TEKNILLINEN KORKEAKOULU
Elektroniikan, tietoliikenteen ja automaation tiedekunta
Tietoliikenne- ja tietoverkkotekniikan laitos

Tuomas Tallqvist

Automatisaation etujen analysointi liiketoiminnan ennustamispalvelun taustatoiminnoissa

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi
diplomi-insinöörin tutkintoa varten Espoossa 27.2.2008

Työn valvoja: Professori Heikki Saikkonen
Työn ohjaaja: Diplomi-insinööri Jani Sauhke

Tiivistelmä

Tekijä: Tuomas Tallqvist

Työn nimi: Automatisaation etujen analysointi liiketoiminnan ennustamispalvelun taustatoiminnoissa.

Päivämäärä: 27. helmikuuta 2008 **Sivumäärä:** 85

Tiedekunta: Elektroniikan, tietoliikenteen ja automaation tiedekunta

Professuuri: Ohjelmistojärjestelmät

Työn valvoja: Heikki Saikkonen, Professori

Työn ohjaaja: Jani Sauhke, Diplomi-insinööri

Tiivistelmän teksti: Diplomityön kohdejärjestelmä on ASP-mallilla toimiva palvelu. Palvelu tarjoaa myynnin ennustamistoimintoja päivittäis- ja kappaletavarateollisuudessa toimiville yrityksille. Palvelun käyttäjät voivat palvelun avulla tehdä esimerkiksi myynnin ennusteita käyttäen hyväksi ainutlaatuista menetelmää.

Palvelun tarkoituksenmukainen käyttö vaatii, että rutiininomaiset taustatoiminnot suoritetaan säännöllisesti, esimerkiksi viikoittain. Tällä hetkellä esimerkiksi myynnin asiakkaan toteutuneen myyntitiedon tuominen järjestelmään suoritetaan manuaalisesti erilaisilla tietokantaskripteillä suoraan tietokantatasolla. Manuaalinen työ vaatii paljon aikaa ja on virhealtista. Diplomityön tavoitteena on analysoida mahdollisuuksia manuaalisen työn vähentämiseksi taustatoiminnoissa, esittää ratkaisuehdotus automatisaatiomoduulin toteutukselle, toteuttaa ratkaisu ja analysoida uuden ratkaisun tuottamia hyötyjä liiketoimintavaatimusten pohjalta.

Automatisaatioratkaisun toteuttamiseen käytetään samoja ohjelmistokehyksiä ja komponentteja, joilla varsinainen järjestelmä on toteutettu. Järjestelmän runko on toteutettu J2EE ohjelmistokehitysalusta Jboss-palvelimelle. Käyttöliittymä toteutetaan internet-selaimelle käyttäen standarditekniikoita, kuten xhtml:a ja css:ia. Työn tuloksena syntyy automatisaatio- ja ajastusmoduuli, jolla voidaan hallita automatisointien ajastuksia ja monitoroida automaattisten toimintojen suorituksia, virheitä ja tiloja. Työkalun suunnitellussa oleellisena vaatimuksena on geneerisyys. Moduulin päälle tulee voida rakentaa ajastettavia toimintoja riippumatta niiden yksityiskohtaisesta toiminnallisuudesta. Näin ollen samaa työkalua voidaan käyttää muissa järjestelmän ylläpitotoiminnoissa toteumatiedon siirron lisäksi.

Toteutuksen toimivuutta testataan tutkimalla, miten hyvin automatisaatiomoduuli toimii sen ensisijaisessa käyttötarjoituksessa. Ensisijaisena tarkastelukohteena on manuaalisen työmäärän väheneminen ja virhetilanteista selviämisen nopeus. Geneerisyysvaatimuksen toteutumista analysoidaan uuden toiminnon lisäämisen kompleksisuuden näkökulmasta kooditasolla.

Avainsanat: Ajastus, automatisointi, Jboss, J2EE, EJB ajastuspalvelu

Abstract

Author: Tuomas Tallqvist

Name of the Thesis: Analyzing the benefits achieved by automatization of back-end functions in Business Forecasting service

Date: February 27, 2008 **Number of pages:** 85

Faculty: Faculty of Electronics, Communications and Automation

Professorship: Software Systems

Supervisor: Heikki Saikkonen, Professor

Instructor: Jani Sauhke, M. Sc. (Tech)

Summary text: Target system of this thesis is an ASP service providing business forecasting functions for daily and piece goods industry. Users of the Rank & Share service are able to make forecasts for example total sales using an unique method.

An effective use of the service requires some routine background functions to be executed regularly, as for example weekly basis. At the moment these functions are made manually. For example, customer's sales history data is imported manually executing SQL-scripts in database level. This kind of manual work is time consuming and error prone. The main objective of this thesis is to analyze possibilities to reduce manual work with background functions and implement automatization module that can be used to attain that goal. The advantages of the implemented module will be analyzed and compared with business requirements set to the module.

The automatization module has been implemented by using the same technology used in the existing Rank & Share service. The service is made with the J2EE platform and it runs on Jboss application server. The user interface will be implemented using standard technology, such as xhtml and css. The main result of the thesis is an automatization and scheduling module integrated in the Rank & Share application. The module must provide functions to manage scheduled tasks and monitor executions, errors and states of the scheduled tasks. In addition, the module must be so generic that new task types can be easily implemented into the module. Consequently, the module can be used in all new schedulable administration tasks regardless of their inner behavior.

The Module's primary purpose and goal is to reduce the manual work and speed up the error recovering. The simplicity of adding new task in the system is an indicator of the generic nature of the automatization module. This will be measured in software developers point of view.

Keywords: Scheduling, automatization, J2EE, Jboss, EJB Timer Service

Esipuhe

Tämä diplomityö on tehty Rank & Share Oy:lle vuosina 2007 ja 2008.

Kiitän työkavereita Rank & Share Oy:ssä tuesta ja kannustuksesta, jota olen saanut työn tekemisen aikana. Erityisesti haluan kiittää työn ohjaajaa Jani Sauhketta neuvoista ja ohjeista.

Kiitos työn valvojalle Heikki Saikkoselle joustavasta suhtautumisesta työhön sen tekemisen aikana ja ripeästä toiminnasta työn valmiiksi saattamiseksi.

Kiitos vanhemmille kaikesta tuesta, jonka olen heiltä opiskeluni aikana saanut. Kiitos myös ystäville kaikista niistä hetkistä, joina sain työn teon lomassa rentoutua kanssanne.

Espoossa 27.2.2008

Tuomas Tallqvist

Sisällysluettelo

Lyhenteet

AJAX	Asynchronous JavaScript And XML, tekniikka vuorovaikutteisten verkkosovellusten luomiseen.
API	Application programming interface, ohjelmointirajapinta.
BO	Business Object, J2EE kerrosmallissa sovelluslogiikkakerroksen olio.
CORBA	Common Object Request Broker Architecture, standardi joka määrittelee eri ohjelmointikielillä toteutettujen komponenttien yhteistoiminnan.
CSS	Cascading Style Sheets, www-dokumenttien tyyliohjeiden määrittelykieli.
CSV	Comma-Separated Values, dokumenttimuoto jossa arvot ovat eroteltu puolipisteillä.
DAO	Data Access Object, J2EE kerrosmallissa tiedonkäsittelykerroksen olio.
DTD	Document Type Definition, XML kielten yhteydessä käytetty rakennemäärittelytapa.
DWR	Direct Web Remoting, sovelluskirjasto AJAX-tekniikan ja Java-ohjelmointikielen yhteiskäyttöön.
EJB	Enterprise JavaBeans, komponenttipohjainen arkkitehtuuri palvelinpohjaisten palveluiden toteuttamiseen.
FTP	File Transfer Protocol, tiedostojen siirtoon suunniteltu tiedonsiirtoprotokolla.
HTML	HyperText Markup Language, www-dokumenttien toteuttamiseen käytetty kuvauskieli.
HTTP	HyperText Transfer Protocol, www-dokumenttien siirtoon suunniteltu tiedonsiirtoprotokolla.
J2EE	Java Platform, Enterprise Edition, ohjelmistokehitysalusta monitasorakenteisten Java-sovellusten kehitykseen.
J2SE	Java Platform, Standard Edition, ohjelmistokehitysalusta Java-sovellusten kehitykseen.
JAAS	Java Authentication and Authorization Service, sovellusten käyttäjien autentikointiin ja käyttöoikeuksien hallintaan käytetty tekniikka.
JDOM	Luokkakirjasto XML-tiedostojen esittämiseen Java-olioina.
JNDI	Java Naming and Directory Interface, ohjelmointirajapinta nimi- ja hakemistopalveluiden toteuttamiseen.
JSP	Java Server Pages, tekniikka html- ja xml-dokumenttien dynaamiseen luomiseen www-palvelimella.
JSTL	Java Server Pages Standard Tag Library, JSP-tekniikan laajennos.

JTA	Java Transaction API, ohjelmointirajapinta transaktioiden hallintaan javassa.
JVM	Java Virtual Machine, javan virtuaalikone, jossa ohjelmien suoritus tapahtuu.
Log4J	Kirjasto lokituksen tekemiseen javassa.
MVC	Model-View-Controller, graafisten käyttöliittymien suunnittelemisessa käyetty suunnittelumalli.
SQL	Structured Query Language, tietokantojen hallintaan kehitetty standardoitu kieli.
TSV	Tab-Separated Values, dokumenttimuoto jossa arvot ovat eroteltu sarkaimella.
URL	Uniform Resource Locator, käytetään osoittamaan WWW-sivuja.
VO	Value Object, J2EE kerrosmallissa arvo-olio.
XML	eXtensible Markup Language, rakenteellinen kuvauskieli.

Johdanto

Rank & Share on liiketoiminnan ennustamispalvelu päivittäis- ja kappaletavarateollisuudessa toimiville yrityksille. Ennustamisprosessissa lähdetään ajatuksesta jonka mukaan on hyvin todennäköistä, että tulevaisuudessa myynti korreloi vahvasti toteutuneiden myyntien kanssa, kun markkinatoimenpiteet ja olosuhteet ovat samankaltaiset. Rank & Share -palvelu tarjoaa ennustamiseen ainutlaatuisen menetelmän, joka perustuu edellä mainittuun oletukseen. Palvelu on toteutettu J2EE ohjelmistokehitysalustaa käyttäen Jboss sovelluspalvelimelle.

Ennusteprosessin tehokas käyttö vaatii säännöllistä tiedon käsittelyä, ajantasaisen myyntitiedon tuomista järjestelmään, sekä ennusteiden analysointia ja käyttöä muissa järjestelmissä. On siis selvää että palvelun käyttö vaatii erilaisten tausta- ja tukitoimintojen säännollistä suorittamista. Tässä työssä tutkitaan näiden toimintojen automatisointimahdollisuuksia. Suuri osa taustatoiminnoista on hajautettuna eri ohjelmistojen hoidettavana, joten työssä esitetään ratkaisuehdotus niiden toteuttamiseen ja automatisointiin keskitetysti Rank & Share -palvelun sisällä.

Työn tavoitteena on suunnitella ja toteuttaa automatisaatio- ja ajastusmoduuli, johon taustatoiminnot voidaan toteuttaa ja integroida. Moduuli liitetään Rank & Share -sovellukseen ylläpitotoimintona. Moduuli tarjoaa toisaalta käyttöliittymät tehtävien ajastamiseen ja automatisointiin ja toisaalta se tarjoaa sovelluskehittäjille helpon ja nopean kehyksen uusien tausta- ja tukitoimintojen toteuttamiseen ja integrointiin moduulin käyttöön. Työssä käytetään J2EE ohjelmistokehitysalustaa kuten Rank & Share -sovelluksessa muutenkin. Lisäksi otetaan käyttöön DWR-sovelluskirjasto paremman käytettävyyden saavuttamiseksi.

Työn käytännön tavoitteena on tehdä Rank & Share -sovelluksesta tehokkaammin käytettävä sekä asiakkaan että ylläpidon näkökulmasta katsoen. Moduulin käyttöönoton tulee vähentää palvelun ylläpidon työmäärää taustatoimintojen suorittamisen suhteen ja taustatoiminnot tulee voida siirtää asiakkaan ylläpidon vastuulle.

1 Rank & Share –palvelu

1.1 Palvelun toiminnot

1.1.1 Yleistä

Rank & Share –palvelu on yrityskäyttöön suunniteltu liiketoiminnan ennustamis- ja raportointijärjestelmä, jonka avulla yritys voi ohjata liiketoimintaansa ja ennakoida tarkasti myyntiin vaikuttavien tekijöiden vaikutusta kokonaisliikevaihtoon, kannattavuuteen ja pääomakustannuksiin. Palvelun tavoitteena on tarjota asiakkaille kattavat työkalut myynnin ennustamiseen erityisesti kampanjoiden ja tuotelanseerausten aikana. Palvelun ydinkohderyhmänä on päivittäis- ja kappaletavarateollisuudessa toimivat yritykset.

Palvelun erityispiirteenä voidaan pitää sen käyttämää ainutlaatuista menetelmää, jossa ennusteiden luomiseen käytetään toteumatiedoista kerättyä informaatiota kokonaismyynnistä, valikoimista sekä tuotekohtaisten myyntien jakautumisesta tuoteryhmän sisällä. Näin myyntiin vaikuttavien tekijöiden vaikutuksia voidaan tarkastella ja analysoida erikseen kolmesta eri näkökulmasta.

Palvelun toiminnot on jaettu kahteen työkalumoduuliin sekä kahteen ylläpitomoduuliin. Ennustusmoduuli sisältää toiminnot liiketoiminnan operatiiviseen ennustamiseen ja ennustetarkkuuden mittaamiseen. Raportointimoduulissa on toiminnot toteutuneiden myyntien raportointiin ja vertailuun. Yritystilin ylläpitomoduuli pitää sisällään asiaksyrityksen käyttäjätilien ja taustatoimintojen ylläpidon. Neljännen moduulin toiminnallisuudet kohdistuvat koko palvelun ylläpitoon ja yritystilien hallinnointiin sekä palvelun taustatoimintojen suorittamiseen ja ylläpitoon.

1.1.2 Liiketoiminnan ennustamismoduuli

Liiketoiminnan ennustamismoduulissa itse ennustaminen on jaettu kolmeen perustoiminnallisuuteen: kokonaismyynnin ennustaminen, jakauman ennustaminen sekä valikoiman ja ranking-järjestyksen ennustaminen. Kukin perustoiminnallisuus pitää sisällään sekä ennustustoiminnallisuuden että hallintatoiminnallisuuden. Ennustustoiminnallisuus sisältää varsinaisen ennusteiden asettamisen. Hallintatoiminnallisuus sisältää järjestelmään tallennettujen jakaumien ja valikoimien luomiseen ja muokkaamiseen käytettävät työkalut. Näiden lisäksi moduuli sisältää lisämyyntien ennustamistoiminnon, jonka kautta tuoteryhmän tuotteille voidaan asettaa yksittäisiä, menetelmän ulkopuolelle jätettäviä ennusteita.

Ennustaminen tapahtuu käytännössä siten, että käyttäjä asettaa päiväkohtaisen ennusteen kullekin ennustekohteelle (asiakas-, -alue ja tuoteryhmä kombinaatio) ja komponentille (kokonaismyynti, jakauma ja valikoima) tietyksi periodiksi, esimerkiksi viikoksi, eteenpäin. Kun ennusteen osakomponentit on asetettu ja tallennettu järjestelmään, ennuste voidaan laskea auki tuotekohtaiseksi ennusteeksi. Yrityksen liiketoiminnan ennusteprosessista riippuen ennusteet lasketaan auki tuotekohtaiseksi ennusteeksi ja ”jäädytetään” tietyin aikavälein, esimerkiksi kerran viikossa. Jäädytetty ennuste on siis virallinen ennuste, jota

yritys voi käyttää apuna liiketoimintansa ohjauksessa. Ennusteen laskemista tuotekohtaiseksi ennusteeksi kutsutaan ennusteen revisioinniksi, ja sen suorittaa palvelun ylläpitäjä omasta käyttöliittymästään.

Laskentaprosessiin kuluva aika on suoraan verrannollinen tuoteryhmien lukumäärään sekä tuoteryhmän sisällä olevien tuotteiden lukumäärään. Päivittäistavarateollisuudessa nämä lukumäärät ovat yleensä suuria, joten tuotekohtaisten ennusteiden laskentaa ei voida tehdä aina ennusteiden tallentamisen yhteydessä. Tämän takia se tehdään erikseen ylläpitokäyttöliittymän kautta.

Liiketoiminnan ennustamismoduulin osana on ennustamistoimintojen lisäksi ennustetarkkuuden raportointityökalu. Ennustetta verrataan kyseisen ajanjakson toteumatietoihin ja näin saadaan tietoja ennusteen onnistumisesta ja ennustetarkkuudesta. Ennustetarkkuutta tarkastellaan sekä tuoteryhmäkohtaisesti että tuotekohtaisesti yksikkömyynnin, liikevaihdon ja katteen osalta. Lisäksi raportista käy ilmi, miten hyvin ennusteeseen asetettu valikoiman tuotejärjestys ja jakauma ovat osuneet kohdalleen.

1.1.3 Raportointimoduuli

Rank & Share –palvelun raportointimoduulissa on mahdollista generoida erilaisia raportteja yrityksen toteumatiedoista sekä järjestelmään tallennetuista ennusteista. Raportteja voi ajaa useammasta kohteesta kerralla, jolloin kohteiden vertailu on mahdollista. Raportointimoduulin ominaisuuksiin kuuluu muun muassa kokonaismyynnin raportti, jossa esitetään valitun kohteen kokonaisyksikkömyynti, kokonaisliikevaihto, kate ja pääomankustannus valitulla ajanjaksolla sekä graafisesti että taulukkomuodossa. Moduulissa on myös jakaumien raportointimahdollisuus, jossa esitetään tuotteiden yksikkö-, liikevaihto-, ja katemääräiset jakaumat tuoteryhmän sisällä.

1.1.4 Asiakastilin ylläpitomoduuli

Asiakastilin ylläpitomoduuli on suunniteltu ainoastaan asiakasyrityksen pääkäyttäjän käyttöön. Se sisältää toiminnot yrityksen käyttäjätilien luomiseen ja hallintaan sekä ennusteprofiilin päivittämiseen ja ennusteiden raportointitietokannan päivittämiseen.

1.1.5 Palvelun ylläpitomoduuli

Palvelun ylläpitäjän yksinoikeudella käytössä oleva moduuli sisältää työkalut uuden asiakastilin luomiseen, toteumatiedon tuontiin, asiakkaan ennusteiden luontiin ja revisiointiin sekä asiakkaan ennusteiden raportointitietokannan päivittämiseen. Lisäksi moduulissa toiminto asiakkaan asiakas-, alue- ja tuoteryhmähierarkian päivittämiseen.

1.2 Palvelun taustatoiminnot

1.2.1 Yleistä

Taustatoiminnot ovat palvelussa suoritettavia prosesseja, joista peruskäyttäjän ei tarvitse tietää, mutta jotka ovat palvelun toimivuuden ja asianmukaisen käytön kannalta kriittisiä. Palvelun taustatoimintojen tehtävänä on pitää ennusteet ja

toteumatiedot ajantasalla sekä suorittaa muita asiakkaita palvelevia ja palvelua ylläpitäviä prosesseja. Palvelun taustatoiminnot ajetaan eräajoina tietyissä tilanteissa tarpeen vaatiessa tai tietyin ennalta määrätyin väliajoin. Osa taustatoiminnoista vaatii raskasta laskentaa, joten niiden suorittaminen palvelun normaalikäytön yhteydessä reaaliaikaisesti ei ole järkevää.

1.2.2 Toteumatiedon tuonti

Rank & Share –palvelu käyttää yrityksen toteumatietoja kahdessa eri yhteydessä. Ensisijaisesti toteumatietoja käytetään uusien ennusteiden tekemisessä jakaumien, valikoimarankingien ja kokonaismyyntiennusteiden pohjana. Toiseksi toteumatietoista voidaan generoida raportteja, joita käytetään ennustetarkkuuden mittaamisessa. Ennusteprosessin sujuvuuden ja asiakkaan palvelusta saaman hyödyn maksimoimiseksi toteumatietoa pitää tuoda järjestelmään mahdollisimman pian sen jälkeen, kun sitä on saatavilla. Esimerkiksi päivätasolla edellisen päivän toteumatiedot voidaan tuoda järjestelmään heti seuraavana aamuna.

Toteumatieto sisältää yrityksen liiketoimintaprosesseista ja ennusteprofiilista riippuen joko päivä-, viikko- tai kuukausikohtaista myyntititetoa tuotetasolla tai tuoteryhmätasolla. Järjestelmään tuodaan tuotteen tai tuoteryhmän yksikkömyynti, euromääräinen liikevaihto ja kate sekä varastoarvo ajanjaksoittain. Toteumatieto luetaan joko csv- tai tsv-tiedostosta ja se siirretään tietokantatauluihin erityisillä tietokantaskripteillä. Toteumatiedon siirto palvelimelle tapahtuu asiakkaan kanssa ennalta sovittua menetelmää käyttäen, esimerkiksi ftp-protokollalla.

Toteumatietojen siirto tietokantaan on monivaiheinen prosessi ja vaatii ylläpitäjän aktiivista toimintaa. Tietokantaskriptit, joilla toteumatieto viedään kantaan, on muokattava kullekin asiakkaalle erikseen ja ajettava tietokantakonsolin kautta. Skriptit on toteutettu PostgreSQL-tietokannalle, mutta soveltuvat pienin muutoksin muillekin tietokannoille. Skriptejä on yhteensä viisi kappaletta ja ne on ajettava oikessa järjestyksessä. Toteumatiedon määrästä ja ominaisuuksista riippuen koko prosessi kestää muutamasta minuutista ylöspäin aina 2-3 tuntiin asti. Prosessin pitkä kesto johtuu pääasiassa siitä, että tuotekohtaisista tiedoista lasketaan tietokantaan valmiiksi tuoteryhmäkohtaiset jakaumat sekä muita tunnuslukuja. Laskutoimitusten määrä on suoraan verrannollinen tuotteiden määrään, joka vastaavasti vaikuttaa suoraan laskentaprosessin kestoon. Laskenta vie luonnollisesti runsaasti prosessoriaikaa, joten toteumatiedon tuonti rasittaa palvelinta ja häiritsee normaalikäyttöä. Näin ollen prosessi tulee suorittaa sellaisena vuorokaudenaikana, jolloin palvelimen käyttöaste on alhaisimmillaan. Käytännössä tämä tarkoittaa kello 22 ja 06 välistä aikaa. Toteumatiedon tuonti järjestelmään on siis verrattain pitkä prosessi ja vie ylläpidolta resursseja.

1.2.3 Ennusteen revisiointi

Käyttäjän asettamat ennustetiedot kokonaismyynnin, valikoiman, rankingjärjestyksen ja jakaumien suhteen tallennetaan tietokantaan sellaisenaan. Näistä tiedoista pystytään laskemaan auki tuotekohtaiset ennusteet tuoteryhmän sisällä. Tuotekohtaiset ennusteet pitävät sisällään tunnusluvut yksikkömyynnin, liikevaihdon ja katteen osalta. Ennustetietojen käsittely ja laskenta suoritetaan suurelta osin tietokantatasolla sovellustason sijaan, mutta vaati silti runsaasti

laskentatehoa ja -aikaa. Näin ollen revisiointiprosessia ei voida ajaa palvelun normaalikäytön aikana, esimerkiksi ennusteiden tallennuksen yhteydessä.

Ennusteen revisioinnilla on tärkeä rooli myös asiakasyrityksen liiketoimintaprosessissa. Revisiointiajankohta on kiinnitetty asiakaskohtaisesti tiettyyn ajankohtaan ja revisiointi suoritetaan tietyin väliajoin, esimerkiksi kerran viikossa. Tällöin revisioiduista ennusteista tulee virallisia, eikä niihin voi tämän jälkeen tehdä muutoksia. Näin ollen revisiointiperiodi rytmittää yrityksen ennustustoimintaa ja muita liiketoimintaprosesseja, kuten tuotannonohjausta.

1.2.4 Ennusteiden vienti

Ennusteiden vienti –toiminto suorittaa asiakasyrityksen viimeisimpien jäädytettyjen ennusteiden muunnoksen tietokantatasolta sellaiseen muotoon, että asiakas voi käsitellä niitä omassa järjestelmässään. Käytännössä toiminto suoritetaan siten, että tietokannasta haettu ennustetieto kirjoitetaan csv- tai tsv-muotoiseen tiedostoon tietokantaskripteillä ja luotu tiedosto saatetaan asiakkaan saataville palveluun. Tiedoston sisällön muoto on samanlainen, kuin toteumatiedon tuonnissa käytetty tiedostomuoto. Ennusteet kirjoitetaan tiedostoon samalla tasolla (tuote tai tuoteryhmä) kuin millä toteumatieto on tuotu järjestelmään.

1.2.5 Muut

Palvelun taustatoimintoihin luetaan asiakasyrityksen perustietojen tuontitoiminto. Perustiedoilla tarkoitetaan ensisijaisesti tietoja tuote-, asiakasryhmä- ja aluehierarkiasta. Näiden lisäksi asiakas voi toimittaa tietoja kampanjoista, käyttämistään tuottajista, materiaaleista sekä muista tuotteisiin ja tuotantoon liittyvistä asioista. Asiakas toimittaa tiedot xml-tiedostomuodossa. Tiedoston rakenne noudattaa Rank & Share –palvelulle suunniteltua rakennetta.

Perustietojen tuonti suoritetaan kertaluontoisesti uuden asiakastilin luonnin yhteydessä. Tämän jälkeen tuote-, asiakas-, alue- ja muut tiedot päivitetään tarpeen tullen, harvoin kuitenkaan useammin kuin kerran kuukaudessa. Perustietojen tuonnin suorittamiseen on palvelussa käyttöliittymä ja tuonti suoritetaan palvelun ylläpitäjän toimesta.

2 Ongelman määrittely

2.1 Ylläpitoresurssit

Rank & Share –palvelu tarjoaa käyttäjilleen ennustus- ja raportointityökaluja, joiden asianmukainen käyttö vaatii erilaisten tausta- ja tukitoimintojen säännöllistä suorittamista. Tomintojen suorittaminen tapahtuu tällä hetkellä palvelun ylläpidosta vastaavien henkilöiden toimesta. On selvää että asiakasmäärän kasvaessa tausta- tukitoimintojen suorittamiseen kuluva aika kasvaa ja ennen pitkää vie suurimman osan ylläpitohenkilökunnan resursseista. Kaikkea tausta-, tuki- ja ylläpitotoimintoja ei pystytä täydellisesti automatisoimaan, mutta tiettyjä toimintoja on mahdollista siirtää palveluun automaattiseen suoritukseen. Automaattisten toimintojen valvonta ja ajastus tulee lisäksi siirtää palvelun ylläpidolta asiakkaan ylläpidon vastuulle.

2.2 Automatisointi ja ajastus

2.2.1 Yleinen kuvaus

Ylläpitotehtävät vaativat yleensä paljon laskentaresursseja, minkä takia niiden suorittaminen pitää ajoittaa vuorokaudenaikaan, jolloin palvelimen kuorma on pieni ja laskenta vaikuttaa mahdollisimman vähän käyttäjien kokemaan palvelun laatuun. Automatisointitehtävät pitää tämän takia pystyä ajastamaan suoritettavaksi ennalta määriteltynä ajankohtana. Jotta automatisoituja taustatoimintoja voi käyttää joustavasti, palvelussa pitäisi olla lisäksi mahdollisuus konfiguroida ja hallita toimintoja ja niiden ajastuksia.

Manuaalisesti tehty työ on joustavampaa virhetilanteissa ja usein virhetilanteisiin reagoiminen ja niistä toipuminen on sujuvampaa. Ongelmana on pystyä tekemään automatisointi siten, että käyttäjällä on käytettävissä riittävän kattavasti tietoa virhetilanteesta, sen syistä ja seurauksista, jotta virheiden korjaaminen on mahdollista. Oleellisinta asiakkaan kannalta on saada riittävän selkeät tiedot virhetilanteen luonteesta ja syistä, jotta virheen pystyy korjaamaan ilman sen suurempaa teknistä ymmärrystä automatisoidun tehtävän yksityiskohdista.

Palvelun kehitys on vielä melko alkuvaiheessa. Tämän takia on odotettavaa, että uusia automatisoitavia taustatoimintoja on tulossa huomattava määrä lisää. Toimintojen lisääminen järjestelmään on tehtävä sovelluskehittäjille mahdollisimman sujuvaksi ja joustavaksi resurssien säästämiseksi ja sovelluskehityksen tehokkuuden ylläpitämiseksi. Ongelmana on geneerisen sovelluskehyksen puute.

2.2.2 Toteumatiedon tuonti

Työn aikana toteutettava esimerkkitoiminto suorittaa asiakkaan toteumatiedon tuonnin järjestelmään. Tämän toiminnon valinta esimerkkitehtäväksi perustuu sen manuaalisen työn monimutkaisuuteen ja laskennan raskauteen sekä siihen, että manuaalinen työ tulee tällä hetkellä tehdä erillisillä Rank & Share -palvelun ulkopuolisilla työkaluilla.

Manuaalinen työ toteumatiedon tuonnissa vaatii työkaluja, joihin ei pääse käsiksi suoraan palvelun käyttöliittymistä. Tällaisten työkalujen käyttö heikentää palvelun

käyttöä asiakkailla, joilla ei ole resursseja tai intressejä palvelun ulkopuolisten työkalujen hankintaan ja opetteluun. Käytännössä asiakkaan tulee itse huolehtia toteumatiedon tuonnista, jolloin ylimääräisiä, palvelun ulkopuolisia työkaluja ei pidä tarvita.

Asiakkaalta saatavan toteumatiedon vienti tietokantaan on verrattain hankalaa ja monivaiheista käsityötä, jossa moni asia voi epäonnistua. Toteumatiedon käsittely ja luku tietokantaan vaatii useiden erillisten tietokanta-ajojen suoritusta oikeassa järjestyksessä tietokannan hallintasovelluksen kautta. Tietokanta-ajojen suoritukseen on tehty valmiit tietokantaskriptit, mutta niitä pitää muokata aina tilanteen mukaan asiakaskohtaisesti. Suurimpana haasteena on, että asiakkaan omista järjestelmistä tuotettu toteumatieto ei aina ole oikeassa muodossa tai se ei sisällä kaikkea tarvittavaa tietoa. Näin ollen toteumatiedon oikeellisuus pitää ennen tietokantaan kirjoitusta tarkastaa virheiden varalta.

2.3 Työn tavoitteet

2.3.1 Yleiset tavoitteet

Työn kokonaistavoitteena on parantaa Rank & Share -palvelun taustatoimintoja ja tehostaa ylläpitoa vaativien tehtävien suorittamista. Työssä suunnitellaan ja toteutetaan automatisointi- ja ajastusmoduuli, jonka avulla nämä tavoitteet pyritään saavuttamaan. Työn tavoitteiden saavuttamista mitataan toteuttamalla palveluun uusi automatisoitu tehtävätyyppi ja arvioimalla sen toteutuksen sujuvuutta ja käytössä saavutettuja etuja.

Työn tuloksena syntyy kehys taustatoimintojen kehittämiseen. Kehyksen avulla uusien taustatoimintojen automatisoinnin toteuttaminen ja integrointi järjestelmään helpottuu. Lisäksi syntyy käyttöliittymät ajastusten hallintaan ja monitorointiin sekä sovelluslogiikka näiden toimintojen suorittamiseen.

Työ jaetaan kolmeen vaiheeseen. Ensimmäisessä vaiheessa toteutetaan ajastusmoduulin sovelluslogiikka ja käyttöliittymät. Toisessa vaiheessa suunnitellaan kehys uusien automatisoitujen tehtävätyyppien kehittämiseen. Kolmannessa vaiheessa tehdään esimerkkitoteutus käyttäen toisessa vaiheessa luotua kehystä sekä analysoidaan kehyksen toimivuutta ja ajastusmoduulin tuomia etuja ylläpitotyökaluna.

2.3.2 Ajastettujen tehtävien hallintatyökalu

Työn ensimmäisessä vaiheessa suunnitellaan ja toteutetaan Rank & Share -sovellukseen moduuli ajastettujen tehtävien hallintaan ja monitorointiin. Hallintatyökalun käyttöliittymässä on toiminnot ajastusten asettamiseen, tehtävien konfigurointiin, tehtävien suoritusten monitorointiin ja lokitietojen esittämiseen.

2.3.3 Taustatoimintojen automatisointikehys

Palvelun toimintojen ja työkalujen kehittyessä myös taustatoimintoja on tulossa lisää. Tavoitteena on suunnitella ja toteuttaa kehys, jonka pohjalle uusien toimintotyyppien toteutus ja integrointi ajastusmoduuliin on mahdollisimman helppoa ja suoraviivaista.

2.3.4 Esimerkkitoteutus ja analyysi

Esimerkkitoteutuksena Rank & Share -palvelun automatisaatio- ja ajastusmoduuliin integroidaan yrityksen toteumatietojen tuontiprosessi. Manuaalisesti suoritetusta taustatoiminnosta tehdään automatisoitu ratkaisu ja se lisätään palveluun ajastettavaksi tehtäväksi. Uuden tehtävätyypin lisäämisen aikana tehdyn työn määrää seurataan ja analysoidaan, jotta työn tavoitteiden täyttymisestä tältä osin voidaan tehdä johtopäätökset.

Ajastusmoduulin tarjoamien työkalujen käytöstä saavutetuista eduista on vaikea saada välittömästi näyttöä, koska ylläpito on jatkuva prosessi. Työssä toteutetusta moduulista saavutettavat edut tulevat esiin vasta pidemmän ajan kuluessa asiakasmäärän kasvaessa. Analyysivaiheessa voidaan kuitenkin mitata työn tavoitteiden täyttymistä mittaamalla työn vaatimusmäärittelyvaiheessa löydettyjen vaatimusten täyttymistä.

3 Arkkitehtuuri ja teknologiat

3.1 Toteutustekniikat

Rank & Share –ohjelmisto on toteutettu sovelluspalveluna eli ohjelmistoa voidaan käyttää internetin kautta ilman, että käyttäjän tarvitsee asentaa omalle tietokoneelle erillistä sovellusta. Tällä hetkellä ohjelmiston käyttöliittymä on toteutettu käytettäväksi Internet-selaimella. Rank & Share –ohjelmiston toteutuksessa on pyritty käyttämään mahdollisimman paljon valmiita, standardinomaisia ratkaisuja ja komponentteja. Näin ohjelmiston kehitys nopeutuu ja helpottuu huomattavasti. Oman ohjelmakoodin määrä jää verrattain pieneksi ja samalla ylläpidettävyys ja jatkokehitys helpottuu. [1]

Rank & Share -ohjelmisto on toteutettu J2EE sovelluspalvelimen rakennetta ja komponentteja käyttäen [1]. J2EE on standardinomainen määrittely- ja suunnittelukehys monikerroksisen sovelluspalvelimen toteuttamiseen. J2EE ei itsessään ole valmis sovelluspalvelinratkaisu, vaan se määrittelee sovelluspalvelimen rakenteen ja komponentit, jotka J2EE standardin mukaisen sovelluspalvelimen tulee toteuttaa [2]. Yksi tällainen sovelluspalvelinratkaisu on JBoss.

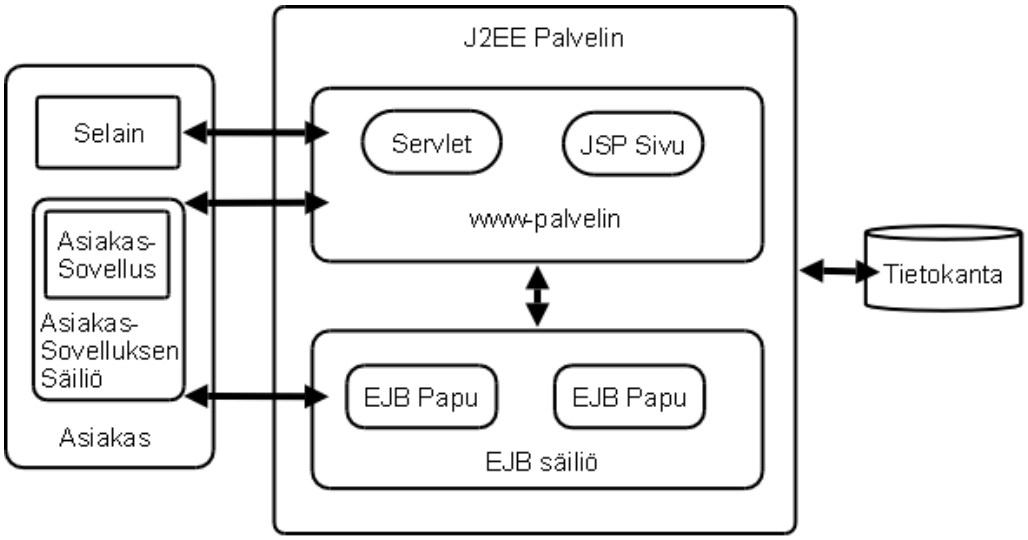
Rank & Share –ohjelmisto on toteutettu JBoss-sovelluspalvelimen pohjalle ja sen ohjelmistokomponentteja käyttäen. JBoss sisältää kaikki J2EE standardissa määritellyt komponentit sekä lukuisia lisäominaisuuksia ja ohjelmistokehitystyötä nopeuttavia komponentteja. JBoss-palvelimessa on muun muassa valmiit toteutukset käyttäjien autentikointiin ja oikeuksienhallintaan sekä transaktioiden hallintaan (JAAS ja JTA). JBoss on siis hyvin monipuolinen sovelluspalvelinkokonaisuus, jossa on hyvät konfigurointimahdollisuudet sekä laajat muokkausmahdollisuudet ja valmiit työkalut [3]. Näistä syistä JBoss on myös melko vaikeaselkoinen ja monimutkainen kokonaisuus, jonka tehokas käyttö vaatii paljon opettelua. Tässä luvussa esitellään ainoastaan J2EE arkkitehtuurin geneerinen kerrosmalli ja sovelluksen rakenne sekä komponentit niiltä osin kuin se on ajastus- ja automatisaatiomoduulin toteuttamisen kannalta tarpeellista. Lisäksi luvussa esitellään ohjelmiston toteutuksessa käytetyt erilliset sovelluskehykset ja -kirjastot, kuten Struts ja DWR.

J2EE arkkitehtuurin kerrosmallin ymmärtäminen on oleellista toteutettavan automatisaatio- ja ajastusmoduulin suunnittelun ja toteuttamisen kannalta. Kerrosmalli määrittelee hyvin pitkälle, mihin osaan sovelluksen arkkitehtuuria mikäkin toiminnallisuus sijoittuu. Lisäksi on oleellista ymmärtää muutamia asioita JBoss:n konfiguroinnista, jotta käytettyjen komponenttien kuten EJB:n käyttö on mahdollista. Sovelluskehysten ja -komponenttien yksityiskohtaiseen käyttöön ja teknisiin ominaisuuksiin ei tässä työssä ole syytä paneutua.

3.2 J2EE Arkkitehtuuri

J2EE arkkitehtuurin mukaisesti rakennetussa ohjelmistossa voidaan ajatella olevan kolme pääkomponenttia: asiakasohjelmisto, sovelluspalvelin sekä tietokanta. Näiden pääkomponenttien rajapinnoissa toimivat omat alikomponentit, jotka huolehtivat tiedon välityksestä pääkomponenttien välillä. Edelleen pää- ja

alikomponentit voidaan yksityiskohtaisessa tarkastelussa purkaa pienempiin osiin. [2]



Kuva 1 J2EE komponentit

Käyttäjän asiakasohjelmistona toimii tässä tapauksessa internet-selain, jolla ohjelmiston käyttöliittymiä voidaan käyttää. J2EE-palvelimen osaksi on liitetty www-palvelin, joka vastaanottaa asiakasohjelmiston lähettämät http-pyynnöt ja generoi vastauksen pyyntöön dynaamisesti käyttämällä JSP/Servlet-tekniikkaa. Tämä lisäksi palvelin pitää yllä käyttäjän istunnonaikaisia tietoja.

Vaikka www-palvelin on upotettu osaksi sovelluspalvelinta, sen voidaan ajatella olevan itsenäinen komponentti, joka huolehtii tietojen ja palvelupyyntöjen välittämisestä käyttäjän ja varsinaisen sovelluspalvelimen välillä. Palvelupyyntö voi olla esimerkiksi pyyntö hakea tieto tietokannasta ja muokata se esitysmuotoon tai tiedon lähetys sovelluspalvelimelle tietokantaan kirjoitusta varten. Kuten kuvasta 1 nähdään, sovelluspalvelimen palveluita voidaan käyttää muillakin asiakasohjelmistolla kuin internet-selaimella. Sovelluspalvelimen näkökulmasta www-palvelin on tavallaan asiakasohjelmisto, joka käyttää hyväksi sovelluspalvelimen palveluita. Internet-selainmen voidaan ajatella olevan puhtaasti käyttöliittymän esitykseen käytettävä sovellus. [2]

Varsinaisella sovelluspalvelimella toimii joukko EJB-komponentteja, jotka tarjoavat erilaisia palveluja asiakasohjelmiston käyttöön. EJB-komponentit ovat sovelluspalvelimen ydintoiminto ja kanava, jota kautta sovelluspalvelimen palveluihin, sovelluslogiikkaan ja tietokantaan tallennettuun tietoon pääsee käsiksi [2][4]. EJB-komponenttiarkkitehtuuriin perehdytään myöhemmissä kappaleissa tarkemmin.

3.3 J2EE Kerrosmalli

3.3.1 Suunnitteluperiaatteet

J2EE-kerrosmalli noudattaa kahta oliokeskeisen suunnittelun tärkeää periaatetta, eli luokkien välistä heikkoa kytkentää ja vahvaa yhtenäisyyttä. Heikko kytkentä (low coupling) tarkoittaa käytännössä sitä, että yksittäisten olioiden riippuvuus ja kytkentä toisiin olioihin pitää olla mahdollisimman vähäistä. Tämä ominaisuus on

tärkeä, sillä sen ansiosta ohjelman rakenteesta tulee helpommin ymmärrettävä. Ihmisen on vaikea hallita suurta määrää monimutkaisia riippuvuussuhteita. Lisäksi heikon kykennän omaavia olioita pystyy käyttämään paremmin uudelleen. [5]

Vahva yhtenäisyys (high cohesion) tarkoittaa, että yksittäisen olion vastuulle annettavien tehtävien tulee olla keskenään mahdollisimman saman kaltaisia. Käytännössä siis yhden olion tehtäväksi ei pidä antaa eri tyyppisiä tehtäviä, esimerkiksi sekä tietokantahakuja että sovelluslogiikkaa. Näin tekemällä oliosta saadaan niin ikään helpommin käytettävä ja ymmärrettävä. Yksi olio siis vastaa vain yhden tyyppisistä toiminnoista. [5]

J2EE kerrosmallissa ohjelmisto on jaettu kerroksiin, joissa kussakin suoritetaan vain tiettyjä toimintoja. Ylemmällä kerroksella olevat oliot käyttävät alemman kerroksen olioiden tarjoamia palveluita. Kuvassa 2 on esitetty geneerisen J2EE-sovelluksen kerrosmalli.

Kuva 2 J2EE kerrosmalli

3.3.2 Esityskerros

J2EE-kerrosmallin ylimmällä tasolla on esityskerros. Esityskerroksen komponentit huolehtivat käyttäjältä tulevien syötteiden käsittelystä, käyttöliittymien toteutuksesta ja kommunikoinnista alemman kerroksen kanssa. Käyttöliittymät voidaan rakentaa J2EE-mallin mukaisesti java-apletteina tai www-sivustona. Rank & Share -palvelun käyttöliittymät on toteutettu www-sivustona. Sivujen generoiminen tapahtuu dynaamisesti J2EE:n JSP/Servlet-komponenttia käyttäen. [6]

Esityskerroksen rakenne ja looginen toiminta Rank & Share -palvelussa noudattaa MVC-suunnitelumallia. Suunnittelumallin perustana on ajatus, että sovelluksen rakenne jaetaan kolmeen erilliseen osaan: malliin, näkymiin ja ohjaimeen. J2EE-kerrosmallia käytettäessä voidaan malli-osan ajatella käsittävän kaikki kerrokset jakelukerroksesta alaspäin. Näkymä-osa on käytännössä JSP-tekniikalla toteutettu WWW-sivu. Ohjain huolehtii syötteiden välityksestä näkymän ja mallin välillä sekä kontrolloi siirtymisiä näkymien välillä.

Suunnittelumallin toteutuksessa käytetään hyväksi Struts-sovelluskehystä. Sturts-sovelluskehys on valmis MVC-mallin toteutus ja huolehtii suurelta osin sovelluskehittäjän puolesta eri komponenttien välisestä kommunkoinnista. Lisää Struts-sovelluskehyksen ominaisuuksista ja käytöstä kappaleessa 3.5.1.

3.3.3 Jakelukerros

J2EE-kerrosmallissa jakelukerros tarjoaa sovelluskohtaisia palveluita ylemmälle kerrokselle. Rank & Share -palvelussa jakelukerros käsittää joukon EJB-komponentteja, jotka välittävät sovelluslogiikan toimintoja esityskerrokselle. EJB-komponenteissa ei itsessään ole varsinaista logiikkaa vaan toimivat vain niin sanottuina kääreluokkina esityskerroksen asiakassovellukselle. Sovelluslogiikan voisi periaattessa toteuttaa suoraan EJB-komponentteihin, mutta tällöin sovelluslogiikkan toimintoja ei ole mahdollista tarjota suoraan muun tyyppisten palveluiden kautta [6]. EJB-komponenttien toiminnan periaatteet sekä eri tyypit esitellään tarkemmalla tasolla seuraavassa luvussa.

Rank & Share -sovelluksessa eri EJB-komponenttien vastuualueet on jaettu vastaamaan palvelussa tarjottavien moduulien toimintoja. Toisin sanoen kullekin moduulille on oma EJB-komponentti, joka tarjoaa kyseiselle moduulille oleellisia palveluita. [1]

3.3.4 Sovelluslogiikkakerros

Sovelluslogiikkakerroksen olioiden tehtävänä on toteuttaa tietokannasta haetun tiedon käsittely muotoon, jossa se voidaan mahdollisimman helposti muuttaa esitysmuotoon esityskerroksella. Vastaavasti esityskerrokselta tuleva tieto muutetaan tietokantaan kirjoitettavaksi. Sovelluslogiikkaoliolla (BO) on vastuu myös erilaisista laskennallisista tehtävistä. Rank & Share -palvelussa sovelluslogiikan tehtävänä on muun muassa laskea ennusteet auki, revisioida ennusteet sekä muodostaa toteumatiedoista raportteja.

3.3.5 Tiedonkäsittelykerros

Tiedonkäsittelykerroksen oliot huolehtivat tiedon välittämisestä tietokannan ja sovelluslogiikkakerroksen välillä. Tiedonkäsittelykerros piilottaa tietokannan rakenteen sovelluslogiikalta muuttamalla tietokannasta haetun tiedon olioiksi ja päinvastoin. Rank & Share -palvelu käyttää tietokannan käsittelyssä erityistä sovelluskehystä Hibernatea [1]. Hibernaten avulla java-oliot voidaan kirjoittaa tietokantaan suoraan ja niitä voidaan hakea tietokannasta. Lisäksi Hibernatessa on valmiit mekanismit transaktioiden hallintaan. Hibernate:n tarkempi esittely on kappaleessa 3.5.2.

3.3.6 Arvo-oliot

J2EE:n kerrosmallissa arvo-oliot edustavat olioita, jotka voidaan hakea suoraan tietokannasta ja vastaavasti kirjoittaa tietokantaan suoraan Hibernaten komponentteja hyväksi käyttäen. Arvo-oliot muodostavat loogisesti yhteen liittyvien tietojen kokoelmia ja yleensä edustavat jotain reaalimaailman oliota. Arvo-oliossa ei siis tule olla sovelluslogiikkaa, vaan no ovat puhtaasti tietojen säilytystä ja siirtoa varten suunniteltuja. Arvo-olioita käytetään läpi kerrosmallin tietokantatasolta esityskerrokselle. [5]

3.3.7 Arkkitehtuurikomponentit

Arkkitehtuurikomponentit ovat yksittäisiä olioita tai komponentteja, joita käytetään läpi sovelluksen kerrosmallin tukitoimintojen tai yleisten työkalujen tarjoajana. Esimerkiksi Rank & Share -sovelluksessa käyteään erityistä Log4J-lokitustyökalua. [1]

3.4 J2EE komponentit

3.4.1 JSP/Servlet

JSP ja Servlet –komponentit edustavat J2EE arkkitehtuurissa esityskerroksen toteuttamisessa käytettävää tekniikkaa. Servlet tarkoittaa www-palvelussa toimintoa, jolla sovelluspalvelin käsittelee käyttäjän lähettämät palvelupyynnöt. Staattisessa www-palvelussa käyttäjä tyypillisesti kutsuu suoraan yksittäistä html-kielellä toteutettua sivua, kun dynaamisessa www-palvelussa käyttäjä kutsuu tiettyä servlet:iä, joka generoi html-sivun käyttäjän antamien parametrien ja käyttäjän istuntokohtaisen tiedon perusteella. Servlet käyttää sovelluspalvelimen tarjoamia EJB-komponentteja tiedon hakuun ja käsittelyyn, vastaanottaa EJB:n palauttaman tiedon ja muokkaa sen esityskelpoiseen muotoon. Servlet-luokka voi itsessään generoida html-koodia tai vaihtoehtoisesti siirtää html-koodin generointivastuun JSP-sivulle. [2]

JSP tekniikka mahdollistaa dynaamisten www-sivujen toteuttamisen suoraan tekstitiedostona html-sivun tapaan. JSP-sivu sisältää staattista html-koodia sekä JSP-elementtejä. Jsp-elementit ovat xml-muotoisia elementtejä, joiden avulla sovelluskehittäjä voi lisätä www-sivustoon dynaamista sisältöä. Näiden lisäksi JSP-sivulle on mahdollista kirjoittaa suoraan java-koodia, joka voi käsitellä tietoa ja generoida html-elementtejä. [2]

Rank & Share -sovelluksessa on käytetty Struts-sovelluskehyksen tarjoamaa suunnittelu- ja toimintamallia, joka yhdistää Servlet- ja JSP-tekniikat siten, että MVC-suunnittelumallin mukainen sovellusrakenne toteutuu [1]. Struts:n toimintamalli esitellään luvussa 3.5.1.

3.4.2 Enterprise Java Beans

EJB on komponenttiarkkitehtuuri, joka tarjoaa menetelmän sovelluspalvelimen palveluiden käyttöön verkon yli erillisellä asiakasohjelmalla. EJB-komponentin käytännön toteutuksena voidaan pitää yksittäistä EJB-luokkaa eli papua. Papu asetetaan sovelluspalvelimelle sille erityisesti tarkoitettuun EJB-säiliöön. Säiliö muun muassa huolehtii pavun ja asiakasohjelman välisistä yhteyksistä, tietoturvasta ja hoitaa pavun rekisteröitymisen nimipalveluun. [2] EJB-säiliö siis huolehtii järjestelmätason toiminnoista, jolloin sovelluskehittäjä voi keskittyä ainoastaan sovelluslogiikan toteuttamiseen.

EJB-säiliö ei päästä asiakasohjelmaa suoraan käyttämään papua, vaan sitä varten säiliöön on rakennettava erilliset rajapinnat. Home-rajapinnan kautta asiakasohjelma pääsee luomaan ja poistamaan papuja. Remote- ja local-rajapintojen kautta asiakas pääsee käyttämään pavun tarjoamia metodeita. Remote ja local-rajapinnat eroavat toisistaan pääasiassa siinä, että remote-rajapinnan kautta asiakasohjelma pääsee käyttämään papuja, jotka sijaitsevat eri virtuaalikoneella (JVM) kuin asiakassovellus. Local-rajapinnan kautta papua

käytettäessä asiakasohjelman ja EJB-säiliön pitää sijaita samalla JVM:llä. [7] Rank & Share -sovelluksessa asiakasohjelmisto sijaitsee samalla sovelluspalvelimella kuin EJB-säiliö, joten local-rajapinnan toteuttaminen riittää. Tosin itsenäistä testisovellusta varten myös joitain remote-rajapintoja on toteutettu, mutta ne on asetettu pois päältä tuotantokäytössä.

EJB-komponentin käyttäminen asiakasohjelmassa tapahtuu erityisen asiakasluokan kautta. Asiakasluokka pääsee käsiksi EJB-säiliön pavun paikalliseen rajapintaan sovelluspalvelimen nimipalvelun kautta. Sovelluspalvelimen nimipalvelu on toteutettu JNDI-komponentilla, mutta sen toimintaan ei ole tarpeellista perehtyä tässä yhteydessä. Kun asiakasluokka on onnistunut ottamaan yhteyden EJB-säiliössä olevaan papuun, pavun metodeita voi käyttää kuin ne olisivat paikallisen olion metodeita. [4][7]

3.4.3 EJB Timer Service

Ajastettujen toimintojen toteuttamisessa käytetään J2EE:ssä määriteltyä ajastuspalvelua. EJB-säiliössä on TimerService-luokan olio, joka hallinnoi säiliöön rekisteröityjä ajastuksia. TimerService:n kautta voi asettaa kertaluontoisia ja toistuvia ajastuksia, ja TimerService-olio laukaisee ajastukset ajallaan. TimerService-olion sisältäviä ajastuksia, eli Timer-olioita, voi lisätä ja poistaa pavun kautta. Timer-olioihin pääsee käsiksi TimerHandle-olioiden kautta. [2]

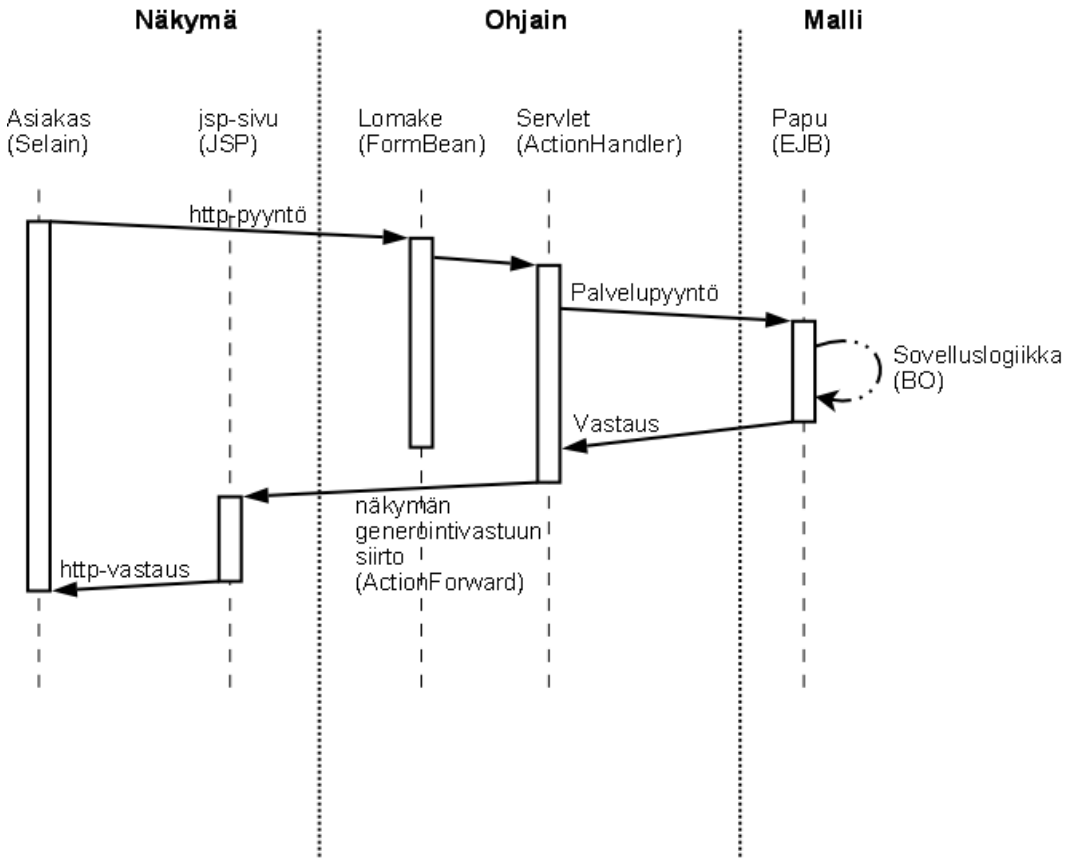
Papu määritellään suorittamaan metodi ajastuksen lauetessa. Tämä tapahtuu toteuttamalla TimedObject-rajapinta. TimedObject-rajapinssa on ejbTimeout-metodi, suoritetaan ajastuksen lauetessa. TimerService-olio on pysyvä papu-olio, joten sovelluspalvelimen sulkeutuessa timer-olioden tilat tallennetaan, ja ne voidaan käynnistää uudelleen palvelimen käynnistyessä uudelleen. Mikäli ajastuksen olisi pitänyt laueta palvelimen ollessa suljettuna, ajastus laukeaa heti palvelimen käynnistyttyä [7]. Tämä ominaisuus on otettava huomioon, jotta Rank & Share -palvelu toimii halutulla tavalla.

Jboss:n TimerService -toteutuksessa on ohjelmointivirhe, joka aiheuttaa sen, että palvelimen sulkeutuessa ja uudelleen käynnistyttyä Timer-oliot eivät pysty kertomaan ajastuksensa jäljellä olevaa aikaa enää oikein. [8] Tämä voi aiheuttaa ongelmia ajastusmoduulin toteutuksessa.

3.5 Sovelluskehykset

3.5.1 Struts

Struts on sovelluskehys J2EE:n kerrosmallin esityskerroksen toimintojen toteuttamiseen. Struts on kehitetty noudattamalla MVC-suunnittelumallia [9] ja näin ollen se tarjoaa kattavat työkalut erityisesti näkymien ja ohjainten kehittämiseen. Malli-osa käsittää Rank & Share -sovelluksessa J2EE:n kerrosmallin esityskerroksen alapuoliset kerrokset.



Kuva 3 Struts:n käyttö Rank & Share -sovelluksessa

Kuvassa 3 on esitetty yksittäisen http-pyynnön käsittelysekvenssi Rank & Share -sovelluksessa Struts:n komponentteja käyttäen. Kuvassa on esitetty kaksi struts:n pääkomponenttia: JSP-sivu ja Servlet eli toiminnon käsittelijä. Käytännössä kukin näkymä käyttöliittymässä on toteutettu JSP-sivuna ja jokasella näkymällä on yksi tai useampi toiminnon käsittelijä, jolle näkymästä annetut komennot ja lomakkeelle syötetyt tiedot siirtyvät http-pyynnön välityksellä. Toiminnon käsittelijä suorittaa kullekin toiminnolle vaadittavat toimenpiteet ja siirtää näkymän generointivastuun JSP-sivun hoidettavaksi. Toiminnon käsittelijä kontrolloi näkymien välisiä siirtymiä ja päättää kussakin tilanteessa, mihin näkymään seuraavaksi siirrytään. Esimerkiksi virheellisen syötteen saadessaan käsittelijä voi siirtää käyttäjän erityiselle virhesivulle normaalin näkymän sijasta.

Struts tarjoaa siis työkalut J2EE:n JSP/Servlet-komponentin käytön helpottamiseksi. Tämän lisäksi Struts pitää sisällään lukuisia muita sovelluskehitystä nopeuttavia työkaluja. [9] Rank & Share -sovelluksessa näitä käytetään hyödyksi. Tärkeimpinä mainittakoon monikielisyyden tuen implementointi, html-lomakkeiden syötteiden validointi sekä www-sivujen modulaarinen rakentaminen. Modulaarinen www-sivujen rakentaminen tarkoittaa käytännössä sitä, että yksittäinen www-sivu rakennetaan kahdesta tai useammasta erillisestä JSP-sivusta. Pohjasivu implementoi sivuston raamit, otsikkorivit ja valikot. Pohjasivun sisään sijoitetaan varsinaiset näkymät, joita käytetään muun muassa lomakkeiden täyttöön ja raporttien esittämiseen.

3.5.2 Hibernate

Hibernate on sovelluskehys olioiden tallentamiseen, hakemiseen ja kutsumiseen tietokantatasolta sovellustasolle. Rank & Share -sovelluksessa Hibernate:a käytetään BO- ja DAO-kerroksilla VO-olioiden tallentamiseen ja hakemiseen [1]. Hibernate:ssa on myös tehokas tiedonhakukieli, joka muistuttaa vahvasti SQL-kieltä, mutta on olio-suuntautunut. Hibernate:n suurin hyöty sovelluskehittäjälle on siis tietokannan toiminnan peittäminen olio-suuntautuneelta maailmalta. [10]

Hibernate-sovelluskehys on liitetty valmiiksi osaksi Jboss-sovelluspalvelinta [3], joten sen käyttöönotto on melko yksinkertaista. Rank & Share –sovelluksessa VO-oliot kuvataan tietokantatasolle erityisillä xml-tiedostoilla, joissa kuvataan, miten kukin VO-olion kenttä kirjataan tietokantatauluihin, ja millaisia suhteita olioiden välillä on.

3.5.3 Direct Web Remoting

Selainkäyttöliittymissä käytettyä javascript-ohjelmointikieltä voidaan käyttää http-pyyntöjen lähettämiseen ja vastausten vastaanottamiseen erityistä XMLHttpRequest-oliota käyttämällä. Näin www-sivulle saadaan enemmän interaktiivisuutta, ja sivun sisältöä voidaan päivittää päivittämättä koko sivua. Tätä tekniikkaa kutsutaan Ajax-teknologiaksi. DWR yhdistää Ajax-teknologian javan oliomaailmaan ja Servlet-tekniikkaan. [11]

DWR on ohjelmistokirjasto, jonka avulla www-sivulta voidaan javascript:n avulla kutsua sovelluspalvelimella sijaitsevien olioiden metodeita ja käsitellä java-olioita javascript:llä niin kuin ne olisivat javascript-oliota. DWR:ssä on mahdollisuus myös kutsua sovelluspalvelimelta Java-olioiden avulla asiakkaan selaimella sijaitsevia javascript-funktioita. Tätä menetelmää kutsutaan reverse-ajax-tekniikaksi. [11]

Rank & Share -sovelluksessa ei tällä hetkellä käytetä DWR-kirjastoa, mutta se otetaan käyttöön ajastusmoduulin käyttöliittymien toteutuksessa. DWR on verrattain helppo ottaa käyttöön, ja sen käyttäminen Struts-sovelluskehyksen kanssa on suoraviivaista.

4 Vaatimusmäärittely

4.1 Automatisaatio- ja ajastusmoduuli

4.1.1 Liiketoimintavaatimukset

Työn tavoitteena on manuaalisten taustatoimintojen ja ylläpitotyön vähentäminen automatisoinnin avulla. Liiketoiminnalliset tavoitteet ja vaatimukset on suurelta osin esitetty jo ongelman määrittelyn yhteydessä luvussa 2. Tässä yhteydessä listataan kuitenkin oleellisimmat ja työn tavoitteiden kannalta tärkeimmät vaatimukset.

Automatisaatio- ja ajastusmoduulille asetetaan liiketoiminnan kannalta neljä oleellista vaatimusta:

- Manuaalisen työn määrä tausta-, tuki-, ja ylläpitotoiminnoissa pitää saada vähenemään, jotta ylläpidon resursseja voidaan suunnata muualle.
- Automatisoitujen tehtävien suoritusten aikana tapahtuviin virhetilanteisiin reagoiminen pitää saada mahdollisimman pitkälle asiakkaan vastuulle, jotta ylläpidon resursseja voidaan säästää ja asiakas voi heti virheen ilmetessä korjata virheen ja jatkaa palvelun käyttöä täysipainoisesti.
- Uusien tehtävätyyppien kehittäminen ja integrointi ajastusmoduuliin tulee olla suoraviivaista, jotta sovelluskehitys olisi mahdollisimman tehokasta ja nopeaa.
- Automatisointi- ja ajastusmoduulin tulee parantaa palvelun houkuttelevuutta ja helpottaa palvelun myymistä asiakkaalle. Ajastusmoduuliin lisättävät tehtävät voivat olla asiakaskohtaisia ja näin mukautettavissa kunkin asiakkaan tarpeita ja toiveita vastaaviksi.

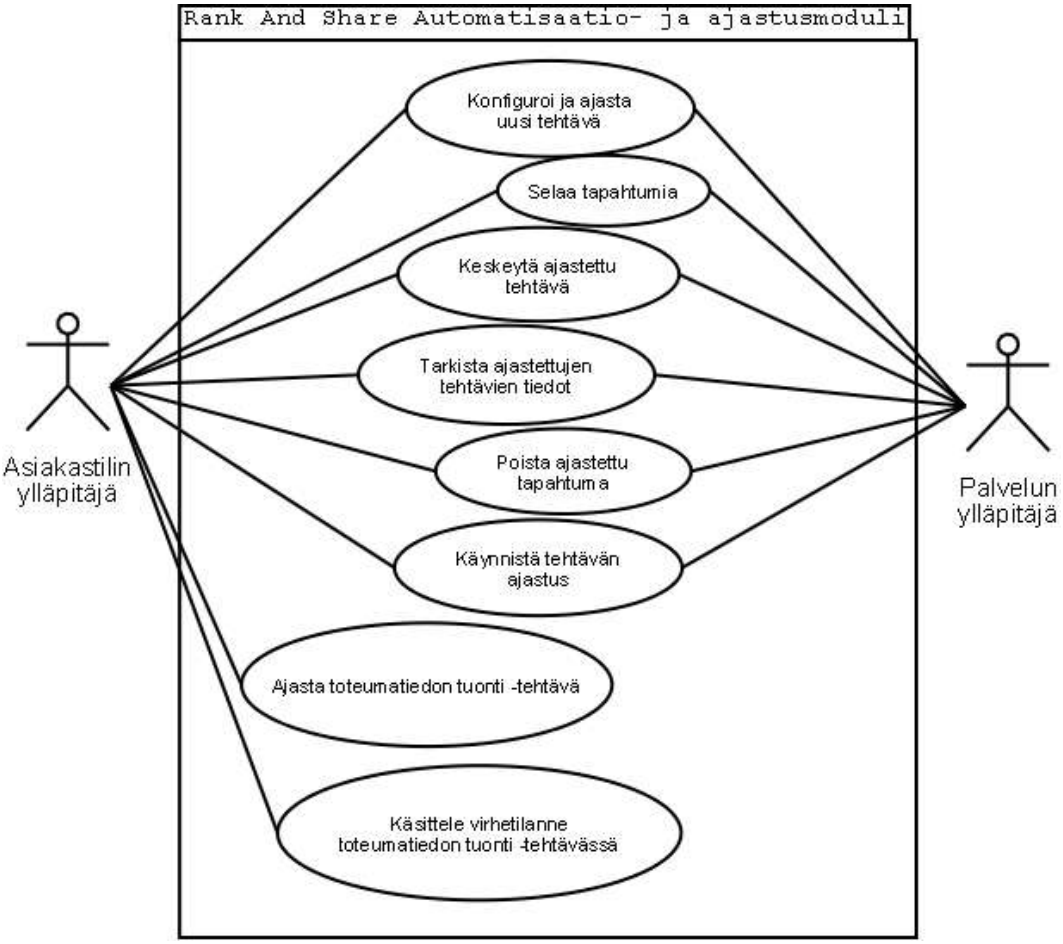
4.1.2 Käyttäjävaatimukset

Käyttäjävaatimuksilla käsitetään tässä yhteydessä niitä vaatimuksia jotka koskevat ajastettujen toimintojen hallintaa ja monitorointia sovelluksen käyttöliittymien kautta. Sovelluskehittäjän näkökulmasta tulevat vaatimukset uusien tehtävätyyppien lisäämisen osalta on käsitelty kappaleessa 4.2.

Käyttäjävaatimukset kartoitettiin ja kirjattiin ylös käyttötapauskuvausten muodossa. Ennen käyttötapauskuvausten keräämistä ja kirjoittamista tunnistettiin kaksi pääkäyttäjäryhmää, joiden käyttöön ajastusmoduuli pääasiallisesti tulee. Käyttäjäryhmät ovat järjestelmän käyttäjärooleja mukaillen palvelun ylläpitäjä ja asiakastilin ylläpitäjä. Peruskäyttäjillä ei ole pääsyä ajastusmoduuliin. Asiakastilin ylläpitäjän edustajiin ei voitu olla yhteydessä, joten kaikki käyttötapauskuvaukset tulivat palvelun ylläpitäjän edustajilta. Käytännössä palvelun ylläpitäjän roolissa olevat henkilöt olivat tavoitettavissa koko moduulin kehityksen ajan, joten uusien käyttötapausten kirjaaminen ja jo tehtyjen käyttötapausten tarkentaminen oli mahdollista ja sitä tehtiin aina kun siihen nähtiin tarvetta.

Käyttötapaukset kirjattiin ylös palaverien pohjalta, jotka käytiin käyttäjäryhmien edustajien kanssa. Käyttötapaukset on jaettu kahteen erilliseen kategoriaan, joista toinen käsittelee moduulin yleistä toiminnallisuutta ja toinen

esimerkkitehtävätyypiksi valittua toteumatiedon tuonti –tehtävätyypin käyttöä. Toteumatiedon tuonti –tehtävä toimii hyvänä esimerkkinä, sillä siinä virhetilanteiden esiintyminen on todennäköistä ja virhetyyppejä on useita.



Kuva 4 Vaatimusmäärittelysssä löydetyt käyttötapaukset

Toimijoina ovat palvelun ylläpitäjä ja asiakastilin ylläpitäjä. Kuvassa 4 on yhteenvetona esitetty tärkeimmät käyttötapaukset. Liitteessä 1 on kirjoitettu auki kukin käyttötapaus ja kussakin tapauksessa on esitetty seuraavat tiedot:

ID:	Käyttötapauksen yksilöivä tunniste
Käyttötapauksen nimi:	Lyhyt kuvaava nimi
Toimija:	Kuka on pääasiallinen toimija
Tavoite:	Mikä on suoritettavan tehtävän tavoite käyttäjän näkökulmasta.
Alkutilanne:	Lyhyt kuvaus tilanteesta, jossa ollaan käyttötapauksen alkaessa.
Kuvaus:	Kuvaus käyttötapauksen aikana tapahtuvista toiminnoista.
Poikkeukset:	Kuvaus oletetusta toiminnasta poikkeavista tapahtumista ja erikoistilanteista käyttötapauksen aikana.

Käyttötaajuus:	Miten usein käyttötapauksessa kuvattu tehtävä suoritetaan.
Tärkeys:	Kriittinen, tärkeä tai lisäominaisuus

4.2 Tehtävätyypin kehityksen vaatimukset

Uuden tehtävätyypin sovelluskoodin kehittäminen ja integrointi automatisointi- ja ajastusmoduuliin tulee olla mahdollisimman modulaarista, eli tehtävätyyppi voidaan kehittää erillään moduulista ja integroida siihen itsenäisenä osana. Uuden tehtävätyypin sovelluskehitys tulee siis erottaa integroinnista mahdollisimman suurelta osin. Alla on listattu yhdessä sovelluskehittäjien ja ylläpidon kanssa löydetyt vaatimukset liittyen tehtävätyypin sovelluskehitykseen ja integrointiin.

- Uusi tehtävätyyppi voidaan ohjelmoida olemassa olevaan järjestelmään ja siinä voidaan hyödyntää olemassa olevia komponentteja ja toimintoja esimerkiksi tietokantakäsittelyissä.
- Tehtävätyyppien konfigurointi pitää olla keskitettyä ja tehtävätyypeille pitää voida asettaa käyttöoikeuksia viitaten Rank & Share - sovelluksessa yleisesti käytettyihin rooleihin.
- Tehtävätyypin suorittaminen vaatii yleensä parametreja. Parametrien tyyppejä tai määrää ei saa rajoittaa. Parametrien syötön tulee tapahtua omassa käyttöliittymänäkymässä käyttäjän suorittaessa tehtävän ajastusta. Jokaiselle tehtävätyypille pitää pystyä helposti toteuttamaan parametrien syöttönäkymä.
- Tehtävätyypin toteuttavalle sovelluskehittäjälle pitää tarjota helppo tapa kirjoittaa lokitietoja tehtävätyypin sovelluslogiikan yhteyteen.
- Kaikki tehtävätyypeille yhteiset toiminnallisuudet pitää siirtää pois tehtävätyypin kehittäjän harteilta. Esimerkiksi lokitietojen tallentaminen pitää olla automatisointimoduulin toiminto.
- Tehtävän käyttämiä resursseja ei saa rajoittaa. (Pääsy tietokantaan ja palvelimella oleviin tiedostoihin).
- Tehtävätyypin integroinnin ja testauksen yhteydessä pitää järjestelmästä tulla kattavat virheilmoitukset ja lokitus asioista joita järjestelmä suorittaa tehtävätyypin ajastuksen ja ajon aikana, jotta sovelluskehittäjä pystyy selvittämään integroinnin aikana tapahtuneiden virheiden syyt ja korjaamaan virheet.

Yllä mainitut vaatimukset priorisoidaan kriittisiksi. Kaikki vaatimukset tulee olla täytettynä työn valmistuttua.

4.3 Tekniset vaatimukset

Automatisaatio- ja ajastusmoduulin toteutuksessa on otettava huomioon Rank & Share -sovelluksen rakenne ja luokkajako sekä rajapinnat. Olemassa oleva sovellus on toteutettu J2EE-kerrosmallia noudattaen, joten myös uuden toteutettavan moduulin tulee noudattaa samaa jakoa. Kerrosmalli asettaa näin olleen selkeät säännöt luokkajaon toteutukselle.

Automatisaatio- ja ajastusmoduuli tulee itsenäiseksi osaksi olemassa olevaa sovellusta, joten suunnitteluvaiheessa tulee ottaa huomioon ehto, jonka mukaan moduulin totetus ei saa vaatia olemassa olevan järjestelmän luokkien laajamittaista muokkausta. Näin ollen kullekin kerrosmallin kerrokselle on toteutettava omat luokat ajastusmoduulin toimintojen toteutukseen. Esimerkiksi jakelukerrokselle on toteutettava oma papu, jonka palvelut tarjotaan asiakasohjelmiston käyttöön paikallisen rajapinnan kautta. Sovelluslogiikka- ja tiedonkäsittelykerroksille toteutetaan niin ikään omat luokat automatisaatio- ja ajastusmoduulin käyttöön.

VO-luokkien käyttö tapahtuu Hibernaten kautta ja luokat kirjoitetaan tietokantaan käyttäen Hibernaten xml-pohjaista kuvausta. Hibernaten lisäksi sovelluksessa on mahdollista käyttää SQL-kyselykieltä tietokanta-operaatioiden toteutuksessa, mutta tämä ei ole suositeltavaa. Poikkeuksena tähän ovat erityisen raskaat tietokantaoperaatiot, jotka suorituskykysyistä on parempi toteuttaa suoraan SQL-kielellä Hibernaten sijaan.

Tietokantana käytetään PostgreSQL-tietokantapalvelinta. SQL-kielellä toteutettavat kyselyt pitää mahdollisimman pitkälle toteuttaa standardi-SQL:llä, jotta kyselyiden toiminta ei riipu käytettävästä tietokantapalvelimesta. Näin tietokantapalvelimen vaihto aiheuttaa mahdollisimman vähän muutoksia DAO-toteutuksiin.

Esitystapakerroksen toteutus ja käyttöliittymät tehdään Struts-sovelluskehystä käyttäen. Käyttöliittymätoteutuksessa käyttöön otetaan ajax-teknologia, mutta sen käyttöä on harkittava tapauskohtaisesti. Ajax-teknologia on otettava huomioon myös toiminnonkäsittelijöiden suunnittelussa, sillä DWR-ohjelmistokirjaston käyttämien metodien toteuttaminen poikkeaa ratkaisevasti normaaleista Struts:n käyttämistä metodeista [11].

Käyttöliittymien toteuttamisessa käytetään html-kieltä ja css-määrittelyitä. Olemassa olevia tyylimäärittelyitä tulee käyttää yhteneväisesti muun sovelluksen kanssa. Java ohjelmakoodin kirjoituksessa tulee noudattaa Sun:n määrittelemiä ohjelmointikäytäntöjä. Tietoa ohjelmointikäytännöistä löytyy lähteestä [12].

5 Suunnittelu

5.1 Automatisaatio- ja ajastusmoduuli

5.1.1 Yleistä

Rank & Share -sovellus on toteutettu Java-ohjelmointikielellä. Java on puhtaasti oliopohjainen kieli, joten luonnollisesti suunnitteluvaiheessa käytetään olio-pohjaista suunnittelua. Suunnittelussa pyritään noudattamaan kappaleessa 3.3 esitettyjä olio-suunnittelun laatukriteereitä: heikkoa kytkentää ja vahvaa yhtenäisyyttä. Luokkajako ja luokkien vastuualueet on pyrittävä suunnittelemaan siten, että nämä kaksi kriteeriä pystytään täyttämään mahdollisimman hyvin.

Rank & Share -sovelluksen rakenne ja eri kerroksissa olevien luokkien suhteet, yhteistyö ja vastuualueet on valmiiksi määritelty, joten suunnittelu tältä osin helpottuu huomattavasti. Sovelluksessa olevat valmiit luokat ja niiden tarjoamat metodit tarjoavat paljon valmista toiminnallisuutta, jota ajastusmoduulin toteutuksessa voi käyttää hyväksi.

Moduulin rakenteen suunnittelun tuloksena syntyy luokkamalli, jossa esitetään moduulin toteutuksen kannalta oleelliset luokat, niiden sisältämät tiedot ja luokkien väliset suhteet. Luokkamallin lisäksi esitellään J2EE-kerrosmallin mukaiset luokat jakelukerroksella ja sen alapuolella, luokkien sisältämät toiminnallisuudet pääpiirteissään sekä rajapinnat jakelukerroksen ja esityskerroksen välissä. VO-luokkien kuvautuminen tietokantatauluihin sekä mahdolliset muut tietokantarakenteet esitellään myös.

5.1.2 Luokkamalli

Luokkamallin suunnittelussa ja luokkien tunnistamisessa käytetään menetelmää, jossa käyttötapauksista etsitään substantiivit. Kunkin substantiivin kohdalla mietitään onko se sopiva luokaksi vai luokan attribuutiksi. Vastaavasti verbit ovat toiminnallisuuksia, jotka voidaan määrittää tietyn luokan metodeiksi. Ajastusmoduulin tapauksessa toiminnallisuudet siirtyvät suoraan sovelluslogiikkakerrokselle BO-luokan hoidettavaksi. Näin ollen luokkamalliin pitää löytää asianmukaiset VO-luokat, jotka kuvaavat järkevästi moduulin toteutuksessa tarvittavia tietokokoelmia.

Käyttötapauksista kerätyt luokkaehdokkaat on esitetty taulukossa 1. Taulukossa on lisäksi kunkin ehdokkaan kohdalla merkitty sen kelpoisuus luokaksi tai luokan attribuutiksi.

Taulukko 1 Käyttötapauksissa esiintyvät substantiivit

Nimi	**Tyyppi**	**kuvaus**
Asiakastilin ylläpitäjä	Attribuutti	Käyttäjän rooli
Järjestelmän ylläpitäjä	Attribuutti	Käyttäjän rooli
Ajastettu tehtävä	Luokka	Yksittäisen ajastetun tehtävän tietojen kokelma.
Tehtävän parametrit	Attribuutti	Tehtävälle konfiguroidut asetukset

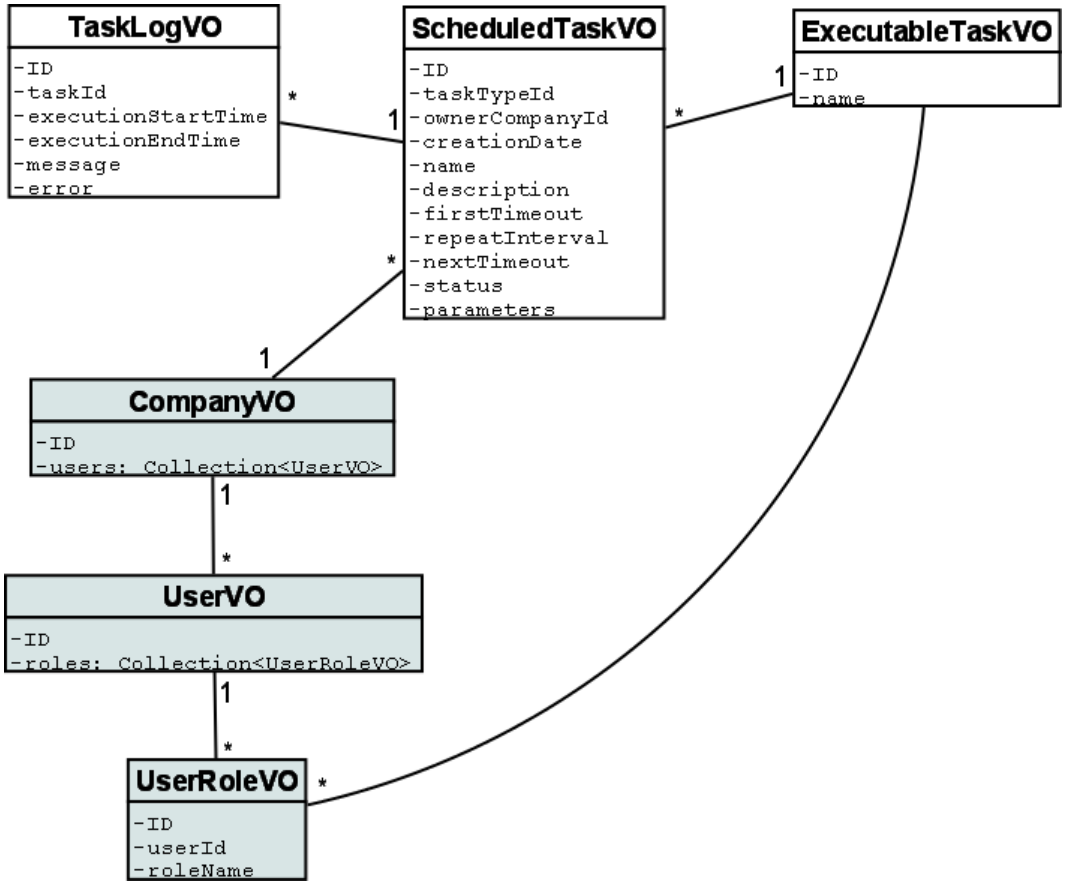
Käyttäjä	Luokka	Sovellusta käyttävä henkilö.
Palvelu	muu	Yleisnimitys koko järjestelmästä
Suoritusten aikaväli	Attribuutti	Tehtävälle asetettu tieto. Kuvaa toistuvasti suoritettavan tehtävän suoritusten välisen ajan.
Tehtävän nimi	Attribuutti	Tehtävälle asetettu nimi.
Tehtävän kuvaus	Attribuutti	Tehtävälle asetettu vapaamuotoinen lisätieto.
Ensimmäinen suoritusajankohta	Attribuutti	Tehtävälle asetettu ensimmäisen suorituksen ajankohta.
Tehtävän tyyppi	Attribuutti	Ajastetulle tehtävälle määritellyn suoritteen tyyppi.
Tehtävätyyppi	Luokka	Järjestelmässä ajastettavissa oleva tehtävätyyppi.
Tehtävän informaatio	Muu	Yleisnimitys ajastetun tehtävän tiedoista käyttäjän näkökulmasta.
Tehtävän seuraava suoritusajankohta	Attribuutti	Ajastetun tehtävän seuraavan suorituksen ajankohta.
Tehtävän tila	Attribuutti	Ajastetun tehtävän tila. Esimerkiksi”suoritettu onnistuneesti” tai ”suoritus keskeytynyt virheen takia”
Yritys	Luokka	Käyttäjän edustama yritys.
Tehtävän luontipäivämäärä	Attribuutti	Ajankohta, jolloin tehtävä on luotu järjestelmään.
Lokitieto	Luokka	Sisältää kokoelman tietoja tehtävän suorituksesta.
suorituksen aloitusajankohta	Attribuutti	Koska ajastettu tehtävä on aloittanut suorituksen.
suorituksen kesto	Attribuutti	Kertoo, miten kauan ajastetun tehtävän suoritus on kestänyt.
suorituksen tila	Attribuutti	Suoritetun tehtävän tila suorituksen jälkeen.
Tehtävän omistajayritys	Attribuutti	Mikä yritys omistaa ajastetun tehtävän.
Virhe	Attribuutti	Ajastetun tehtävän suorituksen aikana tapahtunut virhe.
Virheilmoitus	Attribuutti	Virheen yhteydessä oleva selite.

Käyttötapausten perusteella tunnistetut luokat, attribuutit ja suhteet on esitetty kuvassa 5. Kuvassa on merkitty harmaalla luokat, jotka on jo toteutettu sovellukseen aikaisemmin. Kaikki luokat ovat VO-luokkia, eli sisältävät pelkästään joukon kenttiä. Kaikki VO-luokat sisältävät erityisen ID-kentän, jota käytetään tietokantatasolla olion yksilöivänä tunnisteena.

Keskeisenä luokkana on ScheduledTaskVO-luokka, joka edustaa yksittäistä ajastettua tehtävää. Ajastettu tehtävä luodaan käyttäjän antamien tietojen perusteella ja tallennetaan tietokantaan. Ajastettuun tehtävään liitetään tiedot muun muassa ajastuksista, omistajayrityksestä sekä tehtävätyypin tunniste. Tehtävälle tallennetaan tehtävätyyppikohtaiset parametrit, jotka välitetään tehtävän suoritusmetodille ajon alkamisen yhteydessä eli kun ajastus laukeaa.

Tehtävätyyppiä edustaa ExecutableTaskVO-luokka. Luokan rooli moduulissa on kertoa yksityiskohtaisesti, mikä sovelluslogiikan mukainen metodi suortetaan ajastetun tehtävän ajastuksen lauetessa. Tehtävätyypille määritellään lisäksi käyttäjärooli tai -roolit, joilla on oikeus ajaa tehtävätyypin toiminto.

TaskLogVO edustaa yksittäistä lokitietoa. Ajastettu tehtävä generoi lokitiedot jokaisesta suoritetusta tehtävästä. TaskLogVO sidotaan yksittäiseen tehtävään tehtävän tunnisteella eli ID:llä. Lokitietoon tallennetaan suorituksen alkamis- ja loppumisajankohta, joista voidaan laskea suorituksen kesto. Virheellisen suorituksen tuloksena lokitietoon tallennetaan tiedot virheestä.



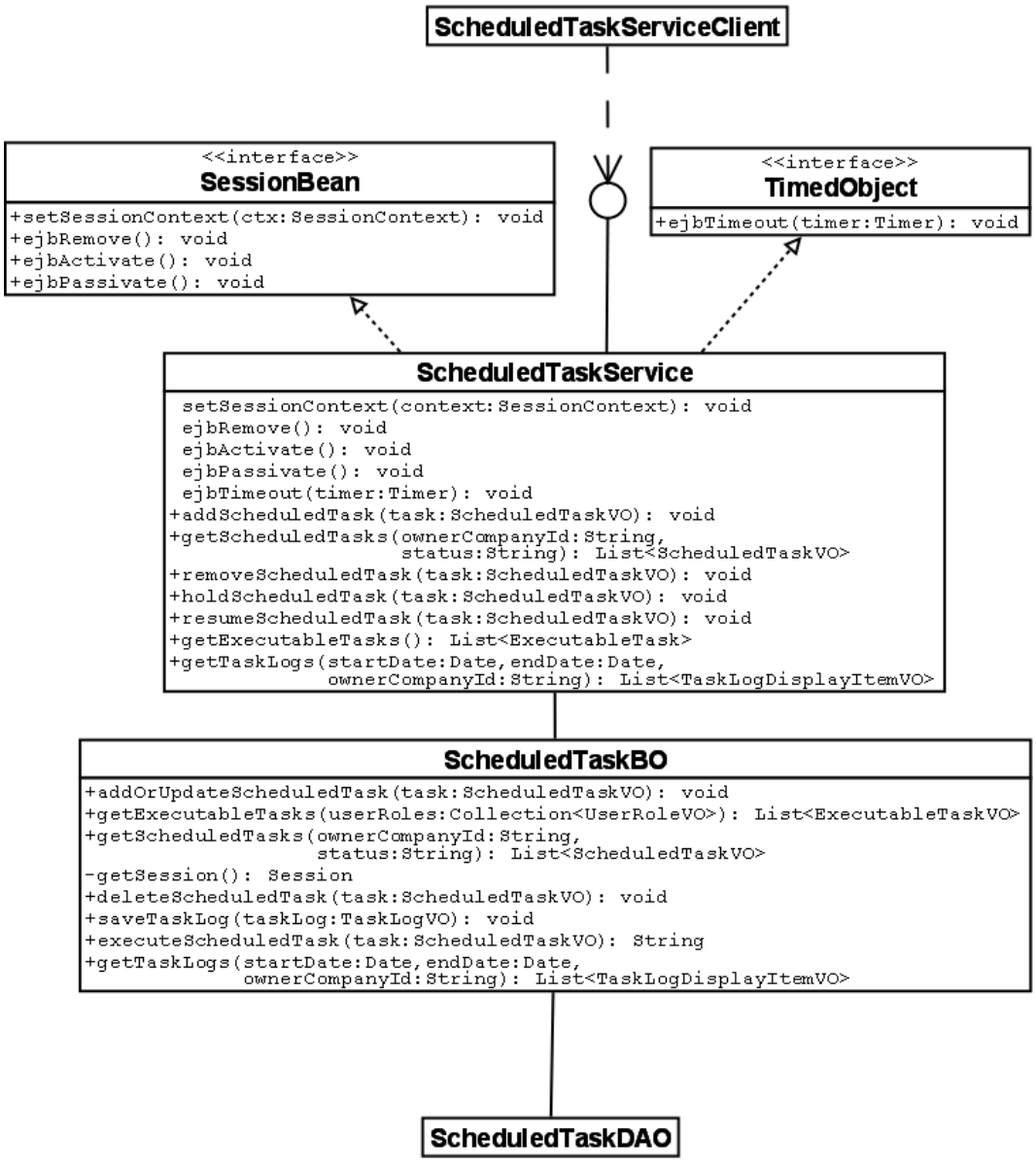
Kuva 5 Käyttötapausten pohjalta sunniteltu luokkamalli

Käyttötapausten perusteella luotu tietomalli on hyvin yksinkertainen ja käsittää ainoastaan kolme luokkaa olemassa olevien luokkien lisäksi. Luokkien kentistä on

esitetty kuvassa 5 ne, jotka käyttötapausten mukaan löydettiin. Kenttien määrä tulee toteutusvaiheessa mahdollisesti muuttumaan. Kenttien tyypit määritellään niin ikään lopullisesti vasta toteutusvaiheessa.

5.1.3 Kerrosmalli ja rajapinnat

Automatisaatio- ja ajastusmoduulin kerrosmallin mukainen toteutus jakelu-, sovelluslogiikka ja tiedonkäsittelykerroksien osalta toteutetaan yksittäisinä luokkina kullakin kerroksella. Kuvassa 6 esitetään kunkin kerroksen luokat sekä oleellisimmat rajapinnat. ScheduledTaskServiceClient- ja DAO-luokkien metodien määritykset ovat käytännössä samat kuin ScheduledTaskServicen ja -BO:n joten niitä ei kuvassa ole esitetty.



Kuva 6 Moduulin sovelluslogiikka ja EJB-papu sekä rajapinnat

ScheduledtaskService-luokka edustaa varsinaista EJB-papua, jonka kautta ScheduledTaskBO:ssa toteutetut toiminnallisuudet tarjotaan asiakasohjelmiston käyttöön. Kuvassa 6 on esitetty luokkien tärkeimmät toiminnallisuudet

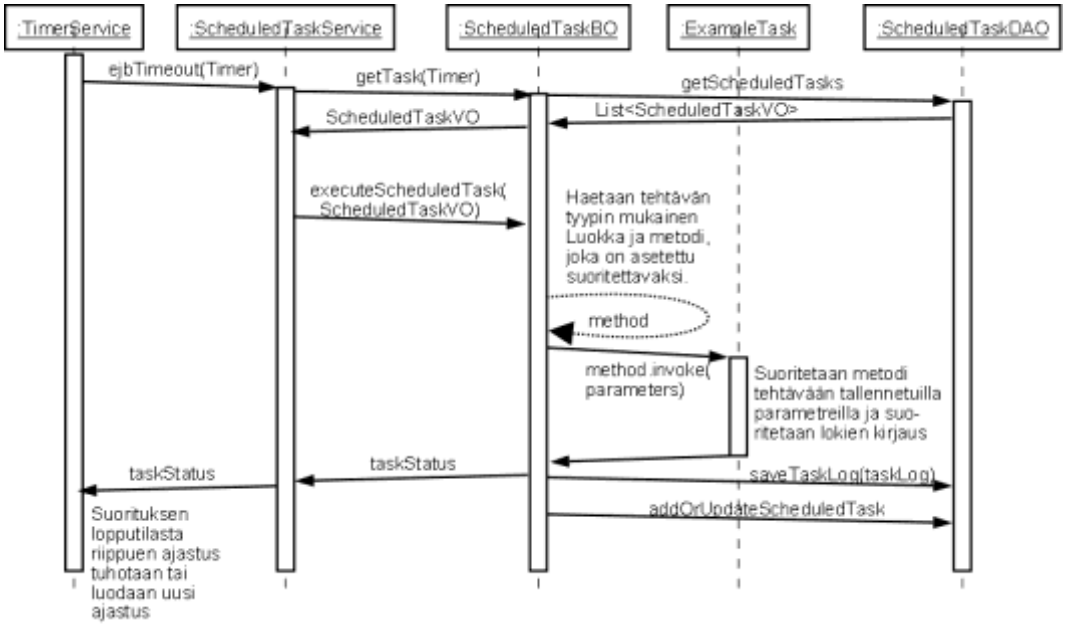
metodinimien muodossa. Nämä toiminnallisuudet tarjotaan asiakasohjelmistoille EJB rajapintojen ScheduledTaskServiceLocal ja ScheduledTaskServiceRemote kautta.

Tietokanta- ja xml-käsittelyt toteutetaan DAO-luokassa. Hibernaten kautta käytetyt oliot voitaisiin tallentaa ja ladata suoraan BO-tasolla, mutta selkeyden vuoksi kaikki tietokantakäsittelyt tehdään DAO:ssa. Hibernaten transaktioiden ja istuntojen hallinta suoritetaan BO-tasolla, joten DAO:ssa suortetut tietokantatoimenpiteen liittyvät ylemmällä tasolla aina yksittäiseen istuntoon.

ScheduledTaskService-luokassa on kaksi erityistä ominaisuutta, joiden avulla EJB Timer Service -ajastuspalvelua käytetään. SessionContext-kentän kautta päästään käsiksi sovelluspalvelimelle asetettuihin ajastimiin eli Timer-olioihin. SessionContext-olio tarjoaa pääsyn TimerService-olioon, jonka kautta ajastusten lisäämiset ja poistot suoritetaan.

5.1.4 EJB Timer Service

Ajastuksen laukeamisen tapahtuessa ajastuspalvelu kutsuu TimedObject-rajapinnan ejbTimeout-metodia. Metodin toteutus ScheduledTaskService-luokassa toimii liitoskohtana tietokannassa olevan ScheduledTaskVO-olion ja sovelluspalvelimella olevan Timer-olion välillä. Kukin Timer-olio kytketään yhteen ScheduledTaskVO-olioon ajastuksen luontivaiheessa. Kytkentä tapahtuu Timer-olion info-kentän avulla. Info-kentän arvo vastaa ScheduledTaskVO-olion timerInfo-kentän arvoa. Näin löydetään tehtävä, joka kunkin ajastuksen lauetessa tulee suorittaa. Kuvassa 7 on esitetty sekvenssikaavio ajastuksen laukeamisesta.



Kuva 7 Ajastuksen laukeaminen sekvenssikaaviona

Ajastetun tehtävän suorittaminen siirtyy ejbTimeout-metodista BO-luokan executeScheduledTask-metodille. Metodissa suoritetaan tehtävälle asetetun tehtävätyypin ajo, lokititetojen tallennus, mahdollisten virhetilanteiden käsittely sekä ajastetun tehtävän tietojen päivitys ja tallennus tietokantaan.

J2SE tarjoaa menetelmän, jolla luokan nimen perusteella voidaan luoda olio kyseisestä luokasta [13]. ScheduledTaskVO luokassa olevia kenttiä className ja

methodName käytetään suoritettavan metodin etsimiseen. Oikean metodin löytymisen jälkeen metodia kutsutaan ScheduledTaskVO-olioon tallennetuilla tehtävätyyppikohtaisilla parametreilla. Suorituksen jälkeen tehtävän tiedot päivitetään ja tehtävän suorituksen aikana kerätyt lokitiedot kirjoitetaan tietokantaan.

Lokitietojen kerääminen tehtävän suorituksen aikana tapahtuu erityisen LoggableTask-luokan avulla. LoggableTask-luokka on kaikkien tehtävätyyppiluokkien yliluokka ja tarjoaa lokitustyökalut tehtävätyypiluokalle. LoggableTask-luokka mahdollistaa ajon aikaisiin lokitietoihin pääsyn suorituksen loputtua executeScheduledTask-metodissa.

5.1.5 Tietokanta

Tietokantataulujen rakenne määräytyy käytännössä ScheduledTaskVO- ja TaskLogVO-luokkien kenttien mukaisesti. Luokat kuvataan tietokantatauluiksi Hibernaten kuvauskielellä xml-muotoisena. Suunnitteluvaiheessa ei voida vielä tarkasti tietää, mitä tietoja kyseiset luokat pitävät sisällään, joten tietokannan rakennetta ja määrittelyitä ei ole tarpeellista suunnitella. Hibernaten luonteen ja käyttötarkoituksen mukaisesti tietokantataulujen suunnittelua ei pitäisi tarvita tehdä lainkaan, vaan ne muotoutuvat Hibernate-kuvausten perusteella.

5.2 Käyttöliittymät

5.2.1 Yleistä

Käyttöliittymien suunnittelun tavoitteena on määritellä käyttöliittymän näkymät, niissä olevat toiminnallisuudet, esitettävät tiedot ja siirtymät toisiin näkymiin. Suunnittelun tuloksena saadaan hahmotelmat näkymien rakenteesta ja sisällöstä. Lisäksi luodaan lista informaatiosta, jonka kukin näkymä pitää sisällään. Suunnitteluprosessin tärkeimpänä tavoitteena on luoda kaaviokuva, jossa on esitetty näkymät, näkymien väliset siirtymät, toiminnonkäsittelijät sekä lomakeluokat ja muut apuluokat joita käytetään näkymien generoimisessa ja tietojen välittämisessä. Suunnitteluvaiheessa lyödään myös lukkoon JSP-sivujen ja esityskerroksella käytettyjen luokkien nimet.

Käyttöliittymien suunnittelussa käytetään ennen kaikkea käyttäjävaatimuksissa esiin tulleita käyttötapauksia ja toiminnallisia vaatimuksia. Näistä pystytään tunnistamaan käyttöliittymän eri näkymät ja niissä olevat toiminnallisuudet. Liitteessä 3 on esitetty vaatimusmäärittelyn perusteella löydetyt näkymät ja niiden ominaisuudet, siirtymät ja toiminnonkäsittelijät. Käyttöliittymien toiminnallisuuksissa vaadittava sovelluslogiikka voidaan melko suoraviivaisesti siirtää toiminnonkäsittelijöiden kautta EJB-pavulle ja siitä eteenpäin sovelluslogiikkakerrokselle.

5.2.2 Näkymät

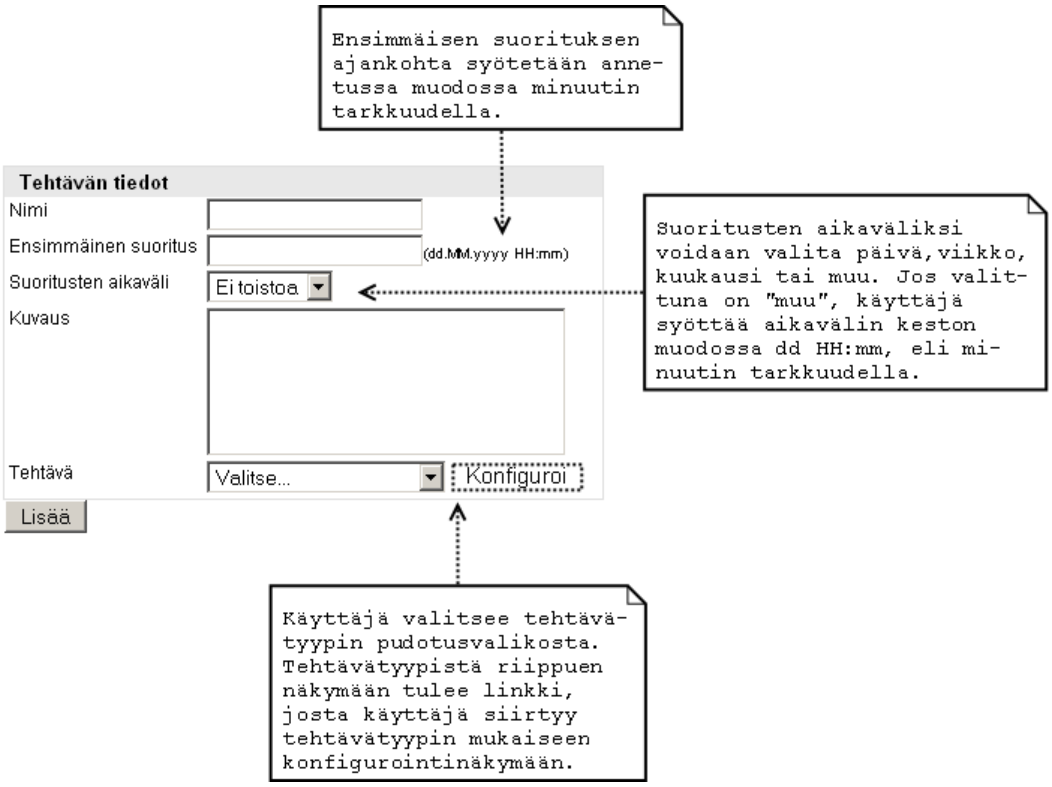
Rank & Share -palvelun yhtenä tärkeimmistä eduista pidetään ennusteprosessin läpiviemisessä säästettävän ajan määrää perinteiseen ennustamiseen verrattuna. Palvelun käyttäjän pitää pystyä suorittamaan ennusteprosessi nopeasti ja johdonmukaisesti. Näin ollen käyttöliittymien pitää olla yksinkertaisia ja helppoja

käyttää. Yksinkertaisuutta noudatetaan myös ajastusmoduulin käyttöliittymien suunnittelussa.

Automatisaatio- ja ajastusmoduulin käyttöliittymän toiminnot voidaan toteuttaa neljällä erillisellä näkymällä. Liitteen 3 mukaisen listauksen perusteella näkymät voidaan tunnistaa:

1. Ajastettavan tehtävän lisääminen
2. Tehtävätyypille ominaisten asetusten syöttö
3. Ajastettujen tehtävien ylläpito
4. Ajastettujen tehtävien loki

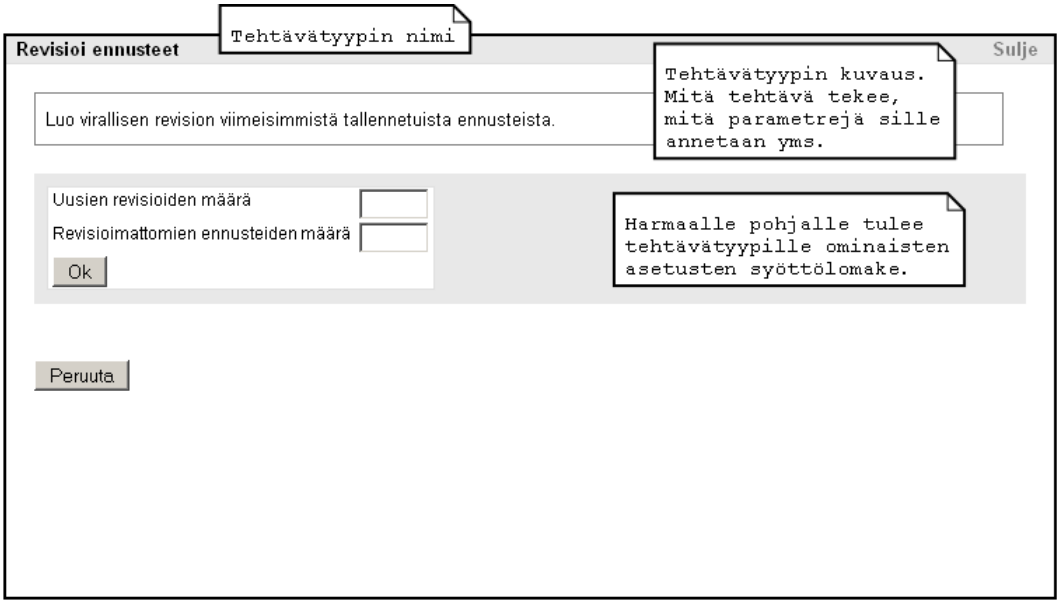
Ajastettavan tehtävän lisääminen järjestelmään tapahtuu ylläpitäjien toimesta. Tehtävää ajastettaessa täytyy tehtävälle määritellä nimi, ensimmäisen suorituksen ajankohta, toistoväli ja tehtävätyyppi. Käyttäjä voi lisätä tehtävälle myös vapaamuotoisen kuvauksen. Palvelun ylläpitäjä voi lisäksi liittää ajastettavan tehtävän yksittäiseen yritykseen. Tällöin näkymässä on pudotusvalikko, josta ylläpitäjä valitsee yrityksen nimen. Kuvassa 8 on esitetty hahmotelma tehtävän lisäämisnäkymästä.



Kuva 8 Ajastettavan tehtävän lisääminen

Ajastettavan tehtävän lisäämisnäkymän alimmaisena valikkona on pudotusvalikko, josta käyttäjä valitsee tehtävätyypin, joka suoritetaan ajastuksen lauetessa. Tehtävätyypin asianmukainen suoritus yleensä vaatii, että tehtävätyypille annetaan tarkentavia parametreja. Käyttäjä voi siirtyä tehtävätyypin parametrien asetusnäkymään valitsemalla pudotusvalikon vierestä ”konfiguroi”. Jos tehtävä ei ota vastaan parametrejä, ”konfiguroi”-linkkiä ei näytetä.

Tehtävätyypin parametrien syöttönäkymässä esitetään tehtävätyypin kuvaus sekä tehtävätyypille ominainen lomake, johon käyttäjä syöttää parametrit tehtävätyypille. Kuvassa 9 on esitetty esimerkki mahdollisesta tehtävätyypin konfigurointinäkymästä. Harmaa alue näkymän keskellä on tarkoitettu tehtävätyypin lomakkeelle. Kuvassa 9 näkymä toteutetaan omana html-kehyksenä (iframe), joka ilmestyy tehtävän ajastusnäkymän (Kuva 8) päälle käyttäjän painaessa ”Konfiguroi”-linkkiä. Tehtävätyypin konfigurointinäkymästä voi poistua joko oikean yläkulman linkistä tai peruuta-napista. Parametrien hyväksyminen tapahtuu ”Ok”-painikkeesta. Tällöin konfigurointinäkymä sulkeutuu ja fokus palautuu näkymään 1.



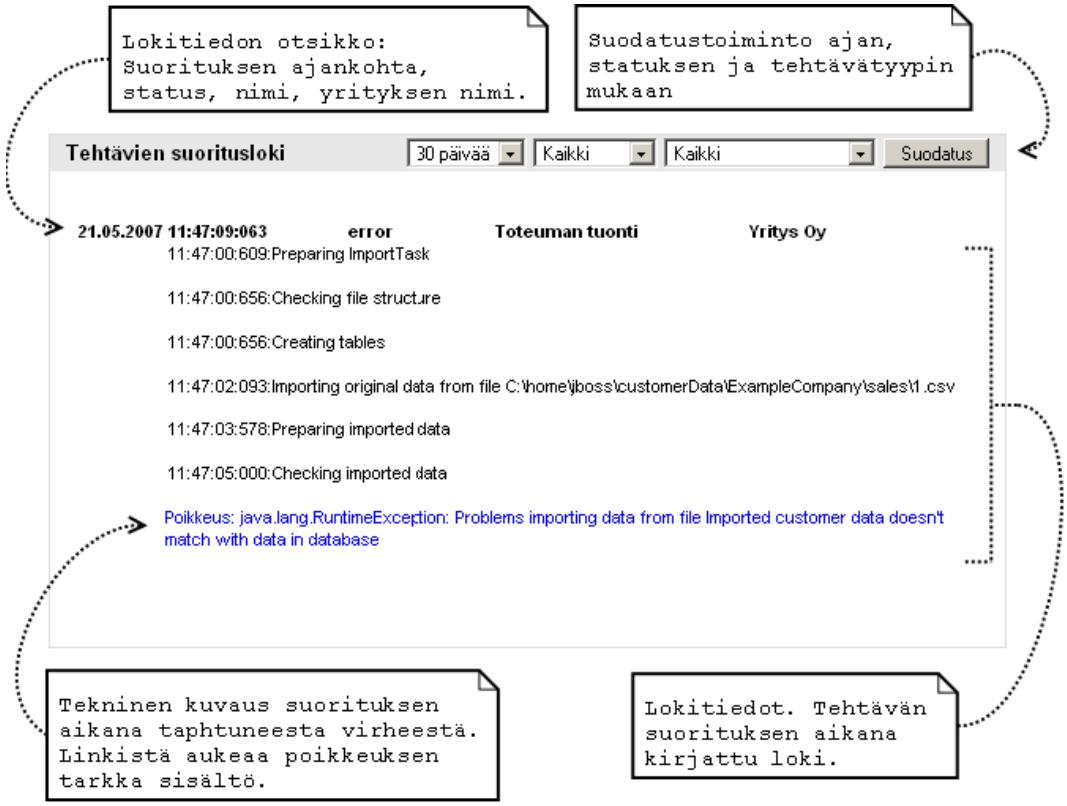
Kuva 9 Tehtävätyypille ominaisten asetusten syöttö

Ajastettujen tehtävien ylläpitonäkymässä esitetään lista kaikista käyttäjän yrityksen ajastetuista tehtävistä. Jokaisen tehtävän kohdalla on esitetty tehtävälle oleelliset tiedot. Kuvassa 10 on esitetty hahmotelma ylläpitonäkymästä. Tehtävälistan oikealla laidalla on linkit ylläpitotoimintoihin. Ajoita-toiminto kysyy käyttäjältä ensimmäisen suorituksen ajankohtaa erillisessä kehyksessä, joka tulee näkyville ylläpitonäkymän yhteyteen. Ajankohdan syöttö tapahtuu antamalla päivämäärä ja kellonaika minuutin tarkkuudella.

Kuva 10 Ajastettujen tehtävien ylläpito

Ajastettujen tehtävien lokinäkymän tehtävänä on esittää ylläpitäjälle tietoja ajastusmoduulissa tapahtuneista toiminnoista ja suorituksista. Kuvassa 11 on esitetty hahmotelma lokinäkymästä. Näkymässä esitetään kahdentyyppisiä toimintoja: Tehtävien suorituslokeja ja tehtävien ylläpitolokeja. Tehtävien

suorituslokit kertovat yksittäisen tehtävän ajon aikaisista tapahtumista ja ylläpitoloki tehtävien lisäyksistä, poistoista ja ajastuksen keskeytyksistä. Molemmat lokityypit esitetään samassa listassa. Jokaisesta yksittäisestä lokitiedosta näytetään perustiedot. Tehtävän suorituksen aikana tapahtuneesta virheestä käyttäjälle näytetään virheen kuvaus ja linkki, josta voi lukea virheen yksityiskohtaisempaa tietoa. Yksityiskohtainen tieto aukeaa suoraan virhelinkin alle.



Kuva 11 Ajastettujen tehtävien loki

Lokitietojen suodattaminen tehdään valitsemalla pudotusvalikoista halutut parametrit ja painamalla suodata-nappia. Tällöin lokinäkymässä olevat lokitiedot päivitetään vastaamaan suodatusparametrejä. Ajan mukaan suodattaminen tapahtuu valitsemalla pudotusvalikosta tietty määrä päiviä. Esimerkiksi valitsemalla ”30 päivää”, lokitedot näytetään viimeisen 30 päivän ajalta. Statuksen mukainen suodatus voidaan tehdä seuraavilla parametreilla: ”error”, ”completed”, ”invalid” tai ”kaikki”. Tehtävätyypin mukaisessa suodattamisessa voi valita kaikkien käytössä olevien tehtävätyyppien väliltä. Yrityksen mukainen suodattaminen on mahdollista ainoastaan järjestelmän ylläpitäjän roolissa. Tällöin pudotusvalikossa näkyy kaikki järjestelmään rekisteröityneet yritykset.

5.2.3 Toiminnonkäsittelijät ja siirtymät

Tässä kappaleessa esitellään edellisessä kappaleessa esitettyjen näkymien toiminnallisuuksien toteuttamiseen tarvittavat komponentit eli toiminnonkäsittelijät, JSP-sivut ja VO-luokat. Lisäksi esitellään sovelluspalvelimella olevan EJB-pavun asiakasluokka ja sen kautta käytössä olevat toiminnallisuudet. Kunkin näkymän päätoiminnallisuudet esitetään omissa

kaavioissaan, johon on merkitty tarvittavat luokat ja siirtymät sekä tarvittavat parametrit JSP-sivuilla ja lomakeluokissa.

Käyttöliittymän toiminnallisuuksien toteuttaminen Rank & Share -sovelluksessa tapahtuu Struts-sovelluskehyksen tarjoamilla työkaluilla ja sen rakennetta noudattaen. Suunnitteluvaiheessa tämä on otettava luonnollisesti huomioon. Struts-kehyksen toiminnalliset periaatteet on esitetty kappalessa 3.5.1.

Rank & Share -sovelluksessa noudatetaan tiedostojen nimeämisessä tiettyä käytäntöä, jotta erilaisten luokkien erottaminen on helppoa. Toiminnonkäsittelijät saavat nimen loppuun sanan ”Action” ja lomaketietoja välittävät luokat ”FormBean”.

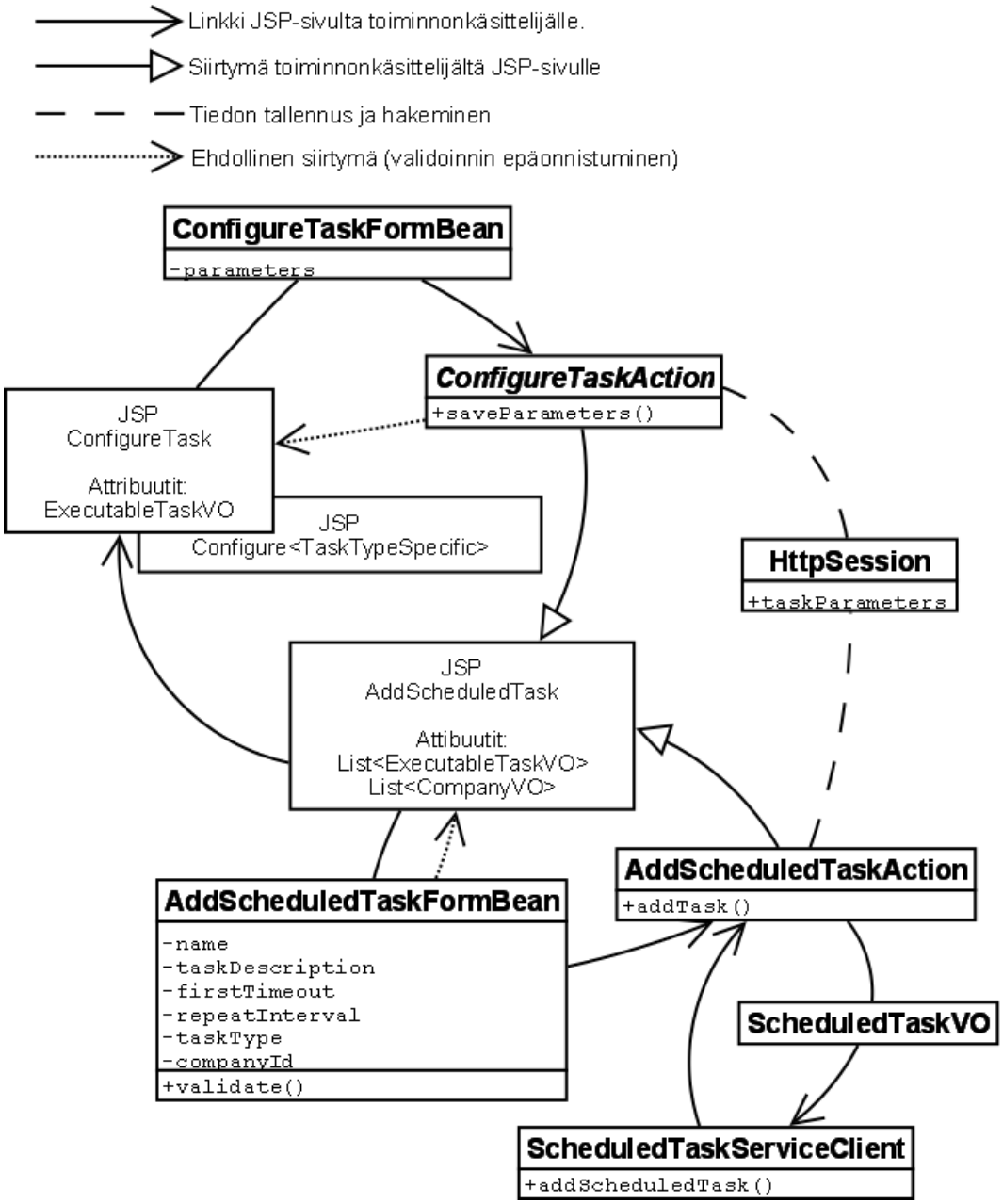
5.2.3.1 Ajastetun tehtävän lisääminen

Ajastetun tehtävän lisääminen tapahtuu kahdessa vaiheessa. Ensimmäisessä vaiheessa käyttäjä syöttää ajastettavalle tehtävälle perustiedot, ajastusasetukset sekä valitsee tehtävätyypin, joka suoritetaan tehtävän ajastuksen lauetessa. Toisessa vaiheessa käyttäjä syöttää valitsemalleen tehtävätyypille ominaiset asetukset erillisestä näkymästä. Kuvassa 12 on esitetty näiden toimintojen toteuttamisessa käytettävät luokat, JSP-sivut ja siirtymät. Lisäksi JSP-sivuihin on merkitty tiedot, jotka tarvitaan JSP-sivun kääntämisen yhteydessä. Nämä tiedon on tallennettu näkymään tultaessa istuntoon eli HttpSession-luokan olioon.

Toisen vaiheen tietojen syöttö tapahtuu ConfigureTask-sivulla. ConfigureTask JSP-sivu käsittää iframe-kehyksen, jossa esitetään tehtävätyypin perustiedot ja tekniset tiedot. Tämä sivun yhteyteen lisätään tehtävätyypille ominaisten parametrien syöttölomake. Lomake ottaa vastaan parametrit ja ne välitetään tehtävätyypin toiminnonkäsittelijälle ConfigureTaskFormBean-luokkaa käyttäen. Tässä luokassa ei suoriteta tietojen oikeellisuuden tarkistusta eli validointia, vaan validointi tapahtuu vasta tehtävätyypin asetusten toiminnonkäsittelijässä. Jokaiselle tehtävätyypille kirjoitetaan siis oma toiminnonkäsittelijä, jossa suoritetaan parametrien validointi ja niiden tallettaminen istunto-olioon. Väärien tai vajaiden tietojen syöttö aiheuttaa paluun takaisin tehtävätyypin parametrien syöttönäkymään. Tehtävätyypin parametrien asettamisen jälkeen siirrytään takaisin tehtävän perustietojen syöttönäkymään. ConfigureTaskAction on tehtävätyyppikohtaisten toiminnonkäsittelijöiden abstrakti yliluokka, jonka tarkempi kuvaus ja toiminta esitellään kappaleessa 5.3.

AddScheduledTask JSP-sivulta käyttäjä syöttämät tiedot välitetään toiminnonkäsittelijälle AddScheduledTaskFormBean-luokkaa käyttäen. Lomakkeelle syötettyjen tietojen kelvollisuuden tarkistus suoritetaan validate()-metodissa. Mikäli lomaketiedot ovat puutteellisia tai väärän muotoisia, siirrytään takaisin JSP-sivulle. Sivulle välitetään virheviestinä syy, miksi lomaketietojen validointi epäonnistui. Onnistuneen validoinnin jälkeen kontrolli siirtyy AddScheduledTaskAction-toiminnonkäsittelijälle, joka hakee istunto-oliolta tehtävätyypin parametrit.

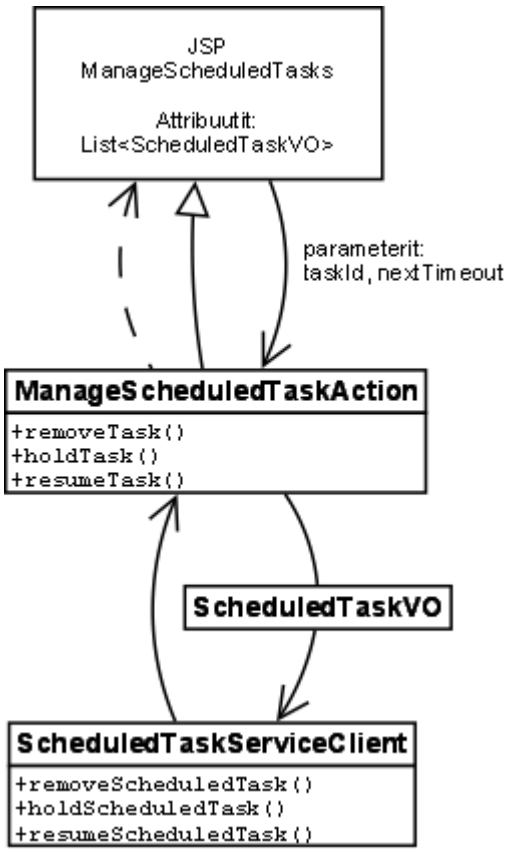
Perustiedot, ajastusasetukset ja tehtävätyypin parametrit kerätään kokoon AddScheduledTaskAction-toiminnonkäsittelijässä. Tiedot kootaan ScheduledTaskVO-olioksi ja lähetetään sovelluslogiikalle ScheduledTaskServiceClient-luokan avulla.



Kuva 12 Ajastetun tehtävän lisääminen

5.2.3.2 Ajastettujen tehtävien ylläpito

Ajastettujen tehtävien ylläpitonäkymä toteutetaan ManageScheduledTasks JSP-sivuna. Näkymässä esitettävät tiedot voidaan hakea ScheduledTaskVO-olioina kannasta ilman, että esitystapakerroksella tarvitaan erillistä apuluokkaa. Ylläpitonäkymän toiminnot eli ajastuksen keskeytys, tehtävän poisto ja tehtävän uudelleenajastus toteutetaan ManageScheduledTaskAction-luokassa. Kuvassa 13 on esitetty näkymään liittyvät komponentit. Näkymän toiminta on suoraviivaista eikä siirtymiä muihin näkymiin ole.



Kuva 13 Ajastettujen tehtävien ylläpito

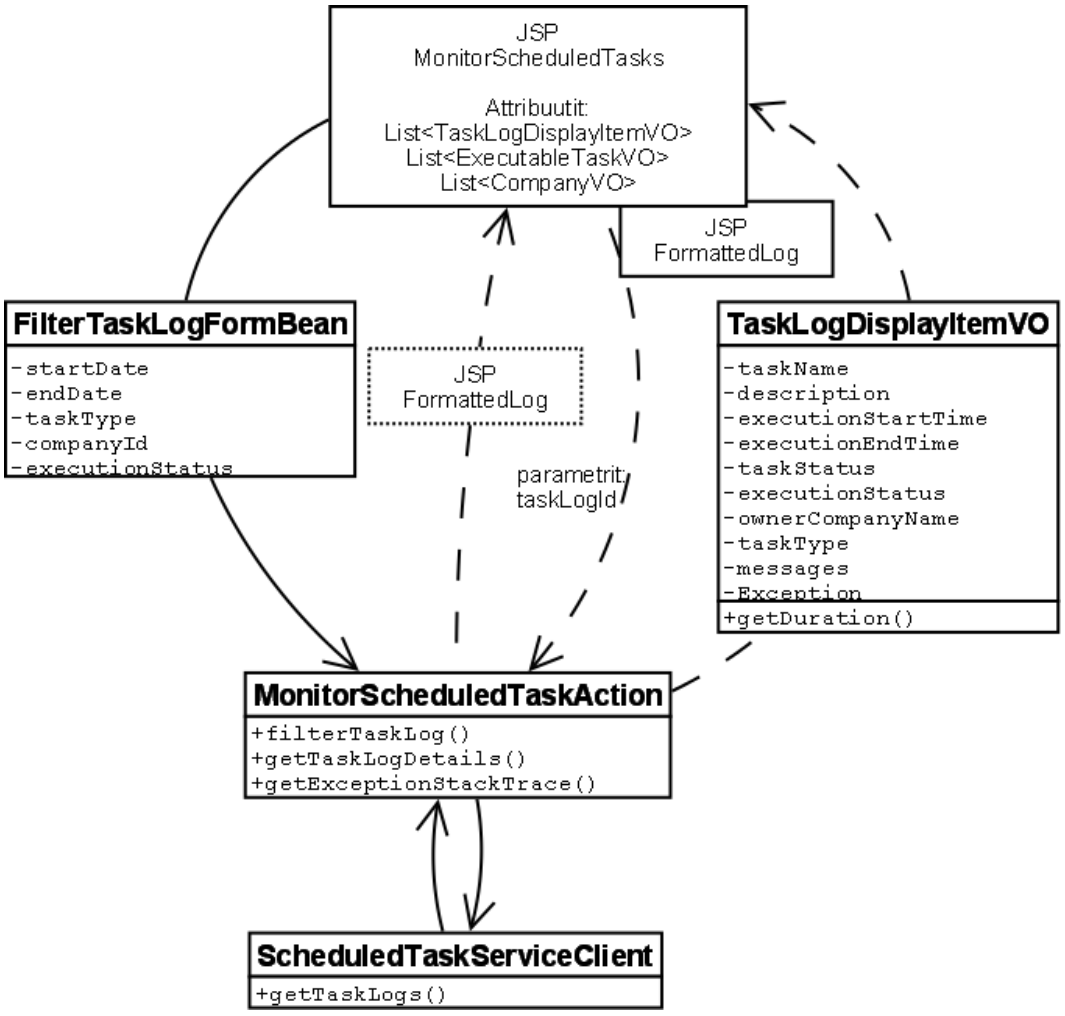
Ajastetun tehtävän poistaminen ja ajastuksen keskeyttäminen voidaan toteuttaa ilman erillistä lomakeluokkaa, koska toimintojen suorittamiseen riittää tieto valitusta tehtävästä. Tieto valitusta tehtävästä saadaan suoraan näkymässä olevasta listasta ja se voidaan välittää tehtävän id:n muodossa url-parametrina. Toiminnonkäsittelijä etsii tiedon perusteella istunto-olioon tallennetusta listasta valitun ScheduledTaskVO-olion, muuttaa tarvittavat tiedot ja lähettää olion sovelluslogiikalle tietojen päivitystä varten. Poisto-operaatio ei aiheuta näkymän päivitystä, vaan valitun tehtävän tiedot päivitetään dynaamisesti Ajax-tekniikkaa käyttäen. Keskeytystoiminnon suorittaminen päivittää koko näkymän.

Tehtävän uudelleen ajastaminen tapahtuu valitsemalla listasta halutun tehtävän, jonka jälkeen näkymään ilmestyy dialogi, jossa käyttäjä syöttää päivämäärän ja kellonajan. Dialogin toiminta toteutetaan javascript:lla ja css-määrittelyillä sekä dwr:llä. Näin olleen näkymää ei tarvitse päivittää toiminnonkäsittelijän kautta. Seuraavan suorituksen ajankohta asetetaan valmiiksi syöttökenttään. Seuraavan ajastuksen ajankohta haetaan palvelimelta dwr:ää käyttäen ja sen määritellään seuraavaksi täyttyväksi minuutiksi palvelimen kellon mukaan. Ajastustoiminnon parametrit, eli tehtävän id ja seuraavan suorituksen ajankohta välitetään toiminnonkäsittelijälle url-parametrina, joten erillistä lomakeluokkaa ei tarvita. Parametrien validointi tapahtuu toiminnonkäsittelijässä ja toiminnon suorituksen jälkeen ylläpitonäkymä päivitetään kokonaan.

5.2.3.3 Ajastettujen tehtävien loki

Ajastettujen tehtävien lokitiedot esitetään omassa näkymässä MonitorScheduledTask JSP-sivulla. Näkymässä tarvittavat tiedot ja toimintojen

vaatimat luokat sekä siirtymät on esitetty kuvassa 14. Kuvassa esitetty TaskLogDisplayItemVO-luokka toimii lokitietojen välittäjänä tiedonkäsittelykerrokselta esityskerrokselle. Luokkaan kerätään tietokantatasolta kaikki tarpeellinen tieto. Tämä VO-luokka ei siis toimi Hibernaten kautta kuten muut ajastusmoduulin VO-luokat.



Kuva 14 Ajastettujen tehtävien loki

Näkymän toiminnallisuuksien toteuttamisessa käytetään MonitorScheduledTaskAction-luokkaa, johon toteutetaan kolme metodia. Tärkein toiminnallisuus on lokitietojen suodatustoiminto. Tätä toimintoa varten toteutetaan FilterTaskLogFormBean, jonka kautta käyttäjän asettamat suodatusparametrit välitetään toiminnonkäsittelijälle. Lomaketietojen validointia ei toistaiseksi toteuteta, koska kaikki valinnat tehdään pudotusvalikoista kuvan 11 mukaisesti. Tällöin virhesyötteiden todennäköisyys on pieni. Lokitiedot haetaan annettujen parametrien perusteella sovelluslogiikalta käyttäen ScheduledTaskServiceClient-luokkaa. Parametrien perusteella haettu tieto välitetään sovelluslogiikalta listana TaskLogDisplayItemVO-oliota. Listasta generoidaan html-muotoinen esitys käyttämällä erillistä FormattedLog JSP-sivua. Sivun luoma lista lähetetään käyttäjän loki-näkymään Ajax-tekniikalla. Suodatustoiminnon toteutuksessa näkymän sisältöä siis muutetaan päivittämättä koko näkymää.

Kaikkea lokitietoa ei ole järkevää välittää html-sivulle heti lokitietojen haun yhteydessä, koska tietoa on paljon. Näin ollen yksittäisen lokitapahtuman tiedot välitetään Ajax-tekniikalla käyttäjän valitessa lokinäkymästä yksittäisen tapahtuman. Kyseisen tapahtuman tiedot siirtyvät TaskLogDisplayItemVO-oliona käyttäjän selaimelle ja tiedot esitetään erillisenä kehyksenä lokinäkymässä. Myös lokitapahtumassa oleva tieto ajon aikana tapahtuneesta poikkeuksesta välitetään käyttäjälle asynkronisesti.

5.3 Tehtävätyypit

5.3.1 Tehtävätyypin sovelluslogiikka

Automatisaatiomoduulin tehtävätyyppien toteutuksessa lähdetään siitä, että yksittäinen tehtävätyyppi voi olla mikä tahansa yksittäinen metodi sovelluspalvelimella. Tehtävätyypin koodi tulee sijaita J2EE kerrosmallin mukaan sovelluslogiikkakerroksella ja mahdolliset tietokantakäsittelyitä vaativat osatehtävät toteutetaan erillisinä metodeina muiden moduuleiden DAO-luokkiin. Mahdollisuuksien mukaan voidaan käyttää myös jo olemassa olevia metodeita.

Ajastusmoduulin toteutuksen näkökulmasta tehtävätyypin sovelluskoodin sijainti tulee pystyä identifioimaan luokkanimen ja metodin otsikon yhdistelmällä. Käytännössä on siis mahdollista toteuttaa useita tehtävätyyppejä samaan luokkaan. Myös kuormitettujen metodien määrittely eri tehtävätyypeiksi on mahdollista, sillä tehtävätyyppi tunnistetaan metodin nimen ja sen vastaanottamien parametrien yhdistelmällä.

Metodien kutsuminen Java Reflection API:n kautta vaatii, että metodin kutsuja tietää ennalta metodin vastaanottamien parametrien tyypit [13]. Suunnitteluvaiheessa tehdään yksinkertaisuuden vuoksi päätös, jonka mukaan tehtävätyypin toteuttava metodi vastaanottaa ainoastaan yhden parametrin. Parametrin tyypin voi metodin toteuttava sovelluskehittäjä itse määritellä, mutta sen tulee kuitenkin olla sarjallistettavissa olevaa tyyppiä. Vain sarjallistetut oliot voidaan kirjoittaa ja lukea suoraan tietokannasta jolloin parametrien välittäminen ja metodin kutsuminen on mahdollista. Tässä yhteydessä tulee kuitenkin huomata, että vaikka parametrien määrä on näennäisesti rajoitettu yhteen, voi sovelluskehittäjä määritellä parametrin tyypiksi esimerkiksi kokoelman. Map-rajapinnan toteuttavat kokoelmat toimivat hyvin useamman parametrin välittämisessä. Tällöin on kuitenkin muistettava, että kokoelman sisältö on oltava sarjallistettavissa.

5.3.2 Lokitiedot

Tehtävätyypin ajonaikainen lokitus toteutetaan siten, että tehtävätyypin toteuttava luokka perii LoggableTask-luokan, joka tarjoaa lokitustyökalut ja huolehtii lokitietojen säilyttämisestä ajon aikana. Käytännössä LoggableTask-luokka luo aliluokan käyttöön TaskLogVO-olion, johon voidaan lisätä lokiviestejä LoggableTask-luokan metodien kautta. LoggableTask-luokka huolehtii myös lokiviestien lisäämisestä järjestelmän lokiin Log4J-lokitustyökalulla. Jos tehtävätyypin toteuttava luokka ei peri LoggableTask-luokkaa, ajonaikainen lokitus ei ole mahdollista muuten kuin Log4J-työkalulla järjestelmälokiin. Tehtävän suorituksen loputtua ajon aikainen loki-olio päivitetään statuksen osalta ja kirjoitetaan tietokantaan.

Tehtävän ajon aikana tapahtuvasta, suorituksen keskeyttävästä virhetilanteesta tulee aina heittää poikkeus. Tällä tavoin ajastusmoduuli pystyy tunnistamaan virheen, muuttamaan ajastetun tehtävän statuksen ja lokittaa virheviesti asianmukaisesti. Tehtävätyypin toteuttajan vastuulle jää poikkeuksen asianmukainen luominen ja viestikentän muotoilu siten, että se kertoo tapahtuneesta virheestä riittävästi.

Tehtävätyypin lisääminen tulee olemaan prosessi, jossa sovelluskehittäjä toteuttaa tarvittavat komponentit Rank & Share -sovellukseen. Prosessi pyritään tekemään mahdollisimman selkeäksi ja yksinkertaiseksi, mutta virhetilanteisiin on silti varauduttava. Tämän takia puuttuvista asetuksista, väärin konfiguroiduista tehtävätyypeistä, puuttuvista tai vääristä luokka- ja metodinimistä sekä muusta tehtävätyypin kehityksen ja testauksen aikana ilmenevistä virhetilanteista kirjoitetaan ilmoitukset järjestelmälokiin debug-tasolla.

5.3.3 Tehtävätyypin parametrit

Tehtävätyypin metodille välitettävät parametrit syötetään käyttöliittymästä ajastetun tehtävän lisäämisen yhteydessä. Kullekin tehtävätyypille toteutetaan näkymä josta parametrit syötetään. Näkymä lisätään osaksi konfigurointinäkymää, kuten kappaleessa 5.2.3.1 mainittiin. Jotta tehtävätyyppejä voidaan käsitellä ajastusmoduulissa geneerisesti, tulee näkymien ja parametrien syöttölomakkeiden sekä parametrien tallennuksen toimia samalla tavalla tehtävätyypistä riippumatta. Näin ollen kullekin tehtävätyypille toteutettava näkymä ja siihen liittyvä toiminnonkäsittelijä tulee toimia tiettyjen ennalta määriteltyjen ja muun sovelluksen ymmärtämien sääntöjen mukaan.

Tehtävätyyppi tunnistetaan sen yksilöivällä tunnuksella. Tunnus määrittelee tehtävätyypin näkymän ja toiminnonkäsittelijän nimen. Tunnuksen perusteella siis pystytään hakemaan valitun tehtävätyypin parametrien syöttösivu sekä toiminnonkäsittelijä. JSP-sivun nimen tulee olla muotoa <Tunnus>.jsp ja se asetetaan /pages/scheduledTasks/ –kansioon. JSP-sivuun toteutetaan parametrien syöttölomake ja toiminto, jolla kutsutaan toiminnonkäsittelijää. Parametrien välittäminen toteutetaan ConfigureTaskFormBean-luokalla. Luokka välittää parametrit avain-arvo -pareina tehtävätyypin toiminnonkäsittelijälle. Avain-arvo –parien välittämisen aikana ei oteta kantaa arvojen tyyppien suhteen, joten ratkaisu toimii tehtävätyypistä riippumatta. Arvojen validointi tehdään toiminnonkäsittelijässä ja virhetilanteissa validointi palauttaa kontrollin takaisin parametrien syöttösivulle virheviestin kera.

Tehtävätyypin toiminnonkäsittelijä nimetään muotoon <Tunnus>Action.java ja asetetaan ...ui.actionhandlers.scheduledTasks-pakettiin. Toiminnonkäsittelijä perii abstraktin yliluokan ConfigureTaskAction. Tämä luokka tarjoaa metodit konfigurointinäkymän alustukseen, parametrien tallentamiseen istunto-olioon sekä logiikan siirtymän tekemiseen. ConfigureTaskAction siis toimii rajapintana tehtävätyyppikohtaisen toiminnonkäsittelijän ja geneerisen konfigurointinäkymän välillä. Oikean näkymän ja toiminnonkäsittelijän löytäminen määritellään geneerisesti Struts-config.xml tiedostossa. Tehtävätyyppikohtainen toiminnonkäsittelijä tallettaa käyttäjän syöttämät parametrit istunto-olioon muodossa joka voidaan kirjoittaa suoraan tietokantaan ScheduledTaskVO-olion kenttänä ja jonka tehtävätyypin suoritusmetodi voi vastaanottaa.

Erikoistapauksena on tehtävätyyppi, joka ei tarvitse parametrien syöttönäkymää. Tällainen tapaus voi esimerkiksi olla tehtävätyyppi, joka lähettää asiakastilin ylläpitäjälle sähköpostitse raportin viimeisimmän ennusteen tarkkuudesta. Tässä tapauksessa tehtävätyyppi tarvitsee parametreiksi ainoastaan yrityksen tunnuksen ja mahdollisesti sähköpostiosoitteen. Nämä tiedot voidaan hakea suoraan järjestelmästä käyttäjätunnuksen perusteella. Nyt tehtävätyypin toiminnonkäsittelijä suorittaa tietojen haun automaattisesti, eikä parametrien syöttönäkymää tarvitse toteuttaa lainkaan. Automaattinen parametrien asettaminen toteutetaan omana metodina toiminnonkäsittelijässä.

5.3.4 Tehtävätyypin integrointi

Uuden tehtävätyypin toteuttamisen yhteydessä sovellukseen toteutetaan siis itse sovelluslogiikan lisäksi näkymä ja toiminnonkäsittelijä. Näiden näennäisesti erillisten komponenttien liittäminen toimivaksi kokonaisuudeksi toteutetaan konfigurointitiedostossa. Executable-tasks.xml tiedostoon asetetaan kullekin tehtävätyypille tarvittavat tiedot, jotta tehtävätyyppiä voi käyttää Rank & Share -sovelluksessa. Seuraavat tiedot ovat välttämättömiä tehtävätyypin parametrien syöttämisen ja tehtävätyypin suorittamisen toteuttamiseksi:

- Täydellisesti määritelty luokkanimi

  Kertoo, mistä luokasta tehtävätyypin toteuttava metodi löytyy. Luokkanimen tulee sisältää sekä pakettihierarkian mukainen alkuosa että itse luokan nimi.

- Metodin nimi

  Metodi, joka suoritetaan tehtävätyypin tullessa suoritukseen. Metodin nimi tulee olla täsmälleen samassa muodossa kuin luokassa.

- Parametrin tyyppi

  Metodi ottaa vastaan parametrin vain tässä muodossa. Parametrin tyyppi pitää olla täydellisesti määritelty luokkanimi. Parametrikenttää ei tarvita jollei suoritusmetodi ota vastaan parametreja.

- Tehtävätyypin yksilöivä tunnus

  Tehtävän yksilöivää tunnusta käytetään esityskerroksella näkymän ja toiminnonkäsittelijän tunnistamisessa. Tunnuksen tulee olla mahdollisimman kuvaava eikä se saa sisältää välilyöntejä. Tunnuksen muodon tulee noudattaa yleisesti hyväksyttyä Java-luokkien nimeämiskäytäntöä.

- Tieto parametrien asettamisen automatisoinnista

  Totuusarvo, joka kertoo, suoritetaanko parametrien automaattinen hakeminen ja asettaminen toiminnonkäsittelijässä. Tällöin käyttäjällä ei ole mahdollisuutta syöttää tehtävätyypin parametreja.

Vaatimusmäärittelyvaiheessa esiin tullut tarve käyttöoikeuksien määrittämisestä toteutetaan niin ikään konfigurointitiedostossa. Käyttöoikeudet määrittelevät, mitkä käyttäjäroolit saavat ajastaa tehtävätyypin mukaisia tehtäviä. Käyttöoikeudet tarkistetaan ajastettavan tehtävän lisäysvaiheessa AddScheduledTaskAction-luokassa. Lisäksi tehtävätyypille voi antaa

vapaamuotoisen kuvauksen. Kuvaus näytetään tehtävätyypin parametrien syöttönäkymässä.

- Tehtävän vapaamuotoinen kuvaus

  Tehtävätyypin suorittaman toimenpiteen kuvas, parametrien tyypit sekä muuta käyttäjälle mahdollisesti arvokasta tietoa.

- Käyttöoikeuksien määrittely

  Käyttöoikeudet määritellään samoilla roolinimillä kuin mitä järjestelmässä on muuten käytetty. Tehtävätyypille voi antaa useita käyttäjärooleja.

Tehtävätyyppien konfigurointi tapahtuu Executable-tasks.xml –tiedostossa, joka sijoitetaan erilleen muusta sovelluksesta /home/jboss/customerData/-kansioon. Tiedoston tarkka muoto määritellään lopullisesti vasta toteutusvaiheessa, mutta sen sisältö tulee olla edellä esitellyn kaltainen. Xml-tiedoston luku sovellukseen toteutetaan ScheduledTaskDAO-luokassa. Tiedoston sisällöstä muodostetaan lista ExecutableTaskVO-oliota, jokaiselle tehtävätyypille omansa. Xml-tiedoston lukemisessa käytetään JDOM-kirjaston SAXBuilder-luokkaa, joka osaa validoida tiedoston muodon ja rakenteen tarvittaessa. Validointi vaatii DTD-määrittelyn toteuttamisen xml-tiedostolle. DTD-tiedoston muoto määritellään niin ikään vasta toteutusvaiheessa. [14][15]

6 Toteutus

6.1 Sovelluslogiikka

6.1.1 Kerrosmalli

J2EE kerrosmallin mukainen jako jakelu-, sovelluslogiikka- ja tiedonkäsittelykerrokseen toteutettiin suunnitellusti. Mainituista kerroksista selkein on tiedonkäsittelykerros, joka selkeästi keskittyy ainoastaan tiedonhakuun tietokannasta ja xml-tiedostoista. Automatisaatio- ja ajastusmoduulin toteutuksessa DAO-kerroksen toiminta keskittyi kyseisiin toimintoihin. Jakelu- ja sovelluslogiikkakerroksen välinen raja on huomattavasti häilyvämpi, joten tarkkaa roolijakoa ei työn aikana näiden kahden välille tehty. Tämän päätöksen tekemiseen ajoi lisäksi se seikka, että ajastusten hallinta täytyy toteuttaa jakelukerroksella, koska EJB Timer Service on tiukasti sidottu papu-luokan istunto-olioon. Näin ollen sovelluslogiikkaa on toteutettu sekä ScheduledTaskService- että ScheduledTaskBO-luokassa. Kappaleessa 6.1 sovelluslogiikalla käsitetään DAO-, BO-, ja papu-luokkien yhdessä toteuttamien toimintojen logiikka.

6.1.2 Testaus

Sovelluslogiikan testausta suoritettiin jatkuvasti uusien ominaisuuksien toteuttamisen yhteydessä. Varsinaisia yksikkötestejä luokille tai metodeille ei tehty, vaan testaus tapahtui erillisellä java-sovelluksella, johon toteutettiin tuki uusille ominaisuuksille sitä mukaa, kuin niitä toteutettiin sovelluslogiikan puolelle. Testaussovellusta varten papu-luokalle piti toteuttaa remote-rajapinta, jonka kautta sovelluspalvelimella olevaan EJB-säiliöön pystyttiin ottamaan yhteys erilliseltä Java virtuaalikoneelta.

Testaussovellus ei missään vaiheessa ollut täysiverinen asiakassovellus automatisaatio- ja ajastusmoduulin toimintoihin, vaan sitä muokattiin jatkuvasti tiettyä testaustarvetta varten. Se kuitenkin sisälsi komentorivipohjaisen käyttöliittymän, jonka kautta ajastuksia pystyi hallinnoimaan. Lisäksi testaussovellusta käytettiin kehitysvaiheessa tapahtuneiden tietokantaoperaatioiden suorittamiseen ja tietokannan siivoamiseen sekä EJB Timer Service:n tilan monitorointiin ja Timer-olioiden luontiin ja lopettamiseen.

6.1.3 Toteutusjärjestys

Sovelluslogiikan toteuttaminen jaettiin kolmeen erilliseen osakokonaisuuteen. Ensimmäinen osakokonaisuus piti sisällään ajastustoimintojen hallintatoimenpiteiden toteuttamisen käyttöliittymän ja pavun ajastuspalvelun välillä. Lisäksi ensimmäisessä vaiheessa toteutettiin Timer Service:sta tulevien ajastusten käsittely niiden laukeamisen yhteydessä. ScheduledTaskService-luokan ejbTimeout-metodissa toteutettu logiikka vaati paljon erityishuomiota ja testausta, koska sen toteutuksessa piti ottaa huomioon lukuisia erikoistilanteita. Esimerkiksi ajastuksen laukeaminen palvelimen ollessa sammutettuna aiheutti ongelmia Timer Service:ssa olleen ohjelmointivirheen takia.

Tehtävän suorituksen yhteydessä tarvittava sovelluslogiikka toteutettiin toisessa vaiheessa. Lisäksi toteutettiin tehtävän suorituksen jälkeiset sovelluslokin käsittelyrutiinit sekä lokien tallennus tietokantaan. Tässä vaiheessa päähuomion sai ScheduledTaskBO-luokan executeScheduledTask-metodi, jossa toteutettiin tehtävätyypin sovelluskoodin haku, kutsuminen, onnistuneen suorituksen käsittely sekä virhetilanteiden käsittely suorituksen keskeytymisen jälkeen.

Kolmannessa vaiheessa keskityttiin lokien hakuun tietokannasta sekä filtteröintiominaisuuksien toteuttamiseen. Ajastettujen tehtävien lokien seurannan toteuttaminen painottui pitkälti esityskerrokselle, joten sovelluslogiikan osalta tämä oli selkeästi nopein ja vähiten testausta vaatinut vaihe.

6.1.4 Ajastusten hallinta

Tehtävän ajastaminen järjestelmään tapahtuu EJB Timer Service-palvelun kautta. Palvelu on käynnissä sovelluspalvelimen EJB-säiliössä ja pitää huolen siihen ajastettujen Timer-olioiden laukaisemisesta ajallaan. Uuden Timer-olion luonti ja tallennus palveluun tapahtuu ScheduledTaskService-luokan SessionContex-olion kautta.

Uutta tehtävää ajastettaessa ScheduledTaskService-luokan addScheduledTask-metodi vastaanottaa ScheduledTaskVO-luokan olion, joka sisältää kaiken tarpeellisen tiedon tehtävän ajastamista varten. Uuden Timer-olion luominen EJB Timer Service-palveluun suoritetaan antamalla oliolle kolme parametriä: ensimmäisen laukeamisen ajankohta, laukeamisten aikaväli ja info-kenttä. Kaksi ensimmäistä tietoa saadaan ScheduledTaskVO-oliolta ja kolmas eli info-kenttä generoidaan käyttäen järjestelmän kelloa. Käytännössä info-kenttä pitää siis sisällään Timer-olion luomisajankohdan millisekunteina. Generoitu info-kentän arvo asetetaan ScheduledTaskVO-olion timerInfo-kenttään, jollon saadaan aikaan yhteys Timer-olion ja tehtäväolion välille. Lopuksi ajastetun tehtävän tiedot tallennetaan tietokantaan hibernate:n kautta.

Uuden tehtävän ajastuksen yhteydessä luodaan siis kaksi erillistä oliota. EJB säiliön ajastuspalveluun tallennetaan Timer-olio joka laukeaa sille määriteltynä ajankohtana. Tätä ajastusoliota käyttävä ScheduledTaskVO-olio taas tallennetaan tietokantaan ja haetaan Timer-olion ajastuksen lauetessa.

Tehtävän ajastuksen keskeytystoiminto käsittelee saamaansa ScheduledTaskVO-oliota muuttamalla sen statustiedot sekä poistamalla ajastustiedot. EJB Timer Service –palvelu ei tarjoa Timer-oliolle toimintoa, jolla olion ajastuksen saisi keskeytettyä. Näin ollen ajastuksen keskeyttämiseksi ScheduledTaskVO-oliota vastaava Timer-olio pitää poistaa ajastuspalvelusta kokonaan. Näin myös ScheduledTaskVO-olion timerInfo-kenttä nollataan ennen olion tallennusta tietokantaan.

Vastakkaisena toimintona tehtävän ajastuksen keskeyttämiselle on tehtävän ajastuksen uudelleenkäynnistys. Tämä toiminto on toteutettu ScheduledTaskService-luokan resumeTask-metodissa. Metodi ottaa vastaan ScheduledTaskVO-olion, johon on asetettu seuraavan ajastuksen laukeamisen ajankohta. Nyt EJB-säiliön ajastuspalveluun luodaan uusi Timer-olio uusilla parametreillä, aivan kuten uuden tehtävän ajastuksessakin.

Ajastetun tehtävän poisto tapahtuu ScheduledTaskService-luokan removeTask-metodissa. Metodissa Timer-olio poistetaan EJB-säiliön ajastuspalvelusta ja vastaanotetusta ScheduledTaskVO-oliosta poistetaan Timer-olioon liittyvä timerInfo-kentän arvo sekä nextTimeout-arvo. Lisäksi ScheduledTaskVO-olion statuskenttä asetetaan arvoon REMOVED. Tehtäväoliota ei siis fyysisesti poisteta tietokannasta, mutta sitä ei tulla koskaan käyttämään uudestaan. Tällainen ratkaisu johtuu siitä, että kaikkien ajastusten hallintaan liittyvien toimenpiteiden yhteydessä kirjoitetaan ylläpitoloki tietokantaan. Tietokantaan kirjoitettava loki-olio sisältää viittauksen käsiteltyyn tehtäväolioon. Näin ollen tehtäväolion poisto rikkoisi tietokannan eheyttä eikä ylläpitolokia poistettujen tehtävien osalta pystyttäisi näyttämään.

6.1.5 Tehtävätyypin suorittaminen ja lokitus

Ajastetun tehtävän suorittaminen alkaa kun EJB:n ajastuspalvelu laukaisee Timer-olion ajastuksen. Ajastuksen laukeamisen tuloksena ajastuspalvelu kutsuu ScheduledTaskService-luokan ejbTimeout-metodia. Metodi vastaanottaa parametrina lauenneen Timer-olion. Metodissa tietokannasta haetaan Timer-olioon liittyvä ScheduledTaskVO-olio. Olio pystytään tunnistamaan Timer-olion info-kentän ja ScheduledTaskVO-olion taskInfo-kentän yhteneväisyyden avulla.

Kun oikea ScheduledTaskVO-olio on haettu, siirretään se suoritukseen kutsumalla ScheduledTaskBO-luokan executeScheduledTask-metodia. Metodi vastaanottaa parametrina suoritettavan ScheduledTaskVO-olion. Metodissa haetaan tehtävälle asetetun tehtävätyypin luokka ja ajettava metodi. Metodia kutsutaan tehtävään tallennetuilla parametreilla. Jos parametrejä ei ole määritelty ScheduledTaskVO-olioon, suoritusmetodia kutsutaan ilman parametreja. Jos tehtävätyypin luokka tai metodi ei ole kelvollinen, tehtävän suoritus keskeytyy ja ScheduledTaskVO-olion statukseksi asetetaan INVALID. Jos tehtävän suorituksen aikana tapahtuu virhe, executeScheduledTask-metodi vastaanottaa sen ja lisää sen lokitietoihin. Onnistuneen ajon jälkeen tehtäväolion statukseksi asetetaan joko ACTIVE tai COMPLETED sen mukaan, onko tehtävä ajastettu suoritettavaksi vain kerran vai toistuvasti. Olion saama status palautetaan ejbTimeout-metodille, joka tarkastaa statuksen ja suorittaa tehtävän Timer-olion tuhoamisen tai antaa Timer-olion jatkaa toimintaansa. Jos tehtävän status on ACTIVE, Timer-olio voi jatkaa toimintaansa ajastuspalvelussa ja laueta sille määritellysti myöhemmin. COMPLETED status kertoo, että tehtävä on onnistuneesti suoritettu, eikä sitä tulla suorittamaan uudelleen. Tällöin Timer-olio tuhotaan ajastuspalvelusta. INVALID status niin ikään kertoo, että ajastusolio tulee tuhota.

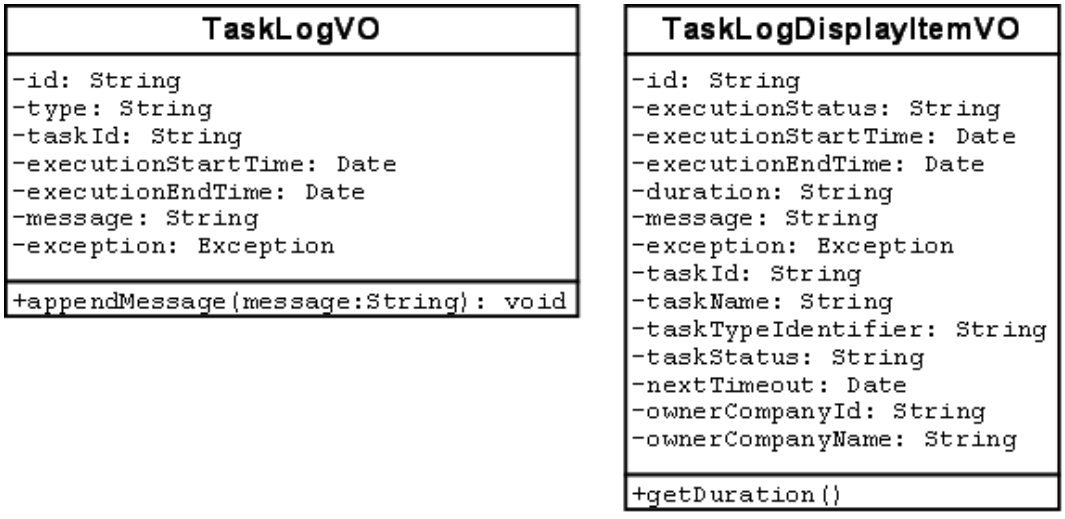
Tehtävän suorituksen aikana tapahtuvaan lokitukseen päästään käsiksi executeScheduledTask-metodissa muuntamalla tehtävätyypin luokka LoggableTask-luokan olioksi ja kysymällä tältä TaskLogVO:ta, johon suorituksen aikainen lokitieto on tallennettu. Vastaanotettu TaskLogVO täydennetään tiedolla suorituksen kestosta, mahdollisella virheilmoituksella ja statustiedoilla, jonka jälkeen se voidaa tallettaa tietokantaan. Mikäli tehtävätyyppin toteuttava luokka ei jostain syystä peri LoggableTask-luokkaa, ei suorituksen aikaista lokia kirjoiteta, vaan lokitietoihin tulee ainoastaan tiedot suorituksen kestosta ja statuksesta.

6.1.6 Ajastettujen tehtävien loki

Automatisaatio- ja ajastusmoduulissa lokitietoa kuvaa TaskLogVO-luokka. Luokan oliot edustavat yksittäistä lokitietoa. Lokitieto voi olla tieto suoritetusta tehtävästä, sen aikana tapahtuneista toiminnoista sekä suorituksen kestosta ja tehtävän statuksesta tehtävän suorituksen jälkeen. Lokitieto voi vaihtoehtoisesti pitää sisällään ajastettujen tehtävien ylläpitotietoja. Lokitieto voi siis liittyä tehtävän suorittamiseen tai tehtävien hallintaan. TaskLogVO-luokassa on type-kenttä, johon tallennetaan tieto, minkä tyyppisestä lokitiedosta on kysymys. Tätä tietoa tarvitaan haettaessa lokitietoja erilaisilla kriteereillä lokitietojen suodatustoiminnossa. TaskLogVO-luokan rakenne on esitetty kuvassa 15.

Lokitietojen tallennus on suoraviivaista. Tallennus suoritetaan DAO-kerroksessa hibernate:n kautta. Lokitiedot tallennetaan TaskLogVO-oliona tasklogvos-tietokantatauluun. Lokitietojen suodattamista ja esittämistä varten on tietokantaan toteutettu tasklogdisplayitemvos-taulunäkymä, josta kaikki lokitiedot haetaan. TaskLogVO-luokan olioita ei siis koskaan haeta tietokannasta, niitä ainoastaan tallennetaan. Lokitietojen hakuun käytetään TaskLogDisplayItemVO-luokkaa. Tasklogdisplayitemvos-tietokantanäkymästä kerrotaan tarkemmin kappaleessa 6.1.7. TaskLogDisplayItemVO-luokan rakenne on esitetty kuvassa 15.

Lokitietojen hakutoimintoihin on toteutettu kaksi metodia ScheduledTaskDAO-luokkaan. getTaskLogs-metodilla haetaan lokitiedot tietylle yritykselle tietyltä aikaväliltä. Metodi vastaanottaa halutun alkupäivämäärän, loppupäivämäärän ja yrityksen id:n ja palauttaa listan TaskLogDisplayItemVO-olioita. Metodin toteutuksessa tietokantahaku on tehty SQL-kyselynä tasklogdisplayitemvos-tietokantanäkymään. Metodin toteutusta voi käyttää myös ilman yrityksen id:tä, jolloin metodi palauttaa kaikkien yritysten lokitiedot aikaväliltä. Lokituskäyttöliittymän suodatustoiminnot toimivat siis aikavälin ja yrityksen suhteen sovelluslogiikkatasolla. Tehtävätyypin ja statuksen perusteella suoritettava suodatus on toteutettu lokinäkymän toiminnonkäsittelijässä.



Kuva 15 Lokiluokat TaskLogVO ja TaskLogDisplayItemVO

Yksittäisen tehtävän lokitietojen hakuun on toteutettu metodi getLogsForTask, joka vastaanottaa ScheduledTaskVO-olion ja palauttaa listan lokitiedoista, jotka kuuluvat annetulle tehtäväoliolle. Tätä ominaisuutta ei ole toistaiseksi käytetty käyttöliittymissä, vaan se on toteutettu testausta varten.

6.1.7 Tietokanta

Tietokannan toteutus Rank & Share -sovelluksessa on yleisesti hoidettu Hibernate-kuvauksilla. Automatisaatio- ja ajastusmoduulissa on kaksi VO-luokkaa jotka kuvataan olioina tietokantaan hibernate-kuvauksen kautta. ScheduledTaskVO- ja TaskLogVO-luokkien tietokantataulun rakenne on esitetty liitteessä 4. Näiden taulujen tietokantaoperaatiot, eli olio-tietokanta -muunnokset on toteutettu ScheduledTaskDAO-luokassa Hibernate-istunnossa olio-rajapinnan kautta.

Ajastettujen tehtävien monitorointinäkymää varten tehty TaskLogDisplayItemVO-luokka on ScheduledTaskVO- ja TaskLogVO-luokista poiketen toteutettu tietokantaan näkymänä. Käytännössä tämä tarkoittaa sitä, että TaskLogDisplayItemVO-luokasta ei ole tehty hibernate-kuvausta, eikä luokan olioita myöskään koskaan kirjoiteta kantaan. Luokan tehtävänä on toimia ScheduledTaskVO- ja TaskLogVO- ja CompanyVO-luokkien tietojen välittäjänä käyttöliittymään. Luokan olioiden luonti tapahtuu ScheduledTaskDAO-luokassa käyttämällä SQL-kieltä tietojen hakuun ja muodostamalla tiedoista TaskLogDisplayItemVO-olio. Tietokantanäkymä tasklogdisplayitemvos luodaan SQL-kielellä. Näkymän luontiin käytetty SQL-käsky on esitetty kuvassa 16.

```
create view tasklogdisplayitemvos as
  select
    tasklogvos.id,
    tasklogvos.type as executionStatus,
    tasklogvos.executionstarttime,
    tasklogvos.executionendtime,
    tasklogvos.message,
    tasklogvos.exception,
    scheduledtaskvos.id as taskId,
    scheduledtaskvos.name as taskName,
    scheduledtaskvos.status as taskStatus,
    scheduledtaskvos.nexttimeout,
    scheduledtaskvos.ownercompanyid,
    companyvos.name as companyname
  from tasklogvos
  inner join scheduledtaskvos
    on tasklogvos.taskid=scheduledtaskvos.id
  join companyvos
    on scheduledtaskvos.ownercompanyid = companyvos.id;
```

Kuva 16 TaskLogDisplayItemVOs tietokantanäkymän luonti

6.2 Käyttöliittymät

6.2.1 Yleistä

Automatisaatio- ja ajastusmoduulin neljän käyttöliittymän toteutuksesta tässä yhteydessä kerrotaan vain kahdesta, eli ajoitusten hallinta- ja ajoitettujen tehtävien loki–näkymistä. Ajastettavien tehtävien lisäysnäkymän ja tehtävätyypin parametrien syöttönäkymän toteutukset ovat vahvasti sidoksissa toisiinsa, joten ne esitellään yksityiskohtaisesti kappaleessa 6.3.2.

Kaikkien tässä työssä toteutettujen käyttöliittymänäkymien rakenteellisessa toteutuksessa on käytetty JSTL-kirjaston ja Struts-kehyksen elementtejä. Ulkoasun määrittelyssä on käytetty mahdollisuuksien mukaan valmiita css-

määrittelyitä. Tässä luvussa ei esitellä jsp-sivujen rakennetta tai ulkoasuun liittyviä asioita, vaan keskitytään toiminnallisuuksien toteutukseen.

Käyttöliittymien toteutuksessa uutena tekniikkana otettiin käyttöön DWR-kirjaston tarjoamat Ajax-ominaisuudet. DWR:n käyttöönotto vaati muutamien tiedostojen ja asetusten lisäämistä järjestelmään. DWR-kirjaston luokat lisättiin dwr.jar-pakettina sovelluksen luokkapolkuun, dwr.xml lisättiin WEB-INF-hakemiston alle ja dwr20.dtd .../customerData/dtds-hakemiston alle. Lisäksi web.xml-tiedostoon määriteltiin DWR:lle oma Servlet-komponentti sekä polku toiminnonkäsittelijään.

6.2.2 Ajoitusten hallinta

Ajoitusten hallintanäkymä sisältää käytännössä yhden taulukon joka listaa järjestelmässä olevat ajastetut tehtävät. Tehtävät haetaan listana sovelluslogiikalta käyttäjän edustaman yrityksen perusteella. Palvelun ylläpitäjä näkee kaikkien yritysten ajastetut tehtävät. Yleisen käytännön mukaisesti näkymään tultaessa sivulla näytettävät tiedot haetaan ja asetetaan istunto-olioon toiminnonkäsittelijän prepare-metodissa.

Näkymässä esitettävän listan tiedot kunkin tehtävän osalta saadaan suoraan ScheduledTaskVO-olion kentistä. Kenttien tietojen lokalisointi käyttäjän selaimen kielen mukaan on toteutettu JSTL:n fmt-elementeillä kuten Rank & Share -sovelluksessa yleisesti on tehty. Toteutusvaiheessa havaittiin, että valmista käyttökelpoista toimintoa tehtävän keston esittämiseen ei ole. Jotta kesto-tieto voidaan esittää järkevästi, sitä varten toteutettiin LogicoELFunctions-luokkaan formatDuration-funktio, joka muotoilee vastaanotetun long-muotoisen kesto-tiedon helposti ymmärrettävään muotoon. Funktion käyttö JSP-sivulla tapahtuu EL-kielellä. Samaa funktiota käytetään myös ajastettujen tehtävien lokinäkymässä.

Ajastusten hallintanäkymässä on kolme toiminto. Tehtävän ajastuksen keskeytys, ajastuksen uudelleen käynnistäminen ja tehtävän poisto. Ajastuksen keskeytys on toteutettu puhtaasti dwr-kirjastoa käyttäen. Toiminnonkäsittelijäluokkaan on toteutettu holdTask-metodi ja JSP-sivulle holdTask-funktio, jossa toiminnonkäsittelijän metodia on kutsuttu. Metodin kutsuminen javascript-funktiossa vaatii, että se asetetaan aktiiviseksi dwr.xml tiedostoon. Kuvassa 17 on esitetty java-luokan metodien esittely ja aktivointi dwr.xml tiedostossa, toiminnonkäsittelijässä toteutettu metodi sekä javascript-funktio JSP-sivulla, jonka kautta toiminnonkäsittelijän metodia kutsutaan. HoldTask-metodi siis vastaanottaa keskeytettävän tehtävän id:n ja palauttaa vastaavan ScheduledTaskVO-olion, joka sisältää päivitetyt tiedot keskeytetystä tehtävästä. Päivitetyt tiedot asetetaan hallintanäkymän taulukkoon tehtävän kohdalle.

```
<create
  creator="new"
  javascript="ManageScheduledTasks"
  scope="application">
  <param
    name="class"
    value="fi.logico.frank2.ui.actionhandlers.
           scheduledTasks.ManageScheduledTasksAction" />
  <include method="holdTask" />
  <include method="removeTask" />
  <include method="getServerDate" />
</create>
```

```
public ScheduledTaskVO holdTask(String taskId)
throws AjaxException {
  WebContext ctx = WebContextFactory.get();
  HttpServletRequest request = ctx.getHttpServletRequest();
  try {
    ScheduledTaskVO task = getTask(taskId, request);
    ScheduledTaskServiceClient taskService =
      new ScheduledTaskServiceClient();
    taskService.holdScheduledTask(task);
    return task;
  } catch(Exception e) {
    throw GeneralUtils.createAjaxException(request, e);
  }
}
```

```
holdTask = function(taskId) {
  ManageScheduledTasks.holdTask(taskId, {
    callback:function(task) {
      if(task != null) {
        dwr.util.setValue("status_" + taskId, task.status);
        dwr.util.setValue("nextTimeout_" +
          taskId, task.nextTimeout);
        dwr.util.setValue("timeLeft_" + taskId, "");
      }
    }
  });
}
```

Kuva 17 Ajax:n käyttö dwr:n avulla. dwr.xml, toiminnonkäsittelijän metodi ja javascript-funktio JSP-sivulla.

Tehtävän poisto on metodina toteutettu DWR:ia käyttäen ja sitä kutsutaan javascript-funktiona JSP-sivulla. Poisto-operaation jälkeen näkymä kuitenkin päivitetään kokonaisuudessaan päinvastoin kuin AJAX:n ja DWR:n periaatteisiin kuuluu. Näin ollen poisto-operaatio ei käytä AJAX:a tehokkaasti hyväkseen.

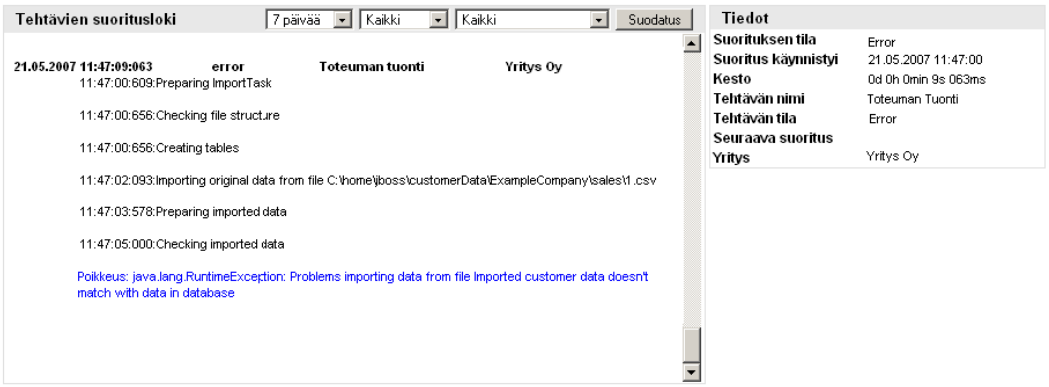
Tehtävän uudelleenajastaminen on toteutettu perinteisenä toiminnonkäsittelijäkutsuna. Siinä käytetään kuitenkin DWR-kutsua palvelimen kellonajan hakuun. Kellonaika ja päivämäärä haetaan uudelleenajastusikkunaan oletusarvoksi seuraavan ajastuksen laukeamisen ajankohdaksi. Näin ollen, jos käyttäjä haluaa suorittaa tehtävän heti, riittää, että hän hyväksyy oletusarvon. Tälloin seuraava ajastus laukeaa seuraavan täyden minuutin täytyttyä. Syötetyn kellonajan ja päivämäärän muodon validointi sekä tarkistus, että asetettu ajankohta on tulevaisuudessa, tehdään toiminnonkäsittelijässä.

Uudelleenajastustoiminnon käsittelijä on toteutettu ManageScheduledTaskAction:n resumeTask-metodissa. Seuraavan suorituksen ajankohta ja ajastettavan tehtävän id välitetään metodille url-parametreina.

6.2.3 Ajoitettujen tehtävien loki

Ajoitettujen tehtävien lokinäkymässä on kaksi erillistä lohkoa. Näkymän lopullinen muoto on esitetty kuvassa 18. Kuvassa vasemmalla puolella on varsinainen lokitietojen esitysosa, jossa esitetään lokitiedot kronologisessa järjestyksessä uusin alimpana. Tämän lohkon tiedot vastaavat suunnitteluvaiheessa esitettyä rakennetta. Oikeanpuoleinen lohko sen sijaan on ratkaisu vaatimusmäärittelyvaiheessa esiin nousseisiin tarpeisiin. Siinä esitetään vasemmapuoleisesta lohkosta valitun lokitietorivin tarkemmat tiedot. Tiedot saadaan esiin valitsemalla hiirellä tummennettu lokitietorivin otsikko. Tiedot-lohkossa esitetään siis yksittäisen lokirivin tiedot kuten kuvassa on esitetty.

Ajoitettujen tehtävien lokinäkymä on toteutettu puhtaasti AJAX-tekniikalla. Se sisältää suodatustoiminnot, lokitietojen yksityiskohtaisten tietojen haun ”Tiedot”-lohkoon sekä virhetilanteissa poikkeuksen yksityiskohtaisten tietojen haun.



Kuva 18 Ajoitettujen tehtävien loki

Suodatustoimintojen toteutus on tehty toiminnonkäsittelijässä normaalina Struts-toiminnonkäsittelijämetodina. Metodi vastaanottaa FilterTaskLogFormBean-lomaketieto-olion ja siirtää käyttäjälle JSP:nä toteutetun html-sivun. Toimintoa ei kuitenkaan käytetä normaalisti lähettämällä html-sivulta lomaketiedot suoraan toiminnonkäsittelijää kutsumalla vaan AJAX:n kautta. Palvelupyyntö rakennetaan javascript:llä ja lähetetään XMLHttpRequest-oliona. Oliolle asetetaan vastauksen käsittelijä joka vastaanottaa toiminnonkäsittelijän palauttaman html-sivun. Vastauksen käsittelijä asettaa vastaanotetun html-sivun ”Tehtävien suoritusloki”-kenttään lokinäkymään. Generoitu html-sivu rakennetaan FormattedLog.jsp-sivulla.

Virheeseen päättyneen tehtävän suorituksen poikkeusloki haetaan käyttäjän valitessa poikkeuksen kuvaus lokinäkymästä. Loki haetaan DWR-funktiona ja asetetaan poikkeuskuvauksen alle lokinäkymään. Kuvaus välitetään merkkijonona.

Yksittäisen lokitiedon lisätietojen haku oikeanpuoleiseen lohkoon on näkymässä toteutettu niin ikään DWR-funktiokutsuna. Nyt toiminnonkäsittelijä palauttaa kokonaisen TaskLogDisplayItemVO-olion. DWR muuttaa tämän java-olion javascript-olioksi, jolloin sitä voi käyttää html-sivulla. TaskLogDisplayItemVO-

olion tiedot asetetaan ”Tiedot”-lohkoon DWR-funktiokutsun vastaanotettua olio toiminnonkäsittelijältä.

6.3 Tehtävätyypit

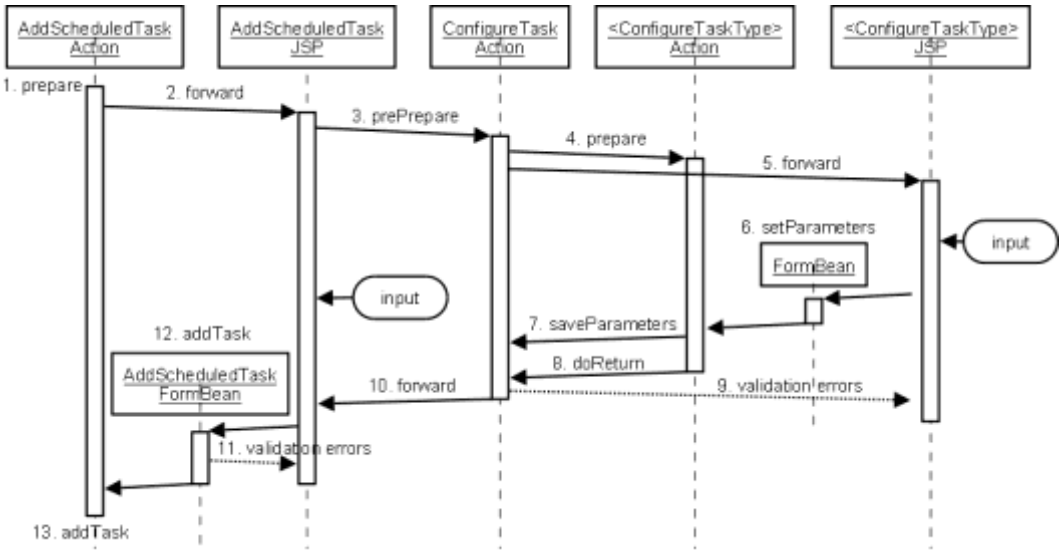
6.3.1 Sovelluslogiikka

Uuuden tehtävätyypin sovelluslogiikan suunnittelussa ja toteutuksessa on otettava huomioon käytännössä ainoastaan lokitus ja virheviestien luominen, kun ajatellaan tehtävätyypin integrointia Rank & Share -sovellukseen. Lokituksen käsittely toteutetaan LoggableTask-luokan perinnällä. LoggableTask-luokka luo kullekin tehtävätyyppi-luokan oliolle yhden TaskLogVO-olion. Tehtävän suorituksen aikana TaskLogVO-oliolle pystyy välittämään lokitietoja LoggableTask-luokan addLogMessage-metodin kautta. AddLogMessage-metodi vastaanottaa lokiviestin ja lisää sen TaskLogVO-olion message-kenttän jatkoksi. Kukin lisätty lokiviesti kirjoitetaan omana rivinään TaskLogVO-viestikenttään. Lisäksi kullekin lokiviestille annetaan aikaleima, joka asetetaan lokiviestirivin alkuun. Näin pystytään muodostamaan selkeä lokiviestien ketju, joka voidaan sellaisenaan esittää käyttöliittymässä. LoggableTask-luokka lokittaa lisäksi kaikki suorituslokit myös järjestelmän lokiin info-tasolla.

Tehtävän aikana tapahtuva suorituksen keskeyttävä poikkeus heitetään aina ulos suoritusmetodista. Tämä on ainoa tapa kertoa automatisaatiomoduulille, että suoritus on keskeytynyt virheellisesti.

6.3.2 Käyttöliittymä

Tehtävätyypin parametrien välittäminen tapahtuu käyttöliittymän kautta. Jokaiselle tehtävätyypille pitää toteuttaa oma parametrien syöttönäkymä sekä toiminnonkäsittelijä joka ottaa parametrit vastaan ja välittää eteenpäin. Kuvassa 19 on esitetty sekvenssikaaviona parametrien vastaanottaminen ja välittäminen ajastettavalle tehtävälle. Kuvaan on merkitty siirtymät toiminnonkäsittelijöiden ja JSP-sivujen välillä. ConfigureTaskAction on tehtävätyyppikohtaisen toiminnonkäsittelijän (kuvassa <ConfigureTaskType>Action) yliluokka, mutta selkeyden vuoksi ne on esitetty kuvassa omina luokkinaan. Input-toiminnot kuvaavat käyttäjän tekemiä syötteitä näkymässä.



Kuva 19 Tehtävätyypin konfigrointi käyttöliittymässä sekvenssikaaviona

Tehtävän lisäysnäkymä alustetaan AddShceduledTaskAction-luokan prepare-metodissa. Metodissa istunto-olioon haetaan lista tehtävätyypeistä sekä lista yrityksistä mikäli käyttäjä on kirjautunut palvelun ylläpitäjän tunnuksilla. Näiden tietojen avulla tehtävän lisäysnäkymä -JSP-sivu voidaan luoda.

Tehtävän lisäysnäkymässä käyttäjä valitsee haluamansa tehtävätyypin. Jos valitulle tehtävätyypille tulee asettaa parametreja, ilmestyy näkymään linkki tehtävätyypin konfigurointisivulle. Kuvan 20 siirtymät 3-5 edustavat siirtymää tehtävätyyppikohtaisten parametrien syöttösivulle. Kohdassa 3. kutsutaan tehtävätyypin toiminnonkäsittelijän prePrepare-metodia, joka alustaa tehtävätyypin konfigurointinäkymän hakemalla tehtävätyypin perustiedot istunto-oliolta esittämistä varten. PrePrepare-metodi kutsuu tämän jälkeen tehtävätyypille räätälöityä prepare-metodia (siirtymä 4). Metodi on määritelty abstraktiksi ConfigureTaskAction-luokassa, joten se täytyy toteuttaa tehtävätyypin toiminnonkäsittelijään. Prepare-metodin tarkoitus on antaa sovelluskehittäjälle mahdollisuus hakea konfigurointinäkymässä tarvittavat oliot järjestelmästä JSP-sivun käyttöön ja asettaa ne istunto-olioon.

```
<action
  path="/ConfigureTask*"
  parameter="method"
  type="fi.logico.frank2.ui.actionhandlers.scheduledTasks.{1}Action"
  name="lazyForm"
>
  <forward
    name="configurationPage"
    path="/pages/scheduledTasks/ConfigureTask.jsp?confPageName={1}"
  />
</action>
```

Kuva 20 Tehtävätyypin konfigurointinäkymän kuvaus struts-config.xml tiedostossa

Prepare-metodin suorituksen jälkeen prePrepare-metodi siirtää käyttäjän tehtävätyyppikohtaiselle konfigurointisivulle. Tehtävätyyppikohtaisten konfigurointisivujen ja toiminnonkäsittelijöiden kuvaukset struts-config.xml –tiedostossa on esitetty kuvassa 20. Kuvan xml-listauksesta nähdään, että tehtävätyypin parametrien syöttösivua kutsuttaessa kutsutaan itse asiassa

ConfigureTask JSP-sivua. Sivulle välitetään configPageName-parametri, joka kertoo, minkä tehtävätyypin konfigurointisivu tulee esittää. Tehtävätyypin konfigurointisivu sisällytetään Struts:n tiles-tag-kirjaston avulla:

```
<tiles:insert
page="/pages/scheduledTasks/${param['confPageName']}.jsp" />
```

Tehtävätyypin konfigurointisivun tulee siis olla hakemistossa /pages/scheduledTasks/ ja sivu pitää olla nimetty tehtävätyypin tunnisteen kanssa saman nimiseksi ja päätteenä pitää olla jsp.

Konfigurointikäyttöliittymässä syötetyt parametrit välitetään LazyValidatorForm-luokan avulla. Näin ollen erillistä FormBean-luokkaa ei tarvitse toteuttaa parametrien välitykseen. Se ei olisi käytännössä edes mahdollista, koska samaa FormBean-luokkaa pitää käyttää kaikissa tehtävätyyppikohtaisissa konfigurointinäkymissä. Tämä johtuu Struts-config.xml-tiedoston rajoitteista.

Lomaketiedot validoidaan tehtävätyypin toiminnonkäsittelijässä. Toiminnonkäsittelijä ottaa vastaan JSP-sivulla syötetyt tiedot, tarkistaa niiden oikeellisuuden ja luo niistä tarkoitukseen sopivan olion. Luotu olio on sama, mikä tullaan välittämään tehtävätyypin sovelluslogiikan toteuttavalle metodille. Parametriolio tallennetaan istunto-olioon. Tallentaminen tapahtuu ConfigureTaskAction-luokan saveParameters-metodin kautta. Metodi tallentaa parametriolion ennalta määritellyllä nimellä. Tällä nimellä parametriolio osataan löytää myöhemmin AddScheduledTaskAction-luokan addTask-metodissa, kun luodaan lopullinen ScheduledTaskVO-olio ajastusta ja tallennusta varten.

Tehtävätyypin toiminnonkäsittelijäluokkaan toteutettu metodi parametriolion luontia varten ei itse siirrä kontrollia JSP-sivulle, kuten toiminnonkäsittelijöissä yleensä tehdään. Metodin lopuksi tulee kutsua yliluokan doReturn-metodia, joka tyhjentää lomaketiedot ja siirtää käyttäjän tehtävän lisäysnäkymään sekä sulkee tehtävätyypin konfigurointi-ikkunan. Jos tehtävätyypin parametreissa on löydetty validoinnin aikana virheitä, käyttäjän täytyy asettaa virheviestit istunto-oliolle. Tällöin doReturn-metodi havaitsee virheet, asettaa ilmoituksen JSP-sivulle, eikä sulje tehtävätyypin konfigurointinäkymää (Kuva 20, siirtymä 9).

Onnistuneen tehtävätyypin konfiguroinnin jälkeen käyttäjä siirretään takaisin tehtävän lisäysnäkymään. Käyttäjä syöttää puuttuvat parametrin ja hyväksyy tehtävän lisäämisen. Tällöin kutsutaan AddScheduledTaskAction-luokan addTask-metodia. Tässä metodissa otetaan vastaan ajastettavan tehtävän parametrit sekä haetaan istunto-oliolta aikaisemmin siihen tallennettu tehtävätyypin parametriolio. Jos tehtävätyyppille ei ole toteutettu konfigurointinäkymää vaan sen parametrit haetaan automaattisesti järjestelmästä, metodissa ajetaan tehtävätyypin toiminnonkäsittelijässä sijaitseva autoConfigure-metodi. Tämän jälkeen suoritetaan vielä validointi, joka tarkistaa että valitun tehtävätyypin sovelluslogiikkametodi on oikeasti järjestelmässä ja se pystyy vastaanottamaan parametriolion, jota sille tullaan tarjoamaan.

Ajastettavan tehtävän parametrit ja tehtävätyyppikohtaiset parametrit kerätään lopuksi kokoon ja niistä muodostetaan ScheduledTaskVO-olio. Olio lähetetään lopuksi ScheduledTaskServiceClient-luokan kautta jakelukerrokselle ja käyttäjälle ilmoitetaan onnistuneesta tehtävän ajastuksesta.

6.3.3 Tehtävätyypin asetukset järjestelmässä

Tehtävätyypin integrointi ja konfigurointi Rank & Share -sovellukseen on suunniteltu kappaleessa 5.3.4. Xml-tiedoston muoto on toteutettu DTD rakennemäärittelynä ja se on esitetty kuvassa 21. Rakennemäärittelyssä on määritelty kullekin suunnitteluvaiheessa esitetylle tiedolle oma elementtinsä lukuunottamatta tehtävätyypin tunnistetta ja automaattisen konfiguroinnin määrittelevää totuusarvoa. Nämä on asetettu elementtien attribuuteiksi.

```
<!ELEMENT executable-tasks (task-type*)>

<!ELEMENT task-type (name,description,parameters,method,class,permissions)>
        <!ATTLIST task-type identifier CDATA #REQUIRED>

<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA)>

<!ELEMENT parameters (description?,parameter-type?)>
        <!ATTLIST parameters auto-configure (true|false) "false">

<!ELEMENT parameter-type (#PCDATA)>

<!ELEMENT method (#PCDATA)>
<!ELEMENT class (#PCDATA)>
<!ELEMENT permissions (role-name*)>
<!ELEMENT role-name (#PCDATA)>
```

Kuva 21 Executable-task.dtd

Tehtävätyyppien asetukset kirjoitetaan siis xml-muodossa executable-task.xml tiedostoon. Tiedostoon määritellään task-type –elementtien sisään kullekin tehtävätyypille omat asetukset. Kaikkien tehtävätyyppien asetukset määritellään siis samaan tiedostoon. Esimerkki tehtävätyypin määrittelystä xml-tiedostosta on esitetty kuvassa 22. Toteutuksessa ei ole tällä hetkellä otettu huomioon kansainvälistämistä, eli tehtävätyypin ja parametrien kuvaukset kirjoitetaan suoraan tiedostoon. Kuvausten kansainvälistäminen esimerkiksi erilliseen tiedostoon siirretään jatkokehitysideaksi.

Tehtävätyyppien asetukset luetaan järjestelmään tarvittaessa. Xml-tiedoston luku on toteutettu ScheduledTaskDAO-luokassa getExecutableTasks-metodissa. Metodille välitetään käyttäjälle annetut roolit listana UserRoleVO:ja ja metodi palauttaa listan ExecutableTaskVO-olioita. Metodissa xml-tiedostosta luetaan tehtävätyyppien asetukset ja tarkistetaan onko käyttäjällä samoja rooleja kuin kyseiselle tehtävätyypille on määritelty. Jos käyttäjällä on yksi tai useampi sama rooli kuin tehtävätyypillä, käyttäjällä on tällöin oikeus ajastaa tehtävätyyppiä käyttäviä tehtäviä.

```
<task-type identifier="ImportCompanyData">
  <name>Perustietojen tuonti</name>
  <description>
        Tuo perustiedot yrityksen tuotteista.
  </description>
  <parameters>
    <description>
      Tarvitsee tiedon mitä tietotyyppejä päivitetään
    </description>
    <parameter-type>
      fi.logico.frank2.valueobjects.HistoryReqParamVO
    </parameter-type>
  </parameters>
  <method>importCompanyData</method>
  <class>
    fi.logico.frank2.businessobjects.CompanyDataImportTask
  </class>
  <permissions>
    <role-name>SuperUser</role-name>
    <role-name>SiteAdministrator</role-name>
  </permissions>
</task-type>
```

Kuva 22 Esimerkki tehtävätyypin asetuksista

Metodissa käytetään JDOM-kirjaston SAXBuilder luokkaa ja tiedoston muoto validoidaan ennen lukua [14][15]. Lisäksi metodissa tarkistetaan kunkin tehtävätyypin luokan ja metodin olemassaolo Java:n Reflection –API:a käyttäen. Metodiin on toteutettu kehitys- ja testausvaiheen lokitusta tehtävätyypin kehittäjän avuksi. Järjestelmä kertoo muun muassa jos konfigurointitiedostoa ei löydy, sen muoto on virheellinen tai jos tehtävätyypille asetettua luokkaa tai metodia ei löydy.

7 Tehtävätyypin implementointi

7.1 Ohjeistus

7.1.1 Suunnittelu

Uuden tehtävätyypin suunnittelussa on otettava huomioon muutamia seikkoja. Ennen tehtävätyypin suunnittelun aloittamista on määriteltävä, onko tarvittava toiminto mahdollista toteuttaa ajastettavana tehtävänä vai onko toiminto toteutettava muuhun olemassa olevaan moduuliin normaalina ominaisuutena. Ajastetun tehtävän käyttöä rajoittaa lähinnä kaksi seikkaa. Tehtävä konfiguroidaan kerran ja samaa konfiguraatiota käytetään sen jälkeen kaikissa kyseisen ajastetun tehtävän ajoissa. Mikäli uusi ominaisuus vaatii ajokertakohtaisia asetuksia, ei ominaisuuden toteuttaminen ajastettavana tehtävätyyppinä ole järkevä ratkaisu. Toisena rajoitteena on, että käyttäjä ei voi saada palautetta käyttöliittymätasolle ajastetun tehtävän suorittamista toimista muuten kuin lokitietojen kautta. Esimerkiksi raporttien esittäminen ajastetun tehtävän kautta ei ole mahdollista.

Suunnitteluvaiheen alussa määritellään yksityiskohtaisesti, mitä toteutettavan tehtävän tulee tehdä ja minkälaisia tuloksia suorituksesta saadaan. Tässä vaiheessa tulee myös miettiä mihin luokkaan sovelluslogiikka toteutetaan. Sovelluslogiikan rakennetta ja lokitusta sekä uuden tehtävätyypin nimeämistä kannattaa niin ikään miettiä tässä vaiheessa. Varsinkin tehtävätyypin yksilöivän tunnuksen muuttaminen jälkeenpäin on verrattain työlästä, joten sen lukkoon lyöminen tässä vaiheessa on järkevää.

Seuraavaksi tulee määrittää mitä parametreja tehtävän suoritusmetodin tulee vastaanottaa tehtävän suorittamiseksi. Myös parametrien tyypit kannattaa miettiä valmiiksi jo tässä vaiheessa. Olemassa olevien luokkien käyttäminen on suositeltavaa. Jos valmista luokkaa parametrien välitykseen ei löydy, kannattaa käyttää kokoelmaluokkaa, esimerkiksi Map-rajapinnan toteuttavat luokat ovat käteviä.

Suunnitteluvaiheessa kannattaa ottaa huomioon tehtävätyypin käyttäjäryhmät. Määritellään kenelle tehtävän suoritus on kohdennettu. Jos tehtävän suorittaminen sallitaan pelkästään yritystilin ylläpitäjälle, yrityksen id:n konfigurointia ei tarvitse syöttää tehtävätyypin konfigurointinäkymässä, vaan sen haku toteutetaan tehtävätyypin toiminnonkäsittelijässä. Jos tehtävä on suoritettavissa myös palvelun ylläpitäjän roolissa, käyttäjä liittää tehtävän ajastusnäkymässä yksittäiselle yritykselle. Tällöin valittu yritys voidaan asettaa tehtävän parametriksi tehtävätyypin toiminnonkäsittelijän autoSetParameters-metodissa.

Käytännössä kaikki executable-tasks.xml-tiedostoon määriteltävä tehtävätyypin tieto kannattaa suunnitella etukäteen ja konfiguroida uusi tehtävätyyppi xml-tiedostoon valmiiksi ennen kuin itse luokan ja jsp-sivujen toteutus aloitetaan.

7.1.2 Tehtävätyypin toteutusjärjestys

Tehtävätyypin toteuttaminen voidaan jakaa kolmeen vaiheeseen.

1. Tehtävätyypin asetusten lisääminen Executable-tasks.xml-tiedostoon.

2. Suoritusmetodin sovelluslogiikan ja lokituksen toteutus.
3. Konfigurointikäyttöliittymän ja toiminnonkäsittelijän toteutus.

Toteutusjärjestys määräytyy sen mukaan, miten testaus suoritetaan. Jos testausta varten toteutetaan erillinen testisovellus, jonka kautta suoritusmetodin toimivuus testataan, voidaan sovelluslogiikka toteuttaa ensin. Jos testaus suoritetaan Rank & Share -sovelluksen ajastusmoduulin kautta, pitää konfigurointikäyttöliittymä ja toiminnonkäsittelijä toteuttaa ennen sovelluslogiikan toteuttamista/testaamista.

Seuraavassa on esitetty ohjeet tehtävätyypin toteuttamiseen ilman erillistä testisovellusta.

1. Luo luokka johon tehtävän suorituksen sovelluslogiikka toteutetaan. Sijoita luokka mielellään fi.logico.frank2.businessobjects-pakettiin ja nimeä luokka selkeästi. Ota nimeämisessä huomioon, että saman luokan sisälle voi tulla useampia tehtävätyyppejä. Huomaa, että luokan tulee periä LoggableTask-luokka, jotta suorituksen aikainen lokitus on mahdollista.
2. Luo metodi, joka sisältää tehtävän suorituksen sovelluslogiikan. Metodin vastaanottaman parametrin tyyppi on määritelty suunnitteluvaiheessa. Jos parametrejä pitää olla useita, käytä Map-rajapinnan luokkaa. Viimeistään tässä vaiheessa kannattaa miettiä valmiiksi käytettävät avain-arvot ja parametrien tyypit Map-olion sisällä. Muista, että kaikkien Map-olion sisällä olevien arvojen tulee toteuttaa Serializable-rajapinta. Konfigurointikäyttöliittymän toimivuuden testaamiseksi metodiin kannattaa kirjoittaa parametrien tyyppien ja arvojen tulostus järjestelmälokiin tai suoraan tehtävän suorituslokiin. (Jälkimmäistä en suosittele, sillä testivaiheen lokitietojen poisto tulee tehdä käsin tietokannasta.)
3. Konfigurointikäyttöliittymän toteutus vaatii JSP-sivun sekä toiminnonkäsittelijän. JSP-sivu tulee asettaa pages/scheduledTasks/ hakemistoon ja se tulee nimetä samaksi kuin tehtävätyypin tunniste. Toiminnonkäsittelijä tulee asettaa fi.logico.frank2.ui.actionhandlers.scheduledTasks –pakettiin ja nimetä muotoon <tunniste>Action.java. Toiminnonkäsittelijän pitää lisäksi periä ConfigureTaskAction-luokka.
4. Käytä konfigurointisivun lomakeen action-määrittelynä <html:form action="/ConfigureTask<tunniste>?method=setParameters">. SetParameters-arvo viittaa metodiin, joka asettaa konfiguraatioparametrit istunto-olioon toiminnonkäsittelijässä. SetParameters vastaa parametreiltaan normaalia toiminnonkäsittelijän execute-metodia (kts. Struts-sovelluskehyksen Action-luokan määritys lähteestä [16]). SetParameters-metodi on asetettu käyttämään LazyValidatorForm-luokan oliota lomaketietojen välittämiseksi. Kyseisen luokan käytön voi tarkistaa lähteestä [16]. Lomaketietojen validointi tulee suorittaa setParameters-metodissa. Virheviestit tallennetaan saveErrorMessages-metodilla ja paluu JSP-sivulle pitää suorittaa doReturn-metodilla.
5. Muodosta setParameters-metodissa olio parametrien tallennusta varten. Parametrit pitää säilöä Seriaziable-rajapinnan totetuttavaan objektiin ja

tallennettava ConfigureTaskActionin saveParameters-metodilla. setParameters-metodi lopetetaan doReturn metodilla.

6. Luo tehtävätyypin toiminnonkäsittelijään prepare-metodi. Jos haluat asettaa istunto-olioon jotain ennen konfiguraatiosivulle menemistä, se tulee tehdä tässä metodissa. Prepare-metodi ajetaan ennen konfiguraatiosivulle siirtymistä.
7. Luo autoSetParameters-metodi. Tähän metodiin tulee luoda parametrien tallennuslogiikka siinä tapauksessa, että konfigurointia ei suoriteta käyttöliittymän kautta vaan automaattisesti. Palvelun ylläpitäjän valitsema yritys on asetettu CompanyVO-oliona istunto-olioon nimellä Constants.SESSION_COMPANY. Jos käyttäjä on yritystilin ylläpitäjän roolissa, on hänen edustamansa yritys asetettu niin ikään kyseisellä nimellä istunto-olioon. Muista tehdä parametrien tallennus saveParameters-metodin kautta.
8. Varmista, että tehtävätyypin asetukset on oikein Executable-tasks.xml-tiedostossa.
9. Testaa tehtävätyypin konfiguroinnin toiminta suorittamalla sille ajastus käyttöliittymästä.
10. Toteuta tehtävätyypin sovelluslogiikka. Metodin sisällä lokituksen tekemiseen tulee käyttää addLogMessage-metodia. Virhetilanteissa heitä poikkeus ulos suoritusmetodista. Kirjoita poikkeuksen viestiin riittävästi tietoa virheen syystä, jotta ylläpitäjät pystyvät korjaamaan virheen.

7.2 Esimerkkitehtävän toteuttaminen

7.2.1 Toteumatiedon tuonti

Automatisaatio- ja ajastusmoduulin toiminnan todentamiseksi ja moduulin etujen analysoimiseksi Rank & Share -sovellukseen toteutetaan tämän työn puitteissa yksi tehtävätyyppi. Toteumatiedon tuonti on toiminnallisuutena oleellinen Rank & Share -sovelluksen käytön nopeuttamiseksi ja ylläpitotyön vähentämiseksi. Toteumatiedon tuonti on lisäksi hyvä esimerkki tehtävätyypistä, jonka suorituksen aikainen loki on tärkeässä roolissa tehtävätyypin sujuvan käytön varmistamiseksi. Toteumatiedon tuonti on manuaalisesti toteutettuna verrattain hankala ja monimutkainen prosessi. Tehtävän automatisointi tarjoaa mahdollisuuden siirtää toteumatiedon tuonti asiakkaan ylläpitäjän vastuulle ja vähentää näin palvelun ylläpitäjän työmäärää.

7.2.2 Esimerkkitehtävätyypin suunnittelu

Manuaalisesti toteumatiedon tuonti on toteutettu tietokantaskripteillä, joiden ajo vaatii suoraa tietokantayhteyttä sekä skriptien manuaalista muokkaamista asiakaskohtaisesti. Tietokantaskriptit siirretään tehtävätyypin suoritusmetodin ajettavaksi ja asiakaskohtaisten tietojen asettaminen välitetään metodille parametreina. Suoritusmetodille luodaan oma luokka ImportTask.java. Suoritusmetodi on nimeltää execute ja se ottaa vastaan yhden String-tyyppisen parametrin. Parametri on yrityksen nimi, jonka toteuma halutaan tuoda järjestelmään. Toteuma haetaan tiedostosta, jonka sijainti palvelimella on ennalta

määritelty. Kyseisestä hakemistosta haetaan käsittelyyn uusin tiedosto. Tiedoston hakuun käytetään valmista DataImportDAO-luokassa olevaa metodia.

Tehtävätyypin toteuttava luokka käsittää useita metodeita, joista kukin tekee pienen osan kokonaisprosessista. Näin sovelluslogiikka saadaan ylläpidettävämmäksi ja helpommin ymmärrettäväksi. Metodeissa muun muassa luodaan väliaikaisia tietokantatauluja, validoidaan toteumatieto-tiedoston sisältö, kopioidaan toteuma väliaikaisiin tauluihin, suoritetaan laskenta ja asetetaan auki lasketut toteumatiedot lopullisiin tietokantatauluihin. Tietokantayhteys muodostetaan hibernate-palvelun kautta ja transaktion avaaminen ja sulkeminen toteutetaan käsin.

Kuvassa 23 on esitetty toteutettavan tehtävätyypin asetukset Executable-tasks.xml-tiedostossa. Kuvasta nähdään että tehtävätyypin yksilöiväksi tunnukseksi määritellään SalesHistoryDataImport. Suoritusmetodi tarvitsee parametriksi ainoastaan yrityksen nimen, joten tehtävän konfigurointi voidaan tehdä automaattisesti, toisin sanoen tehtävätyypille ei tarvitse tehdä erillistä konfigurointinäkymää.

```
<task-type identifier="SalesHistoryDataImport">
  <name>Toteumatietojen tuonti</name>
  <description>Tuo viimeisimmät toteumatiedot CSV-tiedostosta</description>
  <parameters auto-configure="true">
    <description>Tarvitsee yrityksen nimen</description>
    <parameter-type>java.lang.String</parameter-type>
  </parameters>
  <method>execute</method>
  <class>fi.logico.frank2.businessobjects.ImportTask</class>
  <permissions>
    <role-name>SuperUser</role-name>
    <role-name>SiteAdministrator</role-name>
  </permissions>
</task-type>
```

Kuva 23 Toteumatiedon tuonti -tehtävätyypin asetukset

7.2.3 Esimerkkitehtävätyypin toteutus

Esimerkkitehtävä voidaan konfiguroida automaattisesti, joten sille ei tarvitse luoda konfigurointikäyttöliittymää eikä siis JSP-sivua. Toiminnonkäsittelijä kuitenkin täytyy toteuttaa, ja siihen autoSetParameters-metodi parametrien tallennukseen. Ohjeiden mukaisesti luodaan SalesHistoryDataImportAction-toiminnonkäsittelijä joka toteuttaa ConfigureTaskAction-yliluokan vaatimat abstraktit metodit. Toiminnonkäsittelijän ohjelmalistaus on esitetty kuvassa 24. AutoSetParameters-metodissa haetaan palvelun ylläpitäjän valitsema tai yritystilin ylläpitäjän yritys-olio istunto-oliosta. CompanyVO-oliolta haetaan yrityksen nimi ja se tallennetaan saveParameters-metodilla istunto-olioon myöhempää käsittelyä varten.

```
package fi.logico.frank2.ui.actionhandlers.scheduledTasks;

import javax.servlet.http.HttpServletRequest;
import fi.logico.frank2.valueobjects.CompanyVO;
import fi.logico.frank2.valueobjects.Constants;

public class SalesHistoryDataImportAction
  extends ConfigureTaskAction {

  @Override
  public void prepare(HttpServletRequest request)
    throws Exception {
    // TODO Auto-generated method stub
  }

  @Override
  public void autoSetParameters(HttpServletRequest request)
    throws Exception {
    String companyName =
      ((CompanyVO)request.getSession().getAttribute(
        Constants.SESSION_COMPANY)).getName();
    saveParameters(request, companyName);
  }
}
```

Kuva 24 SalesHistoryDataImport-luokan ohjelmalistaus

Tehtävätyypin sovelluslogiikan runko toteutetaan ImportTask-luokkaan execute-metodiin. Lopullinen versio execute-metodista on esitetty kuvassa 25. Kukin osatehtävä toteumatiedon tuonnissa on toteutettu omaan metodiin ja metodit ajetaan suoritus-metodissa. Kunkin metodin ajon aluksi lokitetaan tieto, mikä osatehtävä suoritetaan. Jokaisen osatehtävän aikana voi tapahtua yksi tai useampia virhetilanteita. Osatehtävistä tulevat virhetilanteet otetaan kiinni execute-metodissa jotta suorituksen aikana tehdyt tietokantatransaktiot voidaan peruuttaa ja tietokantaistunto sulkea.

Suoritusmetodi alkaa parametrin validoinnilla ja asiakaskohtaisten muuttujien luomisella. Tämän jälkeen haetaan toteumatieto palvelimen levyltä ja tarkistetaan että sen muoto on oikea. Toteumatiedon tulee olla csv-muodossa eli arvojen erottimena tulee olla puolipiste. Kun tiedoston muoto on validoitu, luodaan tietokantaan väliaikaiset taulut tuotavalle tiedolle. Tiedostosta luetaan arvot väliaikaisiin tietokantatauluihin, tarkistetaan niiden vastaavuus olemassa olevaan tietoon muun muassa tuotetietojen osalta. Seuraavaksi tuotu toteumatieto lasketaan auki ja kirjoitetaan lopullisiin tauluihin. Lopuksi väliaikaiset tietokantataulut poistetaan järjestelmästä.

Esimerkkitehtävätyyppinä toteumatiedon tuonti osoittaa, että tehtävätyypin lisääminen Rank & Share -sovellukseen on nopeaa. Tehtävätyypin sovelluslogiikan toteutus vei aikaa huomattavasti enemmän kuin tehtävätyypin integrointi. Integroinnin toteuttamiseksi Executable-tasks.xml-tiedostoon lisättiin tehtävätyypin asetukset (kuva 23) ja tehtävätyypille toteutettiin toiminnonkäsittelijä (kuva 24).

```
public void execute(String companyName)
  throws Throwable, IllegalArgumentException {
  try {
    InitialContext ctx = new InitialContext();
    SessionFactory factory  = (SessionFactory)ctx.lookup(
                              "java:/hibernate/SessionFactory");
    this.session = factory.openSession();
    this.session.beginTransaction();

    addLogMessage("Preparing ImportTask");
    this.prepareImport(companyName, Constants.CSV_COLUMN_SEPARATOR);

    addLogMessage("Checking file structure");
    this.checkFile();

    addLogMessage("Creating tables");
    this.createTables();

    addLogMessage("Importing original data from file " +
                  this.dataFile.getAbsolutePath());
    this.importDataFromFile();

    addLogMessage("Preparing imported data");
    this.prepareData();

    addLogMessage("Checking imported data");
    this.checkImportedData();

    addLogMessage("Creating views");
    this.createViews();

    addLogMessage("Populating temporary tables");
    this.populateTables();

    addLogMessage("Summing duplicate rows");
    this.sumDuplicateRows();

    addLogMessage("Copying data to correct (real) tables");
    this.copyDataToRealTables();

    addLogMessage("Dropping temporary views & tables");
    this.dropTables();

    //commit if everything went well.
    session.getTransaction().commit();

  }
  catch(Throwable t) {
    if(this.session.getTransaction()!= null) {
      this.session.getTransaction().rollback();
    }
    throw t;
  }
  finally {
    if(this.session.isOpen()) {
      this.session.close();
    }
  }
}
```

Kuva 25 Toteumatiedon tuonti -tehtävätyypin suoritusmetodi

8 Analyysi

8.1 Vaatimusten toteutuminen

8.1.1 Automatisaatio –ja ajastusmoduuli

Moduulin ajastustoiminto on toteutettu EJB Timer Service-komponentilla JBoss-sovelluspalvelimen tarjoamana palveluna. Ajastustoiminnallisuuden sovelluslogiikka toteuttaa vaaditut toimenpiteet ajastettaville tehtäville. Vaaditut toimenpiteet ovat uuden tehtävän lisääminen ajastuspalveluun, ajastuksen poisto, keskeytys sekä uudelleenkäynnistys. Ajastuspalvelussa olevien tehtävien hallinta ja uusien tehtävien lisääminen tapahtuu Rank & Share -sovelluksen ylläpitomoduuliin toteutettujen käyttöliittymänäkymien kautta. Käyttöliittymien vaatimusmäärittelyt on esitetty liitteessä 3. Toteutukset täyttävät vaatimusmäärittelyn kaikilta osin.

Ajastettujen toimintojen suorituksen aikaiset lokitiedot tallennetaan tietokantaan. Lokitieto-alkion sisältö käsittää kaikki vaatimusmäärittelyvaiheessa määritellyt tiedot. Lokitietojen esittäminen ja suodatus eri kriteereillä on toteutettu Rank & Share -sovellukseen ylläpitomoduulin osaksi. Käyttöliittymässä esitetyt tiedot ja toiminnot on toteutettu kuten vaatimusmäärittelyssä määriteltiin. Ajastettujen tehtävien tapahtumaloki on sisällytetty samaan lokinäkymään suorituslokin kanssa. Tapahtumaloki ja suoritusloki voidaan esittää erillään suodatustoiminnon avulla. Virheloki käsittää virheviestien lisäksi Java-ohjelmointikielestä tuodun poikkeuslistauksen. Tätä ominaisuutta ei määritelty vaatimusmäärittelyvaiheessa, mutta osoittautui käytännölliseksi testauksen aikana. Vaatimusmäärittelyssä löydetyistä toiminnallisista vaatimuksista (Liite 2) toteuttamatta jäi vaatimus V12. Ylläpitäjää ei informoida virheellisestä tehtävän suorituksesta sähköpostiviestillä. Tämä ominaisuus toteutetaan tarvittaessa myöhemmin. Lisäksi vaatimusmäärittelyssä esiin tullut ominaisuus lokitietojen suodattamisesta valittavalle aikavälille siirtyy toteutettavaksi myöhemmin.

Automatisaatio- ja ajastusmoduulin käyttöliittymät on todettu käyttäjien toimesta yksinkertaisiksi ja helppokäyttöisiksi ja ne täyttävät niille asetetut tavoitteet ja vaatimukset. Teknisten vaatimusten osalta voidaan todeta, että ne on täytetty suurimmalta osin. Tietokantakyselyt eroavat Postgre- ja MSSQL-versioissa hieman. Tämä ei kuitenkaan ole ongelma, koska Rank & Share -sovelluksesta on toteutettu joka tapauksessa eri versiot eri tietokannoille. DWR-komponentin käyttö oli lopulta suunniteltua vähäisempää, mutta uuden tekniikan käyttöönotto on auttanut sovelluksen muiden uusien ominaisuuksien kehittämistä ja käytettävyyttä huomattavasti.

8.1.2 Tehtävätyypin lisääminen ja integrointi

Uusien tehtävätyyppien toteuttamiselle ja integroinnille määriteltiin kaksi tärkeää vaatimusta. Uuden tehtävätyypin lisääminen Rank & Share -sovellukseen tulee olla mahdollisimman yksinkertaista ja nopeaa, eikä integroinnin toteutus ja tehtävätyypin sovelluslogiikka saa riippua toisistaan. Käytännössä siis tehtävätyypin sovelluslogiikka voidaan toteuttaa ja testata täysin erillään muusta sovelluksesta ja integroida vasta myöhemmin. Nämä vaatimukset pystyttiin täyttämään hyvin. Tämä voidaan todeta esimerkkitehtävätyypin

toteutusselostuksesta. Tarkempi analyysi tehtävätyyppien kehityksestä ja integroinnista on esitetty kappaleessa 8.2.

Tehtävätyypin vaatimukset on esitetty kappaleessa 4.2. Kappaleessa listatut vaatimukset oli priorisoitu kriittisiksi, joten niiden toteuttaminen tämän työn puitteissa oli välttämätöntä. Vaatimukset pystyttiin täyttämään hyvin, mutta muutamia kompromisseja jouduttiin tekemään. Kompromissit ovat aiheuttaneet joitain ongelmia sovelluksen ylläpidossa ja jatkokehittämisessä. Lokitustoiminto LoggableTask-luokan avulla suoritetaan tällä hetkellä ainoastaan yhdellä tasolla. Tämä ei tule jatkossa riittämään. Automatisaatiomoduulin jatkokehityksessä tähän ongelmaan palataan ja se tullaan korjaamaan. Toisena puutteena on rajoite, jonka mukaan tehtävätyypin suoritusmetodi ottaa vastaan ainoastaan yhden parametrin. Tämä haittaa jonkin verran ohjelmakoodin ymmärrettävyyttä ja ylläpitoa. Tähänkin epäkohtaan palataan jatkokehityksen aikana. Kaikki jatkokehitysehdotukset on esitetty tarkemmin kappaleessa 9.2.

8.2 Automatisaation edut

Toteutetun automatisaatio- ja ajastusmoduulin ensisijaisena tavoitteena oli ratkaista muutamia Rank & Share -sovelluksen ylläpitoon liittyviä haasteita. Nämä tavoitteet on esitetty liiketoimintavaatimuksina kappaleessa 4.1.1. Tärkeimpänä tavoitteena oli manuaalisen työn vähentäminen palvelun taustatoimintojen suorittamisen yhteydessä. Tämä tavoite voidaan saavuttaa työssä toteutetulla automatisaatio- ja ajastusmoduulilla toteuttamalla sen käyttöön tehtävätyyppejä, jotka voidaan automatisoida huolehtimaan ylläpito- ja taustatoiminnoista. Toteutettu moduuli ei siis yksinään pysty täyttämään ylläpidon vähenemiseen liittyviä vaatimuksia, vaan ne saavutetaan epäsuorasti tulevaisuudessa moduulin avulla. Työn suorat vaikutukset ylläpidon työmäärän vähenemiseen voidaan todeta laajemmassa mittakaavassa toteutettujen tehtävätyyppien määrän kasvaessa. Suora vaikutus työn puitteissa toteutetuilla toiminnoilla ylläpidon työmäärään on ainoastaan toteumatiedon tuonnin kohdalla.

Automatisaatio- ja ajastusmoduulin toteuttamisen ja käyttöönoton jälkeen sovelluskehitystä on jatkettu toteuttamalla uusia tehtävätyyppejä moduulin käyttöön. Moduuli on otettu testikäyttöön pilottiprojektissa ja siitä saadut kokemukset ovat olleet positiivisia. Asiakkaan ylläpito on pystynyt käyttämään automatisoituja toimintoja pääosin sujuvasti ilman palvelun ylläpidon tukea. Suorituksen aikainen lokituksen ymmärrettävyys on ollut tyydyttävää, mutta ei aina riittävää virhetilanteesta selviämiseen. Tähän tulee kiinnittää tulevaisuudessa enemmän huomiota.

Sovelluskehityksen näkökulmasta työn puitteissa toteutetulle moduulille asetettiin muutamia vaatimuksia. Moduuliin lisättävät tehtävätyypit tulee olla helposti integroitavissa olemassa olevaan järjestelmään ja käyttöliittymänäkymien luominen tehtävätyypeille tulee olla nopeaa. Parametrien määrää ja tyyppejä ei ole rajoitettu. Kehittäjiltä tulleen palautteen perusteella voidaan sanoa, että suurin hyöty tehtävätyyppien toteutusvaiheessa oli se, ettei erillisiä käyttöliittymänäkymiä tarvinnut luoda erikseen jokaiselle uudelle ominaisuudelle, vaan tarvittavat parametrit pystyi vastaanottamaan pienellä vaivalla. Kehittäjät olivat myös tyytyväisiä tehtävätyypin suorituksenaikaisen lokituksen

toteuttamisen helppouteen ja yksinkertaisuuteen. Kokemuksia testauksesta erillisellä testiohjelmalla ei tähän mennessä ole saatu.

Tällä hetkellä toteutettuja tehtävätyyppejä on seitsemän ja niistä aktiivisessa käytössä on neljä. Esimerkkitehtävätyyppinä toteutetun toteumatiedon tuonti -tehtävätyypin lisäksi järjestelmään on toteutettu ja käyttöön otettu ennusteiden revisiointi, yrityksen perustietojen tuonti sekä ennustetietojen vienti.

9 Johtopäätökset

9.1 Työn tavoitteet

Työssä saadut tulokset ja kokemukset ovat suurimmalta osin positiivisia, ja totetutettu moduuli on käyttökelpoinen lisä Rank & Share -sovellukseen. Automatisaation toteuttaminen EJB Timer Service:n avulla oli helppo tapa toteuttaa ajastustoiminto sovelluspalvelimelle, koska sen käyttöönotto ja ominaisuuksien opettelu oli nopeaa. Kyseinen komponentti osoittautui kuitenkin lopulta hyvin rajoittuneeksi ja yksinkertaiseksi, eikä tarjonnut juuri mitään kehitystyötä helpottavia ominaisuuksia. Mikäli kyseistä komponenttia ei olisi valittu etukäteen työssä käytettäväksi, olisi jonkin muun ajastuskomponentin käyttö ollut järkevämpää. Ohjelmien ajastettu suorittaminen on tietojenkäsittelyssä perustoiminnallisuus, jonka toteuttamiseen on löydettävissä paljon valmiita komponentteja. Työn aikana ja varsinkin sen jälkeen on voinut huomata osin keksineensä pyörän uudestaan. Perusteellinen etukäteiskartoitus mahdollisista valmiista ajastuskomponenteista olisi nopeuttanut ja helpottanut automatisaatio- ja ajastusmoduulin toteuttamista.

Yleiset hyödyt taustatoimintojen automatisoinnista ovat kiistattomat. Käytännössä ilman moduulin toteuttamista Rank & Share -palvelun myynti asiakkaalle on hyvin vaikeaa. Ylläpidosta johtuvat kustannukset ovat vähentyneet huomattavasti ja sovelluskehitykseen on saatu moduulin myötä helppo tapa toteuttaa uusia automatisoitavia ominaisuuksia. Myös Rank & Share -palvelun ydintoimintoon, eli liiketoiminnan ennustamiseen on nyt mahdollista toteuttaa uudenlaisia toimintoja, kun erilaiset laskennalliset toiminnot voidaan toteuttaa ja automatisoida helposti. Muun muassa ennustamisessa käytettävät keskiarvo- ja normaalimyyntilaskelmat tarjoavat automatisoituna mahdollisuuden automatisoida myös itse ennustusprosessia entistä pidemmälle. Tällä hetkellä ennustamisprosessi Rank & Share -palvelussa on suurimmilta osin käsityötä, jossa hyvän ennustetarkkuuden saavuttamiseen tarvitaan käytännön kokemusta ja tietoja.

Yhteenvetona voidaan todeta, että itse automatisaatio- ja ajastusmoduulin toteuttaminen olisi voitu tehdä nopeammin ja helpommin. Kuitenkin lopputuloksena toteutunut moduuli on hyvä ja tärkeä lisä Rank & Share -palvelulle. Automatisaation edut voidaan nähdä jo nyt, ja tulevaisuudessa tehtävätyyppien määrän kasvaessa sekä asiakaskunnan lisääntyessä vielä selkeämmin.

9.2 Moduulin jatkokehitys

Tämän työn puitteissa toteutetut ominaisuudet ovat riittävät taustatoimintojen automatisointiin. Käyttöliittymien suunnittelussa keskityttiin ennen kaikkea helppouteen ja yksinkertaisuuteen. Tulevaisuudessa käyttöliittymien käytettävyyttä pitäisi tutkia ja parantaa. Tämä tullaan toteuttamaan osana koko Rank & Share -sovelluksen käytettävyysparannusprojektia. Uusia ominaisuuksia ei näillä näkymin ole tarvetta tehdä, mutta moduulin kehitys otetaan normaaliksi osaksi tuotekehitysprosessia ja käyttäjien palaute tulee vaikuttamaan vahvasti moduulin kehitykseen.

Tehtävän suorituslokin toteuttaminen ainoastaan yhdellä tasolla on tällä hetkellä riittävä, mutta se voitaisiin toteuttaa usealla tasolla. Esimerkiksi kehitys- ja testausvaiheen loki joudutaan tällä hetkellä poistamaan tuotantoversiosta. Tämä ei ole käytännöllistä. Jakamalla lokitus esimerkiksi debug- ja info-tasoihin, voidaan kaikki testiloki säilyttää ja esittää vain tarvittaessa käyttöliittymässä. Myös eri tasoisten lokitietojen suodatus tulee tällöin toteuttaa.

Tällä hetkellä tehtävätyyppien oikeuksien hallinta on toteutettu ainoastaan käyttäjän roolin mukaisesti. Tulevaisuudessa tehtävätyyppien käyttöoikeudet pitää voida määritellä käyttäjäkohtaisesti. Tämän ominaisuuden toteuttaminen vaatii entistä tarkempia käyttäjäkohtaisia asetuksia sovellustasolla. Tällä hetkellä on suunnitelmissa toteuttaa käyttäjäkohtaisia asetuksia muihinkin Rank & Share -sovelluksen työkaluihin. Näin ollen koko käyttäjäoikeusjärjestelmän uudistaminen tulee eteen ennemmin tai myöhemmin. Moduulin tehtävätyyppien oikeuksien hallinta kannattaa ottaa osaksi tätä uudistusta.

Tehtävätyyppien kehityksen kannalta tärkein heikkous automatisaatiomoduulissa on suoritusmetodille välitettävien parametrien rajoitettu määrä. Jatkokehitys tämän asian suhteen tulee olemaan jossain vaiheessa tärkeä. Parametrien tyypit tullaan määrittelemään executable-tasks.xml-tiedostossa ja parametrin pystytään automaattisesti välittämään tämän tiedon avulla sellaisinaan suoritusmetodille ilman Map-luokkien käyttöä.

Lähdeluettelo

[1] Sauhke, J. 2006. Architecting Java 2 Enterprise Edition Application Service. Teknillinen korkeakoulu, tietoliikennetekniikan osasto. Espoo.

[2] J2EE 1.4 tutorial: http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html [viitattu 13.2.2008]

[3] Fleury, Mark & Stark, Scott & Richards, Norman: Jboss The Offical Guide.

[4] Roman, Ed & Amber, Scott W. & Jewell, Tyler: Mastering Enterprice JavaBeans. 2nd edition.

[5] Ashmore, Derek C.: The J2EE Architect's Handbook.

[6] Designing Enterprise Applications with the J2EETM Platform, Second Edition: http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html [viitattu 13.2.2008]

[7] jGuru Enterprise JavaBeans Fundamentals: http://java.sun.com/developer/onlineTraining/EJBIntro/EJBIntro.html#EJBTechnology [viitattu 13.2.2008]

[8] jboss.org JIRA #JBAS-4290: http://jira.jboss.org/jira/browse/JBAS-4290 [viitattu 13.2.2008]

[9] Struts Key Technologies Primer: http://struts.apache.org/primer.html [viitattu 13.2.2008]

[10] Hibernate Reference Documentation: http://www.hibernate.org/hib_docs/reference/en/html_single/ [viitattu 13.2.2008]

[11] DWR overview: http://getahead.org/dwr/overview/dwr [viitattu 13.2.2008]

[12] Code Conventions for the Java Programming Language: http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html [viitattu 13.2.2008]

[13] The Java Reflection API: http://java.sun.com/docs/books/tutorial/reflect/ [viitattu 13.2.2008]

[14] JDOM API: http://www.jdom.org/docs/apidocs/ [viitattu 13.2.2008]

[15] SAX Project: http://www.saxproject.org/ [viitattu 13.2.2008]

[16] Struts API: http://struts.apache.org/1.2.7/api/index.html [viitattu 13.2.2008]

Liite 1 Käyttötapaukset

Käyttötapauksissa, joissa puhutaan yleisesti ylläpitäjästä, tarkoitetaan kumpaa tahansa edellä mainittua toimijaa. Käyttötapaukset on esitetty seuraavaa rakennetta noudattaen:

ID:	Käyttötapauksen yksilöivä tunniste
Käyttötapauksen nimi:	Lyhyt kuvaava nimi
Toimija:	Kuka on pääasiallinen toimija
Tavoite:	Mikä on suoritettavan tehtävän tavoite käyttäjän näkökulmasta.
Alkutilanne:	Lyhyt kuvaus tilanteesta, jossa ollaan käyttötapauksen alkaessa.
Kuvaus:	Kuvaus käyttötapauksen aikana tapahtuvista toiminnoista.
Poikkeukset:	Kuvaus oletetusta toiminnasta poikkeavista tapahtumista ja erikoistilanteista käyttötapauksen aikana.
Käyttötaajuus:	Miten usein käyttötapauksessa kuvattu tehtävä suoritetaan.
Tärkeys:	Kriittinen, tärkeä tai lisäominaisuus

ID	YKT1
Käyttötapauksen nimi	Ylläpitäjä konfiguroi ja ajastaa uuden tehtävän
Toimija	Ylläpitäjä
Tavoite	Ajastaa tehtävä suoritettavaksi tiettynä ajanhetkenä ja asettaa tehtävän suoritukselle vaadittavat parametrit.
Alkutilanne	Käyttäjä on kirjautunut palveluun ylläpitäjän tunnuksilla.
Kuvaus	1. Ylläpitäjä siirtyy uuden tehtävän lisäämislomakkeeseen valitsemalla ylläpitovalikosta ”Lisää ajastettu tehtävä”. 2. Ylläpitäjä kirjoittaa lomakkeeseen tehtävän nimen, kuvauksen ja ensimmäisen suorituksen alkamisajankohdan. 3. Ylläpitäjä asettaa suoritusten aikavälin. 4. Ylläpitäjä valitsee suoritettavan tehtävän tyypin valikosta. 5. Ylläpitäjä siirtyy tehtävätyypin konfigurointinäkymään ja syöttää tehtävätyypille ominaiset asetukset lomakkeeseen.

	6. Ylläpitäjä hyväksyy uudelle tehtävälle annetut asetukset. 7. Ylläpitäjä siirtyy pois näkymästä.
Poikkeukset	Käyttäjä ei ole syöttänyt lomakkeisiin kaikkia tietoja tai tiedot eivät ole kelvollisia. Tällöin käyttöliittymään tulee virheilmoitus ja kehoitus korjata puutteet.
Käyttötaajuus	Muutamia kertoja vuodessa, palvelun käyttöönoton aikana useasti.
Tärkeys	Kriittinen

ID	AYKT1
Käyttötapauksen nimi	Asiakastilin ylläpitäjä tarkistaa ajastettujen tehtävien tiedot
Toimija	Asiakastilin ylläpitäjä
Tavoite	Saada informaatiota yrityksen ajastettujen tehtävien seuraavista suoritusajankohdista ja suoritettujen tehtävien tiloista.
Alkutilanne	Käyttäjä on kirjautunut palveluun asiakastilin ylläpitäjän tunnuksilla.
Kuvaus	1. Ylläpitäjä siirtyy ajastettujen tehtävien ylläpito -näkymään ylläpitovalikon kautta. 2. Ylläpitäjä näkee listan yrityksensä nimissä olevista tehtävistä. 3. Listasta ylläpitäjä lukee tehtävän nimen, kuvauksen, luontipäivämäärän, seuraavan suorituksen ajankohdan, paljonko seuraavaan suoritukseen on aikaa sekä suoritusten aikavälin ja tilan. Tilatiedoista ylläpitäjä näkee onko tehtävä aktiivinen, suoritettu onnistuneesti vai onko suoritus keskeytynyt virheen takia.
Poikkeukset	-
Käyttötaajuus	Päivittäin
Tärkeys	Kriittinen

ID	AYKT2
Käyttötapauksen nimi	Asiakastilin ylläpitäjä tarkistaa ajastusmoduulin tapahtumat
Toimija	Asiakastilin ylläpitäjä
Tavoite	Saada informaatiota viimeisimmistä suoritetuista tehtävistä.

Alkutilanne	Käyttäjä on kirjautunut palveluun asiakastilin ylläpitäjän tunnuksilla.
Kuvaus	1. Ylläpitäjä siirtyy ajoitettujen tehtävien loki -näkymään ylläpitovalikon kautta. 2. Lokinäkymässä ylläpitäjällä näkee lokitiedot viimeisimmistä suoritetuista tehtävistä. Tehtävän kohdalla näkyy tehtävän nimi, suorituksen aloitusajankohta, kesto ja tila. 3. Lokitiedoissa on myös maininnat tehtävien lisäyksistä, poistoista ja keskeytyneiden tehtävien uudelleen käynnistämisistä.
Poikkeukset	-
Käyttötaajuus	Päivittäin
Tärkeys	Kriittinen

ID	PYKT1
Käyttötapauksen nimi	Palvelun ylläpitäjä tarkistaa ajastettujen tehtävien tiedot
Toimija	Palvelun ylläpitäjä
Tavoite	Saada informaatiota järjestelmään ajastettujen tehtävien seuraavista suoritusajankohdista ja suoritettujen tehtävien tiloista.
Alkutilanne	Käyttäjä on kirjautunut palveluun asiakastilin ylläpitäjän tunnuksilla.
Kuvaus	1. Ylläpitäjä siirtyy ajastettujen tehtävien ylläpito -näkymään ylläpitovalikon kautta. 2. Ylläpitäjä näkee listan järjestelmään ajastetuista tehtävistä. 3. Listasta ylläpitäjä lukee tehtävän nimen, kuvauksen, luontipäivämäärän, seuraavan suorituksen ajankohdan, paljonko seuraavaan suoritukseen on aikaa sekä suoritusten aikavälin ja tilan. Lisäksi palvelun ylläpitäjä näkee yrityksen nimen, jolle tehtävä kuuluu. 4. Tilatiedoista ylläpitäjä näkee onko tehtävä aktiivinen, suoritettu onnistuneesti vai onko suoritus keskeytynyt virheen takia.
Poikkeukset	-
Käyttötaajuus	Päivittäin
Tärkeys	Kriittinen

ID	PYKT2
Käyttötapauksen nimi	Palvelun ylläpitäjä tarkistaa ajastusmoduulin tapahtumat
Toimija	Palvelun ylläpitäjä
Tavoite	Saada informaatiota ajastettujen tehtävien seuraavista suoritusajankohdista ja suoritettujen tehtävien tiloista.
Alkutilanne	Käyttäjä on kirjautunut palveluun palvelun ylläpitäjän tunnuksilla.
Kuvaus	1. Ylläpitäjä siirtyy ajoitettujen tehtävien loki -näkymään ylläpitovalikon kautta. 2. Lokinäkymässä ylläpitäjä näkee lokitiedot viimeisimmistä järjestelmässä suoritetuista tehtävistä. Tehtävän kohdalla näkyy tehtävän nimi, suorituksen aloitusajankohta, kesto ja tila sekä yrityksen nimi, jolle tehtävä kuuluu. 3. Lokitiedoissa on myös maininnat tehtävien lisäyksistä, poistoista ja keskeytyneiden tehtävien uudelleen käynnistämisistä.
Poikkeukset	-
Käyttötaajuus	Viikottain
Tärkeys	Kriittinen

ID	YKT2
Käyttötapauksen nimi	Ylläpitäjä poistaa ajoitetun tehtävän
Toimija	Ylläpitäjä
Tavoite	Poistaa tarpeeton ajastettu tehtävä järjestelmästä
Alkutilanne	1. Käyttäjä on kirjautunut palveluun ylläpitäjän tunnuksilla. 2. Järjestelmässä on tarpeeton ajastettu tehtävä, joka pitää poistaa.
Kuvaus	1. Ylläpitäjä siirtyy ajastettujen tehtävien ylläpito -näkymään ylläpitovalikon kautta. 2. Ylläpitäjä näkee listan järjestelmään ajastetuista tehtävistä. 3. Ylläpitäjä etsii listasta poistettavan kohteen ja painaa ”poista” nappia. 4. Onnistuneen operaation päätteeksi kohde poistuu listalta.

Poikkeukset	-
Käyttötaajuus	Muutamia kertoja vuodessa.
Tärkeys	Kriittinen

ID	YKT3
Käyttötapauksen nimi	Ylläpitäjä keskeyttää ajoitetun tehtävän
Toimija	Ylläpitäjä
Tavoite	Keskeyttää ajastettu tehtävä.
Alkutilanne	1. Järjestelmään on lisätty ajastettu tehtävä, jonka ajoitus halutaan keskeyttää. Tehtävä ei ole suorituksessa. 2. Käyttäjä on kirjautunut palveluun ylläpitäjän tunnuksilla.
Kuvaus	1. Ylläpitäjä siirtyy ajastettujen tehtävien ylläpito -näkymään ylläpitovalikon kautta. 2. Ylläpitäjä näkee listan järjestelmään ajastetuista tehtävistä. 3. Ylläpitäjä etsii listasta keskeytettävän kohteen ja painaa ”keskeytä” nappia. 4. Onnistuneen operaation päätteeksi ”seuraava suoritus” sarake tyhjenee tehtävän kohdalta listalla.
Poikkeukset	Jos käyttäjä yrittää keskeyttää tehtävää, joka ei ole käynnissä (jo valmiiksi keskeytetty), järjestelmä ei tee mitään, vaan kertoo käyttäjälle että kyseinen tehtävä ei ole aktiivinen.
Käyttötaajuus	Viikottain
Tärkeys	Kriittinen

ID	YKT4
Käyttötapauksen nimi	Ylläpitäjä uudelleen ajoittaa tehtävän
Toimija	Ylläpitäjä
Tavoite	Ajastaa uudelleen keskeytynyt tehtävä.
Alkutilanne	1. Ajastettu tehtävä on aikaisemmin keskeytynyt virheen takia tai käyttäjä on itse keskeyttänyt tehtävän ajoituksen. 2. Käyttäjä on kirjautunut palveluun ylläpitäjän tunnuksilla.

Kuvaus	1. Ylläpitäjä siirtyy ajoitettujen tehtävien ylläpito -näkymään ylläpitovalikon kautta. 2. Ylläpitäjä näkee listan järjestelmään ajastetuista tehtävistä. 3. Ylläpitäjä etsii listasta ajastettavan kohteen ja painaa ”Ajasta” nappia. 4. Käyttöliittymään aukeaa lomake, johon käyttäjä asettaa seuraavan suorituksen ajankohdan. 5. Onnistuneen ajastuksen seurauksena tehtävälistaan tehtävän kohdalle ilmestyy ”seuraava suoritus”-sarakkeeseen annettu ajankohta.
Poikkeukset	Jos käyttäjä yrittää ajastaa tehtävää, joka on käynnissä. Järjestelmä ilmoittaa asiasta käyttäjälle, eikä näytä ajastulomaketta.
Käyttötaajuus	Viikottain
Tärkeys	Kriittinen

ID	ETKT1
Käyttötapauksen nimi	Asiakastilin ylläpitäjä ajastaa toteumatiedon tuoti -tehtävän
Toimija	Asiakastilin ylläpitäjä
Tavoite	Ajastaa viikottainen toteumatiedon tuonti -tehtävä järjestelmään.
Alkutilanne	Käyttäjä on kirjautunut palveluun asaikastilin ylläpitäjän tunnuksilla.
Kuvaus	1. Ylläpitäjä siirtyy uuden tehtävän lisäämislomakkeeseen valitsemalla ylläpitovalikosta ”Lisää ajastettu tehtävä”. 2. Ylläpitäjä kirjoittaa lomakkeeseen tehtävän nimen, kuvauksen ja ensimmäisen suorituksen alkamisajankohdan. 3. Ylläpitäjä asettaa suoritusten aikavälin valitsemalla valikosta ”viikko”. 4. Ylläpitäjä valitsee suoritettavan tehtävän tyypiksi ”Toteumatiedon tuonti”. 5. Toteumatiedon tuonti –tehtävä ei tarvitse konfigurointia, joten ylläpitäjä ainoastaan hyväksyy uudelle tehtävälle annetut asetukset. 1. Ylläpitäjä siirtyy pois näkymästä.
Poikkeukset	Käyttäjä ei ole syöttänyt lomakkeeseen kaikkia tietoja tai

	tiedot eivät ole kelvollisia. Tällöin käyttöliittymään tulee virheilmoitus ja kehotus korjata puutteet.
Käyttötaajuus	Kerran
Tärkeys	Toivottava

ID	ETKT2
Käyttötapauksen nimi	Virhe toteumatiedon tuonti -tehtävän suorituksessa
Toimija	Asiakastilin ylläpitäjä
Tavoite	Korjata virhe sekä suorittaa toteumatiedon tuonti järjestelmään.
Alkutilanne	1. Toteumatiedon tuonti –tehtävän edellinen suoritus on keskeytynyt virhetilanteeseen. 2. Käyttäjä on saanut sähköpostitse ilmoituksen virheeseen keskeytyneestä toteumatiedon tuonti-tehtävän suorituksesta. 3. Käyttäjä on kirjautunut palveluun asiakastilin ylläpitäjän tunnuksilla.
Kuvaus	1. Ylläpitäjä siirtyy ajoitettujen tehtävien loki -näkymään ylläpitovalikon kautta. 2. Lokinäkymässä ylläpitäjä näkee lokitiedot viimeisimmistä järjestelmässä suoritetuista tehtävistä. Viimeisimmän toteumatiedon tuonti –tehtävän lokitiedossa on merkintä, jonka mukaan tehtävän suorituksen aikana on tapahtunut virhe, ja tehtävän suoritus on keskeytynyt. 3. Ylläpitäjä lukee lokitiedoista, mikä on mennyt vikaan. Lokitieto on riittävän yksityiskohtaista, jotta virheen korjaaminen on mahdollista. 4. Ylläpitäjä korjaa virheen muokkaamalla csv-tiedostoa, josta toteumatiedot luetaan. 5. Ylläpitäjä siirtää korjatun tiedoston palvelimelle. 6. Ylläpitäjä siirtyy ajoitettujen tehtävien ylläpito -näkymään ylläpitovalikon kautta. 7. Ylläpitäjä etsii listasta toteumatiedon tuonti-tehtävän ja painaa sen kohdalla ”Ajasta” nappia. 8. Käyttöliittymästä aukeaa lomake, johon käyttäjä asettaa seuraavan suorituksen ajankohdan. 9. Ylläpitäjä asettaa seuraavan suorituksen ajankohdaksi

	tämän ajanhetken, jolloin suoritus käynnistyy heti. 10. Tehtävälistaan ilmestyy toteumatiedon tuonti-tehtävän kohdalle merkintä ”suoritus käynnissä”. 11. Ylläpitäjä odottaa, kunnes suoritus on valmis ja käy varmistamassa loki-näkymästä että suoritus on ajettu onnistuneesti läpi.
Poikkeukset	
Käyttötaajuus	Viikoittain tai kuukausittain
Tärkeys	

Liite 2 Toiminnalliset vaatimukset

Vaatimukset on esitetty seuraavaa rakennetta noudattaen:

ID: Ominaisuuden yksilöllinen tunniste

Vaatimus: Mikä vaatimus täyttyy

Perustelu: Miksi ominaisuus toteutetaan

Tärkeys: Kriittinen, tärkeä tai lisäominaisuus

Viittaus käyttötapaukseen: Mihin käyttötapaukseen ominaisuus liittyy

ID	Vaatimus	Perustelu	Tärkeys	Viittaus käyttötapaukseen
V1	Käyttäjä voi lisätä järjestelmään ajastettavan tehtävän.		Kriittinen	YKT1
V2	Käyttäjä voi konfiguroida järjestelmään lisättävän ajastettavan tehtävän. Konfiguraatiomahdollisuudet riippuvat tehtävän tyypistä.	Yksittäinen tehtävätyyppi voidaan suorittaa erilaisilla asetuksilla, ja käyttäjän on pystyttävä muokkaamaan näitä asetuksia.	Kriittinen	YKT1
V3	Käyttäjä voi selata järjestelmään lisättyjä tehtäviä ja niiden tietoja.	Käyttäjä hallinnoi järjestelmässä olevia ajastettavia tehtäviä.	Kriittinen	AYKT1, PYKT1
V4	Asiakastilin ylläpitäjän ominaisuudessa toimiva käyttäjä näkee ainoastaan oman yrityksensä ajastettavat tehtävät.	Asiakastilin ylläpitäjällä on oikeus suorittaa ainoastaan edustamansa yritykseen kohdistuvia tehtäviä.	Kriittinen	AYKT1, PYKT1
V5	Palvelun ylläpitäjän ominaisuudessa toimiva käyttäjä näkee kaikki järjestelmän ajastettavat tehtävät.	Palvelun ylläpitäjä voi hallinnoida kaikkia järjestelmän tehtäviä.	Tärkeä	PYKT1
V6	Käyttäjä voi poistaa ajastettavan tehtävän.	Kertasuoritteisia tehtäviä ei tarvitse säilyttää järjestelmässä. Jos tehtävälle on annettu väärät asetukset, tehtävä joudutaan poistamaan ja luomaan uusi tehtävä oikeilla asetuksilla.	Tärkeä	YKT2
V7	Käyttäjä voi keskeyttää tehtävän ajastuksen.	Jos esimerkiksi tehtävän vaatimaa tietoa ei ole saatavilla, tehtävää ei kannata suorittaa, vaan sitä	Tärkeä	YKT3

		on lykättävä, kunnes tiedot on saatavilla.		
V8	Käyttäjä voi uudelleen käynnistää tehtävän ajastuksen.	Virheeseen keskeytynyt tehtävän suoritus pitää voida suoritaa käyttäjän toimesta heti, kun virhe on korjattu.	Tärkeä	YKT4
V9	Käyttäjä näkee ajastusmoduulissa tehdyt toimenpiteet.	Varsinkin palvelun ylläpitäjällä pitää olla mahdollisuus seurata moduulissa tapahtuneita tehtävien lisäyksiä, poistoja, ajastuksia jne.	Tärkeä	AYKT2, PYKT2
V10	Käyttäjä näkee tehtävän suorituksen aikana tehdyt toimenpiteet (loki).	Virhetilanteessa suorituksen aikana tehdyt toimenpiteet on hyvä nähdä, jotta voidaan selvittää, mistä virhe johtuu. Onnistuneen suorituksen yhteydessä voi olla tärkeää tietää yksityiskohtia suorituksesta.	Kriittinen	AYKT2
V11	Käyttäjä näkee suoritettujen tehtävien tiedot.	Tila, suoritusaika, kesto ja muu mielenkiintoinen informaatio pitää näyttää käyttäjälle.	Tärkeä	AYKT2
V12	Käyttäjää informoidaan sähköpostitse virheeseen keskeytyneistä tehtävistä.	Virhetilanteen sattuessa ylläpitäjän pitää saada tieto mahdollisimman pian, jotta virhetilanteesta pystytään toipumaan mahdollisimman nopeasti normaalin käytön häiriintymättä.	Lisäominaisuus	ETKT2
V13	Lokitietoja voi suodattaa tehtävätyypin ja suoritusajankohdan (palvelun ylläpitäjä myös yrityksen mukaan).	Ajan mittaan lokitietoa tulee paljon, eikä ole järkevää esittää niitä kaikkia. Käyttäjä voi valita näytettäväksi vain uusimmat tai tietyn tyyppiset tehtävät.	Lisäominaisuus	

Liite 3 Käyttöliittymämääritys

Käyttöliittymämäärityksessä on käytetty lähteenä vaatimusmäärittelyssä kerättyjä käyttötapauskuvauksia ja toiminnallisia vaatimuksia. Niiden perusteella on kirjattu ylös tarvittavat näkymät. Näkymät on numeroitu yhdestä neljään. Kunkin näkymän alle on listattu kyseisen näkymän sisältämät ominaisuudet ja toiminnallisuudet sekä tarvittavat toiminnonkäsittelijät. Lisäksi listaan on merkitty viittaus käyttötapaukseen, jossa kyseinen ominaisuus on mainittu.

1. Ajastettavan tehtävän lisääminen (YKT1)

Syöte	Nimi, kuvaus, ensimmäinen suoritus, suoritusten aikaväli. Aikavälin valinta voi olla "päivä","viikko","kuukausi" tai "muu".	YKT1,ETKT1
Syöte	Tehtävätyyppi	YKT1
Toiminto	Konfiguroi tehtävätyypin asetukset. Kaikilla tehtävätyypeillä ei ole asetuksia, joten tätä toimintoa ei aina näytetä.	YKT1,ETKT1
Siirtymä	Näkymään 2.	YKT1
Lomake	Tehtävän lisäämistietojen välittäminen toiminnonkäsittelijälle.	YKT1
Muu	Lomakkeen validointi	YKT1
Toiminto	Lisää tehtävä	YKT1
Toiminnonkäsittelijä	Käsittelee tehtävän lisäämisen	YKT1
Siirtymä	Näkymään 1.	YKT1

2 Tehtävätyypille ominaisten asetusten syöttö (YKT1)

Muu	jokaiselle tehtävätyypille on olemassa oma näkymä.	YKT1
Info	kuvaus, mitä tehtävätyyppi tekee ja mitä parametreja ottaa vastaan.	YKT1
Syöte	Tehtävätyypin asetusten syöttö.	YKT1
Lomake	Välittää tehtävätyypille ominaiset asetustiedon toiminnonkäsittelijälle.	YKT1
Muu	Lomakkeen validointi.	YKT1
Toiminto	Lisää tehtävätyypin asetukset ajastettavaan tehtävään.	YKT1
Toiminnonkäsittelijä	Tallettaa tehtävätyypin asetukset	YKT1

	ajastettavaan tehtävään.	
Siirtymä	Näkymään 1.	YKT1

3. Ajastettujen tehtävien ylläpito (AYKT1, PYKT1)

Info	Lista yrityksen ajastetuista tehtävistä.	AYKT1
Info	nimi, kuvaus, luontipäivämäärä, seuraavan suorituksen ajankohta, aikaa seuraavaan suoritukseen, suoritusten aikaväli, tila.	AYKT1
Info	Palvelun ylläpitäjä näkee yrityksen nimen, jolle ajastettu tehtävä kuuluu.	PYKT1
Toiminto	Poista ajastettu tehtävä.	YKT2
Toiminnonkäsittelijä	Poistaa ajastetun tehtävän järjestelmästä.	YKT2
Siirtymä	Poiston jälkeen takaisin näkymään 3.	YKT2
Toiminto	Keskeytä ajastettu tehtävä.	YKT3
Toiminnonkäsittelijä	Keskeyttää ajastetun tehtävän järjestelmästä.	YKT3
Siirtymä	Keskeytyksen jälkeen takaisin näkymään 3.	YKT3
Toiminto	Ajasta tehtävä uudelleen.	YKT4
Syöte	Seuraavan suoritusajankohdan syöttö.	YKT4
Lomake	Seuraavan suorituksen ajankohdan välittäminen toiminnonkäsittelijälle.	YKT4
Toiminnonkäsittelijä	Ajastaa tehtävän uudelleen.	YKT4
Siirtymä	Uudelleenajastuksen jälkeen takaisin näkymään 3.	YKT4

4. Ajastettujen tehtävien loki (AYKT2, PYKT2)

Info	Lokitiedot viimeisimmistä suoritetuista tehtävistä: Tehtävän nimi, suorituksen ajankohta, kesto ja tila.	AYKT2
Info	Lokitiedot ajastettujen tehtävien lisäyksistä, poistoista ja keskeytyksistä.	AYKT2
Info	Palvelun ylläpitäjä näkee yrityksen nimen, jolle lokitieto kuuluu.	PYKT1
Toiminto	Lokitietojen suodattaminen tehtävätyypin, suoritusajankohdan ja yrityksen mukaan.	V13

Toiminnonkäsittelijä	Suodatustoiminnon toteutus.	V13

Liite 4 Tietokannan rakenne

ScheduledTaskVO

Sarakkeen nimi	Tyyppi	Selite
Id	Varchar	Tietokannan generoima yksilöivä tunniste
Name	Varchar	Tehtävän nimi. Tehtävän luoja voi määritellä tämän vapaasti tehtävän asiayhteyteen sopivaksi.
Description	Varchar	Tehtävän kuvaus. Vapaamuotoinen lisätietokenttä.
CreationDate	Datetime	Ajankohta, jolloin tehtävä on luotu.
FirstTimeout	Datetime	Ajankohta, jolloin tehtävä ensimmäisen kerran laukeaa.
NextTimeout	Datetime	Seuraavan laukeamisen ajankohta.
RepeatInterval	Numeric	Tehtävän ajoitusten toistoväli millisekunteina.
ClassName	Varchar	Luokan nimi, jossa ajastettavan tehtävän sovelluslogiikka sijaitsee.
MethodName	Varchar	Metodin nimi, jossa ajastettavan tehtävän sovelluslogiikka sijaitsee.
Status	Varchar	Tehtävän tämänhetkinen tila.
OwnerCompanyId	Varchar	Tehtävän omistavan yrityksen id.
TimerInfo	Varbinary	Tunnistekenttä, joka sitoo tehtävän järjestelmän ajastimeen.
MethodParameters	Varbinary	Tehtävän metodille välitettävät parametrit sarjallistetaan oliona tähän sarakkeeseen.
TaskTypeIdentifier	Varchar	Tehtävätyypin tunniste.

TaskLogVO

Sarakkeen nimi	Tyyppi	Selite
Id	Varchar	Tietokannan generoima yksilöivä tunniste
Type	Varchar	Lokitetun toimenpiteen tyyppi. Tyyppi voi olla joko suoritetun tehtävän status-muutos tai tehtävän hallinnointitoimenpide.
TaskId	Varchar	Tehtävän id, jolle lokitieto kuuluu.
ExecutionStartTime	Datetime	Suorituksen alkamisajankohta.
ExecutionEndTime	Datetime	Suorituksen loppumisajankohta.

Message	Text	Tehtävän suorituksen aikana tehty lokitus tallennetaan tekstimuotoisena tähän.
Exception	varbinary	Tehtävän suorituksen keskeyttänyt poikkeus oliona.