Riippuvuuksien hallintatyökalut auttavat tässä prosessissa.

Apache Ivyn valintaan vaikutti se, että se on integroitu Apache Ant -työkaluun, joka on mukana jo oletuksena.

Tässä työssä Ivyn avulla varmistetaan, että projektilla on oikeat versiot JBoss RestEasy ja JIRAn SOAP-kirjastoista. Ivyn konfiguroiminen käyttöä varten on yksinkertaista. Ant-projekteilla on build.xml-tiedosto. Sinne lisätään uusi target-elementti Kuvan 9. mukaisella tavalla.

```
1  <target name="resolve" description="--> retrieve dependencies with ivy">
2      <ivy:settings file="ivysettings.xml"/>
3      <ivy:retrieve/>
4  </target>
```

Kuva 9. Apache Ivy build.xml konfiguraatio

Kun Kuvan 9 mukainen lisäys on tehty, voi komentokehotteella antaa komennon, ant resolve, joka käyttää ivysettings.xml-tiedoston määrittämiä koodisäilöjä, repository, mahdollisten riippuvuuksien ratkomiseen. Riippuvuuksia voi määritellä ivy.xml-tiedostoon. Ivy on yhteensopiva Maven-säilöjen kanssa.

4 RAJAPINNAT JA TIETOMALLIT

Yleisesti ohjelmoinnissa rajapinnalla tarkoitetaan sopimusta käyttäjien ja tuottajien välillä. Tuottaja ilmoittaa ja sopii tietyn rajapinnan kautta, että tietyllä tavalla voi toteuttaa jonkin määrätyn toiminnon. Käyttäjän ei tarvitse tietää miten se toteutetaan, ainoastaan miten sitä käytetään. Rajapinnat myös auttavat koodin päivityksessä, koska rajapinnan molemmilla puolilla voidaan tehdä isojakin muutoksia, ilman että toiseen puoleen tarvitsisi tehdä minkäänlaisia muutoksia. Tässä käsitellyt rajapinnat ovat www-sovelluspalveluja, web service. Nämä rajapinnat ovat siis kutsuttavissa internetin yli.

Tietomallilla tarkoitetaan tässä opinnäytetyössä järjestelmien tapaa tallentaa itseään kiinnostavat tiedot. Tässä opinnäytetyössä ei ole tarvetta käydä läpi kokonaisuudessaan integroitavien järjestelmien tietomallia tai kaikkia rajapintoja. Ne rajoitetaan vain tämän työn kannalta tärkeisiin tietoihin.

4.1 JIRA API

JIRA tarjoaa erilaisia rajapintoja, joilla voidaan suorittaa toimintoja etätoimintoina ulkoisesta kolmannen osapuolen ohjelmistosta. Rajapintoina on REST, SOAP, XML-RPC ja JSON-RPC. Näistä vain REST rajapintaa suositellaan käytettäväksi (Atlassian. n.d. JIRA SOAP deprecated). REST-rajapinta on suositeltu vaihtoehto, mutta se ei vielä työn alkaessa tarjonnut kaikkia JIRA-ominaisuuksia käytettäväksi. Esimerkiksi projektia ei voinut luoda REST-kutsulla.

JIRAn SOAP-rajapinta tarjoaa kaikki tarvittavat metodit, joilla voi muokata ohjelmallisesti JIRAssa olevia projekteja tai lisätä uusia. Tärkeimpiä JIRAn SOAP-rajapinnan tarjoamia funktioita tämän työn kannalta on muun muassa, createProject-, getProjectByKey-, getPermissionSchemes ja getProjectsNoSchemes- functiot.

Kaikki kutsut JIRA-APIa vasten vaativat voimassa olevan istunnon ja yhtenä parametrina security token -parametrin. Sen saa palautustietojen mukana, kun kutsuu kirjautumisfunktiota.

4.2 PlanMill API

PlanMill API on REST- ja RPC-pohjainen. Tässä opinnäytetyössä käytetään REST-rajapintaa. REST-kutsut palauttavat XML-pohjaisen listauksen kaikesta materiaalista, jota kyseinen haku koskee. API-haut vaativat parametriksi aina token-elementin, joka saadaan PlanMill-palvelimelta, kun on ohjelmallisesti kirjauduttu sisään oikeilla käyttäjätiedoilla. PlanMill REST-kutsujen mukaan pitää aina liittää userid- ja userauth-parametrit ja niiden arvot jotka löytyvät PlanMill-järjestelmästä, kun kirjautuu internet selaimella sisään.

Kun suoritetaan hakuja PlanMill kantaa vasten, niin haut suodatetaan haun mukana annetun käyttäjän asetuksien ja kutsun suodatuksien mukaisesti. Jos esimerkiksi käyttäjän projekti-näkymän asetuksissa on suodatus, joka näyttää vain 100 ensimmäistä projektia, niin haku joka yrittää noutaa kaikki

projektit, suodattuu tämän käyttäjän asetuksien mukaisesti ensiksi ja vaikka projekteja vaikka 200 kappaletta, niin paluuviestinä tulisi vain ensimmäiset 100 kappaletta. Tämä on erittäin tärkeää asia muistaa, koska haun tulokset eivät välttämättä vastaa sitä mitä odottaa. Suositeltavaa olisikin käyttää nimenomaan www-sovelluspalveluja varten luotua ja konfiguroitua käyttäjää, jolla ei olisi mitään turhia suodatuksia järjestelmässä. Hakuun saa lisättyä suodatuksia samoin kuin selaimella.

PlanMill-järjestelmässä kaikilla käyttäjillä on käyttäjätaso, joka määrittelee mitä he voivat nähdä, muokata, lisätä tai poistaa. Aikaraporttien kirjaaminen on tässä työssä tarvittavista komennoista ainoa, joka vaatii power-roolin, jotta kirjaaminen onnistuu.

4.3 Tempo API

Tempo API tarjoaa aikakirjaukset haettavaksi REST-rajapinnan välityksellä. Se on suunniteltu koneiden ja palveluiden väliselle keskustelulle ja se ohittaa kaikki JIRAn käyttäjä hierarkiat. Kyselyt siis suoritetaan aina pääkäyttäjä tason käyttäjänä. Jotta kyselyitä ei voisi suorittaa kuka tahansa, niin kyselyn mukaan tulee liittää TempoApiToken-merkkijono, jonka JIRAn pääkäyttäjä pystyy generoimaan. JIRAssa pääkäyttäjä voi määrittää mistä IP-osoitteista REST-kyselyitä voi suorittaa. Tässä työssä käytetään vain kahta APIn tarjoamaa palvelua, GetWorklogs ja UpdateWorklogs.

GetWorklogs on palvelu jonka avulla haetaan kaikki Tempo-aikakirjaukset. Hakutulokseen on mahdollista vaikuttaa erilaisilla suodattimilla, jotka rajaavat palautettavien aikakirjausten määrää. Yksi suodatin, joka helpottaa työn tekemistä huomattavasti on diffOnly-parametri. Jos sen syöttää mukaan hakuun ja asettaa arvoksi tosi, true, niin se palauttaa vain ne aikakirjaukset, jotka on päivittynyt sen jälkeen kun UpdateWorklogs-palvelua on edellisen kerran kutsuttu.

UpdateWorklogs palvelulla voidaan kuitata haetut ja hyväksytyt aikakirjaukset. Jos aikakirjausta ei kuittaa, niin GetWorklogs palauttaa sen taas seuraavan kyselyn aikana. (TM Software. n.d. Tempo Servlet Manual.)

4.4 Tietomallit ja niiden eroavaisuudet

Koska tarkoituksena on luoda PlanMill-tietojen mukaan uusi projekti JIRAan, niin tarvittava tietomalli ei ole kovin iso, suhteessa JIRAn kokonaiseen tietomalliin. Projektin luomista varten kerätään ja luodaan vain välttämättömät tiedot. Näihin tietoihin kuuluvat projektin nimi, projektipäällikkö ja avain-tunnus.

PlanMill-tietomalli on myös erittäin laaja, koska palvelu kattaa projektin hallinnan alusta loppuun. Tätä työtä koskien, se voidaan kuitenkin rajata hyvin pieneen osa-alueeseen. PlanMill tiedoista ainoastaan projekti-data ja niistäkin vain yleistiedot ovat kiinnostavia työn kannalta.

PlanMillin ja JIRAn tietomalleissa on suuriakin eroja, koska ne käsittelevät vain osittain samoja osa-alueita. Esimerkiksi molemmat ylläpitävät tietoja projekteista, vastuullisista, työmääristä ja tehtävistä. PlanMill kuitenkin eri-

koistuu laskutukseen ja ajan seuraamiseen kun taas JIRA on ongelma/tehtävä-keskeinen ohjelmisto, jossa kirjataan ongelmia ja niiden aiheuttamaa työtä.

4.5 REST

REST-on arkkitehtuurinen tyyli, ei protokolla. RESTin tärkeimpiä ominaisuuksia on löyhä riippuvuus keskenään kommunikoivien järjestelmien välillä. Se on tilaton järjestelmä, joka tarkoittaa sitä, että kaikki tietyn kyselyn vaatimat tiedot, täytyy aina kuljettaa pyyntöjen mukana, esimerkiksi kirjautumis- tai käyttöoikeustiedot. REST-arkkitehtuurimalli perustuu HTTP-protokollaan.

```
1 http://www.mydomain.fi/catalog/item/123
2
3 http://www.mydomain.fi/catalog/item?type=figurine&color=red
```

Kuva 10. REST-kutsu esimerkkejä

RESTin tärkein osa on resurssit. Resurssit erotellaan toisistaan käyttämällä URI, Uniform Resource Identifier, osoitetta. Resurssi voi tarkoittaa mitä tahansa verkossa saatavilla olevaa asiaa, yleisimpinä HTML-dokumentit eli tavalliset verkkosivut. Sillä voidaan myös tarkoittaa video-, musiikki-, teksti- tiedostoja tai minkälaista tiedostoa tahansa. Se voi myös olla vain osoite johonkin palveluun, jossa voi suorittaa kyselyjä.

Tunnetuin REST-arkkitehtuurilla toteutettu palvelu on internet. Resursseina internetissä on pääasiassa HTML sivut. Selain lähettää pyynnön haluttuun URL-osoitteeseen ja saa vastaukseksi esimerkiksi näytettävän verkkosivun. (Dr. M. Elkstein. n.d. Learn REST.) (Roy T. Fielding. 2000. Representational State Transfer (REST).)

4.6 RPC

RPC, remote procedure call, on tekniikka jolla kehittäjät voivat kutsua etäjärjestelmän funktioita, välittämättä verkkoprotokollista tai yhteyden toteutusmallista. Sen sijaan, että lähetettäisiin kirjoitusvirhealtis REST-tai JSON-kutsu, niin voidaan koodiin luoda olio, joka kuvastaa järjestelmän tietomallia ja kutsua sen sisäisiä funktioita kuin ne olisivat lokaaleja. Niiden kutsuminen kuitenkin lähettäisi viestin verkon yli etäjärjestelmään. (Dave Marshall. 1999. What Is RPC?)

Esimerkiksi JIRAn API mahdollistaa monien RPC kutsujen tekemisen ja monet vaativat token-parametrin. Sen pyytämistä varten kutsutaan login-funktiota JIRA-järjestelmästä. Sen sijaan, että luotaisiin REST- tai JSON-pohjainen URI, niin kutsutaan jiraServiceConnection olion login funktiota ja API hoitaa itse yhteyden ja URIn generoimisen (Kuva 11.)

```
1  token = jiraServiceConnection().login(userName, password);
2
3  allProjects = jiraSoapService.getProjectsNoSchemes(token);
4
5  remoteSchemes = jiraSoapService.getSecuritySchemes(token);
```

Kuva 11. RPC-esimerkkejä

5 INTEGRAATION TOTEUTUS

Integraation toteutus aloitettiin tyhjästä projektista ja se on toteutettu mahdollisimman suljetussa ympäristössä. JBoss SOA-P ja JIRA sijaitsevat samassa fyysisessä työkoneessa ja koska PlanMill on SaaS-palvelu, sitä ei tarvitse asentaa tai konfiguroida. Järjestelmien asennusta ei tässä työssä käsitellä.

Työssä on kaksi selkeästi toisistaan eroteltavissa olevaa osiota. PlanMill-projektien haku ja vieminen JIRAn ja Tempo-aikakirjausten haku ja vienti PlanMilliin.

Työn ensimmäisen osan toteutus aloitettiin luomalla palvelu, joka ajastimen herättämä suorittaa tietyn aikavälein projektitietojen päivittämisen. Prosessin aikana haetaan PlanMill-järjestelmästä projektitiedot käyttämällä REST-kyselyä. Kyselyn tuloksista etsitään JIRAsta puuttuvat projektit, vertaamalla niitä nykyisiin JIRAssa oleviin projekteihin. Jos uusia projekteja löytyi, niin niistä generoidaan uusi projekti JIRA-järjestelmään. Tämä prosessi kuvataan tarkemmin kohdissa 5.2 – 5.3.

Työn toinen osuus toteutettiin luomalla palvelu, joka ajastimen herättämänä suorittaa tietyn aikavälein aikakirjaus-tietojen kirjaamisen PlanMill-järjestelmään. Aikakirjaukset haetaan Tempo-liitännäiseltä käyttäen REST-kyselyä. Kyselyn tuloksen mukana tulleet aikakirjaukset, kirjataan PlanMill-järjestelmään käyttäen PlanMillin REST-APIa. Tämä prosessi kuvataan tarkemmin kodissa 5.4 – 5.5.

5.1 Tietomallien muunnoksen toteutus

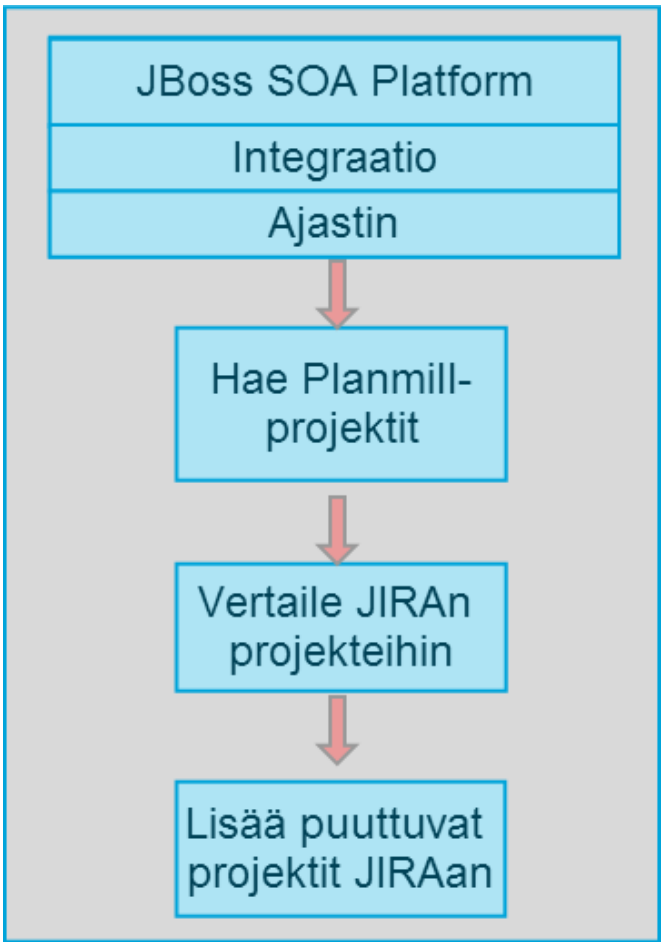
Molempia tietomalleja varten on omat luokat, joiden avulla on mahdollista luoda kyseisen tietomallin mukainen projekti. Luokista löytyy kaikki ominaisuudet joita tarvitaan PlanMill- tai JIRA-luokan vertailuun ja luomiseen.

Kun järjestelmä on hakenut PlanMill-sovelluksesta projektitiedot, niistä luodaan PlanMill-luokan olioita. Tämän luokan keräämien tietojen pohjalta voi luoda JIRA-luokan olion. Tarvittavien tietojen määrä on niin vähäinen, että muunnos on helppo tehdä vain luomalla luokalle rakentaja, joka haluaa parametreiksi kaikki PlanMill-olion tiedot. Tämän jälkeen juuri luotua JIRA-oliota voi käyttää, kun luodaan konkreettista projektia JIRA-järjestelmään.

5.2 Projektien haku ja vertailu PlanMill-järjestelmästä

PlanMill-API ei tarjoa mitään tapahtumia ulospäin, joten projektien lisäystä ei voi mitenkään kuunnella. Kuuntelun sijasta toteutettiin kyselijä-komponentti, joka aika ajoin pyytäisi kaikki projektit ja vertailisi niitä JIRAn tietokannassa oleviin projekteihin.

PlanMill projektien haku palvelun tehtävä on kysellä PlanMill-sovellukselta, mitä projekteja siellä on tallennettuna. Jos projekteja löytyi, niistä luodaan XML-tiedosto, joka jaetaan pienempiin, yhden projektin kokoisiin osioihin ja käynnistetään toinen palvelu, johon yksittäiset projektit lähetetään. Tämän toisen palvelun tarkoitus on lisätä puuttuvat projektit JIRAn tietokantaan. Projektit käydään yksitellen läpi ja nimen avulla tarkistetaan JIRAn tietokannasta, onko siellä kyseinen projekti jo tallennettuna. Jos projektia ei ole niin sellainen luodaan. Kuva 12 kuvastaa prosessia joka käsitellään kohtien 5.2 ja 5.3 aikana.



Kuva 12. Projektien lisäys JIRAan

Integraation toimintaperiaate on ajatustasolla yksinkertainen. Kun käyttäjä lisää PlanMill-järjestelmään uuden projektin, niin seuraavan kerran kun ajastimen heräte toiminto suoritetaan, käynnistyy projektien haku ja uusi projekti havaitaan ja generoidaan JIRAan. Tämä prosessi ei vaadi käyttäjältä mitään erikoistoimenpiteitä.

5.2.1 Ajastimen luonti

Ajastimien käyttö on helppoa ja yksinkertaista. Ensiksi luodaan ajastin, joka tietyn ajan välein herättää sen kuuntelijat. Ajastimen jälkeen luodaan kuuntelija, joka kuuntelee tiettyä ajastinta. Nämä ovat vain XML-määrityksiä, eivätkä vaadi vielä ohjelmoimista. Näiden jälkeen toteutetaan tapahtumakäsittelijä, event handler, ajastimen kuuntelijalle.

Ensimmäinen vaihe kyselyn suorittamisessa, on ajastimen luonti, jotta saadaan määritettävä, tasainen aikaväli kyselylle. JBoss ESB:ssä on oletuksena ajastin-tarjoaja. Tällä voidaan määritellä heräte, joka tapahtuu tietyn ajan välein. Ajastin määritellään jboss-esb.xml-tiedostoon Kuvan 13 esimerkin mukaisesti.

```
<schedule-provider name="PlanmillUpdateSchedulerProvider">
        <simple-schedule frequency="3600" frequencyUnits="seconds"
            scheduleid="Planmill-UpdateScheduler"/>
</schedule-provider>
```

Kuva 13. Ajastimen määritys jboss-esb.xml-tiedostossa

Schedule-provider on ajastintarjoaja, jolle määritellään nimeksi PlanMillUpdateSchedulerProvider. Sille määritellään heräte aikaväliksi 3600 sekuntia, yksi tunti, jonka välein kutsutaan tätä ajastinta kuuntelevia kuuntelijoita. Tämä aika voisi olla mitä vain, mutta testaustarkoitukseen olen valinnut tämän. FrequencyUnits määrittää mitä yksikköä frequency parametrilla tarkoitetaan. Viimeiseksi annetaan ajastimen id, jolla voidaan linkittää tietty kuuntelija kuuntelemaan nimenomaan tätä tarjoajaa.

5.2.2 Ajastin-kuuntelijan luonti

Ajastettu kuuntelija on JBossin oletuskuuntelija ja se on tarkoitettu kuuntelemaan ajastettua tarjoajaa. Kuuntelijat voivat olla JBoss:ssa palveluiden alkupisteitä. Kun kuuntelija havaitsee muutoksen, jota se tarkkailee, se voi käynnistää toimintoketjun, joka toteuttaa jonkin tietyn palvelun. Kun edellä luotu ajastimen heräte käynnistyy, huomaa tämä kuuntelija sen ja käynnistää toimintoketjun joka tarkistaa ja suorittaa mahdolliset muutokset JIRA projekteihin.

Kuuntelija-tapahtumakäsittelijä-luokan täytyy toteuttaa ScheduledEventMessageComposer-rajapinta. Kuuntelija voidaan määritellään jboss-esb.xml-tiedostoon Kuvan 14 esimerkin mukaisesti.

```
<scheduled-listenerevent processor="net.ambientia.PlanmillUpdateScheduleEventHandler"
    is-gateway="false"
    name="SchedulerListener"
    scheduleidref="Planmill-UpdateScheduler"/>
```

Kuva 14. Ajastimen kuuntelija määritys jboss-esb-tiedostossa.

Processor parametri on tapahtumakäsittelijäluokka, joka herätetään ajastetusti. Tämä tapahtumakäsittelijä täytyy periyttää joko ScheduledEventLis-

tener luokasta, josta kutsutaan onSchedule funktiota, minkä jälkeen ei reititetä viestiä eteenpäin tai ScheduledEventMessageComposer luokkaa, josta kutsutaan composeMessage funktiota, jonka avulla luodaan viesti, joka reititetään ESB-järjestelmään. Tässä tapauksessa käytetään jälkimmäistä, jotta voimme luoda viestin järjestelmään. Is-gateway määritys ilmaisee onko viesti ESB-järjestelmästä vai ulkopuolelta. Name antaa nimen tälle kuuntelijalle.

5.2.3 Ajastimen tapahtumakäsittelijä

PlanMillUpdateScheduleEventHandler-luokan tarkoitus on luoda viesti ja reitittää se toimintoketjuun. Jotta viestin luominen ESB-järjestelmään olisi mahdollista, täytyy luokan periytyä ScheduledEventMessageComposer luokasta. Kun ajastin herättää tapahtumakäsittelijän, niin sen composeMessage funktiota kutsutaan. Tässä käsittelijässä haemme PlanMill-tietokannasta kaikki projektit ja luomme viestin, johon tallennetaan löydetyt projektit ja reititetään viesti eteenpäin. Tämä tapahtumakäsittelijä käynnistää

PlanMill-projektien haku tapahtuu tätä työtä varten toteutetulla PlanMillClient-luokalla. Se tarjoaa getProjects-metodin joka suorittaa kyselyn PlanMill-tietokantaan ja palauttaa kaikki löydetyt projektit. Kuvassa 15 on esimerkkikoodia projektien hausta.

```
1  PlanmillClient planmillClient = new PlanmillClient();
2  String planmillProjects = planmillClient.getProjects();
3  message.getBody().add("planmillString", planmillProjects);
4  return message;
```

Kuva 15. Projektien haku ohjelmallisesti PlanMillClient-luokalla ja niiden tallentaminen viestiin.

PlanMillClient-komennot palauttavat projektit aina merkkijonona, jonka sisältö on XML-muodossa. Se merkkijono liitetään ESB-viestiin ja sen jälkeen viesti laitetaan PlanMill_Project_Fetch_Service-palvelun toimintoketjuun, jossa muut toiminnot prosessoivat sitä eteenpäin.

5.2.4 Projekti-merkkijonon käsittely

Ensimmäinen toiminto PlanMill_Project_Fetch_Service-palvelussa on StringToXML (Kuva 16). Tätä toimintoa käytetään myöhemmässäkin vaiheessa, TempoService-palvelussa. Sen tarkoituksena on muuttaa PlanMill:stä saatu merkkijono muuttuja oikeaksi XML-tiedostoksi. Tämä on yksinkertaista ja Javan oletusluokat selviävät tästä helposti.

```
1  xmlString = (String)message.getBody().get("planmillString");
2  result =XmlStringToXmlDocConverter.stringToXml(xmlString);
3  message.getBody().remove("planmillString");
4  message.getBody().add("planmillXml", result);
5  return message;
```

Kuva 16. PlanMill-projektitietojen muunto XML-dokumentiksi

Ajastimen tapahtumakäsittelijä sijoitti PlanMillString nimisen merkkijonon viestiin, joten sen voi sieltä hakea message.getBody().get()-funktiolla merkkijono-muuttujaan. XmlStringToXmlDocConverter funktio (Kuva 17), ottaa parametriksi XML sisältöisen merkkijonon ja palauttaa XML-tiedoston, tarkemmin org.w3c.dom.Document-tyyppisen XML-dokumentin.

```
public static Document stringToXml(String xmlString){
        Document result = null;
        DocumentBuilderFactoryfactory = DocumentBuilderFactory.newInstance();
        InputSource source = new InputSource(new StringReader(xmlString));
        DocumentBuilder builder = factory.newDocumentBuilder();
        result = builder.parse(source);
    return result;
}
```

Kuva 17. XML-merkkijonon muunnos XML-tiedostoksi

Muunnoksen jälkeen, merkkijono poistetaan ESB-viestistä ja sen tilalle tallennetaan PlanMillXml-nimellä, juuri luotu XML-dokumentti. Return message-koodirivi laittaa viestin eteenpäin toimintoketjussa.

5.2.5 Projektien jakaminen ja siirto toiseen toimintoketjuun

Projekteja voi olla satoja, joten helpoin tapa hallita niitä, on erotella ne omiksi viesteikseen ja lähettää yksikerrallaan eteenpäin, jotta seuraavat toiminnot voivat käsitellä projekteja yksitellen.

Projektien jakamisessa käytetään hyödyksi JBossin tarjoamaa ServiceInvoker luokkaa. Sen avulla voimme herättää ohjelmallisesti toisen palvelun ja lähettää viestin sinne käsiteltäväksi. PlanMill-projekteja varten on luotu PlanMillService-palvelu, jossa projekti ja sen tiedot luetaan talteen ja lähetetään JIRAan tarvittaessa.

Toisen palvelun käynnistäminen aloitetaan luomalla uusi instanssi ServiceInvoker-luokasta. Sen jälkeen luodaan viesti käyttämällä MessageFactory-luokkaa, jonka jälkeen sinne laitetaan PlanMill-projektitiedot ja sitten kutsutaan ServiceInvoker-olion deliverAsync-funktiota, joka lähettää viestin halutulle palvelulle. Tämä prosessi on kuvattu koodiesimerkillä Kuvassa 18.

```
ServiceInvoker invoker = new  ServiceInvoker("category","service");
Message payload = MessageFactory.getInstance().getMessage();
payload.getBody().add("project",project);
invoker.deliverAsync(payload);
```

Kuva 18. Palvelun käynnistäminen ServiceInvoker-luokalla.

DeliverAsync-funktio on asynkroninen eli se ei jää odottamaan, että viesti on käsitelty, vaan jatkaa suoritusta olettaen, että se käsitellään ajallaan.

5.3 PlanMill-projektien käsittely-palvelu

PlanMill_Project_Processor_Service-palvelun tarkoitus on yksittäisten Planmil-projekti viestien vastaanottaminen ja niiden instantiointi PlanMill-Project-luokan olioiksi. Oliot siirtyvät seuraavan toimintoon jossa niiden pohjalta luodaan JiraProject-olioita. Olioita verrataan JIRAn projekti-tietoihin ja etsitään sieltä puuttuvia projekteja. Jos puuttuvia on, niin ne lisätään tässä vaiheessa JIRA-kantaan. JiraProject-luokka on vastuussa JIRA-projektin avaimen generoinnista, josta on tarkemmin kerrottu kohdassa 5.3.4.

5.3.1 PlanMill-projektien instantiointi

PlanMillProjectAnalyzer-luokka aloittaa PlanMill_Project_Processor_Service-palvelun. Tätä toimintoa suorittaessa, viestin mukana on yksittäisiä PlanMill-projekti XML-elementtejä. Tämä luokka on vastuussa tuon XML-elementin konvertoinnista PlanMillProject-olioksi.

Suurimman työn konvertoinnissa hoitaa PlanMillProject-luokka itsessään. Kun PlanMillProject-olio on luotu, se sijoitetaan viestiin ja suoritus siirtyy seuraavalle toiminnolle.

5.3.2 JIRA-projektien generointi

JiraProjectGenerator on PlanMillService-palvelun seuraava toiminta. Tässä toiminnassa luodaan JiraProject-olio PlanMillProject-olion pohjalta. PlanMill ja JIRA projektien eroja on muun muassa nimeämis-käytännöt ja JIRAn uniikki projektikohtainen avain. Nimen muunnostyön hoitaa Jira-Project-luokka samalla kun oliota instantioidaan.

Kun olio on luotu, otetaan yhteys JIRAn käyttäen JiraClient-luokkaa. Sen avulla voi muun muassa tarkistaa onko JIRAssa jo tietyn niminen projekti olemassa, onko generoitu avain kelvollinen ja luoda JIRAan projektin.

Jos projektia ei juuri luodun JiraProject-olion nimen perusteella JIRAsta löydy, niin sellainen projekti generoidaan JIRAan. Projektin generoimisesta huolehtii JiraClient-luokka.

5.3.3 PlanMillProject-luokka

PlanMillProject-luokka luo java-olion yksittäisestä XML-elementistä, jossa on yhden projektin kaikki tiedot, jotka tulivat aikaisemmin toteutetun getProjects-funktio kutsun avulla. PlanMillProject-luokka ei vastaa täysin PlanMill-tietomallia, vaan sitä on rajattu siten, että vain tämän työn kannalta tärkeät asiat otetaan talteen.

Tärkeimmän ominaisuudet jotka otetaan talteen ovat projektin id, nimi, projektipäällikkö, asiakkaan id tunnus, projektin tila ja asiakkaan nimi.

PlanMillProject-luokalla on joitakin apu-metodeja, jotka auttavat tiedon keräämisessä XML-tiedostosta.

5.3.4 JiraProject-luokka

JiraProject-luokka instantioidaan PlanMillProject-oliolla. Tärkein ero JIRA ja PlanMill projektilla, Ambientian nimeämiskäytäntöjen puolesta, on projektin nimi. JIRA projektin nimi täytyy luoda yhdistäen PlanMill-projektin nimi, PlanMill-projektin asiakkaan nimi ja PlanMill-projektin id. Nämä ominaisuudet täytyy muuntaa muotoon:

Asiakas, Projektin nimi (ID)

Muunnosta varten on funktion, joka ottaa parametreiksi edellä mainitut ominaisuudet ja palauttaa merkkijonon, joka vastaa Ambientian JIRA projektien nimeämiskäytäntöjä. Kuva 19 näyttää esimerkin funktiosta, jolla nimen muutos toteutetaan.

```
1  public String createName(String pmName, String pmCustomer,
2  String pmId){
3      return pmCustomer + ", "+pmName+" ("+pmId+")";
4  }
```

Kuva 19. Funktio createName, jolla luodaan JIRAan kuvaava nimi projektille.

JiraProject-luokalla on tärkeä metodi, joka generoi JIRAan sen sääntöjen mukaisen uniikin avain projektille. Sen generoiminen tapahtuu kaksi-vaiheisesti.

Ensiksi koitetaan luoda avain projektin nimen perusteella. Tämä on hyvä vaihtoehto, koska avaimesta tulee täten loogisempi ja kenties helpommin muistettava. Kuvassa 20 näkyy avain generaattorin toimintaperiaate. Vaikka projektit ovat kaikki samalla tavalla nimettyjä, niin generaattori kykenee luomaan uniikit avaimet, pysyen vielä loogisuuden rajoissa. Loogisuudella tarkoitetaan sitä, että ”TESTIP” on huomattavasti helpompi muistaa ja lausua kuin täysin satunnainen ”OIJFUAASEQ”.

null, Test OpenAPI (564)	TES
null, test.fi (1492)	TEST
null, testi (1501)	TESTI
null, Testiprojekti (301)	TESTIP
null, Testiprojekti 2 (1401)	TESTIPR
null, testitestitesti (1517)	TESTIT
null, Testjj1 (451)	TESTJ
null, Testjj2 (452)	TESTJJ

Kuva 20. Avain generaattorin generoimat avaimet.

Tämä vaihe on toteutettu sillä periaatteella, että aluksi otetaan kaksi ensimmäistä kirjainmerkkiä projektin nimestä. Huomattakoon, että avaimen luonnissa ei käytetä generoitua JIRA-projekti nimeä, vaan alkuperäistä PlanMill-projekti nimeä. Valitut merkit siistitään kielletyistä merkeistä käyttäen

säännöllisiä lausekkeita. Sääntö lausekkeena on "[^a-zA-Z]+", joka käytännössä tarkoittaa, että avain saa koostua vain aakkosista, ei kirjaimista eikä minkäänlaisista erikoismerkeistä.

Siistimisen jälkeen tarkistetaan onko avain tarpeeksi pitkä. Avaimen oikeellisuus tarkistetaan viimekädessä JIRA-palvelulta. Koska API ei tarjoa suoraan tarkistusta avaimen oikeellisuudelle, niin tarkistus toteutetaan hakemalla projektia, avaimella. Jos jokin projekti löytyy, niin avain on silloin käytössä, joten jatketaan generoimista.

Minimi pituus avaimelle JIRA 5:ssa on kaksi merkkiä. Jos se ei ole tarpeeksi pitkä, otetaan projektin nimestä kolme merkkiä ja toistetaan testaukset (Kuva 21). Tätä jatketaan kunnes on löytynyt sääntöjenmukainen avain tai kunnes ylitetään suurin sallittu avaimen pituus, joka on JIRA 5:ssa 10. JIRA 6:ssa tätä voi pääkäyttäjä muokata JIRAn asetuksista. Tämä huomioiden JiraProject-luokkaan on luotu muuttuja keyCharLimit, joka on oletuksena kymmenen, mutta sen voi muuttaa JIRA 6:sta varten isommaksi. Tämän muuttaminen vaikuttaa suoraan avaimen generoimis-metodiin.

```
1  private boolean createKeyFromInitialLetters(int minLength) {
2      int letterPickCount = 1;
3      while(letterPickCount < keyCharLimit){
4          String tempKey = getInitialLettersFromProjectName(letterPickCount);
5          tempKey = CleanStringFromIllegalCharacters(tempKey);
6          if(tempKey.length() >= minLength){
7              setKey(tempKey);
8              return true;
9          }
10         letterPickCount++;
11     }
12     return false;
13 }
```

Kuva 21. Avaimen generoiminen projektinimen perusteella.

Jos avaimen generoiminen ei onnistu projektin nimen perusteella, niin sen jälkeen se generoidaan satunnaisista kirjaimista maksimi pituudella. Satunnaisuus luodaan kirjaimista jotka ovat kaikki isoja, A:n ja Z:n väliltä.

5.3.5 JiraSoapClient-luokka

Tämän luokan tehtävä on luoda yhteys JIRAan ja muun muassa hakea projekteja, avaimia ja Tempo-työkirjauksia. Se toteuttaa JiraClientInterface-rajapinnan, joka on luotu löyhä riippuvuus ajatuksen kanssa, jotta integraatio tukee uudempiakin yhteystapoja. JiraSoapClient toteuttaa tämän käyttäen SOAP-Apia. Muita mahdollisia olisivat esimerkiksi REST ja XML-RPC. Tässä käytetään SOAPia, koska se oli työtä aloittaessa ainoa API, jonka avulla pystyi luomaan projektin JIRAan.

JIRA SOAP remote API on vanhentunut JIRAn 6 versiossa (Atlassian. n.d. JIRA SOAP deprecated).

JIRAn SOAP-APIn käyttäminen vaatii erilaisia alkutoimenpiteitä. Ensimmäiseksi täytyy luoda JiraSoapServiceServiceLocator-olio. Sen jälkeen kutsutaan sen funktiota, setJirasoapserviceV2EndpointAddress. Parametriksi täytyy antaa osoite JIRAn SOAP-palveluun, esimerkiksi:

```
http://localhost:8080/rpc/soap/jirasoapservice-v2
```

Tämän jälkeen täytyy vielä hakea olio, jolla voi kutsua SOAP-APIa. Sen saa kutsumalla saman JiraSoapServiceServiceLocator-olion toista funktiota, getJirasoapserviceV2, se palauttaa olion, joka on tyyppiä JiraSoapService.

JiraSoapClient luo ja hallitsee yhteyttä JIRAan ja sisältää funktiot uusien projektien luomiselle ja Tempo-liitännäisen ajankirjauksien hakemiselle. JIRAan kirjautuminen, tokenin noutaminen ja projektin luonti tapahtuu SOAPin avulla, mutta Tempo-aikakirjaukset haetaan JBoss Resteasy REST-asiakasohjelmalla.

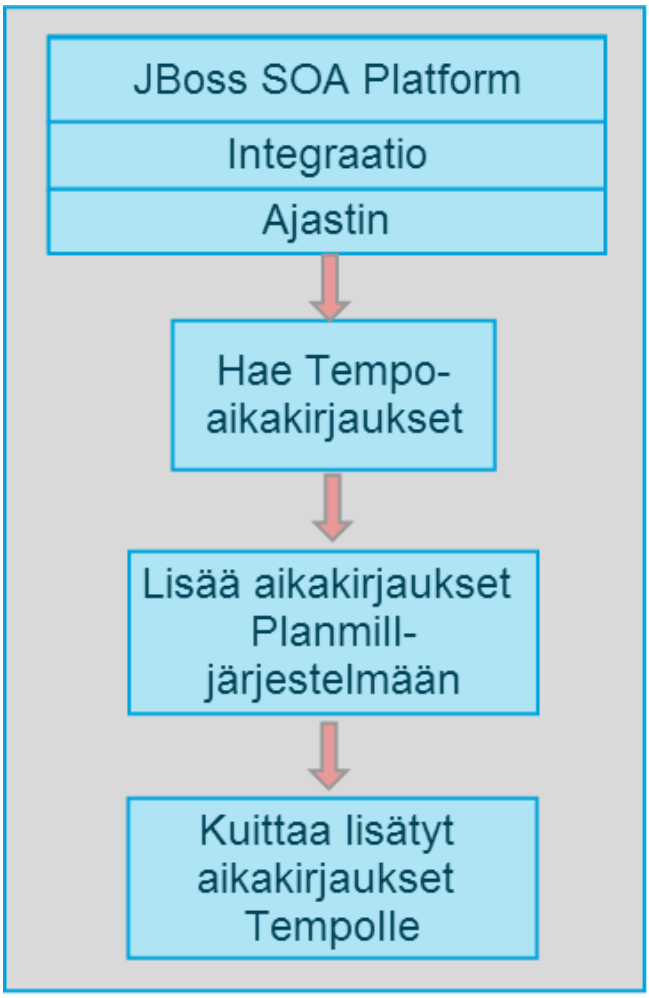
Aina kun tämä luokka instantioidaan, niin se automaattisesti suorittaa login-funktion jolla kirjaudutaan JIRA-palveluun ja haetaan token-muuttuja, jota tarvitaan kaikissa API-kutsuissa. Token sisältää sessio tiedot, joista JIRA tietää, että kyseessä on tunnistettu yhteys.

JiraSoapClient luokka sisältää funktiot projektin luomisessa tarvittaviin oletus skeemojen hakuun. Jokaisella projektilla täytyy olla ainakin oletus skeemat, kun sitä yritetään tallentaa JIRAan. Näitä skeemoja ovat Permission-, Security- ja Notification-skeemat. Vaikka työympäristössä ei ollutkaan muuta kuin oletus-skeemat, niin JiraSoapClient käy kaikki skeemat läpi ja etsii niiden oletusversiot, jotta projektin luonti onnistuisi halutusti, vaikka järjestelmässä olisikin modifioituja skeemoja.

Tempo-ajankirjausten haku on myös tämän luokan tehtävä. Tempo on JIRA liitännäinen, mutta sillä on silti omat palvelunsa, joilla kommunikaatio suoritetaan, joten sitä on käytettävä sen oman APIn kautta. Tempo-API on REST-pohjainen, joten sitä varten tarvitaan REST-toiminnollisuuden mahdollistama luokka. Koska JBoss tarjoaa oman luokan tähän, niin sitä ei ole työtä varten itse toteutettu. REST-asiakasohjelmana toimii JBoss-RESTEasy.

5.4 Tempo aikakirjausten haku

Tempo ei tarjoa oletuksena ulospäin mitään tapahtumia, joita voisi kuunnella. Tätä varten on luotu ajastin, joka ajoittain kysyy Tempo APIn kautta, onko uusia aikakirjauksia tullut. Jos uusia kirjauksia on, ne erotellaan omiksi viesteiksi ja ne lähetetään yksitellen toiseen palveluun jossa niistä luodaan PlanMill-timereport-tyyppinen REST kysely, jolla aika saadaan kirjatuksi myös PlanMill-järjestelmään. Tämän jälkeen varmistetaan että PlanMill sai kirjaukset ja kuitataan Tempolle, että kyseiset aikakirjaukset saatiin. Kuva 22 kuvaa tämän osion prosessia.



Kuva 22. Aikaraportin automaattinen generointi PlanMill-järjestelmään

5.4.1 Ajastimen luonti

TempoWorklogFetch-Scheduler-ajastimen luominen Tempo-liitännäisen uusia aikakirjauksia varten, on aivan sama prosessi kuin kohdassa 5.2.1. Erotuksena ainoastaan aikamääreet, jolloin ajastin lähettää herätteen kuuntelijoille, joka tälle ajastimelle asetettiin 300 sekuntiin.

5.4.2 Ajastin-Kuuntelijan toteutus

TempoWorklogFetchEventHandler-luokan tarkoitus on ajoittain hakea päivitetyt Tempo-aikakirjaukset. Päivitetyillä tarkoitetaan tässä sellaisia kirjauksia, jotka ovat muuttuneet sen jälkeen, kun niitä on edellisen kerran haettu. Jotta ESB-viestin luominen olisi mahdollista, täytyy luokan toteuttaa ScheduledEventMessageComposer-rajapinta. Se myös kuuntelee TempoWorklogFetch-Scheduler-ajastinta ja kun ajastimeen määritetty aika on täynnä, aloittaa se toiminnan.

JiraSoapClient-luokka hoitaa aikakirjausten hakemisen. Sillä on getUpdatedTempoWorkLogs-metodi joka palauttaa merkkijono tyyppisen XML-muotoisen aikakirjaus kokoelman. Kyselyn URL muodostuu JIRAn osoit-

teesta, Tempo-liitännäisen palvelun osoitteesta, aikakirjaus-hakusuodattimista ja TempoApiToken merkkijonosta. Kyselyssä halutaan varmistaa, että vanhat, jo merkatut aikakirjaukset jäävät haun ulkopuolelle. Kun hakuun lisää parametrin, diffOnly=true, niin se suodattaa kuitatut kirjaukset pois automaattisesti. URLn kasaaminen ja kyselyn toteutus voisi näyttää Kuvan 23 mukaiselta.

```
String tempoSearchString = "http://localhost:8080"
+"/plugins/servlet/tempo-getWorklog/?format=xml&validOnly=true&addIssueDetails=true"
+"&tempoApiToken=91918a53-aa2b-49c3-b811-f9ed6fdcd0fc";

request = new ClientRequest(tempoSearchString);
request.accept("application/xml");
ClientResponse<String> response = request.get(String.class);
String response = response.getEntity(String.class);
```

Kuva 23. Tempo-aikakirjausten kysely

Saatuaan päivitetyt Tempo-aikakirjaukset, niistä tehdään merkkijono-muuttuja ja se lisätään ESB-viestiin.

5.4.3 Aikakirjausten jakaminen ja siirto toiseen toimintoketjuun

Aikakirjaukset jaetaan XML-elementtien mukaisesti omiin osioihinsa. Jokainen Worklog-elementti sisältää kaiken tiedon yksittäisestä aikakirjauksesta, kuten Kuvasta 24 näkee. Nämä elementit lähetetään toiseen palveluun yksikerrallaan.

```
<worklog>
   <worklog_id>46445</worklog_id>
   <issue_id>13189</issue_id>
   <issue_key>CLOUD-18</issue_key>
   <hours>8.0</hours>
   <work_date>2011-10-11</work_date>
   <username>erica</username>
   <staff_id>2410724289</staff_id>
   <billing_key>6</billing_key>
   <billing_attributes/>
   <activity_id>v10444</activity_id>
   <activity_name>CloudBay Sprint 4</activity_name>
   <work_description>Review</work_description>
   <parent_key/>
   <reporter>john</reporter>
   <external_id/>
   <external_tstamp/>
   <hash_value>dc11dffc091fcc72e7358067a9488fa1e31ce314</hash_value>
</worklog>
```

Kuva 24. Malli worklog-elementti

Toisen palvelun käynnistäminen koodista onnistuu ServiceInvoker-instanssilla. Ensiksi täytyy luoda uusi Message-olio, johon tallennetaan yksi Worklog-elementti, merkkijono tyyppisenä. Sen jälkeen kutsutaan toista palvelua, joka on suunniteltu käsittelemään yksittäisiä Worklog-elementtejä. Sitä kutsutaan asynkronisesti ja parametriksi annetaan juuri luotu viesti.

5.5 Tempo-aikakirjausten käsittely

Tämän palvelun tarkoitus on suorittaa toimintoja yksittäiselle aikakirjaukselle. Aikakirjaukset ovat tässä pisteessä Worklog-elementtejä, jotka sisältävät kaiken tiedon kyseisestä aikakirjauksesta. Tiedoista instantioidaan TempoWorklog-olio, jotta tiedon käsittely olisi helpompaa, verrattuna XML-tiedostoon. PlanMillClient-luokan avulla aikakirjaus lähetetään PlanMill-palvelulle ja onnistunut kirjaus kuitataan vielä Tempo-liitännäiselle.

5.5.1 Aikakirjausten kirjaaminen PlanMill-TimeReport palveluun

Tempo-aikakirjaukset saapuvat TempoWorklogProcessor-toiminnolle yksitellen, jotta niiden hallinnoiminen olisi helpompaa. Aikakirjaukset ovat viestissä merkkijono tyyppisenä, XML-mallisena tietona. Niistä instantioidaan TempoWorklog-luokan olioita, jotta niitä on helpompi käsitellä. PlanMillClient-luokka sisältää funktion, SendTimeReport, jonka avulla aikakirjauksen lähettäminen PlanMill-palveluun on yksinkertaista. Se vaatii parametreiksi merkkauksen ajankohdan, keston tunneissa, kommentin, laskustatuksen, henkilön id:n ja tehtävän id:n. Nämä tiedot kerätään TempoWorklog-oliolta.

Koska työtä on pitänyt rajata, niin henkilön ja tehtävän id:tä ei kerätä talteen dynaamisesti. Nämä tiedot on suoraan määritelty luokan tietoihin, jotta aikakirjaus olisi mahdollista. Näiden tietojen hankkiminen dynaamisesti olisi kuitenkin mahdollista.

5.5.2 Aikakirjausten kuittaaminen Tempo-liitännäiselle

Kun aikakirjaus on syötetty PlanMill-järjestelmään, niin se pitää vielä kuitata Tempo-liitännäiselle. Tämä täytyy tehdä sen takia, että ensi kerralla kun Tempolta haetaan aikakirjauksia, niin se ei anna samoja, jo kirjattuja tietoja.

Kuittaus tehdään lähettämällä REST-kutsu Tempolle. Se on sovitun mallinen kutsu updateWorklog-palveluun. Se sisältää TempoApiKey-parametrin, jonka avulla todennetaan kutsun oikeus palveluun ja worklogs-parametrin, jonka arvona on XML-dokumentti, jossa on kuitattavien worklogien tiedot.

Kuittauksen jälkeen palvelun suoritus pysähtyy kunnes ajastin herättää sen uudelleen.

6 YHTEENVETO

Työn tarkoituksena oli toteuttaa proof of concept-tyyppinen sovellus, joka integroi JIRA- ja PlanMill-järjestelmät. Integraation tarkoituksena oli automatisoida uusien PlanMill-projektien generoiminen JIRAa ja uusien työaikakirjausten syöttäminen JIRAsta PlanMill-järjestelmään. Integrointiin tuli käyttää JBoss SOA Platformia. Sovelluksen toteutus onnistui ja se suoriutuu vaadituista toimista.

Työ saatiin toteutettua ja mitään ongelmia täysimittaisen integraation toteutukseen ei havaittu. Varsinaisia ongelmia, ei työn aikana esiintynyt. Pientä päänvaivaa aiheutui PlanMill-hakutuloksien suodatus järjestelmä, jossa REST-kutsuja suodatetaan ensisijaisesti kutsun parametrina annetun käyttäjän selain suodatuksien mukaan.

Työn teossa haastavimmaksi osa-alueeksi osoittautui JBoss SOA Platformin hyödyntäminen työssä. Koska aikaisempaa kokemusta ei kyseisestä järjestelmästä ollut, niin alussa ei ollut selkeää suuntaa, miten lähteä liikkeelle. Toinen selkeästi haastavampi asia oli JIRAn APIn tutkiminen. Työn alkaessa tutkittiin eri lähteistä, pääosin blogeista, tietoa JIRA SOAP-asiakasohjelman tekoon. Konkreettista tietoa siitä, miten JIRAa voi etänä kutsua ohjelmallisesti, oli erittäin vähän saatavilla.

Työn aikana olen oppinut valtavasti uutta järjestelmäintegraatioista. Ennen työn aloittamista, minulla ei asiasta ollut minkäänlaista kokemusta. JIRA- ja PlanMill-järjestelmien käytön myötä, olen oppinut perusteet niiden toiminnasta sekä käyttötarkoituksista. Myös molempien järjestelmien rajapinnat ovat tulleet tutuiksi. Eniten uutta asiaa on varmasti tarjonnut JBoss SOA Platform. Työtä varten ei käytetty kuin murto-osaa sen tarjoamista ominaisuuksista. Tulevaisuudessa tulen varmasti tutustumaan siihen tarkemmin ja sen tarjoamiin mahdollisuuksiin.

7 LÄHTEET

Atlassian, n.d. Atlassian company. Viitattu 3.4.2013.
http://www.atlassian.com/company

Atlassian. n.d. Atlassian customers. Viitattu 3.4.2013.
http://www.atlassian.com/company/customers

Atlassian. n.d. JIRA SOAP deprecated. Viitattu 16.11.2013.
https://developer.atlassian.com/display/JIRADEV/Creating+a+JIRA+SOAP+Client

Atlassian. 2013. Documentation for JIRA 6.0. Viitattu 30.11.2013.
http://downloads.atlassian.com/software/jira/downloads/documentation/JIRA061_Documentation_PDF-121013-2021-40.pdf

Atlassian. n.d. Open Source Project License Request. Viitattu 12.12.2013.
https://www.atlassian.com/software/views/open-source-license-request

Atlassian. n.d. Marketplace. Viitattu 12.12.2013.
https://marketplace.atlassian.com/plugins/app/jira/popular?cost=marketplace

Atlassian. n.d. JIRA pricing. Viitattu 12.12.2013.
https://www.atlassian.com/purchase/product/jira

Dave Marshall. 1999. What Is RPC? Viitattu 4.12.2013.
http://www.cs.cf.ac.uk/Dave/C/node33.html

Dr. M. Elkstein. n.d. Learn REST. Viitattu 1.12.2013.
http://rest.elkstein.org/

JBoss ESB Beginner's guide. Viitattu 29.11.2013.
DiMaggio, L., Conner, K., Magesh, K. B. & Cunningham, T. 2012.
JBoss ESB Beginner's guide. Packt, UK

Mulesoft. n.d. Enterprise service bus. Viitattu 18.12.2013.
http://www.mulesoft.org/what-esb

PlanMill Oy, 2005. Kannattavuutta ja kilpailukykyä projektiliiketoimintaan. Viitattu 9.12.2013.
http://www.platinumpartners.fi/uploads/PlanMill_&_kannattavuutta_projektiliiketoimintaan_Fin.pdf

Roy T. Fielding. 2000. Representational State Transfer (REST). Viitattu 1.12.2013.
http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Red Hat. n.d. JBoss SOA-P Administrator guide. Viitattu 1.12.2013.
https://access.redhat.com/site/documentation/en-US/JBoss_Enterprise_SOA_Platform/5/html/Administration_Guide/chap-Preface.html

Red Hat. n.d. JBoss Enterprise Application Platform. Viitattu 10.12.2013 https://access.redhat.com/site/documentation/en-US/JBoss_Enterprise_Application_Platform/5/pdf/Getting_Started_Guide/JBoss_Enterprise_Application_Platform-5-Getting_Started_Guide-en-US.pdf

Red Hat. n.d. JBossESB programmers guide. Viitattu 14.12.2013 http://docs.jboss.org/jbossesb/docs/4.10/manuals/html/Programmers_Guide/index.html#d0e56

Red Hat. n.d. JBoss SOA Install and Configuration. Viitattu 14.12.2013 https://access.redhat.com/site/documentation/en-US/JBoss_Enterprise_SOA_Platform/5/pdf/Installation_and_Configuration_Guide/JBoss_Enterprise_SOA_Platform-5-Installation_and_Configuration_Guide-en-US.pdf

The Apache Software Foundation. n.d. Apache Ant FAQ. Viitattu 14.12.2013 http://ant.apache.org/faq.html

TM Software. n.d. Tempo Servlet Manual. Viitattu 2.12.2013 https://tempoplugin.jira.com/wiki/display/TEMPO/Tempo+Servlet+Manual

TM Software. 2011. The Tempo times. Viitattu 8.12.2013 http://blog.tempoplugin.com/wp-content/uploads/2011/11/the-tempo-times-no1.pdf