KARELIA-AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma

Jussi-Pekka Turunen

WEB- JA MOBIILISOVELLUSTEN INTEGROINTI BPEL-PROSESSIMOOTTORIIN

Opinnäytetyö
Huhtikuu 2013

Karelia AMMATTIKORKEAKOULU	**OPINNÄYTETYÖ** **Huhtikuu 2013** **Tietotekniikan koulutusohjelma** Karjalankatu 3 80200 JOENSUU (013) 260 600

Tekijä
Jussi-Pekka Turunen

Nimeke
Web- ja mobiilisovellusten integrointi BPEL-prosessimoottoriin

Toimeksiantaja
Karelia-ammattikorkeakoulu

Tiivistelmä

Opinnäytetyön tavoitteena oli selvittää, kuinka voidaan yhdistää integroitavat sovellukset liiketoiminnallista prosessia ajavaan prosessimoottoriin ja kuinka voidaan edistää prosessista syntyvän yksittäisen instanssin suoritusta. Opinnäytetyössä esitellään ko. järjestelmän laatimiseen vaadittavat käsitteet ja teknologiat, toteutus, tutkimustyön tulokset sekä jatkokehitysmahdollisuudet.

Tutkimustyö toteutettiin laatimalla palvelukeskeisen arkkitehtuurin mukainen järjestelmä, joka yhdistää prosessimoottorin ja integroitavat sovellukset tietokantaa hallitsevaan sovellukseen. Integroitavat sovellukset liitetään palvelukerroksen kautta järjestelmään, joka toteuttaa prosessimoottoriin integroinnin sekä tietokantayhteyden.

Tutkimustyön tuloksena oli, että integroitavat sovellukset voidaan yhdistää prosessimoottoriin palvelukerroksen kautta. Nämä sovellukset toimivat aktiviteettien suorittajina. Prosessissa syntyvät instanssit yksilöidään liiketoiminnassa syntyvien dokumenttien tunnuksilla ja palvelukerros huolehtii instanssien ja aktiviteettien yhdistämisen.

Tutkimustyössä laadittu järjestelmä mahdollistaa prosessimoottorin jatkotutkimuksen sekä järjestelmän jatkokehittämisen, jotka voivat toimia opintojaksojen opetusmateriaalina. Tärkeimpiä jatkokehitysmahdollisuuksia ovat liiketoiminnallisen prosessin jalostaminen, tietovarastoinnin toteuttaminen sekä prosessin tehokkuuden seurannan toteuttaminen.

Kieli suomi	Sivuja 56 Liitteet 9 Liitesivumäärä 11

Asiasanat
prosessimoottori, palvelukeskeinen arkkitehtuuri, liiketoiminnallinen prosessi, tietovarastointi, toteutettavuustutkimus

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
April 2013
Degree Programme in
Information Technology
Karjalankatu 3
FIN 80200 JOENSUU
FINLAND

Author
Jussi-Pekka Turunen

Title
Integrating Web and Mobile Application to BPEL Process Engine

Commissioned by
Karelia University of Applied Sciences

Abstract

The goal of this thesis was to study how to integrate applications to the business process engine and how to advance the execution of a single instance of the business process. In this thesis all the necessary concepts and technologies are introduced, how these features are implemented, results of the research work and possibilities of further development.

The study was carried out by creating a service oriented system, where a business process engine and client applications are integrated to the core application with access to the database. Client applications are integrated to the system via service layer, which implements integration of the process engine and the database connection.

The results of the study were that client applications can be connected to the process engine through the service layer of the system core application. The system core application connects business process instances to business process activities. These instances, which are created at the process run time, are identified with documents used in the business process.

The system created in this study enables further development of the system and further study of the process engine. These elements can be used as teaching material. The most important possibilities of the further studies are refining of the business process, implementation of data warehousing and implementation of process efficiency monitoring.

Language	Pages 56
Finnish	Appendices 9
	Pages of Appendices 11

Keywords

business process engine, service oriented architecture, business process, data warehousing, feasibility study

Sisältö

Liitteet

1 Johdanto

Liiketoiminnallisten sovellusten on tarkoitus tukea liiketoimintaa automatisoimalla tai helpottamalla liiketoiminnan ydinprosessin eri vaiheita. Tavoitteena olisi, että liiketoiminnallisten prosessien ajaminen prosessimoottorin kautta integroiduilla sovelluksilla pakottaa sovellusten laatijan noudattamaan liiketoiminnallista prosessia ja toteuttamaan prosessien automatisointia integroitavasta sovelluksesta käsin. Tämä tarkoittaa sitä, että järjestelmä olisi laadittava standardinomaisia rajapintoja käyttäen, jolloin siihen olisi mahdollista integroida liiketoiminnallista prosessia tukevia sovelluksia. Prosesseista voi olla myös toimimassa rinnakkain useita eri instansseja, jolloin sovellusten tulisi edistää jokaisen yksilöllisen instanssin toimintaa. Kaikkiin näihin ehtoihin on olemassa joko kaupallisia valmiiksi toteutettuja tai yksittäisiä teknisiä ratkaisuja, joitten sitominen yhteen toimivaksi komponentiksi jää ohjelmoijan tai järjestelmän laatijan vastuulle.

Tänä päivänä teknologian puute ei enää estä kyseessä olevan järjestelmän laatimista. Suurin haaste on teknologioissa käytettävän tiedon löytäminen ja teknologioiden yhteen liittäminen. Opinnäytetyötä lähdettiin laatiman juuri sen takia, että löydettäisiin ratkaisuja ja esimerkkitapauksia, mitä pohjatietoa vaaditaan tällaisen järjestelmän laatimiseen ja minkälaisia teknologioita ko. järjestelmässä voitaisiin käyttää.

Opinnäytetyössä toteutetaan järjestelmä, jolla suoritetaan liiketoiminnallisia prosesseja prosessimoottorilla. Järjestelmä laaditaan palvelukeskeisen arkkitehtuurin mukaisesti ja järjestelmään integroidaan liiketoimintaa tukevia asiakassovelluksia. Näitten asiakassovellusten tehtävänä on välittää järjestelmälle liiketoiminnassa käytettävä tallennettava tieto.

Opinnäytetyön tavoitteena oli tehdä toteutettavuustutkimus jolla osoitetaan konseptin toimivuus ja toteuttamiskelpoisuus. Opinnäytetyössä selvitettiin kuinka voidaan laatia järjestelmä, johon sovellusten integrointi olisi yksinkertaista ja kuinka sovelluksilla voitaisiin käyttää mallinnettua liiketoiminnallista prosessia. Prosessista voisi syntyä myös rinnakkain toimivia instansseja, jolloin järjestelmän tulisi kyetä edistämään yksitäisen instanssin suoritusta.

Opinnäytetyössä käytettiin kaupallisia, avoimen lähdekoodin ja ilmaisjakelussa olevia sovelluksia. Pohjatietona käytettiin palvelukeskeisen arkkitehtuurin laatimiseen perehtyviä teoksia sekä web- ja mobiiliohjelmointiin liittyviä teoksia. Tutkimustyö on laadittu Karelia-ammattikorkeakoulun käyttöön. Tutkimustyössä käytetty materiaali annetaan Karelia-ammattikorkeakoulun käyttöön jatkotutkimuksia varten.

2 Liiketoiminnalliset prosessit

Yrityksen liiketoiminnalliset prosessit muodostuvat liiketoiminnan työvaiheista ja niitten järjestyksestä. Työvaiheita voidaan pitää prosessin aktiviteetteina. Aktiviteettien ja niitten suorittavien toimijoiden tehokkuus määrää prosessin tehokkuuden. Prosessien määrittely vaikuttaa prosessin tehokkuuteen. Mitä paremmin prosessi on määritelty, sitä tehokkaammin yritys voi toimia. Yrityksen liiketoiminnallisten prosessien yksityiskohtien tunteminen on tärkeää, koska siten voidaan tunnistaa liiketoiminnan pullonkaulat ja optimoida liiketoiminnallisia prosesseja. Optimoimalla prosesseja voidaan vähentää työntekijöiden työtaakkaa ja pienentää resurssien käyttöä. Prosessien optimointi parantaa prosessien tehokkuutta. Mitä tehokkaampia prosessit ovat, sitä kilpailukykyisempi yritys on. [1, s. 7–8.]

2.1 Liiketoiminnallisen prosessin havaitseminen

Yrityksen liiketoiminnalle olennaisin asia on yrityksen ydinprosessit. Yritysten liiketoimintaa voidaan yleistää jollain tavalla (esim. hankinta, markkinointi, myynti, jne.) mutta yrityksen ydinprosessit ovat hyvinkin yksilöllisiä. Tämän takia yritykset eivät voi ostaa valmista tuotetta, vaan yritysten on laadittava sovellus yksilöllisiin tarpeisiin. Yritys, joka voi yleistää omat liiketoiminnalliset prosessit, ei ole keskiverto yritystä parempi. Keskivertoa paremmat yritykset haluavat pitää prosessit salassa, koska prosessit ja niitten tehokkuus on yritykselle etu. Yrityksen liiketoiminnalliset prosessit ovat yksilöllisiä yritykselle ja niitten prosessit ovat yleensä monimutkaisia. [1, s. 8–10.]

Yrityksen liiketoimintaa tunnistaessa kannattaa keskittyä yrityksen ydintoimintoihin. Ydinprosessit muodostavat yrityksen kannalta oleellisen liiketoiminnan ja automatisoinnin tehtävänä on helpottaa näitten prosessien toimintaa. Jotta prosessit voidaan tunnistaa, on syytä tietää miten prosessit yleensä kehittyvät.

Prosessit syntyvät liiketoiminnan kehityksen myötä. Prosessit kehittyvät työntekijän tehtävien myötä, joista määräytyy prosessin tehtävät ja työjärjestys. [1, s. 11.]

Tätä asiaa voitaisiin ajatella seuraavan esimerkin kautta:

”Aloittelevassa urakointiyrityksessä on ollut tapana, että asiakas on soittanut yrityksen johdolle ja tehnyt työtilauksen. Työntekijälle on ilmoitettu työtilauksesta ja työntekijä hakee työtilaukselle tarvittavat tarvikkeet. Työntekijä on suorittanut työtilauksen ja laskuttanut asiakasta. Työntekijä on toimittanut raportin yritykselle ja pyytää seuraavaksi uutta työtä.”

Edellä mainitussa esimerkissä työpaikan työntekijä on muodostanut prosessin. Työpaikan yleiset prosessit ovat entisestään tuttuja (markkinointi ja myynti), mutta emme voi yleistää työntekijän työvaiheita, koska ne ovat yritykselle yksilöllinen tapa suorittaa liiketoimi loppuun. Esimerkin työntekijä muodostaa prosessin työvaiheet ja työjärjestyksen. Näin yksinkertaisessa esimerkissä voitai-

siin yleistää prosessi jollain tasolla, mutta yrityksen kasvaessa myös prosessit kasvavat ja prosesseihin liittyy useampi työntekijä ja useampi vaikuttaja. Erityisesti teollisessa liiketoimessa prosessit voivat olla hyvinkin erilaisia eri yritysten välillä. Tämän takia prosesseista kehittyy yritykselle ominaisia ja yksilöllisiä tapoja hoitaa liiketoimi. Nämä ydinprosessit pitäisi tiedostaa, että prosessien mallintaminen olisi mahdollista.

2.2 Prosessin mallintamisen tavoitteet

Prosessin mallintamisella pystytään havainnollistamaan seuraavat asiat prosesseista:

- prosessikartta, josta ilmenee kuinka eri prosessit keskustelevat toistensa kanssa ja miten eri prosessit liittyvät toisiinsa
- roolien ja niitten välisten suhteiden kartta, joka näyttää mitkä roolit liittyvät prosesseihin ja mitkä ovat eri roolien väliset suhteet
- prosessimalli, joka kuvaa jokaisen yksilöllisen olemassa olevan liiketoiminnallisen prosessin. Mallissa näkyy prosessin kulku, sen aktiviteetit ja siihen liittyvät roolit. [1. s. 71.]

3 BPMN

BPMN (Business Process Model Notation) on visuaalinen ilmentymä liiketoiminnallisesta prosessikuvauksesta. Tärkein BPMN:n ominaisuus on se, että se on standardoitu ja Object Management Group (OMG) ylläpitää ja kehittää BPMN-kuvausta. Tämä tarkoittaa sitä, että BPMN-kuvausta ei omista mikään yksittäinen toimittaja tai valmistaja, mikä tekee siitä riippumattoman standardin. [3, s. 3.]

Standardoinnin ansiosta käyttäjätkään eivät ole työkaluriippuvaisia, koska notaatio ei ole toimittajariippuvainen. Henkilö, joka tekee mallinnuksen työkalulla X, voi lukea mallinnuksen joka on tehty työkalulla Y.

Standardoinnin lisäksi toinen suuri asia minkä BPMN tuo, on silta kaupallisen ja teknisen henkilöstön välille. Koska notaatio on standardoitu, mahdollistaa tämä yhteisen kommunikaation kaupallisen ja teknisen henkilöstön kesken. Henkilö, joka tuntee liiketoiminnan yrityksessä voi simuloida siitä prosessin BPMN-notaatiolla, jonka it-alan henkilö automatisoi prosessikoneeksi tai palveluksi.

BPMN-notaatio on myös tällä hetkellä kehitetty versioon 2.0, joka laajentaa liiketoiminnan mallintamisen (M) ajettavaksi kieleksi (L). Näitä työkaluja kutsutaan BPM Suiteksi tai BPML-työkaluiksi (BPMNS ja BPML tarkoittavat suoritettavaa kuvausta). Näitten työkalujen tarkoitus on laajentaa Taso:n 2 kuvaus suoritettavaksi yksityiskohtaiseksi kuvaukseksi.

Kuvaava BPMN on prosessin mallintamista graafisesti ja suoritettava BPMN on datan ja teknisten toiminnallisuuksien kartoittamista.

4 BPEL

BPEL (Business Process Execution Language, liiketoiminnallisen prosessin toteuttava kieli) on kieli, jonka tarkoituksena on määritellä liiketoiminnallinen prosessi. BPEL käyttää XML-pohjaista kieltä liiketoiminnallisien prosessien määrittelemiseen ja kuvaamiseen. Kielellä ei kuvata itse prosessia, vaan kielen avulla ajetaan mallinnettu prosessi. Kielellä koostetaan prosessikuvaus standardoiduksi rajapinnaksi, jota käytetään yritysten liiketoiminnassa laajentamaan järjestelmän integroitavuutta. BPEL tarjoaa helpon ja tehokkaan tavan liiketoiminnan yhteistyökumppaneille toteuttaa oma integraatio järjestelmään. Integrointia helpottaa BPEL-kielen standardointi. [4, s. 38–40.]

BPEL-kielen standardoitu WS-BPEL (Web Service Business Process Execution Language) versio on OASIS (Organization for the Advancement of Structured Information Standards) -yhteisön ylläpitämä standardi. Standardoitu WS-BPEL-kieli pohjautuu Web Service -palveluihin.

5 Apache ODE

Apache ODE (Orchestration Director Engine) on BPEL-kieltä suorittava moottori, joka on laadittu WS-BPEL-standardin mukaisesti. Moottori pystyy kommunikoimaan Web Service -palvelujen kanssa, lähettämään ja vastaanottamaan viestejä sekä käsittelemään prosessissa käytettävää dataa ja huolehtimaan poikkeustilanteista.

Apache ODE BPEL -moottori muuntaa BPMN-kuvauksen toiminnalliseksi kerrokseksi, josta syntyy järjestelmän BPEL-kerros. BPEL-kerros sisältää teknisen toteutuksen prosessikoneesta, jonka aktiviteetteja voi laukaista SOAP-viesteillä WSDL-rajapinnan kautta. Moottorissa ylläpidetään kaikkien prosessien suoritusta.

Moottorin palveluita voi käyttää SOAP-viesteillä. Moottorin sisään rakennettuja palveluita ovat

- InstanceManagement, jolla voidaan etsiä ja tutkia prosessien instansseja.
- DeploymentManagement, jolla voidaan hallita prosessien julkaisuja.
- TaskManagement, jolla voidaan tarkastella prosessimoottorilla olevia aktiviteetteja.

6 Käyttötapaukset

Use case, eli käyttötapaus, on skenaariopohjainen kuvaus järjestelmän toimijoitten tapahtumista. Tapahtumat kuvaa toimijoitten (esim. käyttäjä tai järjestelmä) keskeisiä vuorovaikutuksia joilla on jokin päämäärä järjestelmässä. [5. s. 60]. Käyttötapaus antaa kuvauksen mitä edellytyksiä tai vaiheita käyttötapauksen suorittaminen edellyttää järjestelmältä ja miten käyttäjä ohjaa järjestelmän suoritusta.

Käyttötapauksia käytetään eri tarkoituksiin sovelluksen vaatimusten määrittelyssä, niistä yleisimpiä ovat

- liiketoiminnallisten prosessien kuvaus
- järjestelmän vaatimusten kuvaus
- järjestelmän toiminnalliset määrittelyt.

Käyttötapaukset kuvataan kaaviona tai tekstipohjaisessa muodossa. Käyttötapauskaavio toimii yleensä käyttötapausten sisällysluettelona, jonka sisällä olevat käyttötapaukset kuvaillaan tarkemmin tekstipohjaisessa muodossa. [5, s. 60]. Käyttötapauskaavio kuvaa järjestelmän sisällä olevat tapahtumat tai skenaariot ja niihin liittyvät tekijät. Kaavio ei itsessään kerro, mitä vaiheita tai ehtoja käyttötapaukseen liittyy. Käyttötapaus kuvaa vain tapaukset, jotka tapahtuvat ko. järjestelmän sisällä. Käyttötapauskaavio kuvaa hyvin yleisellä tasolla, mitkä käyttötapaukset liittyvät järjestelmään ja mitkä toimijat käynnistävät käyttötapauksen tapahtuman.

7 Robustness analysis

Robustness analysis on tekniikka, jolla voidaan yhdistää ohjelmistosuunnittelussa käyttötapaukset olioiksi. Tekniikassa on tavoitteena käyttää laadittuja

käyttötapauksia ja luoda niistä yksityiskohtainen kaavio, jossa ilmenee tiedon tai ohjauksen suunta. Robustness analysis toimii ohjelmistosuunnittelussa korkeantason määrittelyn ja sekvenssikaavioiden yhdistävänä menetelmänä. [6, s. 101 - 103.]

Analyysi pakottaa ohjelmoijan tarkastelemaan laadittavaa sovellusta enemmän abstraktilta tasolta ja tarkastelemaan vaatimuksia ennen kuin varsinaista suunnittelutyötä aletaan tehdä. Robustness-kaavio antaa kuvan sovelluksen käsitteellisestä tasosta, kun taas varsinainen ohjelmiston suunnittelu tehdään sekvenssikaaviossa. [6, s. 109.]

Kuvauksen laatiminen painottaa sovelluksen suunnittelussa käyttämään laadittuja käyttötapauksia ja löytämään mahdollisesti Domain (domain-malli toimii projektityössä projektin sanakirjana) -mallista puuttuvia luokkia. Kaaviossa ilmaistaan käyttötapauksen toimija ja siihen liittyvä skenaarion kulku. Skenaarion tapahtumien kulussa otetaan huomioon vaihtoehtoiset polut ja poikkeustilanteet. Analyysi sitoo käyttötapaukset ja ohjelmistonsuunnittelun vahvasti yhteen.

Analyysi toteutuu silloin, kun käyttötapausta työstetään kaaviota piirrettäessä. Analyysin aikana käyttötapausta luetaan ja kuvausta mallinnetaan käyttötapauksen kautta. Analyysia laadittaessa käyttötapausten skenaariot voivat muuttua, jolloin itse käyttötapausta pitää päivittää. Mikäli analyysin aikana löydetään uusia domain-luokkia, tulee myös domain-kaavio päivittää. Analyysin valmistuttua pitäisi olla mahdollista seurata kaaviota lukemalla laadittua käyttötapausta. [6, s. 114 – 115.]

8 Web-sovelluskehitys

Web-sovelluskehityksessä toteutetaan yleensä palvelinsovellus, joka tarjoaa client-sovellukselle pääsyn tietoon jonkin rajapinnan kautta tai käsittelee tiedon sopivaan näytettävään muotoon. Tänä päivänä iso osa web-sovelluksista on

nettiselaimessa pyöritettäviä sovelluksia, joissa tieto muokataan näytettävään muotoon jo palvelimella. Poikkeuksena on webselaimessa tapahtuva muotoilu (mm. JavaScript, CSS (Cascading Style Sheets)), joilla pyritään luomaan reaaliaikaisesti sisältöä web-sivuille. Reaaliaikaisuudessa on tavoitteena, että käyttäjän tarvitsisi vaihtaa sivuja saadakseen uuden näkymän web-sovelluksesta.

8.1 Palvelinohjelmointi

Palvelinohjelmointi web-sovelluskehityksessä tarkoittaa pyynnön tai viestin käsittelyä palvelimella. Palvelin käsittelee tulevan viestin ja palauttaa paluuviestinä dataa. Data voi olla jotain tietoa, mikä myöhemmin tulostetaan asiakkaan päässä client-sovelluksella. Palvelinkieliä ovat mm. PHP, Java ja C#.

Palvelimella toimiva sovellus muokkaa esim. tietokannassa olevan tiedon näytettävään muotoon client-sovellukselle. Tieto itsessään ei ole palvelimella yleensä valmiissa muodossa, että sitä voisi sellaisenaan näyttää.

8.2 Tietokannat

Tietokannat kuuluvat myös oleellisesti serveriohjelmointiin. Tietokantoja käytetään web-sovelluskehityksessä ylläpitämään tietoa ja palauttamaan tietoa hakujen perusteella. Yleisin käytetty tietokantamalli on relaatiotietokanta. Relaatiotietokannassa tieto koostuu tauluista, taulujen ominaisuuksista ja niitten välisistä yhteyksistä. Jokainen taulu pitää sisällään rivejä (esim. taulukko 1), joka myöhemmin muodostetaan sarakkeiksi (esim. taulukko 2). Järjestelmää, joka ylläpitää ja järjestelee tietokannan tietoja, kutsutaan nimellä DBMS (Database Management System). Opinnäytetyössä käytettävän tietokannan hallintasovellusta, joka ylläpitää relaatiotietokannan tietoja, kutsutaan nimellä RDBMS (Relational Database Management System). Tietokanta ja tietokannan hallintajärjestelmä sekoitetaan termeinä usein keskenään. On syytä muistaa, että tietokannalla voidaan tarkoittaa mitä tahansa järjestelmää, joka sisältää tietoa. Tietokannan hallintaohjelmalla tarkoitetaan sovellusta, joka käsittelee tietoa.

Taulukko 1. Relaatiotietokannan taulu.

Auto	
integer	ID
string	Valmistaja
string	Malli
integer	Vuosimalli

Taulukko 2. Relaatiotietokannan tiedot.

Autot taulukko			
ID	Valmistaja	Malli	Vuosimalli
1	Toyota	Corolla	1984
2	Ford	Fiesta	1991

Yllä olevassa esimerkissä on relaatiotietokannan taulu (taulukko 1), joka määrittelee, missä muodossa tiedon pitää olla tietokannassa. Rivi määrittelee tiedon tietotyypin sekä tiedon nimen. Taulukossa 2 on relaatiotietokannan taulu, johon on lisätty kaksi riviä. Taulukossa 1 rivit määrittelevät tiedon muodon ja taulukossa 2 sarakkeet määrittelevät, mitä tietoa taulu pitää sisällään.

8.3 Client-ohjelmointi

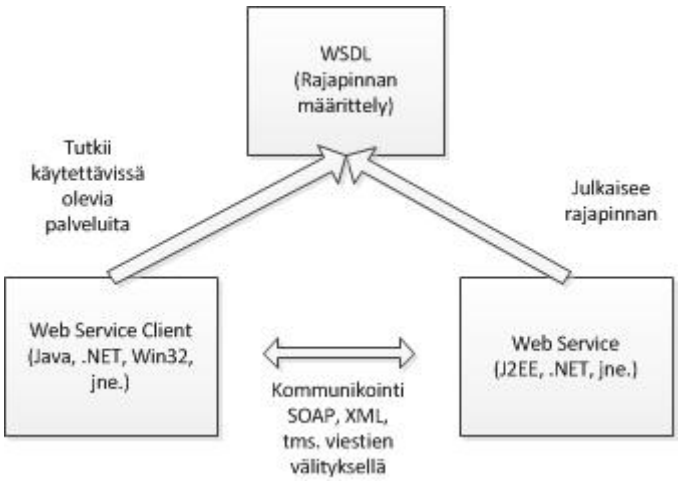
Client-sovelluksella mahdollistetaan käyttäjän pääsy palvelimella sijaitsevaan tietoon. Client-sovellus, joka käyttää palvelimen tietokantaa tai palvelimella sijaitsevia toiminnallisuuksia, voidaan tuottaa melkein millä tahansa modernilla ohjelmointikielellä (C, C++, Java, C# ja jne.). Tällä hetkellä yksi käytetyimmistä client-sovelluksista on web-selain. Client-sovelluksien luokituksessa käytetään ”fat”, ”hybrid” ja ”thin” nimikkeitä, joilla kuvataan sovelluksen tyyppiä. Esim. web-selainta kutsutaan ”thin client” nimikkeellä, koska sovellus itsessään ei säilytä tai käsittele tietoa millään tavalla. Web-selain ainoastaan tulostaa datan luettavaan muotoon, minkä palvelin tarjoaa käyttäjälle. Client-sovellus yhdistetään yleensä palvelimelle TCP (Transmission Control Protocol) tai UDP (User Datagram Protocol) -protokollan välityksellä.

9 Web Services

Web Service on menetelmä, jolla välitetään tietoa Web-sovellusten välillä. Web Servicen ajatuksena on tehostaa sovellusten välistä kommunikointia, joissa kommunikointi tapahtuu palvelimen ja asiakkaan välillä. Web Service välittää tiedon XML (eXtensible Markup Language) tai JSON (JavaScript Object Notation) -muodossa. [7, s. 1–3.]

Web Service takaa yhteensopivuuden alustojen ja teknologioiden välillä jotka käyttävät standardoituja teknologioita kuten XML, SOAP ja HTTP.

Useat eri tahot (W3C, Microsoft, jne.) ovat määritelleet Web Servicejä hieman eri tavalla mutta keskeisin ajatus kaikilla on, että Web Service tarjoaa rajapinnan sovelluksen logiikkaan. Rajapinnan pitäisi olla myös standardin omainen, että liittyminen sovellukseen olisi käyttäjälle mahdollisimman vaivatonta ja liittyminen itsessään toimisi sovitulla tavalla.



Kuva 1. Yleiskuva Web Servicestä.

Seuraavissa luvuissa käydään läpi määritteitä ja käsitteitä, jotka liittyvät olennaisesti Web Service -palveluihin. Näitten käsitteitten riippuvaisuudet on esitetty kuvassa 1.

9.1 XML

XML on tiedostomuoto, jossa tieto voidaan kuvailla tiedoston sisällä (kuva 2). XML-tiedostossa tieto määritellään merkeillä, kuten HTML-tiedostossa, joista myös molempien tiedoston ML (Markup Language) päätteet tulevat. Kyseessä on siis Markup Language -tiedosto. Vaikka molemmat ovat ML-tiedostoja, on niissä kuitenkin muutama merkittävä ero:

- HTML-tiedostossa merkeillä määritellään kuinka tieto (yleensä teksti tai numerot) tulostetaan. XML-tiedostossa merkit viittaavat siihen, kuinka tieto muodostetaan XML-tiedostossa ja mitä tietotyyppejä tieto itsessään sisältää. XML tukee kaikkia yleisimpiä tietotyyppejä kuten kokonaisluku, boolean, merkkijono jne.
- XML-tiedostossa merkkejä voidaan määritellä itse ja luoda XML-tiedostosta sovelluskohtainen. HTML-tiedostossa merkit ovat standardoituja ja ennalta määriteltyjä, jolloin sovelluskohtaisia merkkejä ei voida muodostaa. Tässä suhteessa XML-tiedosto on joustavampi ja näin ollen käytettävämpi tiedonsiirrossa kuin HTML-tiedosto. Vaikka XML-tiedosto olisikin sovelluskohtainen, voidaan sitä silti käyttää luotettavana tiedonsiirtomuotona, koska XML-tiedosto pitää verifioida XML-skeemalla ennen tiedoston siirtämistä palvelulta toiselle.

XML-tiedosto on Web Servicen perusta. XML-tiedosto määrittelee Web Servicen kuvauksen, tiedon varastoinnin ja muodon miten tieto välitetään [7, s. 47].

```
<?xml version="1.0"?>

<note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Kuva 2. XML-viesti.

XML-tiedostolla voidaan myös verifioida, että itse tiedonsiirrolle muodostettu XML-tiedosto on oikean muotoinen Web Servicelle. Tällöin kyseessä on XML-skeema. XML-skeemaa voidaan pitää eräänlaisena metatietotiedostona XML-tiedostolle.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Kuva 3. Kuva XML skeemasta.

Kuten kuvasta 3 nähdään, XML-skeema määrittelee mitä tietoa elementin täytyy pitää sisällään ja mitä elementtejä on sallittu käyttää XML-viestissä.

9.2 WSDL

WSDL (Web Services Description Language) on tiedosto, joka koostetaan standardoidulla tavalla, jotta palvelun käyttäjän ei tarvitse tehdä aina palvelukohtaista tiedonsiirtoratkaisua [7, s. 82]. Tällä tavalla voidaan varmistaa, että eri palvelimella käytettävä Web Service -palvelu noudattaa standardia rajapinnan julkaisumenetelmää, eikä rajapinnan julkaisu muutu eri palvelimien välillä. WSDL määritellään XML-tiedostolla, joka kuvaa miten Web Servicen palveluihin voidaan liittyä ja mitä viestimuotoa Web Service käyttää. Web Servicessä käytettäviä viestejä kuvataan XML-skeemalla, joka kertoo välitettävän viestin sisällöstä.

9.3 SOAP

SOAP (The Simple Object Access Protocol) on protokolla, jonka avulla välitetään XML-viestejä HTTP-protokollan välityksellä [7, s. 111-112]. SOAP välittää XML-dokumentin netin ylitse palvelimella sijaitsevaan Web Service -palveluun. Tämä toiminto laukaisee Web Service -palvelun kohde palvelimella. SOAP-viestin muoto on ennalta määrätty, jotta palvelin voi tulkita SOAP-viestin oikein. SOAP-viestit pitävät sisällään tiedon mitkä palvelut käynnistetään Web Servicellä ja mitä tietoa palveluihin viedään.

9.4 REST

REST (REpresentational State Transfer) rajapinnan pääajatuksena on, että REST on tilaton asiakas-palvelin järjestelmä, eli REST:iä toteuttava järjestelmä ei pidä istuntotietoja yllä (esim- käyttäjätunnuksia tms.). Jokaisella REST:n kautta käsiteltävällä resurssilla pitäisi olla yksilöivä osoite. [5, s. 7–8.]

REST-rajapintaa käytetään HTTP-pyynnöillä. HTTP-pyynnön viestimuoto määrittele, millainen palvelu HTTP-pyynnöllä halutaan suorittaa. REST-rajapinta tarjoaa seuraavien HTTP-pyyntöjen mukaiset palvelut:

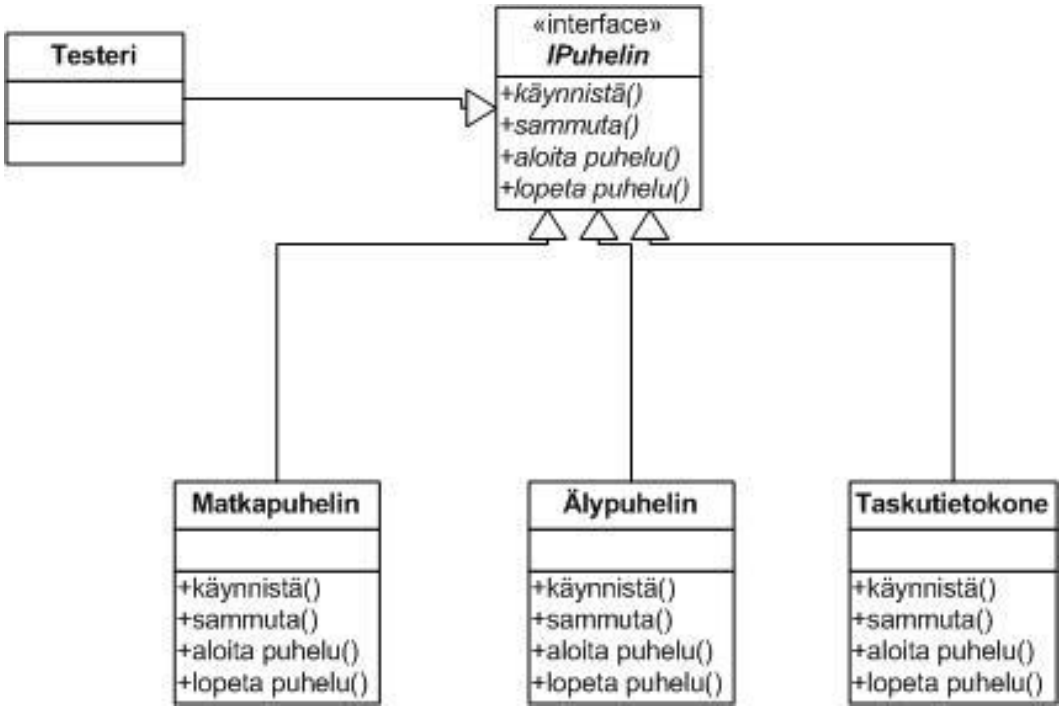
- GET, tietokannasta halutaan noutaa jotain
- POST, tietokantaan halutaan laittaa jotain
- DELETE, tietokannasta halutaan poistaa jotain
- PUT, tietokannassa halutaan päivittää jotain.

Rajapinta palauttaa tiedon HTTP-vastauksena, joka voi sisältää tietoa XML- tai JSON-muodossa.

10 Palvelukeskeinen arkkitehtuuri

SOA (Service Oriented Architecture) on sovelluskehityksen arkkitehtuuri, jonka periaatteena on rakentaa sovellus toisistaan riippumattomista palveluista. Nämä palvelut tulisi luoda siten, että ne suorittavat tietyn osan liiketoiminnallisesta prosessista, joita voitaisiin myös käyttää myöhemmin uudelleen. Yleensä näitten palveluitten rajapinnat ovat standardoituja, mitkä mahdollistavat korkeatasoisen sovelluksen integroinnin palveluihin.

Palvelut ovat toisistaan riippumattomia toiminnallisuuksia, jotka eivät sisällä kutsuja tai riippuvuuksia muihin toiminnallisuuksiin. Tällaista toiminnallisuuksien muodostamista kutsutaan nimellä ”loosely coupled” (kuva 4), joka viittaa ohjelmointityyliin, jossa toimintoja tai funktioita kutsutaan rajapinnan kautta.



Kuva 4. UML-kuvaus loosely coupled -arkkitehtuurista.

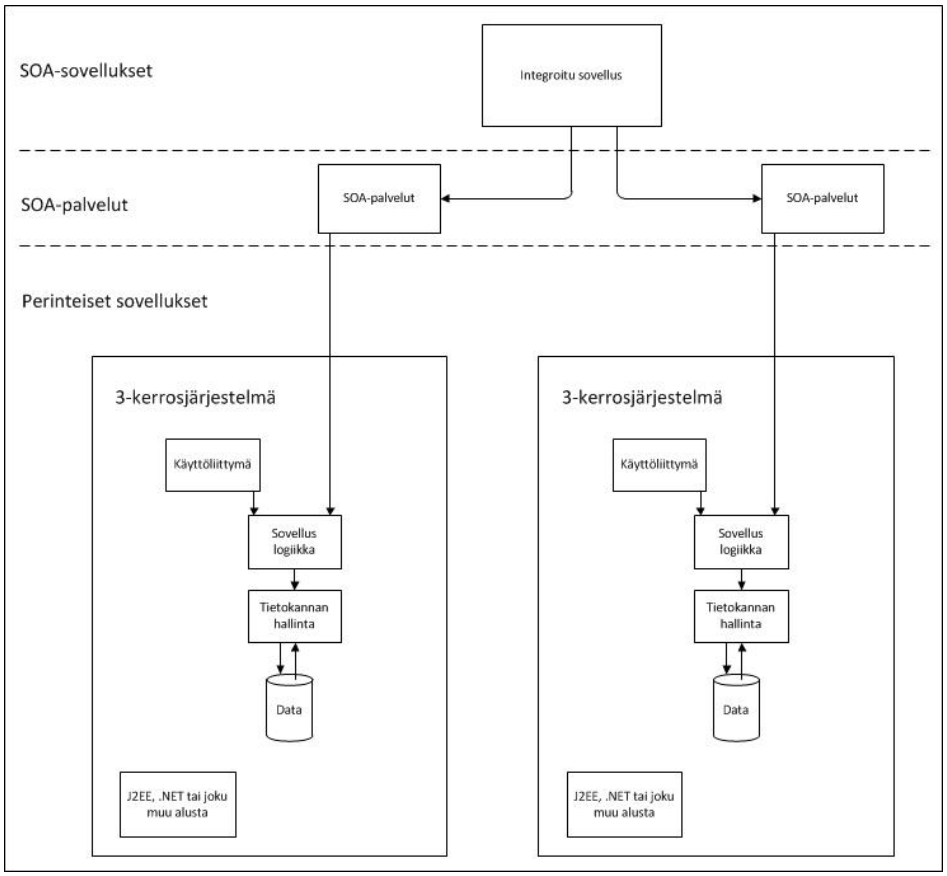
Konkreettiset luokat implementoivat suoraan rajapintaluokan toiminnallisuudet, mutta luokalla Testeri ei ole suoraa yhteyttä näihin luokkiin. Vaikka jälkeenpäin pitäisi lisätä uusi konkreettinen luokka, ei se aiheuttaisi muutosta luokkaan Tes-

teri, koska luokalla ei ole mitään suoranaista yhteyttä toiminnot toteuttaviin konkreettisiin luokkiin.

Vaikka sovellus voitaisiin kehittää ilman standardoituja palveluita ja ohjelmoida sovellus siten, että eri moduulit ovat toisistaan tietoisia, on SOA-mallissa kuitenkin muutama suuri etu lineaariseen ohjelmointiin nähden:

- Palvelut takaavat korkeatasoisen abstrahoinnin, joka helpottaa laajojen järjestelmien ja sovelluksien organisointia [9, s. 78 - 79].
- Rajapintojen standardointi takaa sovelluksen yhteensopivuuden eri ohjelmoijien kesken [9, s. 78 - 79].
- Standardit mahdollistavat järjestelmän hallinnollisen työkalujen kehittämisen, jotka ovat yhteensopivia järjestelmän kanssa. Työkalujen toimittajat voivat varmistaa komponenttien yhteensopivuuden, joka helpottaa sovelluksen kehittäjän työmäärää [9, s. 78 - 79].

Kuvassa 5 on esimerkki siitä, miten arkkitehtuurin voisi muodostaa. Arkkitehtuuria laadittaessa, tavoitteena olisi luoda erilaisia palveluita, joita käyttämällä saataisiin koostettua ylemmän tason sovelluksia, joita käytettäisiin käyttöliittymän kautta. Kuvassa 5 perinteiset sovellukset hoitavat esim. tietokannan käsittelyn, joitten logiikkaa ajetaan jonkin julkaistun rajapinnan kautta. Tämä rajapinta voisi olla esim. Web Service tai REST. Nämä palvelut muodostavat karkeammin jyvitetyn logiikan, joita hyödyntämällä laaditaan käyttöliittymän tarjoava sovellus.



Kuva 5. Palvelukeskeinen arkkitehtuuri.

11 Tutkimustehtävät

Opinnäytetyössä oli tavoitteena rakentaa järjestelmä, joka käyttää WS-BPEL-prosessimoottoria liiketoiminnallisen prosessin suorittamiseen. Järjestelmän pitää kyetä edistämään prosessista syntyvien instanssien muodostumista, sekä tallentamaan liiketoiminnan kannalta oleelliset dokumentit. Instanssien suoritusta ohjataan järjestelmään liitettävillä client-sovelluksilla.

Järjestelmän pohjakerroksena toimii palvelukerros, joka takaa pääsyn liiketoiminnan kannalta oleellisiin dokumentteihin. Palvelukerroksen rajapinta pitää muodostaa siten, että rajapinta on standardoitu ja erilaisten Client-sovellusten integrointi palvelukerrokseen on helppoa.

11.1 Client-sovellusten ja prosessimoottorin liittäminen palvelukerrokseen

Liiketoiminnallisen prosessien aktiviteetteja suorittavat prosessissa mallinnetut roolit. Nämä roolit käyttävät erilaisia liiketoiminnallisia sovelluksia aktiviteettien suorittamiseen. Tutkimustyön tehtävänä on ratkaista, miten voidaan liittää client-sovellukset prosessimoottoriin ja kuinka eri roolit voivat suorittaa prosessin aktiviteetteja client-sovellusten kautta.

Prosessimoottorin tulee edistää prosessissa syntyvien instanssien suoritusta. Instanssit yksilöidään järjestelmässä jollain liiketoiminnan kannalta oleellisella dokumentilla. Tavoitteena on selvittää, miten voidaan yksilöidä prosessissa syntyvät instanssit ja kuinka instanssit voidaan liittää siihen kuuluvan aktiviteetin suorittamiseen.

11.2 Aktiviteettien suorittaminen client-sovelluksen kautta

Client-sovellusten tulee aktivoida prosessissa syntyviä aktiviteetteja. Client-sovellusten kautta ohjataan aktiviteetteja, jotka vaativat liiketoiminnassa esiintyvien roolien suoritusta. Tavoitteena on tutkia kuinka client-sovelluksella voidaan aktivoida halutun instanssin aktiviteetti.

12 Liiketoiminnallisen prosessin suunnittelu ja mallintaminen

12.1 Suunnittelu

Suunnitteluvaiheessa hahmottelin ensimmäiseksi itselleni listan, jossa kerroin muutamalla lauseella mallinnettavassa liiketoiminnassa esiintyvät korkean tason vaatimukset. Näitä vaatimuksia olivat seuraavat:

- Asiakkaan pitää pystyä luomaan työtilaus nettisivujen kautta.
- Työnantajan pitää pystyä tarkastelemaan luotuja työtilauksia ja luomaan työtilauksesta työtehtävä.
- Työntekijän pitää pystyä näkemään hänelle tarkoitetut työtehtävät ja luomaan suoritetusta työtehtävästä loppuraportti.

Esiintyvät vaatimukset mallinnettiin käyttötapauksiksi (liite 3). Käyttötapausten mallinnuksesta muodostui domain-malli (liite 2), joka toimi projektin sanakirjana. Domain-mallia päivitettiin, kun käyttötapaukset muodostettiin robustness-kaavioiksi (liitteet 4 – 8).

Suunnittelu laadittiin Microsoft Visiolla ja siihen ladattavalla UML 2.2 stencil-paketilla. Paketti on ladattu Hrubyn verkkosivulta (2010) lähteestä [14]. Paketti sisältää UML version 2.2 standardoidut piirtomerkit, jossa on myös robustness analysis -piirtomerkit, joita Microsoft Visio ei sisällä vakiona.

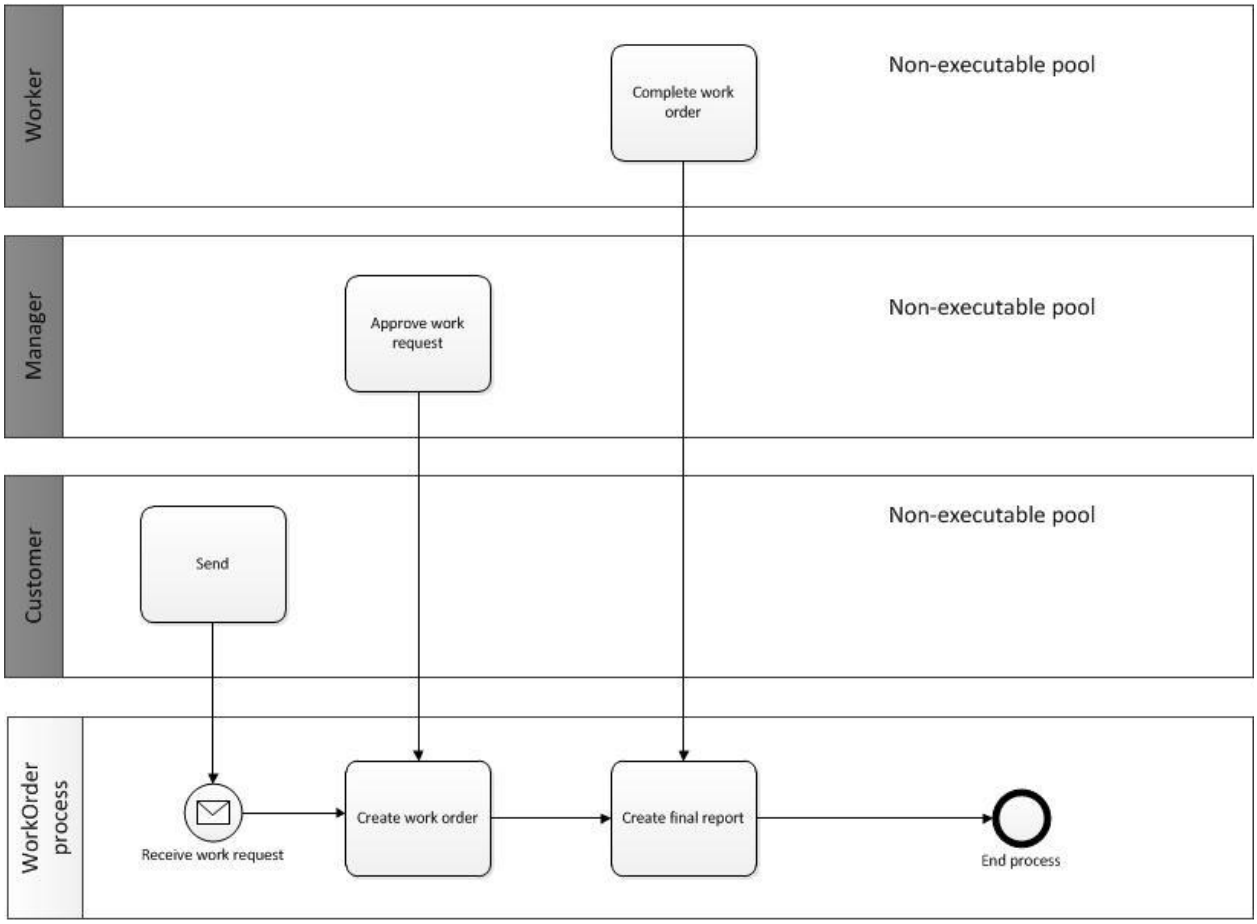
12.2 Liiketoiminnallisen prosessin mallintaminen ja BPMN-kuvauksen laatiminen

Seuraava työvaihe oli luoda prosessikuvaus mallinnettavasta liiketoiminnasta. Liiketoiminnallisen prosessin luomiseen on käytetty Intalio Designer-sovellusta ja Intalion BPMS (Business Process Management Server, prosessimoottorin palvelin) -palvelinta. Liiketoiminnallinen prosessi mallinnetaan graafisesti Designer -sovelluksella, jonka jälkeen prosessi julkaistaan BPMS-palvelimelle.

BPMN-kuvaus on mallinnettu suunnitteluvaiheessa laadituista käyttötapauksista. Käyttötapauksissa oli kolme selkeää työvaihetta, jotka oli helppo muuttaa prosessiksi. Mallinnettava prosessi koostui seuraavista vaiheesta:

1. Asiakas lähettää työtilauksen yritykselle. Tämä aloittaa uuden prosessin.
2. Yrityksen esimies käsittelee työtilauksen ja luo siitä työtehtävän.

3. Yrityksen työntekijä vastaanottaa työtehtävän. Kun työtehtävä on valmis, työntekijä laatii loppuraportin. Loppuraportin lähettäminen päättää prosessin.



Kuva 6. Mallinnettu prosessi.

Kuvassa 6 näkyvässä prosessissa tumman harmaalla merkityt radat ovat ns. black box -ratoja, joiden toimintaa ei tiedetä tai sitä ei voida määritellä. Ei-suoritettavista radoista tulee viestejä, jotka ohjaavat prosessin kulkua prosessimoottorissa. Vaalean harmaana näkyvä rata on prosessi, josta muodostetaan myöhemmin BPEL-kerros. Tämä kerros kuvaa yrityksessä tapahtuvaa liiketoiminnallista prosessia, jonka suorittamiseen vaadittavat aktiviteetit ja tapahtumat tiedetään.

Mallinnettu prosessi on ns. happy path -prosessi, jossa oletetaan, että kaikki menee onnistuneesti alusta loppuun. Tällaista prosessia ei todellisuudessa ole. Prosessissa voisi ilmentyä esim. seuraavia ongelmia, jotka pitäisi ottaa huomioon:

- Esimies ei hyväksy työtilausta.
- Asiakas haluaisi mahdollisesti perua työtilauksen, ennen kuin työtilaus on hyväksytty työtehtäväksi.
- Työntekijä ei viekään työtehtävää loppuun, vaan delegoi työtehtävän muille.

Kaikkien ongelmien vaihtoehtoiset radat pitäisi myös mallintaa prosessikuvaukseen. Tässä prosessimallinnuksessa näitä tekijöitä ei ole otettu huomioon, koska se ei toisi juuri lisäarvoa itse tutkimusongelman selvittämiselle.

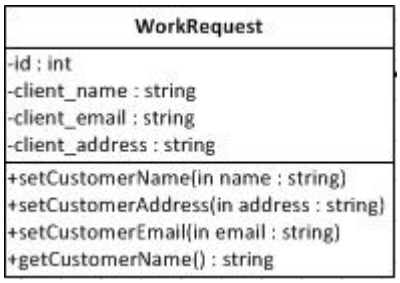
12.3 Prosessien instanssit

Mallinnettavassa prosessissa oli tarkoitus tuoda esille tilanne, jossa prosessista voi muodostua useita eri instansseja toimimaan rinnakkain. Mallinnetussa tapauksessa oletetaan, että yrityksessä on useampi työntekijä ja mahdollisesti myös x-määrä instansseja käynnissä ko. prosessista. Prosessista syntyy aina uusi instanssi, kun asiakas käynnistää uuden tapahtumaketjun. Tämä tapahtuma käynnistää uuden instanssin, joka pitäisi saada yksilöityä, että voitaisiin edistää halutun instanssin suoritusta ja mahdollisesti tarkkailla jokaisen instanssin suoritusaikoja.

Instanssien yksilöimiseen on olemassa Apachen ODE BPEL-moottorissa kaksi erilaista tapaa.

12.3.1 Eksplisiittinen korrelaatio

Eksplisiittisellä korrelaatiolla voidaan instanssi yksilöidä jollain liiketoiminnallisessa dokumentissa esiintyvällä tekijällä. Esim. kuvassa 7 työtilauksen yksilöivä id-tunnus voisi olla tällainen tekijä.



Kuva 7: WorkRequest-olio.

Eksplisiittistä korrelaatiota käytetään kun prosessi halutaan liittää jotain liiketoiminnan kannalta tärkeää tietoa.

12.3.2 Implisiittinen korrelaatio

Implisiittinen korrelaatio on prosessimoottorin tuottama yksilöllinen tunnus. Jokaisella prosessimoottorissa toimivassa instanssissa on oma yksilöllinen tunnus, jolla voidaan jäljittää prosessin instanssi.

Kuva 8: Instanssin tiedot.

Kuvassa 8 on prosessimoottorin muodostama instanssin tunnus kohdassa Identifier. Tällä tunnuksella voidaan jäljittää prosessissa syntyvä instanssi. Esimerkiksi instanssin tiedot voitaisiin noutaa Apache ODE -prosessimoottorilta InstanceManagement-palvelun kautta, aktivoimalla ”getInstanceInfo” palvelu.

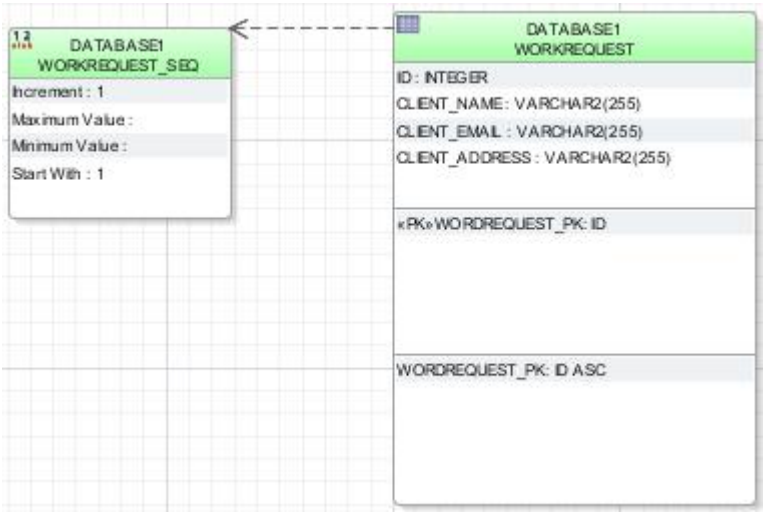
12.4 Prosessien instanssien yksilöinti Intalion BPMN-kuvauksessa

Prosessissa käytettävä tunnus määritellään BPMN-kuvausta laadittaessa. Yksilöivä tunnus liitetään viesteihin, joita välitetään kun prosessin operaatioita halutaan aktivoida. Viestin sisältö määritellään XML-skeeman avulla. Demojärjestelmässä käytin viestinä IdentifierSchema.xsd skeeman elementtiä id, jonka tyyppi on kokonaisluku

Intalio Designer sovelluksessa eksplisiittistä korrelaatiota määritellessä on tiedettävä kolme perusvaatimusta:

1. Viestin sisältö (<bpel:property>). Tämä kertoo prosessimoottorille, että viestissä välitettävä arvo on prosessin instanssin yksilöivä arvo.
2. Sisältöön viittaus (<bpel:propertyAlias>): Tällä viitataan viestin sisältöön tai viestin osaan, joka kuljettaa mukanaan yksilöivää arvoa.
3. Korrelaatiojoukko (CorrelationSets): Korrelaatiojoukolla viitataan prosessin aktiviteetin, jolla yhdistetään aktiviteettiin tuleva viesti ja prosessin instanssin tunnus.

Mallinnetussa järjestelmässä prosessien instanssit on yksilöity eksplisiittisellä korrelaatiolla. Eksplisiittisessä korrelaatiossa käytetään työtilauksen yksilöivää tunnusta. Työtilauksen yksilöivä tunnus on kuvassa 9 olevan Workrequest-taulun id-tunnus.

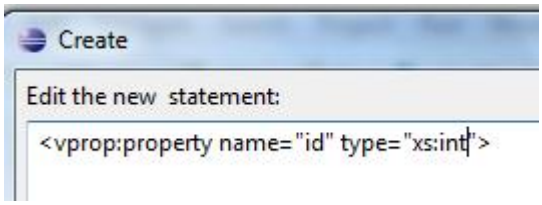


Kuva 9: Workrequest taulu.

12.5 Korrelaation muodostaminen Intalio Designerilla

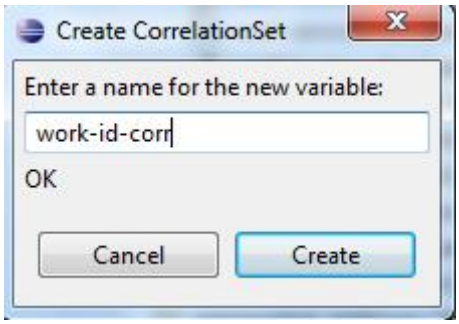
Koska korrelaation muodostaminen oli haasteellista Intalio Designer sovelluksessa, on syytä käydä korrelaation muodostaminen yksityiskohtaisemmin läpi.

Intalio Designer sovelluksessa <bpel:property> pitää määritellä Data Editor -sivulla. Sivulla luodaan uusi <bpel:property>-tyyppinen muuttuja, jolla määritellään instanssin yksilöivä tyyppi ja nimi (kuva 10).



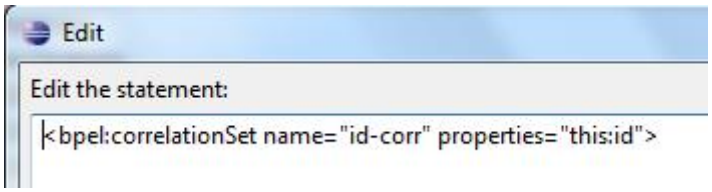
Kuva 10. Create bpel-property -ikkuna.

Tämän jälkeen määritellään prosessissa käytettävä CorrelationSet. CorrelationSet luodaan painamalla hiiren oikealla declarations-lehteä ja valitsemalla Create CorrelationSet (kuva 11).



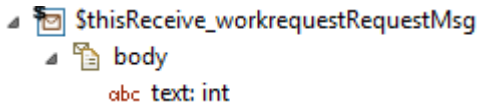
Kuva 11. Create CorrelatioSet -ikkuna.

Korrelaation arvoja pitää muuttaa siten, että korrelaation properties-muuttuja vastaa prosessissa käytettävän bpel:property-muuttujan nimeä (kuva 12).



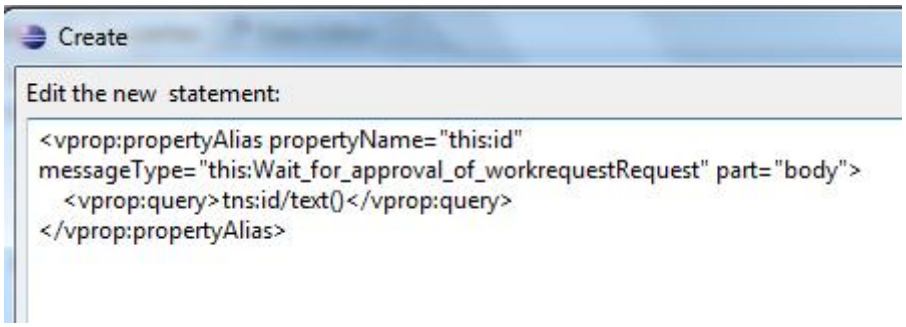
Kuva 12. Edit correlation set -ikkuna.

Kun korrelaatiojoukko on saatu määriteltyä, pitää prosessissa käytettävien viestin kentät yhdistää instanssin yksilöivään tunnukseen. Tämä tapahtuu painamalla hiiren oikealla napilla kuvassa 13 näkyvää viestin sisällön muuttujaa. Tässä tapauksessa text:int-muuttujaan (kuva 13).



Kuva 13. Viestin sisältö.

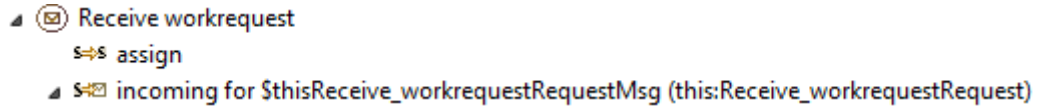
Valikosta valitaan Create <bpel:propertyAlias>), jolla voidaan luoda viittaus BPEL-tunnukseen.



Kuva 14. PropertyAliaksen muodostaminen.

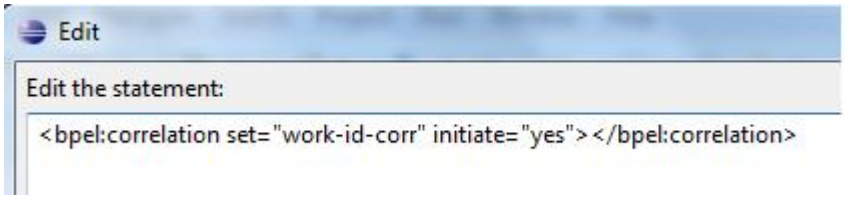
Kun luodaan viittaus tunnukseen, pitää muuttujan propertyName arvoksi muuttaa viittaus prosessi ilmentyvää tunnukseen. Tässä tapauksessa tunnus on id. ja arvo on this:id (kuva 14).

Viimeisin vaihe on linkittää korrelaatiojoukko aktiviteetin viestiin. Tämä tapahtuu siten, että korrelaatiojoukko, joka määriteltiin declarations-lehteen, raahataan aktiviteettiin saapuvan viestin päälle (kuva 15).



Kuva 15: Aktiviteetin viestin linkittäminen korrelaatiojoukkoon.

Jos toiminto onnistui, viestin alle pitäisi ilmestyä <bpel:correlationSet> muuttuja. Linkkiä pitää vielä muuttaa siten, että aktiviteetti käsittelee halutusti instanssissa olevaa korrelaatiojoukkoa.



Kuva 16. Operaation Receive work request -pyyntöviestin korrelaatio.

Kuvassa 16 näkyvässä ikkunassa kenttä initiate määrittelee, miten instanssia käsitellään kun operaatioon tulee viesti, joka yhdistää operaation instanssin korrelaatiomuuttujaan.

Muuttujalle initiate on olemassa kolme erilaista arvoa:

- Yes, joka kertoo, että aktiviteetin täytyy aloittaa korrelaatiojoukko.
- Join, joka kertoo, että aktiviteetin täytyy aloittaa korrelaatiojoukko, mikäli joukkoa ei ole vielä aloitettu.
- No, joka kertoo, että aktiviteetti ei aloita uutta korrelaatiojoukkoa.

13 BPMN-kuvauksen muodostaminen BPEL-kerrokseksi

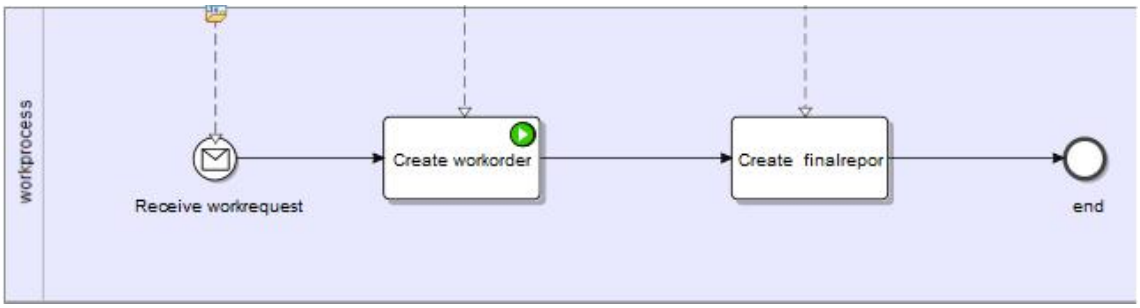
BPMN-kuvaus julkaistaan Apachen ODE -palvelimelle, joka muuntaa kuvauksen BPEL-kielelle ja julkaisee kuvauksesta WSDL-rajapinnan. Kun prosessi on

saatu julkaistua onnistuneesti palvelimelle, on prosessi valmiina ajettavaksi WSDL-rajapinnan kautta.

Apache ODE -palvelinta käytetään Intalion BPMS -palvelimen kautta. Intalio BPMS -palvelin sisältää hallintakonsolin, jonka kautta voi tarkastella ja hallita julkaistuja prosesseja, sekä tarkastella prosessien instansseja. Intalion Designer ja Intalio BPMS on integroitu yhteen, jolloin Designerista voi ladata prosessikuvauksen suoraan BPMS-palvelimelle.

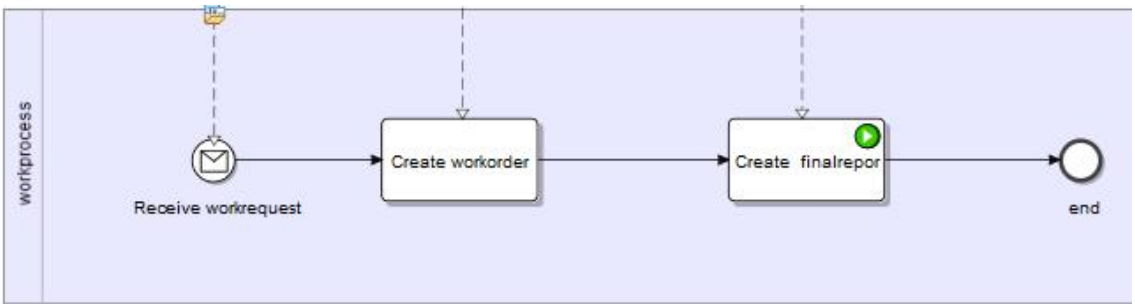
13.1 Prosessimoottorin suoritus

Prosessimoottori muodostaa uuden instanssin mallinnetusta prosessista (kuva 6). Instanssi muodostuu aina kun prosessissa laaditaan uusi työtilaus. Prosessin instanssin suoritusta edistetään eri roolien suorittamilla aktiviteeteilla.



Kuva 17. Instanssin aloitus.

Instanssi muodostetaan (kuva 17) Receive workrequest -aktiviteetilla, joka ohjaa instanssin suorituksen Create workorder -aktiviteettiin. Instanssin prosessi jää odottamaan ko. aktiviteettiin, ennen kuin aktiviteetille tulee viesti, joka kertoo, että kyseinen työvaihe on valmis.



Kuva 18. Instanssin lopetus.

Intanssin suoritus päättyy, kun Create finalreport -aktiviteetti (kuva 18) saa viestin. Tämä kertoo instanssille, että kyseinen työvaihe on valmis ja suoritus voidaan lopettaa end-aktiviteetilla. End-aktiviteettin kuva kertoo tyhjästä lopetusaktiviteetista, joka ei suorita mitään toiminnallisuutta.

13.2 Prosessin instanssin seuranta

BPMN-kuvauksen ja prosessimoottorin iso etu on mahdollisuus tarkastella liiketoiminnallisen prosessin suoritusta ja tarjota mahdollisuus optimoida liiketoiminnallista suoritusta. Instanssien suoritusta voidaan monitoroida esim. ajan perusteella.

Apache ODE -prosessimoottori luo jokaisesta suoritetusta aktiviteetista aikaleiman, jolloin prosessissa tapahtuvien aktiviteettien välistä ajan käyttöä voidaan seurata. Prosessimoottori luo taulukon 3 mukaiset tiedot instansseista. Prosessimoottorista voitaisiin esimerkiksi poimia yksittäisen instanssin tiedot ja laskea instanssissa esiintyvä ajankäyttö.

Taulukko 3. Instanssien tiedot.

Process	State	Started	Last Active
workprocess[v26]	In Progress	2013-03-14 12:03:36	2013-03-14 12:03:36
workprocess[v26]	Completed	2013-03-14 16:26:17	2013-03-14 10:11:59

13.3 Apache ODE -prosessimoottorin testaus

Apache ODE -moottorin BPEL-kerrosta on suositeltavaa testata ennen kuin rajapintaa voidaan käyttää jossain sovelluksessa. Opinnäytetyössä käytin soapUI-sovellusta, jolla voi testata WADL ja WSDL -rajapintoja. Sovellus luo automaattisesti SOAP-viestit käyttäjän määrittelemästä rajapinnasta ja luo automaattisesti pyynnöt palvelimelle. Tämä helpottaa huomattavasti toteuttamisvaihetta, koska käyttäjän ei tarvitse itse laatia testausympäristöä SOAP-viestien lähettämiselle. SoapUI on ilmainen avoimen lähdekoodin sovellus, joka on ladattavissa osoitteesta http://www.soapui.org/.

Tässä on muutama työskentelyä helpottava ominaisuus soapUI-työkalusta:

- Työkalulla voi generoida valmiit SOAP-viestit ja muokata niiden sisältöä vapaasti ja nopeasti.
- Operaation vastauksen saa näkyviin XML-muodossa helposti.
- Työkalu generoi automaattisesti kaikkiin WSDL/WADL-rajapinnan operaatioihin pyyntöviestit.

Hyvä menetelmä testata asiakassovelluksen toiminnallisuutta, oli kirjoittaa asiakassovelluksessa SOAP-viesti XML-tiedostoon. Tiedoston sisältö oli helppo kopioida soapUI:n viestiin ja testata viestin toimintaa lähettämällä viesti soapUIn kautta testattavaan Apache ODE BPEL -moottoriin.

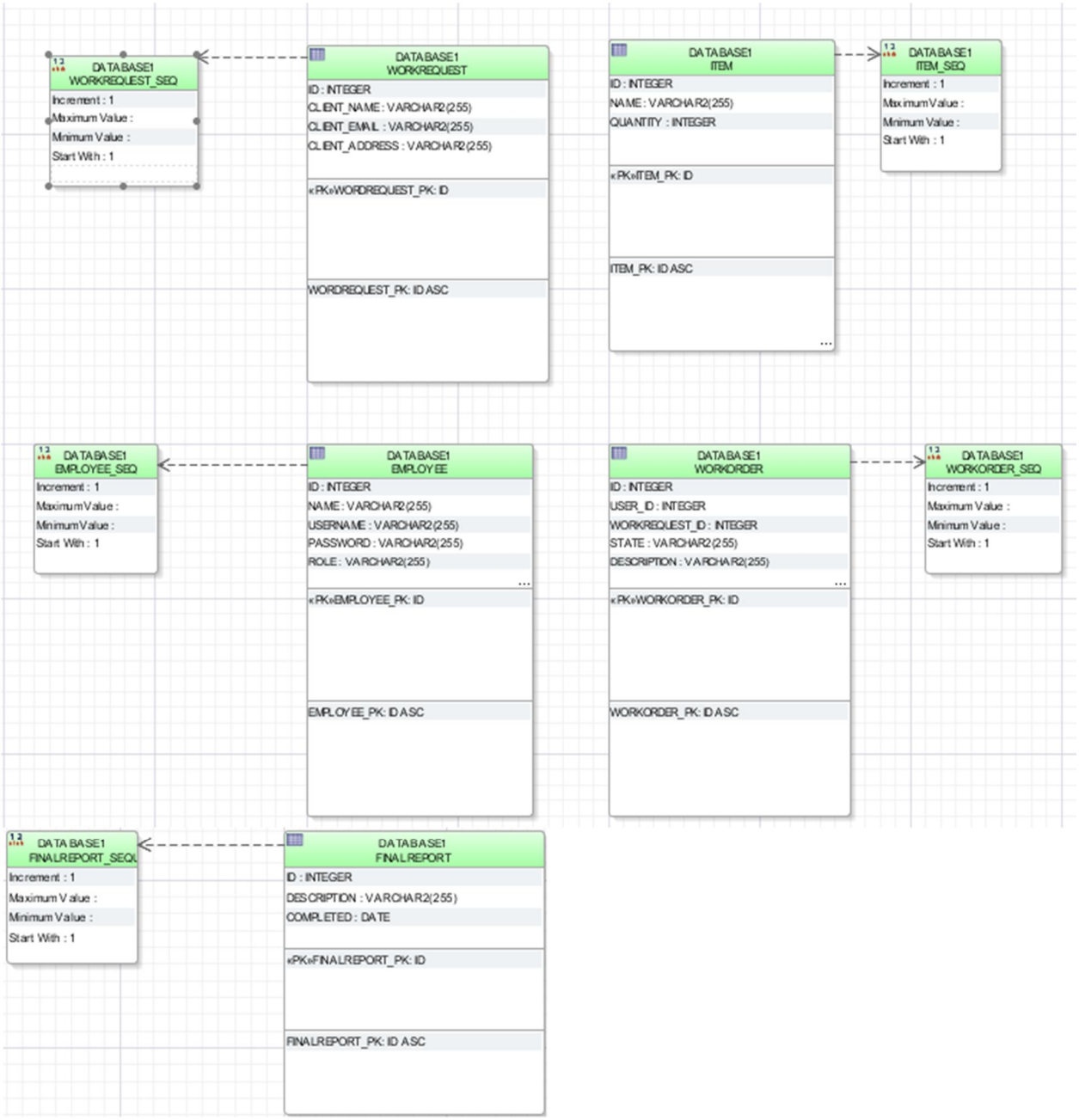
14 Palvelukerroksen toteuttaminen

Demojärjestelmän palvelukerros on oma sovellus, jonka tehtävänä on varastoida liiketoiminnan kannalta välttämättömiä dokumentteja ja ohjata prosessimoottoria välittämällä prosessimoottorille SOAP-viestejä.

Palvelukerros on toteutettu omana komponenttina, koska se korostaisi loose coupling -arkkitehtuuria, joka on osa palvelukeskeistä arkkitehtuuria. Palvelukerroksessa aktivoidaan myös muitten sovellusten palveluita, joka tekee kerroksesta middleware-kerroksen (middleware on hajautettuun järjestelmään viittaava termi). Tavoitteena on, että palvelukerrokseen liittyminen olisi helppoa ja järjestelmän ydintoiminnot voitaisiin yleistää sille tasolle, että toimintoja ei tarvitsisi laatia liitettävissä sovelluksissa uudelleen ja toimintoihin voitaisiin liittyä mahdollisimman helposti. Liitettävien teknologioiden takia, on syytä laatia standardinomainen rajapinta, jonka voi toteuttaa laitteistosta riippumattomasti. Tavoitteena olisi, että liitettävä sovellus tai laite voisi käsitellä rajapinnan tarjoamat viestit ja toimia omana komponenttina, välittämättä rajapinnan toisella puolella tapahtuvista toiminnoista ja käytetystä teknologiasta.

14.1 Tietokanta

Tietokannassa säilytetään liiketoiminnassa tarvittavia dokumentteja. Palvelukerroksen tietokantana toimii Oracle 11g -tietokanta. Tietokannan mallintamiseen on käytetty Oraclen JDeveloper -sovellusta, jolla voidaan laatia tietokanta graafisesti offline-tilassa ja siirtää mallinnettu tietokantakuvaus suoraan tietokantaan. Offline-tietokantakuvauksessa on se etu, että tietokantamallia voidaan päivittää ja muokata ilman, että päivitys vaikuttaa itse ajoympäristöön.



Kuva 19. Tietokantamalli.

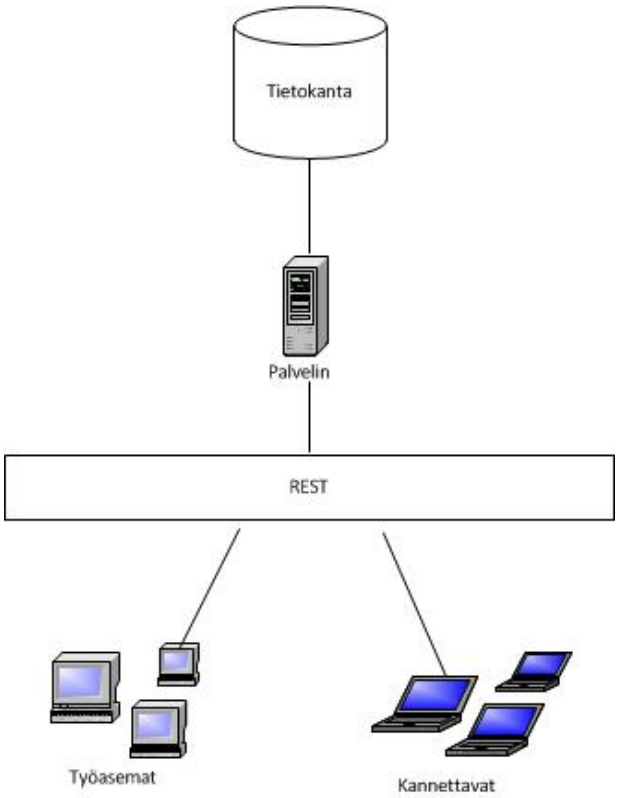
Tietokannassa säilytetään kuvassa 19 näkyvät taulut:

- Workrequest, asiakkaan tekemä työtilaus.
- Item, varastossa oleva tarvike.
- Employee, työpaikan työntekijä. UML-luokkamallissa taulun nimi on ”User”, jota ei voi käyttää Oraclen tietokannassa koska ”User” on varattu nimi.

- Workorder, työnantajan tekemä työtehtävä. Työtehtävä tehdään työtilauksesta.
- Finalreport, työntekijän laatima loppuraportti.

14.2 Palvelukerroksen liittymisrajapinta

Tietokannan palvelurajapintana toimii REST-rajapinta, jonka kautta tapahtuu kaikki tiedonsiirto Client-sovellusten ja tietokannan välillä (kuva 20). REST-rajapinta on toteutettu Netbeans-kehitysympäristön tarjoamalla automaattisella työkalulla. Työkalu generoi tietokannan taulut luokiksi ja laatii luokista REST-rajapinnan. REST-rajapintaan voidaan liittyä lähettämällä HTTP-kutsuja. Palvelu palauttaa vastauksena XML-tiedoston.



Kuva 20. REST-rajapinta.

Sovelluksessa on toteutettu REST-rajapinnasta HTTP:n GET- ja POST-metodit. Sovelluksessa voidaan noutaa tietokannan taulut ja laittaa tietokannan tauluihin uusia rivejä. REST-rajapinta

Tietokantayhteys ja oliopohjainen tietokantamalli on laadittu kehitysympäristön automaattisia toimintoja käyttäen. Netbeans-kehitysympäristö generoi tietokantayhteyden kautta tarvittavat Java-luokat, jotka ovat ilmentymiä tietokannan tauluista. Kehitysympäristö generoi myös vaadittavat toiminnallisuudet luokkien siirtämiseen tietokannan tauluiksi. Käyttäjä voi itse luoda Java-kielestä ajettavia tietokanta hakuja yksinkertaisia Namedqueries-annotaatioita käyttämällä. Tietokantayhteyden luomisen jälkeen voidaan kehitysympäristöllä generoida myös REST-rajapinta. Kehitysympäristö huolehtii URL:n muodostamisen ja kartoittamisen URL:a käyttävälle funktiolle. Käyttäjän tehtävä on tehdä halutut toiminnallisuudet REST-metodien sisälle. Käyttäjä voi tehdä myös omia REST-metodeja luokkien sisälle.

14.3 Prosessimoottorin ja palvelukerroksen välinen yhteys

Tietokantaa käsittelevä kerros huolehtii myös SOAP-viestien välittämisestä Apache ODE -prosessimoottorille. Sovellus huolehtii asianmukaisen SOAP-viestin rakentamisesta ja välittämisestä prosessimoottorille. SOAP-viestit laaditaan palvelimella javax-paketin SOAPFactory-luokalla, jonka tehtävänä on rakentaa standardinomaisia SOAP-viestejä. Käyttäjän tehtävänä on huolehtia viestin sisällön muokkaamisesta, jotta SOAP-viestillä saadan ajettua haluttu operaatio kohdepalvelimella.

Prosessimoottorille välitetään seuraavat SOAP-viestit:

- Workrequest SOAP, joka aktivoi “Receive work request”- aktiviteetin prosessimoottorilla.
- Workorder SOAP, joka aktivoi “Create work order”- aktiviteetin prosessimoottorilla.
- Finalreport SOAP, joka kativoi “Create final report”-aktiviteetin prosessimoottorilla.

15 Client-sovellusten laatiminen

Client-sovellukset toimivat järjestelmässä käyttöliittyminä liiketoiminnallisen prosessin suorittamiseen. Järjestelmään on integroitu seuraavat client-sovellukset:

- Web-sovellus. Sovellusta käyttävät yrityksen asiakas ja esimies. Sovellusta ajetaan Web-selaimen kautta.
- Mobiilisovellus. Sovellusta käyttää yrityksen työntekijä ja sovellusta ajetaan Windows Phone -alustalla.

Client-sovellukset on laadittu kahteen eri ympäristöön, jotta voitaisiin paremmin mallintaa REST-rajapinnan ja standardoitujen viestien käyttämistä eri alustoilla. Tällä todistetaan, että viestien ja rajapinnan standardointi on tehokas ratkaisu, koska se helpottaa erilaisten sovellusten ja järjestelmien integroitavuutta liiketoiminnalliseen sovellukseen.

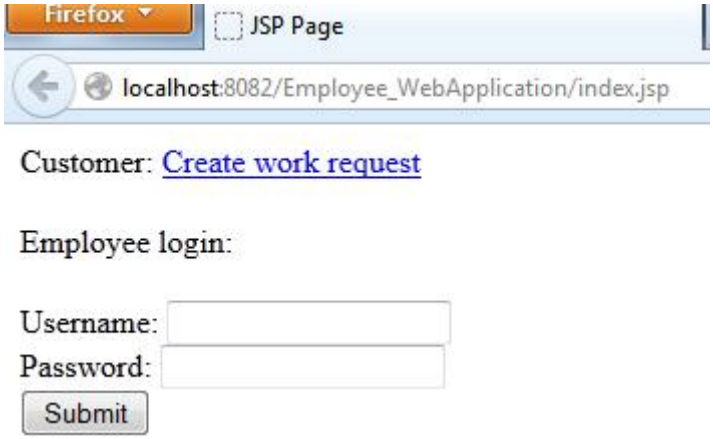
15.1 Web-sovellus

Web-sovellus toteutettiin käyttämällä NetBeans-kehitysympäristön Web Application -projektimallia. Sovellus on koostettu MVC (Model, View, Controller)-arkkitehtuurin mukaisesti. Sovelluksessa servletit toimivat HTTP-pyyntöjen ohjausluokkina, Java-luokat tietokantataulujen malleina ja JSP (JavaServe Page)-sivut näkyminä. Sovelluksessa on myös toteutettu REST-liittymä, jonka tehtävänä on välittää HTTP-pyyntöjä REST-rajapinnalle ja parsia HTTP-pyyntöjen vastauksia.

Web-sovellusta käyttävät asiakkaat ja yrityksen työnantaja. Web-palvelun kautta asiakas laatii työtilauksen ja työnantaja voi hyväksyä työpyynnön jolloin työpyynnöstä tulee työtehtävä.

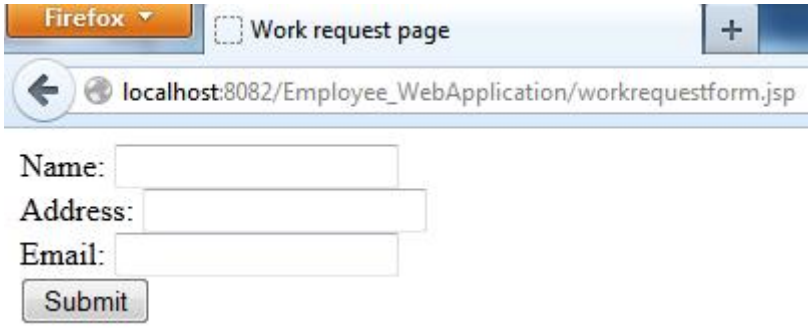
15.2 Web-sovelluksen toiminta

Sovelluksen aloitussivulla voi asiakas siirtyä työtilaussivulle luomaan uuden työtilauksen tai työnantaja kirjautua sisään tarkastelemaan luotuja työtilauksia (kuva 21).

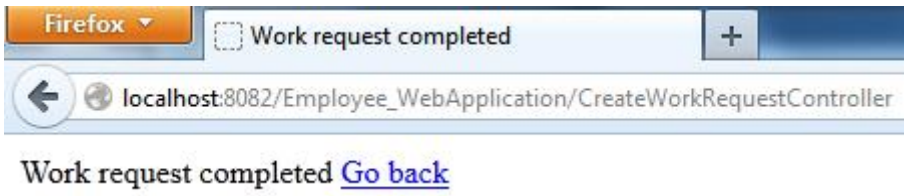


Kuva 21. Aloitussivu.

Työtilaussivulla voi asiakas laatia työtilauksen (kuva 22). Tilauksen laatimisen jälkeen asiakas ohjataan ilmoitussivulle (kuva 23).

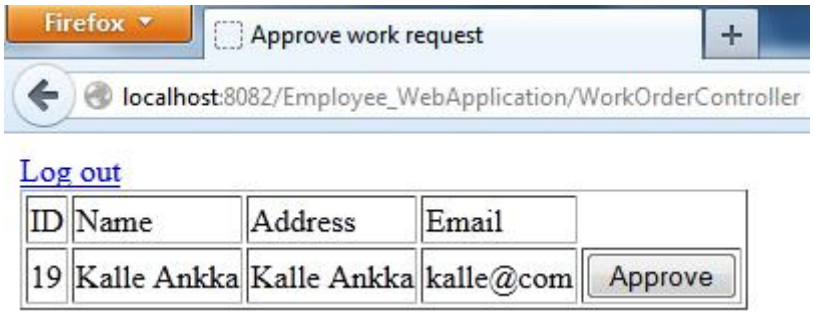


Kuva 22. Työtilaussivu.

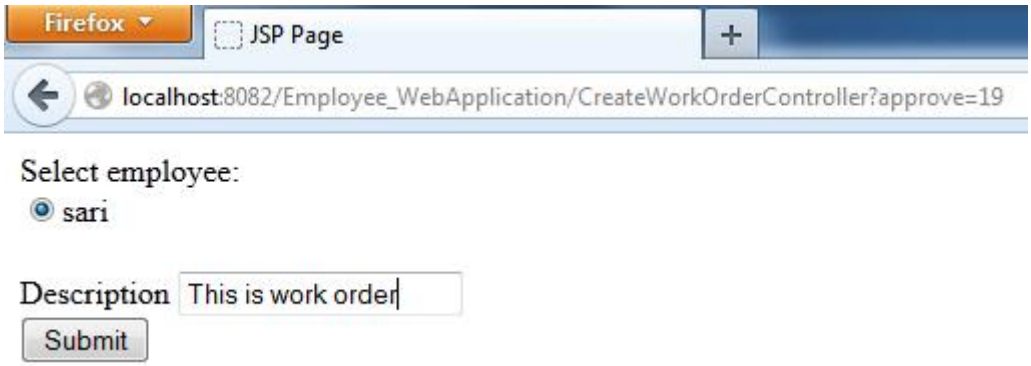


Kuva 23. Ilmoitus onnistuneesta työtilauksesta.

Kuvassa 24 näkyvällä sivulla työnantaja voi hyväksyä laaditun työtilauksen työlistaussivulla painamalla halutun työtilauksen kohdalla Approve-nappia. Tämä ohjaa käyttäjän sivulle jossa voi kirjoittaa työtehtävän kuvauksen ja valita työtehtävälle työntekijä (kuva 25).



Kuva 24: Työlistaussivu.



Kuva 25. Työtehtävän laatiminen.

Kaikki järjestelmässä olevat työntekijät listataan kuvassa 25 näkyvään Select employee -kohtaan, josta työnantaja voi valita haluamansa työntekijän työtehtävälle. Sivulla voi myös kirjoittaa kuvauksen annetusta työtehtävästä.

15.3 Mobiilisovellus

Mobiilisovellusta käyttää liiketoiminnallisen prosessin suorittamiseen työntekijärooli. Työntekijärooli vastaa liiketoiminnallisessa prosessissa loppuraportin tekemisestä, joka kertoo prosessimoottorille, että työ on valmis. Mobiilisovellus on laadittu .NET 4 alustalle ja mobiililaitteen käyttöjärjestelmänä toimii Windows Phone OS 7.1.

15.4 Mobiilisovelluksen toiminta

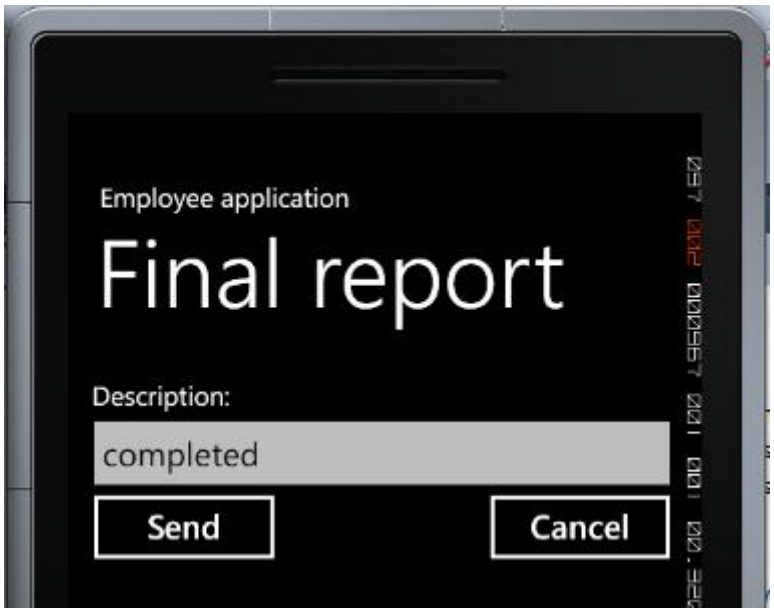
Mobiilisovelluksessa kirjaudutaan sisälle työntekijän tunnuksilla. Kirjautumisikkunassa annetaan käyttäjätunnus ja salasana yhdistelmä (kuva 26).

Kuva 26. Kirjautumisikkuna.

Jos kirjautuminen onnistui, ohjataan käyttäjä työtehtäväsivulle (kuva 27). Työtehtäväsivulla tulostetaan käyttäjälle avoinna olevat työtehtävät. Työtehtävä näkyy vain sille työntekijälle, joka on valittu suorittamaan työtehtävää kun työtehtävä on laadittu. Työntekijä yksilöidään id-tunnuksella, joka haetaan tietokannasta, kun käyttäjä kirjautuu mobiilisovellukseen.

Kuva 27. Työtehtävänäkymä.

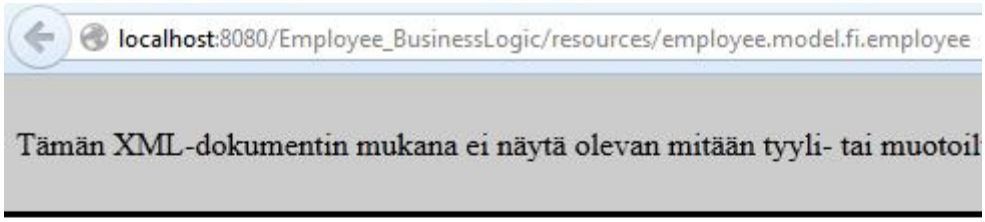
Painamalla kuvassa 27 näkyvää työtehtävää, työntekijä voi kirjoittaa loppuraportin ko. työtehtävästä. Loppuraportti (kuva 28) lähetetään tietokantaan, kun käyttäjä painaa Send-nappia. Loppuraportti välitetään HTTP:n POST-metodin välityksellä tietokantaa hallitsevan sovelluksen REST-rajapinnalle.



Kuva 28. Loppuraporttinäkymä.

15.5 Tiedonsiirto palvelukerroksen ja client-sovellusten välillä

Molemmissa Client-sovelluksissa toteutettiin tietokannan käyttäminen HTTP-viestien välittämisellä ja HTTP-vastauksen parsimisella. Palvelukerroksen palauttamat HTTP-viestit ovat XML-muotoisia. Kuvassa 29 on XML-viestit, jossa employees-sekvenssi on muodostettu XML-elementeillä. Jotta XML-viesti voitaisiin parsia, pitää viestin sisältö tietää tarkasti. Yleensä palveluntarjoaja on dokumentoinut, miten vastaus on koostettu.




```
−<employees>
  −<employee>
      <id>1</id>
      <name>jenni</name>
      <password>1234</password>
      <role>worker</role>
      <username>jenni1</username>
    </employee>
  </employees>
```


Kuva 29. Employees XML-viestit.

Molemmissa tapauksissa toteutustapa on lähes identtinen. Pieniä eroja tosin syntyi viestin lähettämisellä Windows Phone -alustan ja Java-pohjaisen Web-sovelluksen välillä.

Windows Phonessa HTTP-pyyntöjä voidaan välittää synkronisesti tai asynkronisesti toimintoja. Synkroninen menetelmä tarkoittaa sitä, että viestin lähettävä säie jää odottamaan vastausta palvelimelta. Asynkroninen tarkoittaa sitä, että Windows Phone luo jokaisesta HTTP-pyynnöstä uuden säikeen. Tällöin voidaan jatkaa esim. käyttöliittymän päivitystä samalla aikaa kun käyttöjärjestelmä suorittaa HTTP-pyyntöä. Opinnäytetyössä laadittu mobiilisovellus toteuttaa

HTTP-tiedonsiirron asynkronisella menetelmällä. Tällä tavalla voidaan estää käyttöliittymän jäätyminen HTTP-pyynnön ajaksi.

Javalla toteutetussa Web-sovelluksessa kaikki HTTP-pyynnöt ovat synkronoituja sovelluksen muitten toimintojen kanssa. Pyyntö tapahtuu aina samassa säikeessä kuin mistä pyyntö on laukaistu, jolloin sovelluksen suoritus odottaa HTTP-tiedonsiirron valmistumista.

15.6 Integroinnin haaste

Client-sovellus, joka pitää integroida johonkin palvelukerrokseen, sisältää haasteen, joka olisi hyvä tiedostaa palvelukerrosta laadittaessa. Ohjelmoija on täysin riippuvainen palvelukerroksen dokumentoinnista, koska palvelukerros on ns. black box -komponentti, jonka sisäistä toiminnallisuutta ei tiedetä. Esim. luokkamallin laatiminen client-sovellukseen voi olla lähes mahdotonta toteuttaa, jos palvelukerroksesta ei ole olemassa minkäänlaista dokumentaatiota.

Myös kehitysympäristöjen automaattista koodingenerointia ei pysty välttämättä käyttämään, jos rajapinnasta ei ole saatavilla WSDL- tai WADL-tiedostoa. Tällöin ohjelmoijan on itse huolehdittava asianmukaisesta rajapinnan toteutuksesta. Tämä tarkoittaa käytännössä sitä, että ohjelmoijan on laadittava sovellukseen toiminnallisuus, jolla voidaan parsia HTTP-vastauksen sisältö. Tänä päivänä vastaus muodostetaan yleensä XML- tai JSON-muodossa.

Opinnäytetyössä toteutetussa järjestelmässä on REST-rajapinta dokumentoitu liitteen 9 mukaisesti. Dokumentoitaessa REST-rajapintaa on olennaista kuvata seuraavat asiat:

- URL, jonka kautta integrointi tapahtuu
- mitä parametreja mahdollisesti URL:lle välitetään
- lyhyt selitys mitä REST-toiminto tekee järjestelmässä.

Myös vastausviestien sisältö ja muoto olisi dokumentoitava, jotta palveluun liittyminen olisi mahdollisimman vaivatonta.

16 Tulokset

Toteutettavuustutkimuksessa todettiin, että prosessimoottorin integrointi järjestelmään tapahtuu palvelukerroksen toteuttavan järjestelmän kautta. Prosessin aktiviteetteja voidaan suorittaa siten että palvelukerrokseen integroidaan aktiviteetteja ajavat client-sovellukset.

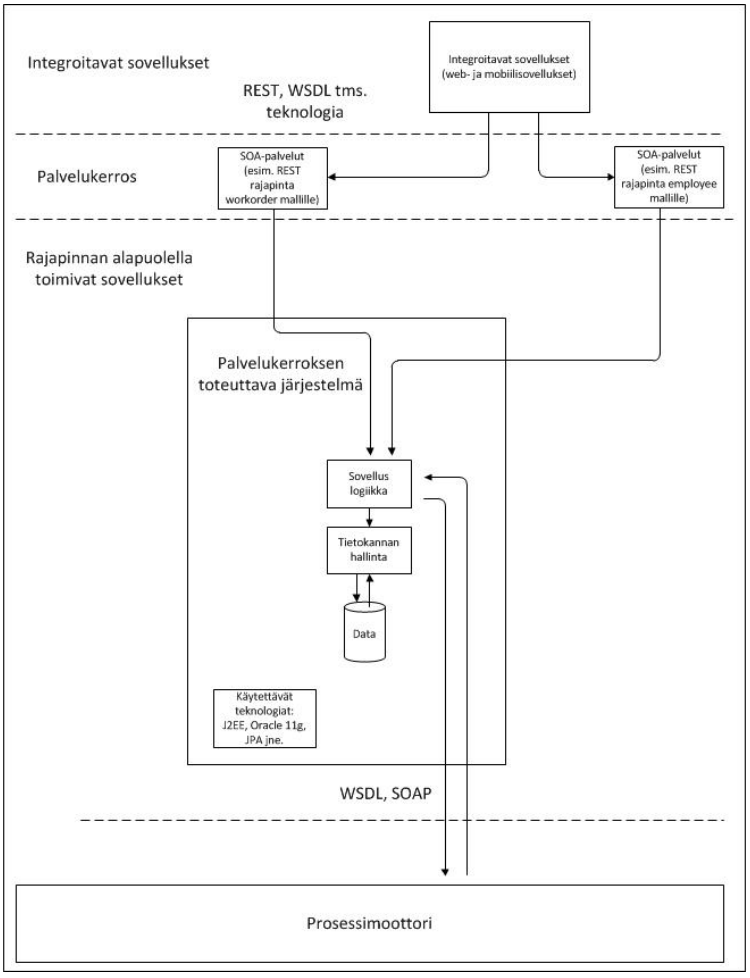
Prosessissa syntyvät instanssit yksilöidään liiketoiminnassa käytetyllä dokumentilla. Laaditussa järjestelmässä käytetty yksilöivä tunnus oli työtilauksen id-tunnus. Tällä tunnuksella voidaan jäljittää haluttu instanssi ja edistää ko. instanssin suoritusta. Prosessissa olevan aktiviteetin liittäminen instanssiin, tapahtuu aktiviteetille välitettävän viestin kautta. Viestin sisällössä oleva instanssin tunnus liitetään aktiviteetin suorittamiseen. Tällä tavalla voidaan edistää yksittäisen instanssin suoritusta.

16.1 Järjestelmän arkkitehtuurin toteutus

Kuvassa 30 on kuvattu kuinka dokumentit virtaavat järjestelmän sisällä. Katkoviivat osoittavat kuvassa rajapintaa, jonka kautta eri tason komponentit keskustelevat järjestelmän kanssa ja nuolet osoittavat dokumenttien virtausta eri komponenttien välillä.

Integroitavat sovellukset ovat käytettävissä käyttöliittymän kautta. Näitten sovellusten toiminnan määrä käytettävissä olevat palvelut, joihin tässä tapauksessa liitytään REST-rajapinnan kautta. Sovellukset koostetaan julkaistuista palveluista, jotta sovellukset toteuttaisivat liiketoiminnan kannalta haluttuja palveluita.

Palvelukerroksen toteuttava järjestelmä pitää sisällään prosessimoottorin suorittamiseen vaadittavat toiminnallisuudet. Prosessimoottori on integroitu palvelukerroksen toteuttavaan järjestelmään. Integrointi tapahtuu WSDL-rajapinnan kautta. Palvelukerros julkaisee korkean tason palvelut tarkasti jyvitetystä logiikasta, joita integroitavat sovellukset käyttävät järjestelmän ajamiseen.



Kuva 30. Dokumenttien virtaus järjestelmän sisällä.

Prosessimoottoria voitaisiin ajaa myös client-sovelluksista käsin, mutta muutoksen sattuessa, client-sovellusten päivitettävyys olisi vaikeampaa kuin palvelukeroksessa sijaitseva logiikan päivittäminen. Dokumenttien tiedon välittäminen prosessimoottorille palvelukerroksesta käsin on myös loogisempaa, koska palvelukerroksen tietokannassa sijaitsee kaikki liiketoiminnan kannalta oleelliset dokumentit. Dokumenttien säilyttäminen client-sovelluksissa vaatisi client-sovellukselta suurempia resursseja, eikä esim. web-selaimen käyttäminen olisi edes mahdollista.

Palvelukerros huolehtii myös instanssien ja aktiviteettien yhdistämisestä. Halutun instanssin prosessin suorittaminen tapahtuu välittämällä instanssin yksilöivä tunnus palvelukerroksesta prosessimoottorille.

16.2 Suunnitteluvaiheen tulokset

Suunnittelumenetelmän käyttäminen oli iso osa sovelluksen laatimista, koska menetelmää käyttäen pystyin laatimaan sovelluksesta loogisesti järkevämmän ja mallintamaan sovelluksessa tapahtuvia ilmiöitä ennen varsinaista toteutusta. Tämä vähentää rinnakkaisuuksia ja päällekkäisyyksiä itse ohjelmointivaiheessa. Myös toteutusvaiheessa ei tule vahingossa laadittua joitain toimintoja, jota ei ole tarkoitettu toteutettavan sovellukseen.

Suunniteltaessa järjestelmää tulisi laatia edes jonkinlaiset korkeantason vaatimukset ja luoda käyttötapaukset vaatimuksista. Näitten menetelmien käyttäminen auttaa keskittymään ratkaistavaan ongelmaan sekä luomaan domain model -diagrammi joka toimii projektissa termistönä. Myös jatkokehitystä varten ko. dokumentointi olisi hyvä toteuttaa.

16.3 Prosessin mallintaminen ja julkaiseminen

Prosessin mallintaminen onnistui Intalio Designer sovelluksella ketterästi. Sovelluksen käyttämiseen vaadittava dokumentointi ei ole kuitenkaan kovin laadukasta ja muutamia asioita saa etsiä keskustelupalstoilta ja erilaisilta websivuilta. Myös BPMN-notaation ymmärtäminen ja ajatus prosessimallintamisesta on lähes välttämätöntä, että työkalua voi käyttää tehokkaasti liiketoiminnallisten prosessien mallintamiseen.

Prosessin käyttäminen Apache ODE -prosessimoottorin kautta onnistuu lähes vaivattomasti, kunhan prosessimoottorille välitettävät SOAP-viestit ovat asianmukaisia.

16.4 Palvelukerroksen toteutusvaiheen tulokset

Toteutettaessa palvelukerrosta on käytetty JDeveloper- ja NetBeans -kehitysympäristöjä. Kehitysympäristöistä molemmat tarjoavat mahdollisuuden generoida Web Service -palveluita. JDeveloperin tarjoama SOAP-viesteihin perustuva Web Service -toteutus oli vaikeasti muokattava. JDeveloperilla oli myös vaikea laatia Web Service -rajapinta, joka antaisi halutun XML-muotoisen vastauksen. Esim. monen XML-elementin sisältämä vastaus oli vaikea toteuttaa JDeveloperilla. JDeveloper-kehitysympäristössä oli muutenkin aika vaikeasti ymmärrettävä pinnan alla tapahtuva XML-viestien sisällön ja Java-luokkien sitominen. JDeveloper-kehitysympäristö on muutenkin ilmeisesti vähän käytetty, koska teknisiin ongelmiin ei löytynyt vastauksia yhtä kattavasti kuin Netbeans-kehitysympäristölle.

Päädyin toteuttamaan palvelukerroksen NetBeansilla, koska NetBeansilla pystyin generoimaan automaattisesti ja lähes vaivattomasti kaiken tarvittavan koodin tietokantayhteyden toteuttamiseen. Luodusta tietokantayhteydestä kehitysympäristö luo myös automaattisesti REST-rajapinnan tai SOAP-viestejä käyttävän Web Service -rajapinnan.

Integroinnissa ja tiedonsiirrossa käytettävä rajapinta oli yksinkertaisinta luoda Netbeans-kehitysympäristöllä REST-rajapinnaksi. WSDL-rajapinnan luominen oli Netbeansilla vaikeampaa kuin REST-rajapinnan luominen. Myös client-sovelluksissa REST-rajapinnan käyttäminen oli helpompaa kuin Web Service -rajapinalla. Tämän takia järjestelmään toteutettiin REST-rajapinta.

Palvelukerroksesta prosessimoottorille välitettävien SOAP-viestien laatiminen onnistuu hyvin Javan SOAP-viestien muodostamiseen tarkoitetuilla luokilla. SOAP-viestien sisältöä ja yhdistettävää rajapintaa kannattaa kuitenkin testata ennen koodin toteutusta esim. luvun 13.3 mukaisella tavalla.

Opinnäytetyössä käytetty Oraclen tietokanta sisältää monia eri toiminnallisuuksia ja mahdollisuuksia, joita ei opinnäytetyössä tarvittu. Tämän takia Oraclen tietokanta on ehkä hieman liian iso sovellus ko. tarkoitukseen. Tosin tulevaisuu-

den kannalta Oraclen tietokannan käyttäminen tarjoaa mahdollisuuden toteuttaa tietovarastointia Oraclen työkalujen avulla ja Oraclen tietokannan opettelua opinnäytetyössä laaditun järjestelmän avulla.

Opinnäytetyössä operatiivista tietokantaa puhdistettiin siten, että instanssin suoritus päättyy, vapautetaan operatiivinen tietokanta instanssiin liitetyistä dokumenteista. Tällaisia dokumentteja ovat Workrequest ja Workorder -taulut. Tällä tavalla voidaan pitää operatiivinen tietokanta mahdollisimman kevyenä ja parantaa operatiivisen tietokannan suorituskykyä. Tietokantaan liittyvää tietovarastointia ja sen mahdollisuuksia on pohdittu lisää luvussa 17.3.5.

16.5 Integroitavien sovellusten liittäminen palvelukerrokseen

Web- ja mobiilisovellusten liittämisessä ei ilmennyt suuria teknologisia ongelmia. Kahden eri sovelluksen integrointi järjestelmään onnistui hyvin, koska molemmat sovellukset pystyivät lukemaan XML-muotoisia vastauksia. Tämän takia rajapinnan pitäisi olla mahdollisimman standardinomainen. Client-sovellusten käyttämät rajapinnat on dokumentoitu liitteessä 9.

Client-sovellusten toteuttaminen kahdelle eri alustalle oli opettavaista, koska se tarjosi näkemykseen siitä miten standardinomaisuus edistää teknologiariippumattomuutta. Koska toteuttaminen tapahtui kahdella eri alustalla ja kehitysympäristöllä, syntyi niissä muutamia pieniä eroavaisuuksia toteutustavassa. Niitä käydään läpi seuraavissa kappaleissa.

16.5.1 Web-sovelluksen toteuttamisvaiheen tulokset

Netbeans-kehitysympäristöllä palvelurajapintaan liittäminen koodingeneroinnin kautta on hieman vaikeaa. Kehitysympäristö saattaa luoda rajapinnassa käytettävistä viesteistä todella oudon nimisiä luokkia, joitten käyttäminen ja ymmärtäminen on vaikeaa. Myös luokkarakenne on aika vaikeaselkoinen, eikä esim. Netbeans-kehitysympäristö osannut luoda ollenkaan REST-rajapinnasta toteu-

tusta. Tämä on puute joka on myös huomattu erilaisilla keskustelupalstoilla. REST-rajapintaa käyttävän luokan toteuttaminen käsin ei kuitenkaan ole kovin suuritöinen. Toteuttaessa toiminnallisuutta pitää kuitenkin tietää miten HTTP-pyyntöjä käsitellään Javan SE -kirjaston kautta. Tämä vaati hieman syvällisempää ymmärrystä Javan tietovirroista, sekä HTTP-vastauksien XML- tai JSON-tiedostomuodon parsimisesta. Opinnäytetyössä tuotetussa lähdekoodissa on olemassa myös esimerkki näitten komponenttien käyttämisestä.

16.5.2 Mobiilisovelluksen toteuttamisvaiheen tulokset

Mobiilisovelluksessa käytetty Visual Studio 2010 ja .NET-kirjasto sisältää hyvät ominaisuudet HTTP-viestien käyttämiseen. Myös Microsoftin MSDN-sivuilla (Microsoft Developer Network) .NET-alusta on hyvin dokumentoitu. Windows Phonen ohjelmoinnissa on kuitenkin hyvä tietää miten Windows Phone ja .NET käyttää säikeitä, koska HTTP-viestien välittäminen tapahtuu .NET-alustalla asynkronisesti, kuten kappaleessa 11.3 on todettu. Käyttöliittymän laatimiseen Windows Phonelle on olemassa useita erilaisia kirjoja, joista itse käytin Lecrenskin & Karli Fonseca-Ensorin (2011) teosta [11].

Microsoft Visual Studio sisältää myös REST- tai Web Service -rajapinnan toteuttamiseen tarvittavan koodin generoinnin. Tämä ominaisuutta en kuitenkaan itse kokeillut. Tulevaisuudessa tätä ominaisuutta voitaisiin myös kokeilla, mikäli sovellusta jatkokehitys ja sovellukseen lisätään uusia erilaisia rajapintoja käyttäviä ominaisuuksia.

16.6 Toteuttamisvaiheen työjärjestys

Järjestelmän toteuttamiseen on käytetty seuraavaa työjärjestystä:

- suunnittelu,
- liiketoiminnallisen prosessin muodostaminen,
- palvelukerroksen toteutus ja prosessimoottorin integrointi,

- client-sovellusten integrointi palvelukerrokseen.

Rekonstruktion kannalta ko. työjärjestys olisi hyvä pitää samanlaisena. Ilman suunnittelua ei voida mallintaa haluttua prosessia ja tuntematta prosessia ei voida laatia järkevää palvelukerrosta. Ilman palvelukerroksen toteuttamista ei voida integroida client-sovelluksia järjestelmään. Vaikka työjärjestys olisi samanlainen, ei toteutusvaiheessa kuvattuja yksittäisten komponenttien toteutusta tarvitse tehdä kuitenkaan täsmälleen samalla tavalla. Se miten eri komponentit luodaan, on tekninen kysymys ja eri tekniikoita käyttäen voi itse toteutus olla erilainen. Tärkeintä työvaiheistuksessa on, että palvelukerros olisi liiketoiminnallisen järjestelmän kannalta järkevä ja palvelukerroksen rajapintaan voitaisiin integroida client-sovellukset helposti. Rajapintojen luonti, että eri komponentit pystyvät keskustelemaan toistensa. Kuten luvussa 15.6 on todettu, että rajapinnan dokumentoinnilla on suuri merkitys jos ohjelmoija ei tunne rajapinnan julkaisevaa toteutusta. Rajapinnan pitäisi olla sellainen, että sitä voidaan käyttää black box -periaatteella.

17 Pohdinta

17.1 Ajatuksesta toteutukseksi

Opinnäytetyössä laaditun järjestelmän toteuttamiseen, piti käydä läpi monta vaihetta, ennen kuin ajatus siitä, kuinka tutkimustyö voitaisiin toteuttaa, kirkastui. Suurimpia haasteita oli sisäistää tarvittava tieto ko. järjestelmästä. Järjestelmän laatimiseen piti saada tarvittavat pohjatiedot palvelukeskeisestä arkkitehtuurista, Windows Phone ohjelmoinnista, web-tiedonsiirto teknologiasta, sekä prosessin mallintamisesta ja prosessimoottorin ajamisesta. Monia näistä asioista ei opeteta koulussa, joten asioitten opiskelu jäi tiedon etsimisen ja lukemisen varaan. Esimerkiksi Windows Phonen HTTP-pyyntöjen lähettämisessä vaadittavia tietoja ei ole opetettu koulussa. Näitä tietoja ovat Windows Phoneen liittyvät säikeistykset, HTTP-pyynnön kirjoittaminen ja HTTP-vastauksen lukeminen. Hyvänä tiedon lähteenä toimi verkkokirjasto ebrary, josta löysin kaikki

tarvittavat teokset. Myös koulun kirjastosta löytyi aiheeseen liittyviä teoksia. Mallintamiseen vaadittavan prosessin hahmottelu oli vaikeaa, koska siihen piti luoda käytännössä tyhjästä jokin liiketoiminnallinen prosessi. Tämän prosessin hahmottamiseen auttoi lähde [6], joka sisältää hyviä menetelmiä, miten kannattaa viedä läpi sovelluksen tai järjestelmän suunnittelu.

Laadittu järjestelmä antoi tulokset ja vastaukset asetettuihin kysymyksiin, eikä suuria teknillisiä ongelmia syntynyt toteutusvaiheessa.

17.2 Eettisyys ja luotettavuus

Opinnäytetyössä on kyseessä kvalitatiivinen tutkimus, jossa ei voida varsinaisesti mitata tai laskea tutkimuksen tuloksia. Kvalitatiivinen analyysi sopii parhaiten tutkimustyöhön, jossa tuloksia ei voida käsitellä numeerisesti, vaan tulosten käsittely jää sanojen ja kuvausten varaan [12, s. 57]. Tämä johtuu siitä, että toteutettava järjestelmä ei tuota numeraalisia arvoja tai järjestelmää ei voida määritellä numeraalisesti.

Tutkimustyön pätevyydellä tarkoitetaan tutkimusmenetelmän valitsemista, jolla voidaan mitata täsmällisesti tutkimuskohdetta [13, s. 216]. Opinnäytetyössä tutkimuksen pätevyyttä on pyritty kohentamaan laatimalla järjestelmä, jossa tulokset olisivat samanlaisia, aina kun tutkimus toteutetaan uudelleen vastaavalla menetelmällä. Tutkimuksen luotettavuutta on pyritty kohentamaan tutkimuksen tarkalla kuvauksella. Tutkimus on pyritty raportoimaan niin tarkasti, että lukijan on mahdollista toteuttaa vastaava tutkimus uudestaan.

17.3 Jatkotutkimus- ja kehitysmahdollisuudet

Tutkimustyössä käytetty järjestelmä annetaan koulun käyttöön jatkokehitystä varten. Järjestelmä sisältää laajat jatkokehitysmahdollisuudet, joita voidaan käyttää liiketalouden ja tekniikan opiskelijoiden opettamiseen. Järjestelmän jat-

kokehittämisellä voidaan tutkia liiketoiminnan automatisointia monesta eri näkökulmasta ja luoda lisäarvoa tässä työssä toteutetulle konseptille.

17.3.1 Liiketoiminnallisen prosessin jalostaminen

Tutkimustyössä käytetyn liiketoiminnallisen prosessin jalostaminen ja jatkokehittäminen voitaisiin toteuttaa liiketalouden opiskelijoilla. Tutkimustyön lopussa prosessi on todellisuudessa liian yksinkertainen, ollakseen täysin toimiva liiketoiminnallinen prosessi. BPMN-notaation opettamiseen prosessia voitaisiin jalostaa toteuttamaan mahdolliset poikkeustilanteet ja luomalla monipuolisempi liiketoiminnallinen prosessi. Oppimisen mittarina voitaisiin käyttää menetelmää, kuinka hyvin jalostettu prosessi saadaan toteutettua teknillisellä puolella. Tämä ajatus tukisi myös kappaleessa 3 esitettyä ajatusta BPMN-notaatiosta.

17.3.2 Toiminnanohjausjärjestelmän integrointi konseptiin

ERP (Enterprise Resource Planning)-järjestelmät eivät voi itsessään toteuttaa liiketoiminnan ydinprosesseja niitten yksilöllisyyden takia, mutta liiketoiminnan tukitoimet ovat varsin yleisiä ja geneerisiä toimintoja, joita ERP-järjestelmät voivat automatisoida. Kaupallisten ERP-järjestelmien liittäminen järjestelmään olisi hyvä jatkokehitysmahdollisuus.

ERP-järjestelmä toimisi järjestelmässä liiketoiminnan tukitoimien automatisoinnissa (laskutus, palkanmaksu, varastonhoito jne.). Tavoitteena olisi, että ERP-järjestelmään voitaisiin virrata tietoa ydinprosessin tapahtumista automaattisesti, jolloin esim. laskutus asiakkaalle tai työkustannuksien seuranta olisi myös automatisoitu. Tästä voisi muodostua todella toimiva kokonaisuus, jossa liiketoiminnallisen järjestelmän automaation aste olisi korkea. Tällä tavalla voitaisiin myös edistää tietämystä ERP-järjestelmien käyttämisestä prosessimoottorin kanssa.

17.3.3 Client-sovellusten jatkokehitys

Client-sovelluksia voidaan käyttää projektitöissä tai opetusmateriaalina. Client-sovelluksissa on käytetty monia eri menetelmiä, mitä ei ole opetettu koulun kursseilla mutta olisi hyvä tietää, jos halutaan toteuttaa web- tai mobiilisovelluksella vastaava REST-rajapintaan liittyminen.

Client-sovelluksissa olisi parannettavaa käytettävyyden ja käyttöliittymän puolella. Eritoten erilaisia virheilmoituksia ja tilailmoituksia pitäisi lisätä mahdollisten poikkeustilanteiden sattuessa. Nämä ominaisuudet voitaisiin laatia myös koulun tulevilla kursseilla, joissa opetetaan web ja Windows Phone -ohjelmointia.

17.3.4 Push-palvelun kehittäminen

Järjestelmään olisi myös hyvä toteuttaa opetus mielessä Push-Notification-palvelu. Push-Notification on tekniikka, jolla välitetään viestejä palvelimelta client-sovellukselle, ilman että client-sovellus joutuisi kutsumaan palvelinta. Esimerkkinä voisi olla sähköposti tai pikaviestipalvelu [11, s.347.]

Aina kun järjestelmään on luotu uusi työtehtävä työnantajan toimesta, voisi järjestelmä ilmoittaa Push-mekanismilla työntekijää uudesta työtehtävästä. Tällä tavalla voitaisiin välittää tietoa työntekijälle uudesta työstä, ilman että työntekijä joutuu päivittämään työtehtävälistan manuaalisesti.

17.3.5 Tietovarastointi

Tietovarastoinnin kehittäminen järjestelmään olisi myös hyvä jatkokehitys mahdollisuus. Kaikki tiedot, jotka on liitetty instanssin suoritukseen, voitaisiin vapauttaa operatiivisesta tietokannasta ja viedä tietovarastoon. Tämä antaa myös uudenlaisen suunnittelunäkökulman liiketoiminnallisessa prosessissa käytettävien dokumenttien hallitsemiseen. Voitaisiin ajatella, että dokumenttien elinkaari on yhtä pitkä kuin dokumentteja käyttävän instanssin pituus. Koska dokumentit kytketään instanssin suoritukseen, voidaan myös dokumentit vapauttaa kun instanssin suoritus päättyy.

Lähteet

1. Juric, M. & Pant, K. Business Process Driven SOA using BPMN and BPEL. Olton Birmingham, GBR: Packt Publishing Ltd. 2008.
2. Manouvrier, B. & Menard, L. Application Integration: EAI, B2B, BPM and SOA. Hoboken, NJ, USA: Wiley-ISTE. 2010.
3. Silver, B. BPMN Method and Style, 2nd Edition, with BPMN Implementer's Guide. Aptos, CA, USA: Cody-Cassidy Press. 2011.
4. Juric, M. & Krizevnik, M. WS-BPEL 2.0 for SOA Composite Applications with Oracle SOA Suite 11g. Olton Birmingham, GBR: Packt Publishing Ltd. 2010. S. 38 – 40.
5. Calero, C., Genero, M. & Mario, P. 2005. Metrics for Software Conceptual Models. London, GBR: Imperial College Press.
6. Rosenberg, D. & Stephens, M. 2007.Use Case Driven Object Modeling with UML: Theory and Practice. New York, USA: Apress.
7. Newcomer, E. Understanding Web Services: XML, WSDL, SOAP and UDDI. Boston, MA. USA: Pearson Education, Inc. 2002.
8. Arora, G., Kishore, S. & NIIT (Corporation) Staff. 2002. Building Web Services with XML. Independence, KY, USA: Premier Press.
9. Singh, M. & Huhns, M. Service-Oriented Computing. Chichester, West Sussex, England: John Wiley & Sons Ltd. 2005.
10. Sandoval, J. RESTful Java Web Services: Master Core REST Concepts and Create RESTful Web Services in Java. Olton Birmingham, GBR: Packt Publishing Ltd. 2009. S. 7 – 8.
11. Lecrenski, N. & Karli Fonseca-Ensor, R. Beginning Windows Phone 7 Application Development : Building Windows Phone Applications Using Silverlight and XNA. Hoboken, NJ, USA: Wrox. 2011.
12. Järventausta, H., Moisala, M. & Toivakka, S. Tutkimalla oppii. Helsinki: WSOY. 1999. S. 57.
13. Hirsjärvi, S., Remes, P. & Sajavaara, P. Tutki ja kirjoita. Helsinki: Tammi. 2004. S. 216.
14. Hruby, P. Visio Stencil and Template for UML 2.2. [Verkkosivu]. 2010. Saatavissa: http://www.softwarestencils.com/uml/index.html

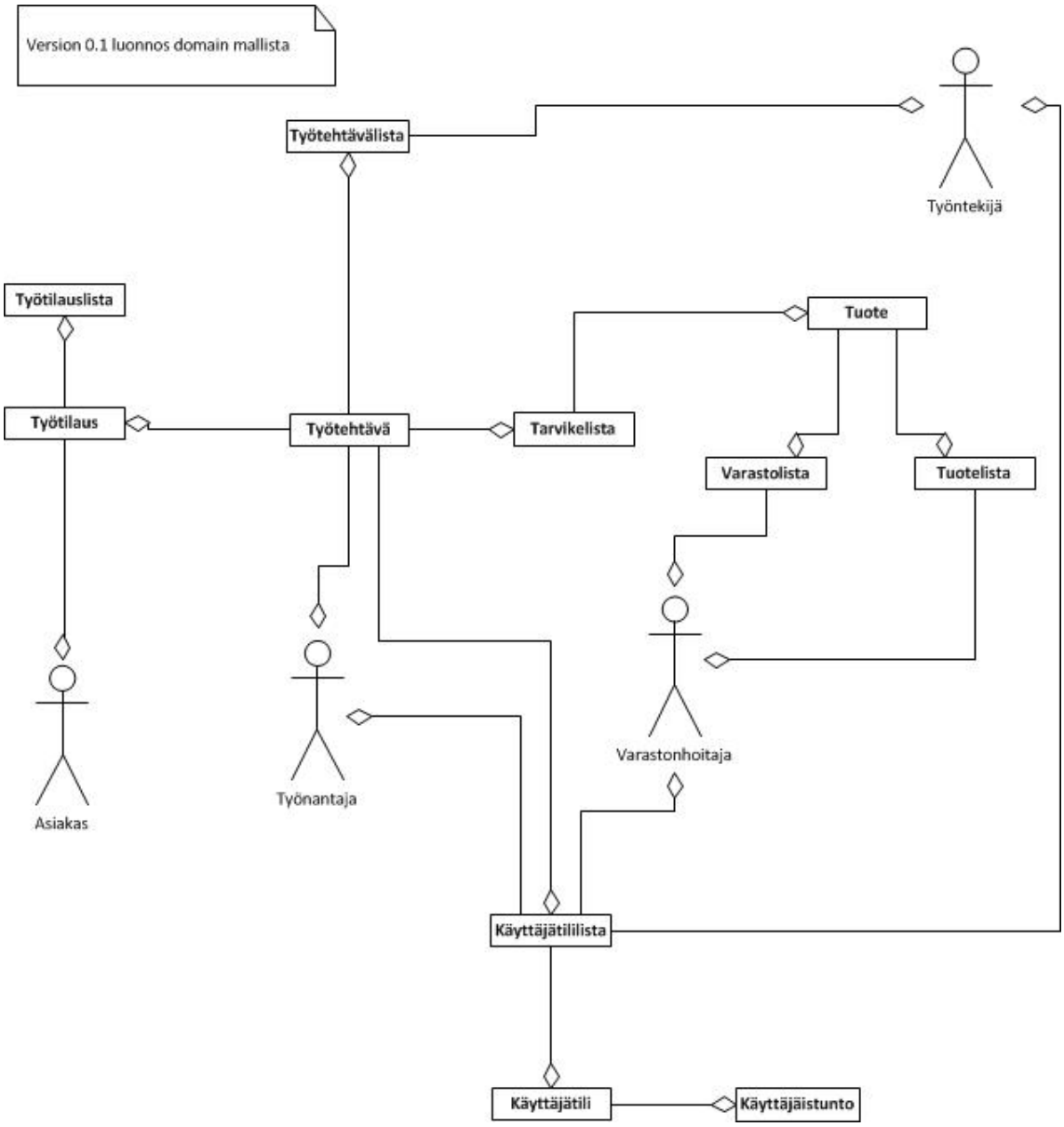
Termit ja lyhenteet

BPEL	Business Process Execution Language on standardoitu kieli BPMN kuvausten toiminnalliseen mallintamiseen.
BPMN	Business Process Modeling Notation on graafinen kuvaus liiketoiminnallisesta prosessista.
BPMS	Business Process Management Server, prosessimoottorin hallintaan käytettävä palvelin.
C, C#, C++	Ohjelmointikieliä, jotka käännetään binääriseen muotoon.
Client	Sovellus tai järjestelmä jolla mahdollistetaan pääsy palvelimen palveluihin.
CSS	Cascading Style Sheets on WWW-dokumenteille kehitetty tyyliohjeiden laji.
DBMS	Database Management System on järjestelmä, joka sisältää tietokannan tietoja.
Domain model	Suunnittelumenetelmissä käytettävä diagrammi, joka kertoo projektissa esiintyvät termit
ERP	Enterprise Resource Planning järjestelmällä toteutetaan sähköisesti liiketoiminnan tukitoimia, kuten laskutus, palkanmaksu ja jne.
HTML	Hypertext Markup Language on tiedostomuoto jolla määritellään WWW-sivun sisältö.
HTTP	Hypertext Transfer Protocol on protokolla jota selaimet ja WWW-palvelimet käyttävät tiedon siirtämiseen.
Intalio	Yritys, joka tuottaa liiketoiminnallisten prossien käsittellyyn ja suunnitteluun liittyviä sovelluksia.
JavaScript	Skriptauskieli, jolla lisätään interaktiivisuutta WWW-sivujen sisältöön.
JSON	JavaScript Object Notation on standardoitu tiedoston rakenne. JSONia käytetään tiedon välittämiseen.

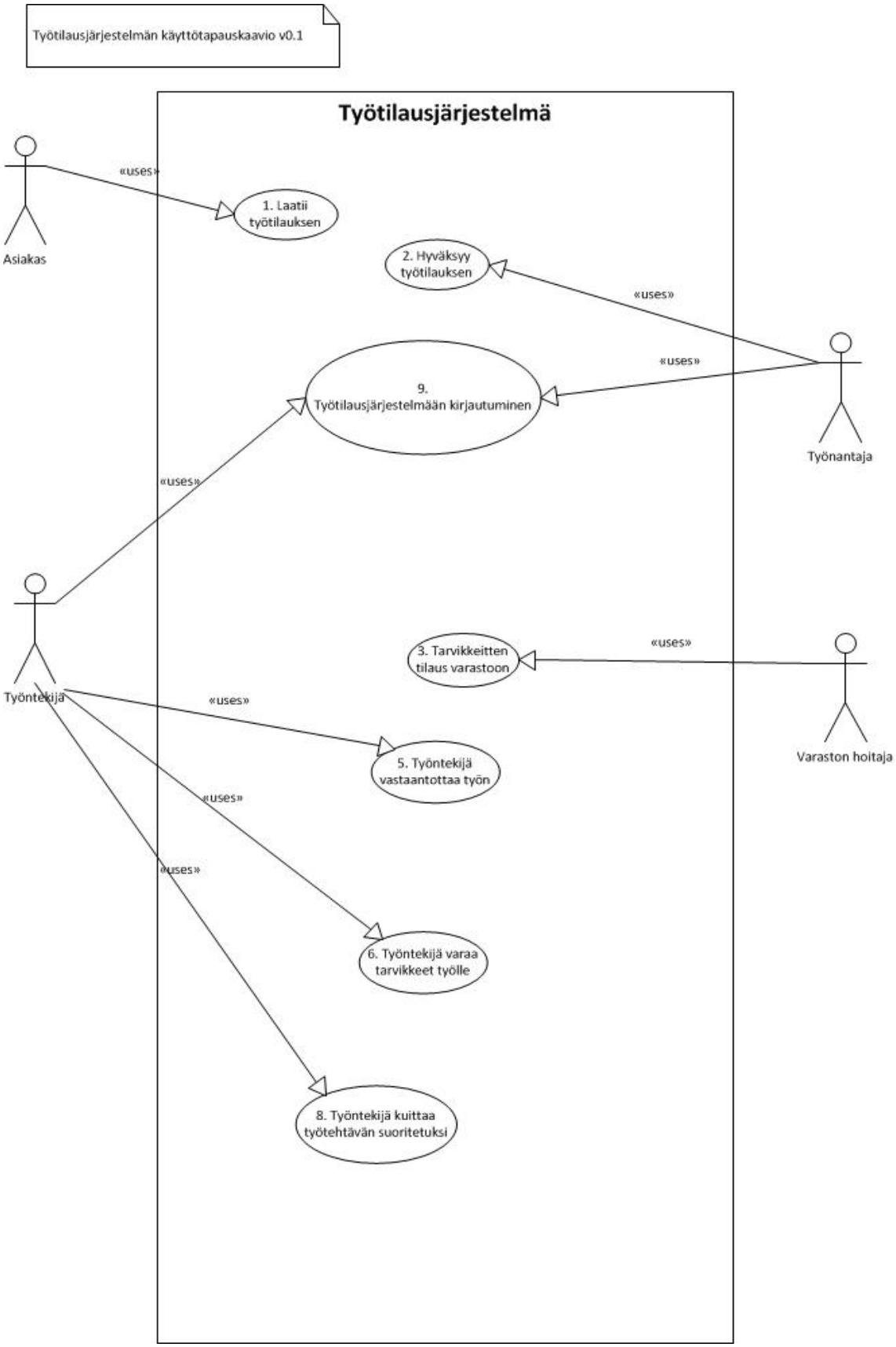
OASIS	Organization for the Advancement of Structured Information Standards on konsortio, jonka tehtävänä on edistää Web Service ja elektronisen kaupankäynnin standardeja
ODE	Orchestration Director Engine, prosessikuvauksen suorittava moottori, joka käyttää WS-BPEL kieltä suoritukseen.
OMG	Object Management Group on konsortio, joka on perustettu oliopohjaisten hajautettujen järjestelmien standardien asettamiseen.
PHP	Skriptauskieli, jolla muodostetaan palvelimella dynaamisesti sisältöä WWW-sivuille.
Push-Notification	Mekanisimi jolla välitetään viesti palvelimelta Client-sovellukselle, ilman että Client-sovellus kutsuisi palvelinta.
RDBMS	Relational Database Managemnt System on sovellus, joka järjestelee ja ylläpitää relaatiotietokannan tietoja.
REST	Representational State Transfer on standardoitu tiedonsiirto menetelmä, joka käyttää HTTP-pyyntöjen metodia halutun palvelun aktivoimiseen.
Schema	Tiedosto joka määrittelee XML-tiedoston sisällön.
SOA	Service Oriented Architecture on filosofia sovelluksen kehittämiselle.
SOAP	Simple Object Access Protocol on protokolla, jonka tarkoituksena on välittää XML-viestejä HTTP-protokollan välityksellä.
TCP	Transmission Control Protocol on netissä tapahtuvaan tiedonsiirtoon tarkoitettu protokolla.
UDP	User Datagram Protocol on netissä tapahtuvaan tiedonsiirtoon tarkoitettu protokolla. UDP protokolassa ei varmisteta pakettien pääsyä perille.
UML	Unified Modeling Language on standardoitu mallinnuskieli, jota käytetään olio pohjaisten sovellusten suunnitteluun ja mallintamiseen.

W3C	World Wide Web Consortium on organisaatio joka määrittelee ja hallinnoi WWW:n standardeja.
Web Service	Palvelimella sijaitsevia palveluita joihin liitytään internetin kautta.
WSDL	Web Services Description File on XML-tiedosto joka kuvailee mitä palveluita Web Service sisältää.
WS-BPEL	Web Service Business Process Execution Language on standardoitu kieli, jolla voidaan kuvata liiketoiminnallisen prosessin suoritus ohjelmistotasolla.
WWW	World Wide Web on internet verkossa toimiva hajautettu hypertekstijärjestelmä.
XML	eXtensible Markup Language on standardoitu tiedostomuoto, jota käytetään tiedonsiirtämiseen.

Projektin sanakirja (domain-model)



Järjestelmän käyttötapaukset



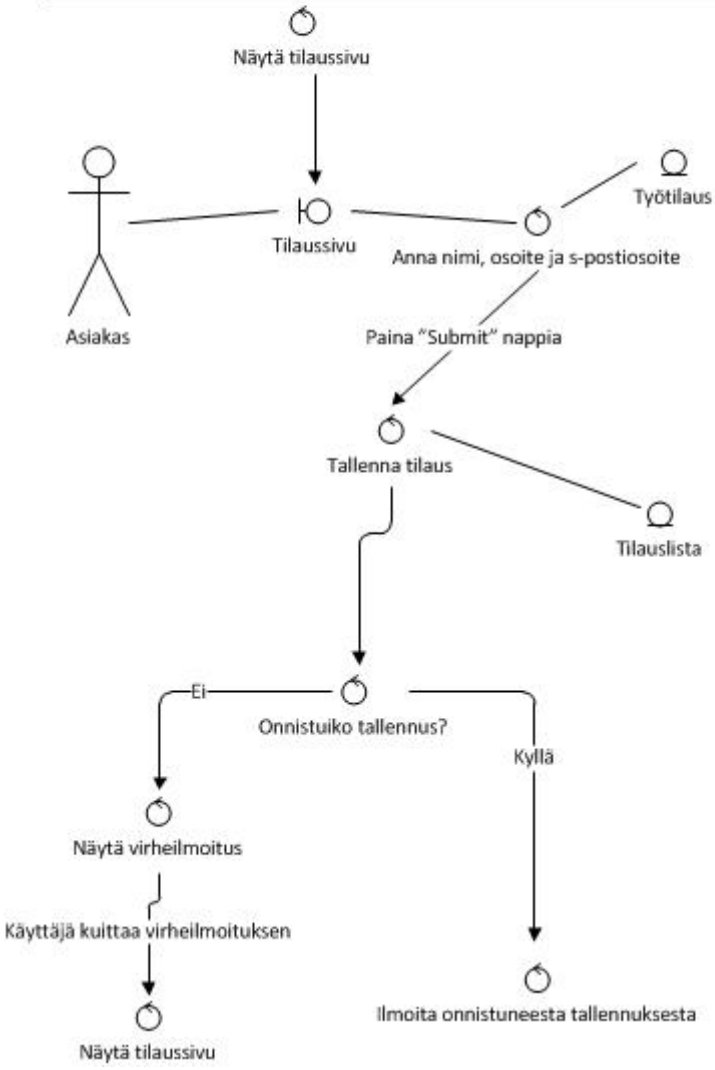
Käyttötapaus 1. robustness diagrammi

UseCase_1

S1, käyttäjä laatii onnistuneesti työtilauksen.
1. Järjestelmä näyttää työtilaus www-sivun.
2. Käyttäjä antaa sivulla nimen, osoitteen ja s-postiosoitteen.
3. Käyttäjä painaa "tallenna"-linnkiä sivulla.
4. Järjestelmä tallentaa työtilauksen.
5. Käyttötapaus päättyy kun käyttäjälle on tullut ilmoitus työtilaukse onnistuneesta tallentamisesta.

Ei vaihtoehtoisia polkuja.

P1. Järjestelmä ei kykene tallentamaan työtilausta.
1. Järjestelmä näyttää virheilmoituksen.
2. Käyttäjä kuittaa virheilmoituksen-
3. Käyttötapaus päättyy kun järjestelmä on ohjannut käyttäjän takaisin tilaussivulle.



Käyttötapaus 2. robustness diagrammi

UseCase 2:

Kuvaus: Työnantaja hyväksyy asiakkaan laatiman työtilauksen.

Aloitus: Käyttäjä on näkymässä, jossa voi tarkastella yksittäistä työtilausta.

Suoritus:

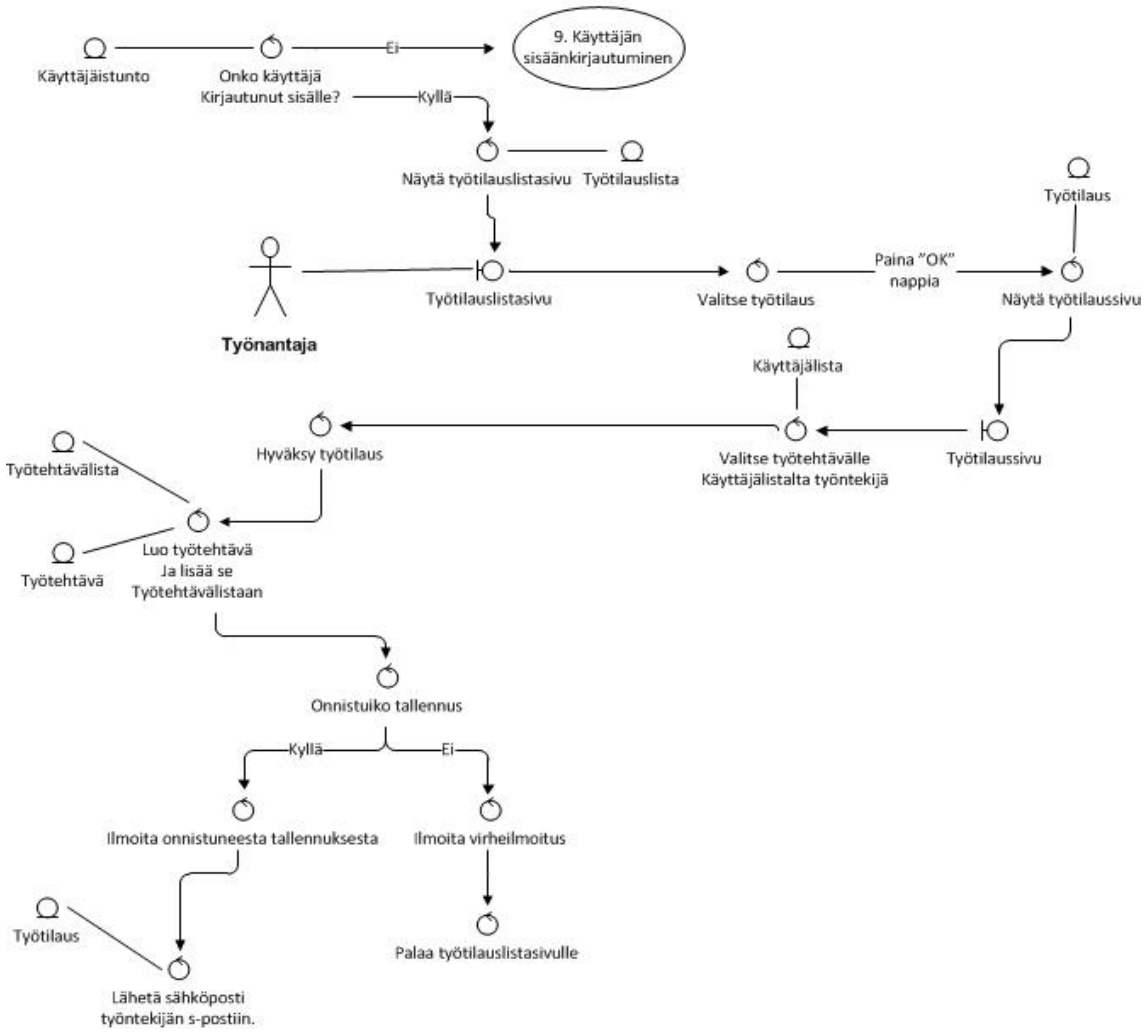
1. Käyttäjä on avannut yksittäisen työtilauksen.
2. Käyttäjä valitsee työtilauksen.
3. Käyttäjä valitsee työtilaukselle työntekijän käyttäjälistalta ja hyväksyy työtilauksen.
4. Järjestelmä luo uuden työtehtävän työtilauksesta.
5. Järjestelmä lisää työtehtävän työtehtävälistaan.
6. Järjestelmä ilmoittaa työntekijää uudesta työtehtävästä. Ilmoitus tapahtuu lähettämällä sähköposti työntekijän s-postiin.
7. Käyttötapaus päättyy kun järjestelmä on ilmoittanut käyttäjää onnistuneesta tallennuksesta.

Vaihtoehtoiset suoritukset:

Käyttäjä ei ole kirjautunut järjestelmään:

1. Käyttäjä ohjataan kirjautumisnäkymään.
2. Kirjautumisen jälkeen käyttäjä ohjataan "Työtehtävälistaussivulle"

Poikkeus 1:
1. Järjestelmä epäonnistuu tallennuksessa.
2. Järjestelmä ilmoittaa epäonnistuneesta tallennuksesta.
3. Käyttötapaus päättyy kun järjestelmä on ohjannut käyttäjän takaisin työtilauslistasivulle.



Käyttötapaus 5. robustness diagrammi

Use Case - <0.1> - <5. Työntekijä vastaanottaa työn>

Kuvaus

Työntekijä vaihtaa työtehtävän tilan, joka kertoo, että työntekijä on vastaanottanut työn.

Toimijat

Työntekijä.

Skenaarion aloitus

Työntekijä on avannut yksittäisen työtehtävän työlistalta.

Esiehdot

Käyttäjä on kirjautuneena järjestelmään työntekijän roolissa.

• Työlistalla on avaamattomia töitä.

Jälkiehdot

• Työtehtävän tila on muuttunut vastaanotetuksi.

Suoritus

1. Käyttäjä valitsee listalta työtehtävän.
2. Käyttäjä muuttaa työtehtävän tilaksi "Avattu".
3. Käyttäjä painaa "Tallenna"-nappia.
4. Käyttötapaus päättyy kun järjestelmä ilmoittaa onnistuneesta muutoksen tallennuksesta.

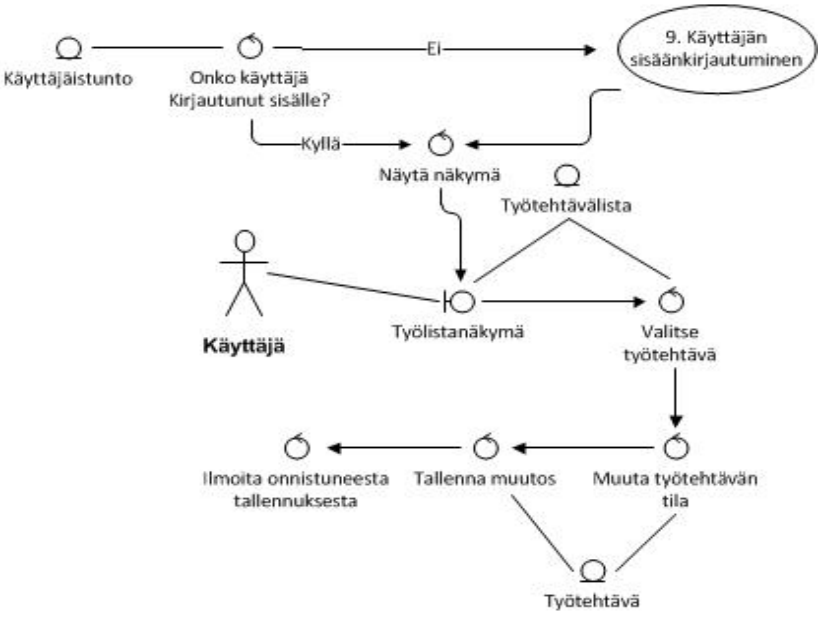
Vaihtoehtoiset suoritukset:

Käyttäjä ei ole kirjautunut järjestelmään:

1. Käyttäjä ohjataan kirjautumisnäkymään.
2. Kirjautumisen jälkeen käyttäjä ohjataan "Työtehtävälistaussivulle"

Erikoisvaatimukset

Käyttöliittymä toimii mobiilialustalla.



Käyttötapaus 8. robustness diagrammi

Use Case - <0.1> - <8. Työntekijä saa työtehtävän valmiiksi>

Kuvaus

Työntekijä muuttaa työn tilan työn edetessä.

Toimijat

Työntekijä.

Skenaarion aloitus

Käyttötapaus alkaa kun työntekijä on avannut yksittäisen työtehtävä työtehtävälistasta.

Esiehdot

- Käyttäjä on kirjautuneena järjestelmään työntekijän roolissa.
- Työtehtävälistalla on valittavia työtehtäviä.

Jälkiehdot

- Työtehtävän tila on muuttunut käyttäjän valitsemaksi tilaksi.
- Työtehtävän poistuu työtehtävälistalta.

Suoritus

1. Käyttäjä valitsee työtehtävän työtehtävälistalta.
2. Järjestelmä näyttää yksittäisen työtehtävän tiedot.
3. Käyttäjä valitsee haluamansa tilan (kts. "Erikoisvaatimukset")
4. Käyttäjä painaa "Tallenna"-painiketta tallentaakseen muutoksen.
5. Käyttötapaus päättyy kun järjestelmä on tallentanut onnistuneesti muutoksen ja ilmoittanut käyttäjää muutoksesta.

Vaihtoehtoiset suoritukset:

Käyttäjä ei ole kirjautunut järjestelmään:

1. Käyttäjä ohjataan kirjautumisnäkymään.
2. Kirjautumisen jälkeen käyttäjä ohjataan "Työtehtävälistaussivulle"

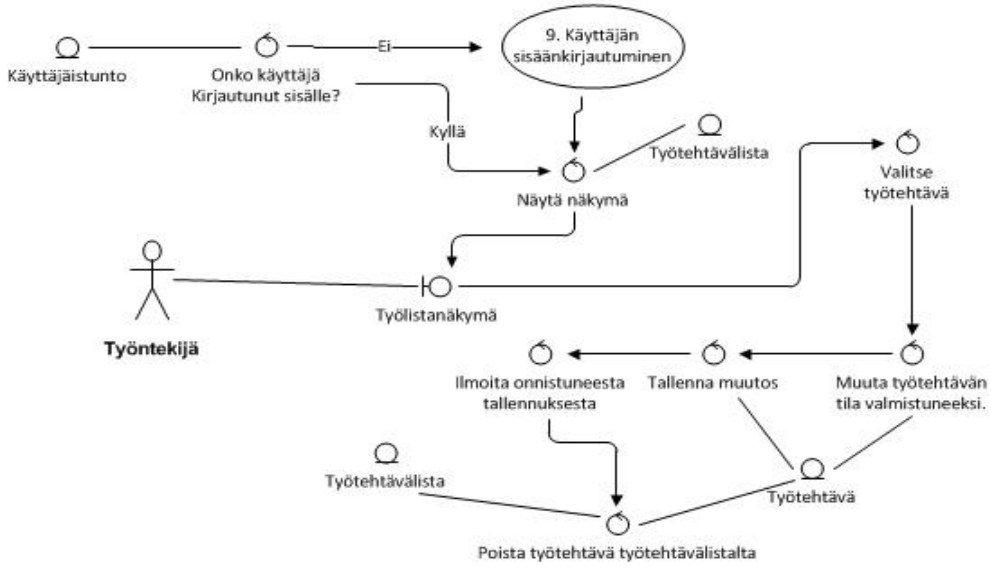
Erikoisvaatimukset

Työtehtävän erilaiset valittavat tilat:

- Valmis
- Aktiivinen

Muita vaatimuksia:

- Järjestelmää käytetään mobiilisovelluksesta.



Käyttötapaus 9. robustness diagrammi

Use Case - <0.1> - <9. Käyttäjän sisäänkirjautuminen>

Kuvaus

Käyttäjä kirjautuu sisään työtilausjärjestelmään.

Toimijat

Työnantaja, varastonhoitaja ja työntekijä

Skenaarion aloitus

Aloitus 1:
• Käyttäjä avaa www-sovelluksen

Aloitus 2:
• Käyttäjä avaa mobiilisovelluksen

Jälkiehdot

Käyttäjä on onnistuneesti kirjautunut sisälle.

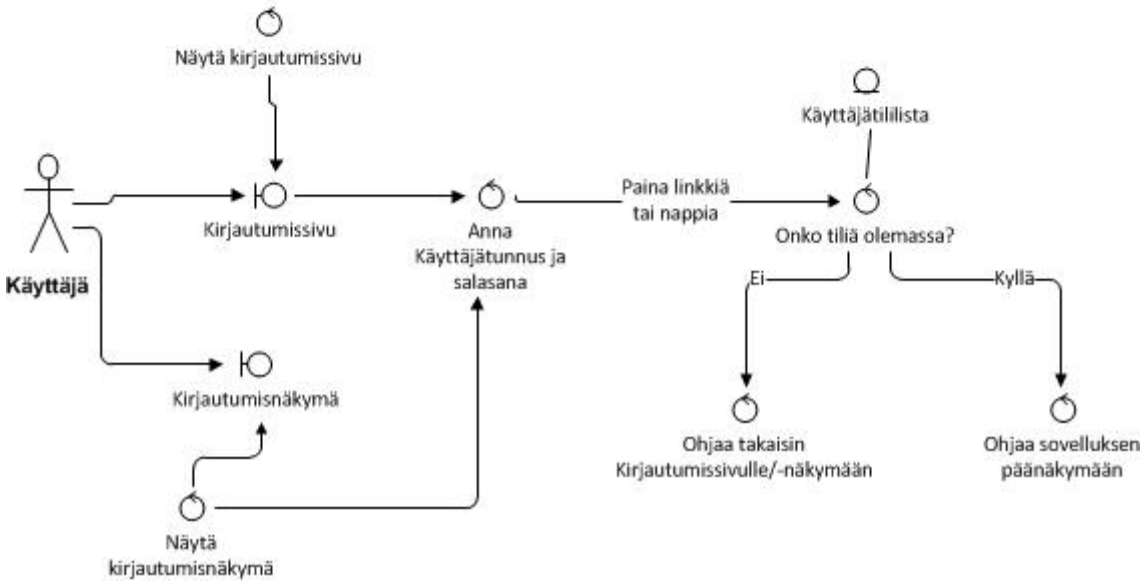
Suoritus

Suoritus 1, käyttäjän kirjautuminen sisälle:

1. Käyttäjä antaa käyttäjätunnuksen "Käyttäjätunnus"-kenttään.
2. Käyttäjä antaa salasanan "Salasana"-kenttään.
3. Käyttäjä painaa "Kirjaudu sisälle"-linkkiä tai nappia.
4. Järjestelmä tarkastaa löytyykö käyttäjätiliä annetuilla arvoilla.
5. Käyttötapaus päättyy kun käyttäjä on kirjautunut sisälle ja järjestelmä ohjannut käyttäjän avaussivulle.

Vaihtoehtoinensuoritus:

1. Jos käyttäjätiliä ei löydy, ohjataan käyttäjä takaisin kirjautumissivulle.



Palvelukerroksen REST API

Met-hod	URL template	Parameters	Class	Description
POST	*\employee.model.fi.employee	{username}, {password}	employee.model.fi.employee	Returns user data. The User is searched by username and password
POST	*\employee.model.fi.finalreport	{workrequest_id}, {description}	employee.model.fi.finalreport	Create new Finalreport
POST	*\employee.model.fi.workorder	{user_id}, {workrequest_id}, {description}	employee.model.fi.workorder	Create new Workorder
GET	*\employee.model.fi.workorder\{id}		employee.model.fi.workorder	Get Workor-ders by user id
POST	*\employee.model.fi.workrequest	{client_name}, {client_address}, {cient_email}	employee.model.fi.workrequest	Create new Workorequest