TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Tutkintotyö

Simon Götz

SYSTEM-KONFIGURAATION KÄYTTÖ I2 INTELLIGENT SELLING SOLUTIONS -OHJELMISTOON PERUSTUVASSA KONFIGUROINTIJÄRJESTELMÄSSÄ

Työn ohjaaja lehtori Erkki Hietalahti
Työn teettäjä TAS Solutions Oy, valvojana DI Tomi Nakari
Tampere 2006

TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikka
Ohjelmistotekniikka

Götz, Simon

System-konfiguraation käyttö i2 Intelligent Selling Solutions -ohjelmistoon perustuvassa konfigurointijärjestelmässä

Tutkintotyö 37 sivua + 10 liitesivua
Työn ohjaaja lehtori Erkki Hietalahti
Työn teettäjä TAS Solutions Oy, valvojana DI Tomi Nakari
Helmikuu 2006
Hakusanat i2, Intelligent Selling Solutions, tuoterakenne

Tiivistelmä

Projektin tarkoituksena oli toteuttaa Moventas Oy:lle Internet-pohjainen sovellus vaihteiden myyntitarjousten luomiseen. Toteutuksessa käytettiin i2-yhtiön tuottamaa Intelligent Selling Solutions -sovellusta, joka vastaa tarjousten kaupankäyntilogiikan hallinnasta. Projektin tarkoituksena oli tehdä tähän sovellukseen asiakkaan haluama käyttöliittymä ja toteuttaa sen eri toiminnot. Toteutustapana oli Java Servlet Pages -teknologia. Tutkintotyössä esitellään tehtyyn sovellukseen toteutettu tuotteiden puumaisen rakenteen hallintatyökalu ja sen suunnittelu, toteutus, testaus ja käyttöesimerkki. Toteutuksessa tuotteet muodostavat puumaisen hierarkkisen rakenteen, josta voidaan erotella versio-, sektio-, tuote- ja moduulitasot. Toteutettu hallintatyökalu jakautuu kolmeen osaan, jotka vastaavat versioiden, sektioiden ja tuotteiden sekä moduuleiden hallinnasta. Tämä työkalu oli keskeinen osa projektin toteutuksessa. Lähes kaikki käyttöliittymän toiminnot käyttävät sitä käsitellessään tarjousta.

TAMPERE POLYTECHNIC
Computer Systems Engineering
Software Engineering

Götz, Simon	Use of system configuration in configurator software based on i2 Intelligent Selling Solutions software
Engineering Thesis	37 pages, 10 appendices
Työn ohjaaja	lehtori Erkki Hietalahti
Commissioning Company	TAS Solutions Oy, supervisor DI Tomi Nakari
February 2006	
Keywords	i2, Intelligent Selling Solutions, product hierarchy

Abstract

The purpose of the project was to implement an internet based solution for Moventas Oy for configuring quotations of gear units. This was done using i2 Intelligent Selling Solutions software for handling financial and logistical aspects of the solution, and integrating a user interface and its functionality into the solution. The project was done using Java Servlet Pages technology. A control tool was build for managing the tree like hierarchical structure of products in quotations. This tool and its design, implementation and testing is described here. In the implemented solution, quotations are made of different levels in product structure: version, section, product and module level. The control tool has three distinct sections for handling versions, sections and both products and modules. This tool was a central part of the project, where almost every functionality uses this tool for accessing and managing the quotation's data.

Sisällysluettelo

Kuvaluettelo

Käytetyt erikoistermit

system-konfiguraatio	TSystemConfiguration-luokka pohjainen tarjousrakenne
TSystemConfiguration	toteutuksen pohjana oleva Java-luokka
TPricingParameters	hinnoittelutiedot sisältävä Java-luokka
TSystemConfigNode	hierarkiarakenteen haaran muodostava Java-luokka
TConfigContent	tuotteen tiedot sisältävä Java-luokka
hierarkiarakenne	tarjouksen tuotteiden hierarkkinen puurakenne
versio, versiotaso	hierarkiarakenteen 1. taso
sektio, sektiotaso	hierarkiarakenteen 2. taso
tuote, tuotetaso	hierarkiarakenteen 3. taso
moduuli, moduulitaso	hierarkiarakenteen 4. taso
TSystemConfigVariable	TSystemConfigNode-luokkaan liittyvien muuttujien Java-luokka
VersionControl	versiotasoa hallitseva Java-luokka
Section	sektiotasoa hallitseva Java-luokka
ProductControl	tuotetasoa hallitseva Java-luokka
aktiivinen versio	käyttäjälle näkyvä versio
perustiedot-sivu	käyttöliittymän päänäkymä
yleistiedot	kaikille tuotteille yhteiset tiedot
muut kustannukset	tuote, joka sisältää mm. kuljetuskustannukset
rakenneryhmä	sektio
käyttöpiste	tuote
rakenneryhmien hallinta	sektioiden hallintanäkymä
sektioiden hallintanäkymä	näkymä, jossa hallitaan sektioita

Käytetyt tekstityylit

`koodirivi`	Java-koodia
kopioi	käyttöliittymässä esiintyvä teksti

1 JOHDANTO

Tutkintotyö keskittyy kuvaamaan TAS Solutions Oy:n Moventas Oy:lle tekemän projektin yhtä osakokonaisuutta. Tämä kokonaisuus on keskeisessä osassa halutun toiminnallisuuden toteuttamisessa tehtyyn projektiin. Tätä osa-aluetta päädyttiin käsittelemään sen keskeisyyden takia ja koska koko projekti on aivan liian laaja tutkintotyön kohteeksi.

1.1 Työn teettäjä ja tavoite

Tutkintotyö on tehty TAS Solutions Oy -yritykselle. Työn tavoitteena on ollut tuottaa Moventas Oy:lle paperikoneen vaihteiden ostotarjouksen luontisovellukseen toiminnallisuus ja käyttöliittymä. Myyntinimikkeiden ja hinnoittelutietojen ja -toimintojen hallinta tapahtuu i2-yhtiön toimittamassa Intelligent Selling Solution (iss) -sovelluksessa. TAS Oy:n tehtävänä on ollut rakentaa asiakkaan haluama toiminnallisuus ja käyttöliittymä ja integroida se iss-sovelluksen toimintaan.

1.2 i2-yhtiö

I2 on amerikkalainen yhtiö, joka on erikoistunut toimitusketjun optimointiin. Yritys tekee ohjelmistoja, joiden avulla voidaan tehostaa tuotteiden tai palveluiden siirtyvyyttä valmistajan ja asiakkaan välillä. Näillä ohjelmilla hallitaan mm. tuotantoketjua, tuotteiden kysynnän muutoksia ja kuljetuslogistiikkaa.

1.3 i2 Intelligent Selling Solution -sovellus

Intelligent Selling Solution -sovellus on i2:n tuottama ohjelmisto, jolla hallitaan sähköisen kaupankäynnin taustalla olevaa logiikkaa. Ohjelmisto pitää kirjaa tarjolla

olevista tuotteista, hinnastosta, tehdyistä tarjouksista sekä tuotteiden valintalogiikasta. Iss-sovellus tarjoaa sovelluskehittäjille Java-rajapinnan, jonka avulla ohjelman toimintaa voidaan ohjata. Tällä tavalla kullekin asiakkaalle voidaan räätälöidä yksilöllinen käyttöliittymä ja tarjousten hallintamalli.

2 VAIHTEIDEN MYYNTITARJOUSTEN LUONTISOVELLUS

Projektin tarkoituksena on ollut luoda Internet-pohjainen sovellus vaihteiden myyntitarjousten luomiseen. Tällöin myyntihenkilöstö voi käyttää järjestelmää mistä tahansa maailmasta ja tiedot tallentuvat keskitetysti sovelluksen käyttämään tietokantaan.

Luodut tarjoukset ovat säilössä tietokannassa, josta niitä voidaan hakea monien eri hakuehtojen perusteella. Luotujen tarjousten muokkausta ei ole rajoitettu.

Tarjouksen sisältö näytetään koko ajan ostoskorissa. Näin käyttäjä näkee aina luomansa tarjouksen tuotteet ja voi muokata tuotteita valitsemalla sopiva toiminto ostoskorissa.

2.1 Tuotehierarkialle asetetut vaatimukset

Projektissa oli tarpeen luoda rakenne, jossa tarjouksen tuotteet ja tiedot voidaan säilöä hierarkkisena rakenteena, jossa kukin taso vastaa tiettyä kategoriaa. Päätasolle tallennetaan tarjouksen tiedot, kuten asiakas ja erääntymispäivämäärä. Toteutetun rakenteen tulee tukea tarjouksen versiointia, jossa valitusta versiosta voidaan luoda uusi identtinen versio. Kukin versio sisältää versioittaisia tietoja tarjouksesta sekä oman tuoterakenteensa.

Käyttöliittymä näyttää käyttäjälle päänäkymässä kaikille versioille yhteiset, tarjouksittaiset, tiedot, sekä aktiivisen version tiedot. Muiden näkymien kautta saadaan tarkempaa tietoa tarjouksen rakenteesta ja tiedoista.

Tarjouksen ensimmäistä versiota ei voida poistaa, muuten versioita voidaan luoda ja tuhota vapaasti. Käyttäjä voi vapaasti valita aktiivisen version olemassa olevista. Nämä toiminnot on mahdollista tehdä käyttöliittymän päänäkymästä.

Tuotteita pitää voida jakaa eri kategorioihin, sektioihin. Kullakin versiolla voi olla rajoittamaton määrä sektioita, jotka sisältävät tuotteita.

Sektioita voidaan tuhota ja luoda tarpeen mukaan. Uusi sektio luotaessa sille annetaan nimi ja sijainti tuoterakenteessa. Sektioiden ja niiden sisältämien tuotteiden järjestystä voidaan muuttaa erillisessä näkymässä. Lisäksi tuotteita voidaan siirtää toisen sektion alaisuuteen.

Tuotteet koostuvat moduuleista, jotka ovat tarjouksen pienimpiä yksikköjä. Kukin moduuli vastaa yhtä tuotenimikettä, esim. nivelakseli. Tuotteen sisältämät moduulit määräytyvät tuotetta luotaessa käyttäjän valintojen perusteella.

Versiot, sektiot, tuotteet ja moduulit muodostavat tarjouksen puumaisen tuotehierarkian, joka esitetään kuvassa 2. Järjestelmä näyttää tuotehierarkian aktiiviselle versiolle ja erottelee kunkin hierarkiatason toisistaan värin perusteella. Esimerkki tästä on hinnoittelunäkymä, joka on kuvattu esimerkkinä luvussa 5.

2.2 Käytetty ympäristö

Sovellus on Internet-pohjainen, joten käyttäjä voi käyttää sitä millä tahansa laitekokoonpanolla, jolla voidaan käyttää graafista Internet-selainta.

Sovellus itse toimii Windows 2000 -palvelimella, BEA Weblogic 8.1 Internet-palvelinohjelmiston avulla. Käyttöliittymä, jota Weblogic ajaa, on yhteydessä i2 Intelligent Selling Solution 6.1.4.1 -sovellukseen, jonka tehtävänä on huolehtia kaupankäyntilogiikasta sekä tietojen tallennuksesta tietokantaan. Tietokantana toimii Oracle 9.2.

Työn teossa on käytetty etupäässä JSP-teknologiaa sekä HTML- ja Javascript-

kieliä. Toteutus pohjautuu hyvin pitkälle iss-sovelluksen tarjoaman Java API -rajapinnan käyttöön.

2.3 JSP-teknologia

Projektissa käytettiin Java servlet pages (JSP) -teknologiaa. Tämä teknologia mahdollistaa Java-koodin liittämisen suoraan HTML-sivulle, jossa se suoritetaan aina ennen sivun näyttämistä.

Java-koodi erotetaan HTML-koodista käyttämällä erityisiä tunnistinmerkkejä, jotka ilmaisevat Java-koodiosuuden alku- ja loppukohdat. Tämä koodi suoritetaan aina sivun latauksen yhteydessä. Java-koodiosuus on näkymätöntä siirrettävässä HTML-koodissa, mutta siinä suoritetut tulostusvirtaa käyttävät tulostukset lisätään HTML-koodiin siihen kohtaan, missä tunnistinmerkit ilmaisivat Java-koodilohkon sijaitsevan. Näin ollen asiakasselain ei koskaan näe Java-koodiosuuksia, mutta ne suoritetaan aina sivu ladattaessa ja niiden avulla voidaan luoda dynaamista sisältöä sivuun.

Internet-palvelinohjelma kääntää Java-koodiosuudet automaattisesti, kun se havaitsee tiedostoa muokatun. Täten sovelluskehitys on helppoa, kun muutokset tulevat voimaan ilman palvelinohjelmiston uudelleenkäynnistämistä eikä muokattua koodia tarvitse etukäteen kääntää. Java-koodin syntaksivirheet tulostuvat palvelinohjelmiston lokiin.

JPS-teknologia valittiin, koska iss-sovellus tarjoaa Java API -rajapinnan ja JSP-teknologia on joustavampaa käyttää, kuin servlet-teknologia. Tämän lisäksi i2-yhtiön oletustoteutus iss-sovelluksen käyttöliittymäksi käyttää JSP-teknologiaa, joten samaa teknologiaa käyttämällä voitiin käyttää hyväksi osia oletustoiminnallisuudesta.

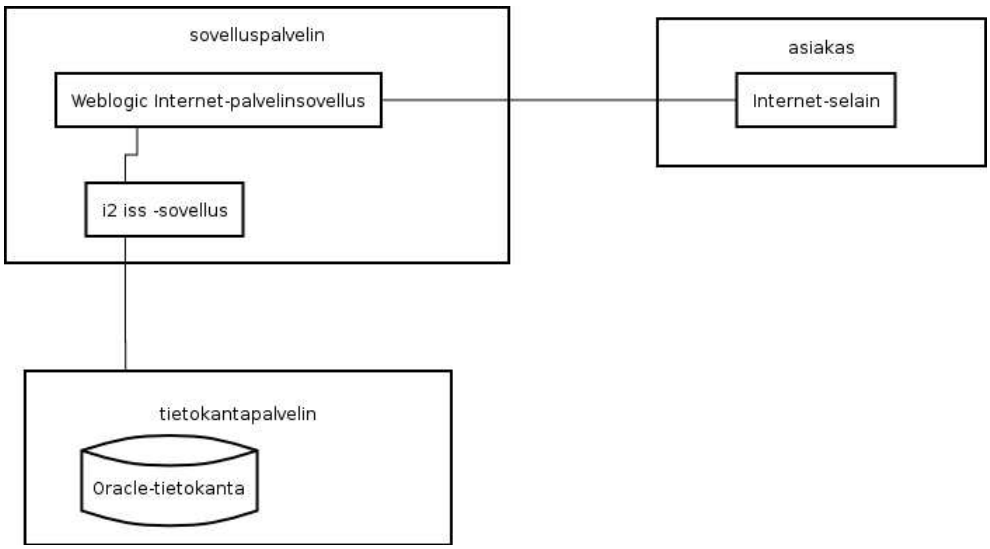
2.4 Tietokannan käyttö

Tietokantaa käytetään iss-sovelluksen tarjoaman Java API -rajapinnan avulla. Käyttöliittymäkoodin ei ole tarkoitus käsitellä tietokantaa suoraan, vaan kaikki tapahtumat suoritetaan tarjotun rajapinnan avulla.

Iss-ohjelmiston Data server -komponentti vastaa yhteydestä tietokantaan. Kun suoritetaan toiminto, joka tarvitsee tietoa tietokannasta, engine server -komponentti suorittaa rajapinnan toimenpiteen ja lähettää kutsun data server -komponentille, joka hakee tiedon tietokannasta.

2.5 Palvelin-asiakas arkkitehtuuri

Toteutus muodostaa kolmitasoisen palvelin-asiakas ohjelmiston, joka on kuvassa 1. Eri tasot ovat asiakas, Internet-palvelin ja tietokantapalvelin. Nämä voivat sijaita fyysisesti joko samassa tai eri palvelimilla. Normaalisti asiakasohjelmisto sijaitsee aina fyysisesti erillään palvelinohjelmistosta ja Internet-palvelin ja tietokantapalvelin ovat eri koneita tehokkuuden parantamiseksi. Kehityskäytössä nämä kaikki tasot sijaitsevat samalla tietokoneella.



Kuva 1. Palvelin-asiakas arkkitehtuuri

3 SYSTEM-KONFIGURAATIO -TOTEUTUS

Projektiin valittiin toteutustavaksi system-konfiguraatio -pohjainen lähestyminen. Tässä tavassa ostoksia hallitaan hierarkiarakenteella, jonka pohjalla on TSystemConfiguration-luokka.

System-konfiguraatio on iss-ohjelmiston versiossa 6 käytetty tapa toteuttaa tarjouksia. Aiemmissa versioissa on ollut käytössä TProject-luokkaan pohjautuva tarjousrakenne.

TProject-luokkaan pohjautuva ratkaisu olisi ollut monin osin parempi vaaditun tuoterakenteen toteuttamiseen, mutta iss-ohjelmiston version 6 myötä i2-yhtiön tarjoama tuki TProject-luokkaan tukeutuville ratkaisuille on vähentynyt. System-konfiguraatio pohjaiseen toteutukseen päädyttiin, koska i2-yhtiö on keskittynyt tuotekehityksessään siihen ja näin ollen sitä tuetaan paremmin ja kauemmin, kuin TProject-luokkaan pohjautuvia ratkaisuja.

3.1 TSystemConfiguration-luokka

TSystemConfiguration-olio on kunkin tarjouksen hierarkian ylin olio. Se sisältää toiminnallisuuden mm. asiakkaan määräämiseksi, koko tarjouksen hintatietojen laskentaan sekä tiedon tarjouksen luojasta ja luontiajasta.

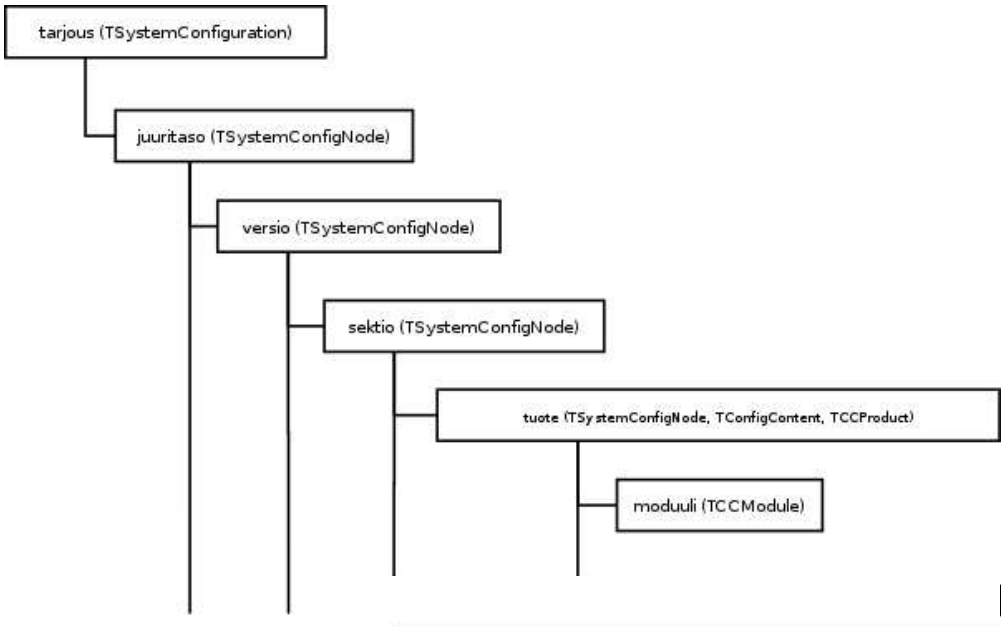
Valitussa toteutuksessa TSystemConfiguration-luokan oliota käytetään tarjouksen juuritason objektina. Siihen tallennetaan tieto tarjouksen luojasta, luontipäivämäärästä ja kellonajasta, tarjouksen nimi sekä asiakas. Luokan sisältämään TPricingParameters-olioon määritellään tarjouksessa käytetty valuutta ja valuutanvaihtokurssi.

TSystemConfiguration-oliolta saadaan TSystemConfigNode-luokan olio, joka on hierarkiassa välittömästi system-konfiguraation alapuolella. Nämä oliot muodostavat tuotehierarkian.

3.2 Tarjouksen hierarkiarakenne

Tarjouksen hierarkia noudattaa kuvan 2 muotoa. Ylimpänä on TSystemConfiguration-luokka, jonka alapuolella on TSystemConfigNode-luokista koostuva puurakenne. Ensimmäisellä alatasolla ovat tarjouksen eri versiot. Versioiden alapuolella ovat sektiot, jotka puolestaan sisältävät tuotteita. Kaikki nämä ovat TSystemConfigNode-luokan olioita. Tuotehaaralta saadaan TConfigContent-tyyppinen olio, joka sisältää tuotteen hinnoittelutiedot TPricingParameters-luokan olion muodossa. TConfigContent-olio sisältää yhden tuoteolion (TCCProduct), joka voi pitää sisällään useita moduuleja (TCCModule).

Toteutettu hierarkiarakenne pohjautuu suoraan iss-sovelluksen aiemmissa versioissa käytettyyn TProject-luokan alaiseen rakenteeseen, jossa oli projekti-, versio-, projektiosa-, tuote- ja moduulitasot. Nämä muodostivat samanlaisen rakenteen, kuin tässä toteutettiin system-konfiguraation avulla.



Kuva 2. Tarjouksen hierarkiarakenne

Hierarkiarakenteen tarkoituksena on ensinnä mahdollistaa tarjouksen eri versiot. Toisekseen täytyi olla mahdollista jakaa tuotteita alikategorioihin kunkin version sisällä. Sektiot toteuttavat tämän vaatimuksen.

Käytännössä sektiot nimetään sen laitteen osion mukaan, johon ne sijoitetaan. Näin tiettyyn kokonaisuuteen kuuluvat vaihteet ovat selkeästi omana ryhmänään. Esimerkiksi paperikoneisiin voi tulla kymmeniä vaihteita. Tällöin sektiot ovat tarpeen selkiyttämään tarjouksen rakennetta.

3.2.1 Juuritaso

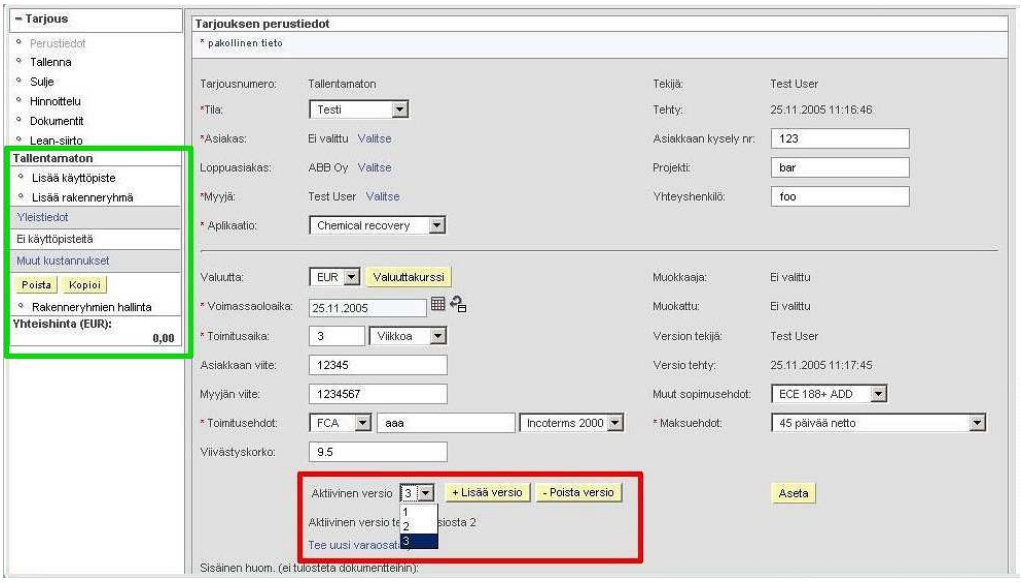
Tarjoushierarkian juuritaso muodostuu TSystemConfiguration-oliosta ja siltä saatavasta TSystemConfigNode-olioista. Näihin kahteen olioon tallennetaan kaikki tieto, joka on yhteistä koko tarjoukselle. Mikäli TSystemConfiguration-luokka sisältää kyseiselle tiedolle tarkoitetun tietokentän, sitä käytetään. Muussa tapauksessa tieto tallennetaan TSystemConfigNode-luokan alaiseksi muuttujaksi luokkaan TSystemConfigVariable. Arvot tallennetaan tekstimuodossa nimillä, jotka ovat yksilöllisiä kyseiselle tiedolle.

3.2.2 Versiotaso

Versiotaso erottaa tarjouksen hierarkiassa eri versiot toisistaan. Kukin versio saa oman järjestysnumeronsa, joka toimii myös version nimenä. Tarjouksen versioittaiset tiedot tallennetaan versioon TSystemConfigVariable-olioina. Versiokohtaisia tietoja ovat mm. tarjouksen numero, projekti ja yhteyshenkilö.

Tarjous sisältää aina vähintään yhden version. Aktiivinen versio määritellään version tietoihin asettamalla muuttujan `IS_ACTIVE` arvoksi `true`. Tarjouksen sisältämien versioiden määrää ei ole rajoitettu.

Käyttäjälle näkyvät kerrallaan yhden, aktiivisen version tiedot. Päästäkseen käsiksi toiseen versioon on käyttäjän vaihdettava aktiivista versiota.



Kuva 3. Tarjouksen päänäkymä. Versioiden hallinnan osio on korostettu punaisella ja ostoskori vihreällä.

Versioita hallitaan käyttöliittymän perustiedot-näkymästä, joka on esitetty kuvassa 3. Näkymästä on mahdollista vaihtaa aktiivista versiota, luoda uusi versio ja tuhota aktiivinen versio. Kuvassa, punaisella korostetulla alueella, näkyy alasvetovalikko, josta valitaan aktiivinen versio ja sen vieressä napit version lisäämiseen ja poistamiseen. Vasemmalla oleva vihreällä ympyröity alue on ostoskori. Ostoskorin yläosassa on linkit uuden tuotteen ja sektion luomiseen, keskellä listaus tarjouksen sisällöstä ja alaosassa napit tuotteen poistamiseen tai kopioimiseen, linkki sektioiden hallintanäkymään ja tarjouksen kokonaishinta. Perustiedot-näkymän ylempi osa sisältää tarjouksittaisia tietoja ja alempi, sivun katkaisevalla viivalla erotettu, osa versioittaisia tietoja.

3.2.3 Sektiotaso

Kukin versio voi sisältää rajattoman määrän sektioita. Ne on tarkoitettu ryhmittämään samaan kokonaisuuteen kuuluvat tuotteet yhteen. Tarjouksessa voisi olla esimerkiksi sektio kuivain, johon kuuluisivat kuivaimeen kuuluvat tuotteet.

Sektio on hierarkiassa TSystemConfigNode-luokan olio. Tähän olioon tallennetaan sektioittainen tieto, jota tarvitaan tuoterakenteen hallinnassa.

Hierarkiassa version alla on aina yksi käyttäjälle näkymätön sektio, jota kutsutaan tilapäissektioksi. Tämän sektion alle tulevat tuotteet, jotka näkyvät käyttäjälle suoraan versiotason alapuolella. Tilapäissektion avulla voidaan luoda tuotteita, jotka eivät käyttöliittymässä näytä kuuluvan mihinkään sektioon, mutta fyysisesti sijaitsevat tilapäissektiossa. Muut sektiot ovat käyttäjän luomia ja näin ollen myös näkyvät normaalisti käyttäjälle.

3.2.4 Tuotetaso

Sektiot pitävät sisällään tuotteita, jotka ovat luokan TCCProduct ilmentymiä. Sektiotason alapuolella on TSystemConfigNode-olio, joka sisältää TConfigContent-olion. TConfigContent-luokka sisältää tuotteen tietoja: hinnoittelutiedot, lukumäärän, asiakkaan ja kielen sekä muita tietoja, jotka ovat tämän projektin kannalta epäolennaisia.

TConfigContent-luokalta saadaan TCCProduct-olio, joka kuvaa yhtä tuotetta. Se sisältää tarkat tiedot siitä, mikä tuote on kyseessä: nimen, koodin ja TBoxItem-olion, joka kuvaa tuotteen ominaisuuksia rakenteessa: luokan, koodin, kuvauksen, makron, nimen sekä voimassa olevan hinnan.

Tuotteet näkyvät käyttäjälle ostoskorissa ja hinnoittelunäkymässä. Kukin tuote näytetään hierarkian mukaisesti sen sektion alapuolella, johon se kuuluu. Poikkeuksen muodostavat tilapäissektion tuotteet, jotka näkyvät hierarkiassa ensimmäisinä, suoraan versiotason alapuolella.

Hierarkian yhdenmukaisuuden ja selkeyden säilyttämiseksi päädyttiin ratkaisuun, jossa käytetään tilapäissektiota versiotason tuotteiden säilyttämiseen. Järjestelmä mahdollistaisi myös näiden tuotteiden liittämisen suoraan versiotason alapuolelle, mutta tämä hankaloittaisi hierarkiarakenteen hallintaa sekoittamalla sektiota ja tuotteita samalle tasolle. Tällöin kaikille tason haaroille tulisi tehdä tyyppitarkastelu, mikä kuluttaisi turhaan resursseja valittuun ratkaisuun verrattuna ja tekisi toteutuksesta turhaan monimutkaisen.

3.2.5 Moduulitaso

Kukin tuote sisältää moduuleita, jotka ovat luokan TCCModule-olioita. Näitä ovat mm. vaihteet ja kytkimet. Moduuleilla on sisäinen hierarkiataso, joka määrää, minkä tyyppinen moduuli on kyseessä. Vain tasolla TCCModule.MODULE olevat moduulit näkyvät käyttäjälle hinnoittelunäkymässä. Tällä hierarkialla ei ole mitään tekemistä kuvatun system-konfiguraatio -hierarkian kanssa.

3.3 System-konfiguraation käyttö

Kukin tarjous sisältää yhden TSystemConfiguration-olion. Tämä olio on kytketty samannimiseen TCart-olioon, joka huolehtii tarjousten tallennuksesta ja tallennettujen tarjousten hausta.

Uutta tarjousta tehtäessä tai olemassa olevaa tarjousta avattaessa, suljetaan ensin mahdollisesti avoinna oleva vanha system-konfiguraatio, minkä jälkeen avataan tai luodaan uusi TCart-olio. Tämä avaa automaattisesti TSystemConfiguration-olion.

3.3.1 System-konfiguraatio -hierarkian hallinta

Tuotteiden hierarkian hallitsemiseksi tarjotut valmiit ominaisuudet i2 iss API:ssä todettiin projektin tarpeisiin riittämättömiksi. Syntyi tarve tehdä oma projektiin soveltuva ratkaisu, joka hallitsee hierarkian juuritasolta moduuleihin. Näin syntyi luokka VersionControl. Luokan alkuperäinen tehtävä oli uusien versioiden luonti, olemassa olevien tuhoaminen ja aktiivisen version vaihtaminen. Tämän lisäksi luokkaan liitettiin uutena piirteenä sektiot, jotka erottelevat eri ryhmiin kuuluvia tuotteita. Tästä osasta tehtiin oma sisäinen luokka Section. Myöhemmässä vaiheessa todettiin tarpeelliseksi lisätä tuotteiden ja moduulien hallintaan liittyviä ominaisuuksia, ja niin tehtiin sisäinen luokka ProductControl.

Hierarkiarakenteen hallintaan voidaan liittää seuraavat käyttötapaukset:

- version luominen
- aktiivisen version tuhoaminen
- aktiivisen version vaihtaminen
- sektion luominen
- sektion tuhoaminen
- sektion siirtäminen
- tuotteen tuhoaminen
- tuotteen siirtäminen
- tuotteen kopioiminen

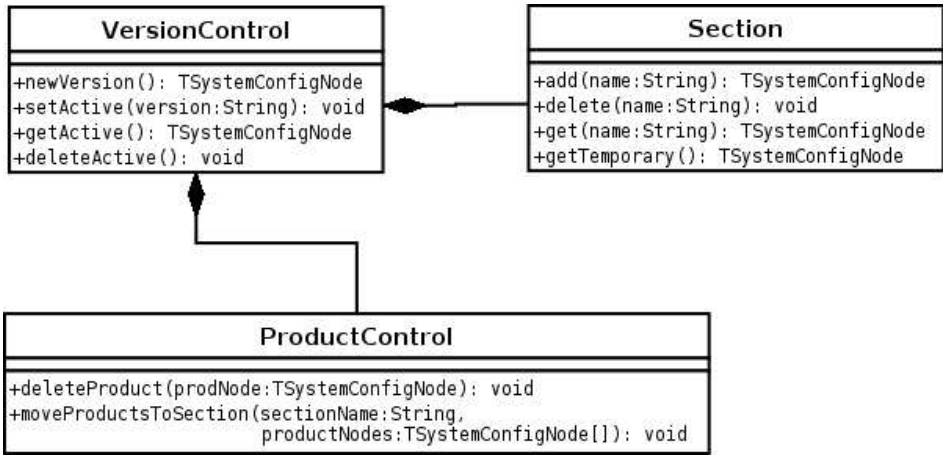
Nämä voidaan vielä jakaa kolmeen osa-alueeseen: versioiden, sektioiden ja tuotteiden käyttötapauksiin. Tuotteen luomista ei ole mainittu, koska se tapahtuu erityisen tähän tarkoitukseen tehdyn käyttöliittymän ja toiminnallisuuden mukaisesti eikä liity toteutettuun hierarkiarakenteen hallintaan.

Luokka VersionControl toteuttaa versioiden käyttötapausten vaatimukset, Section-luokka sektioiden käyttötapausten toiminnallisuuden ja ProductControl-luokka tuotteiden käyttötapausten toiminnot. VersionControl-luokan rakenne ja tärkeimmät metodit on esitetty kuvassa 4.

Sisäisiin luokkiin päädyttiin selkeyden ja ylläpidettävyyden takia. Oli tarpeen jakaa yhä paisuvaa joukkoa hierarkiaa käsitteleviä metodeja, jotka kuitenkin tarvitsivat muita saman luokan metodeja toimiakseen. Toteuttamalla osiot sisäisinä luokkina sallitaan olioiden pääsy ulkoisen luokan attribuutteihin ja metodeihin ja eriytetään sektioiden ja tuotteiden hallinnan toiminnallisuus yleisistä versioittaisista toiminnoista.

Luokalla VersionControl on julkisina attribuutteina sekä Section- että ProductControl-luokkien ilmentymät. Näin sallitaan version hallintaa käytettäessä helppo pääsy sektioiden ja tuotteiden hallintaan.

VersionControl-luokka on keskeisin luokka projektin toteutuksessa. Kaikki hierarkiarakennetta käsittelevät luokat käyttävät sitä joko suoraan tai välillisesti luokan HierarchyPrinter kautta.



Kuva 4. Luokan VersionControl rakenne ja tärkeimmät metodit UML-kaaviona

3.3.1.1 Versiot

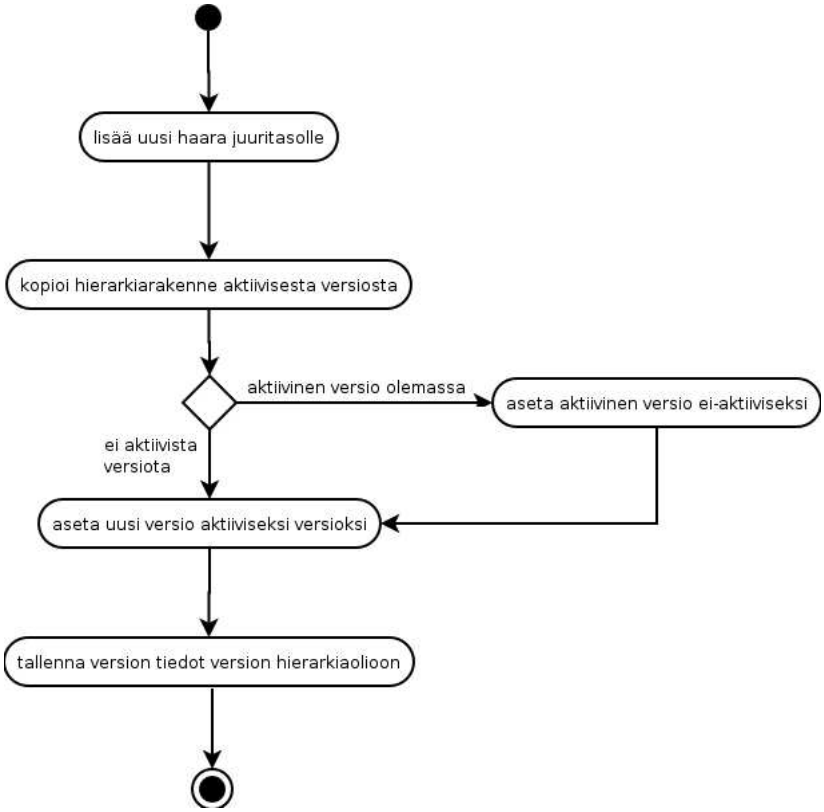
Versiot ovat hierarkiassa välittömästi juuritason alapuolella olevia TSystemConfigNode-luokan ilmentymiä, versio-olioita. Versioon liittyvät tiedot tallennetaan versio-olioon tekstimuodossa. Näitä ovat sekä hierarkiarakenteen hallintaan käytetyt tiedot että tarjouksen versioittaiset tiedot, kuten toimitusaika.

Jokaisessa tarjouksessa on aina vähintään yksi versio, numero 1, jota ei voida poistaa. Uutta versiota luotaessa aktiivisen version sisältämät sektiot, tuotteet ja tarjouksen versioittaiset tiedot kopioituvat uudelle versiolle.

Versioihin liittyvät toiminnallisuudet on toteutettu VersionControl-luokassa. Tärkeimmät toiminnallisuudet ovat version luonti, tuhoaminen ja vaihtaminen. Nämä toteuttavat versiotason käyttötapaukset. Versioihin liittyvien käyttötapausten käyttämä Java-koodi on liitteessä 1.

Uuden version luonti

Uusi versio luodaan lisäämällä hierarkian juuritasolle uusi haara, johon kopioidaan tiedot sillä hetkellä aktiivisesta versiosta. Juuri luotu versio asetetaan aktiiviseksi ja siihen tallennetaan hierarkiarakenteen hallinnassa käytettäviä tietoja. Kuvassa 5 on uuden version luonnin aktiviteettikaavio.



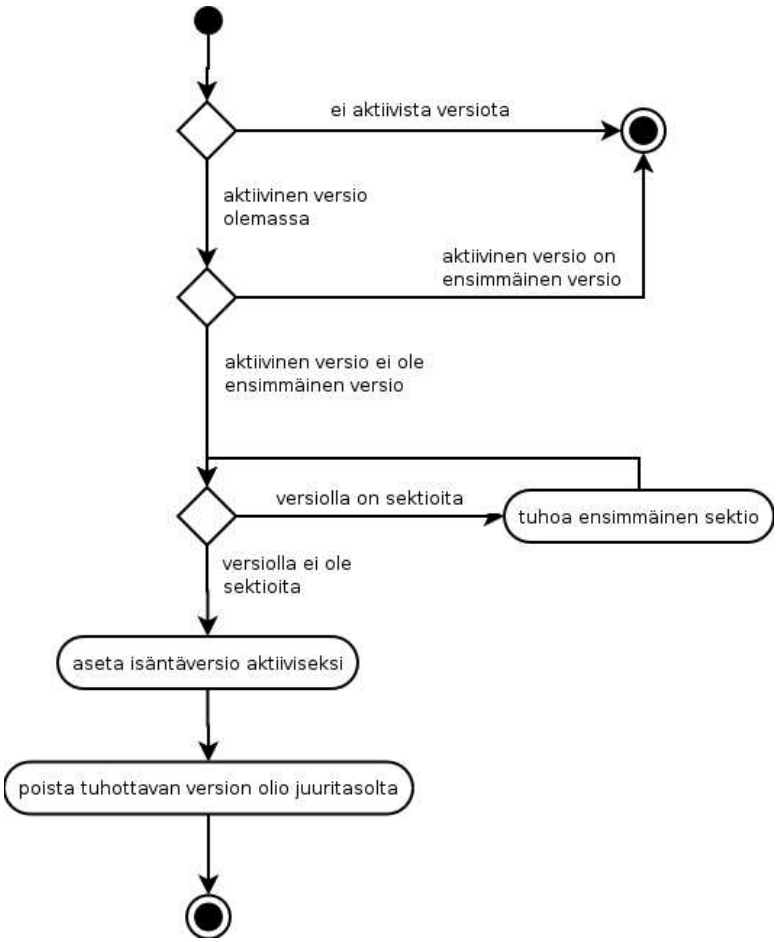
Kuva 5. Uuden version luonnin aktiviteettikaavio

Tätä toiminnallisuutta käytetään käyttäjän valitessa uuden version luonti perustiedot-sivulla. Lisäksi tehtäessä uusi tarjous luodaan samalla automaattisesti tarjoukseen yksi versio. Myös varaosatarjousta tehtäessä tai päivitettäessä luodaan varaosatarjoukseen uusi versio.

Version poistaminen

Vain aktiivinen versio voidaan poistaa. Ensimmäisen version poistaminen ei kuitenkaan ole sallittua.

Tuhottaessa versio, on ensin poistettava sen alainen hierarkiarakenne. Tuhoaminen aloitetaan tuotteista ja niiden jälkeen poistetaan tuotteet sisältänyt sektio. Lopuksi tuhottava versio poistetaan hierarkiasta ja sen isäntäversio asetetaan aktiiviseksi. Aktiviteettikaavio version poistamisesta on kuvassa 6.



Kuva 6. Version poistamisen aktiviteettikaavio

Suljettaessa tallentamaton versio on tarpeen tuhota koko versiorakenne, jotta voidaan varmistua, ettei mitään osia hierarkiasta jää tietokantaan. Tällöin poistetaan kaikki tarjouksen versiot yksi kerrallaan kuvattuun tapaan.

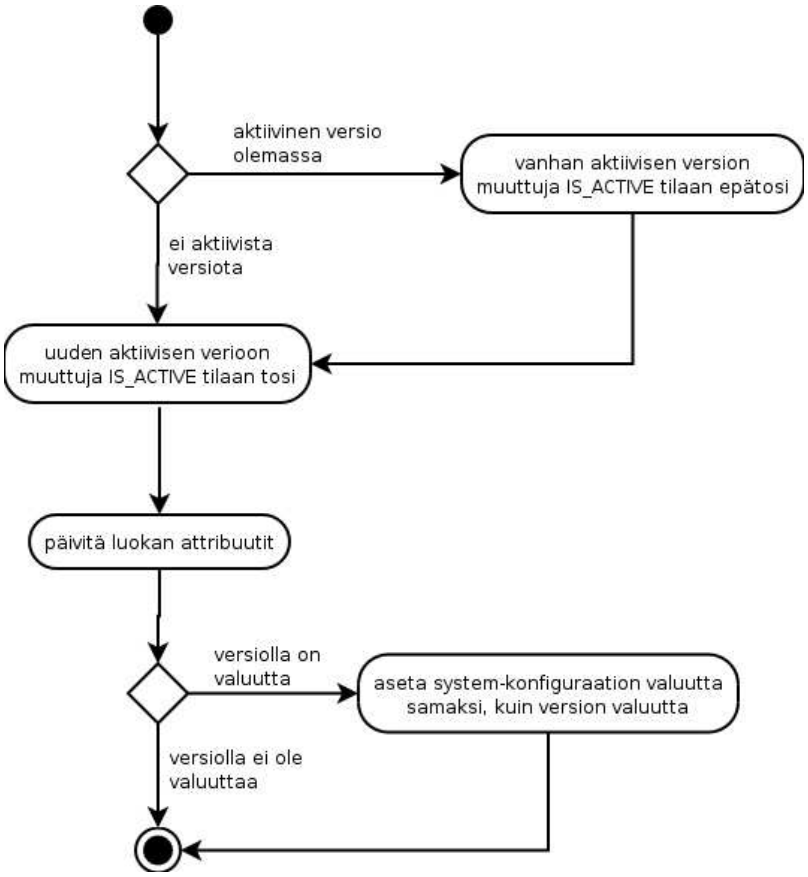
Versio poistetaan käyttäjän valitessa tämä toiminto perustiedot-sivulla. Lisäksi sovellus tuhoaa kaikki versiot, kun suljetaan tallentamaton tarjous.

Aktiivisen version asettaminen

Aktiivinen versio voidaan vaihtaa käyttöliittymän perustiedot-sivulla valitsemalla uusi aktiivinen versio alasvetovalikosta, joka näkyy kuvassa 3. Aktiivinen versio vaihtuu automaattisesti luotaessa uusi versio.

Aktiivinen versio vaihdetaan asettamalla VersionControl-luokan sisäinen luokkamuuttuja osoittamaan valittuun versioon. Lisäksi luodaan uudet oliot sisäisistä luokista Section ja ProductControl siten, että ne liittyvät valittuun versioon. Koska tarjouksen valuutta on versioittainen, se täytyy asettaa samalla, kun aktiivista versiota vaihdetaan.

Version vaihdon johdosta käyttäjälle näkyvät valitun version alaiset sektiot ja tuotteet. Käyttöliittymässä näkyy vain yhden version sisältö kerrallaan. Nähdäkseen toisen version tietoja, käyttäjän on vaihdettava aktiivista versiota. Aktiivisen version asettamisesta on aktiviteettikaavio kuvassa 7.



Kuva 7. Aktiivisen sektion asettamisen aktiviteettikaavio

3.3.1.2 Sektioiden hallinta

Luokka Section, joka on VersionControl-luokan sisäluokka, on vastuussa

sektioiden hallinnasta. Luokkaa käytetään VersionControl-luokan julkisen luokkamuuttujan `sectionControl` kautta, joka viittaa Section-olioon.

Jokaisen version ensimmäinen sektio on niin sanottu tilapäissektio, jonne tallennetaan kaikki versiotasolla näkyvät tuotteet. Tilapäissektio ei sinänsä koskaan näy käyttäjälle, mutta sen sisältämät tuotteen tulostuvat tuotelistauksiin suoraan versiotason alapuolelle, ennen muita version sisältämiä sektioita.

Tilapäissektio sisältää aina kaksi pakollista tuotetta: yleistiedot ja muut kustannukset. Yleistiedot-tuotteen toteutus on sittemmin muuttunut, eikä sitä enää tarvita tilapäissektiossa. Sen olemassaolosta ei kuitenkaan ole mitään haittaa ja se on jätetty sektioon, koska sen poistaminen on turhaa.

Jokainen sektio sisältää yhden tyhjän tuotteen pois luettuna tilapäissektio. Tämä tuote ei näy käyttäjälle ja on olemassa vain i2 iss -ohjelmistossa hinnoittelussa olevan virheen kiertämiseksi.

Sektioihin liittyy läheisesti sektioiden hallintanäkymä, joka on kuvassa 9. Tämä näkymä näyttää tarjouksen rakenteen ja antaa mahdollisuuden muokata sitä luomalla tai poistamalla sektioita ja siirtämällä sektioita tai tuotteita hierarkiarakenteessa. Nämä toiminnot voidaan tehdä kuvassa punaisella rajatulla alueella olevien hallintapainikkeiden avulla.

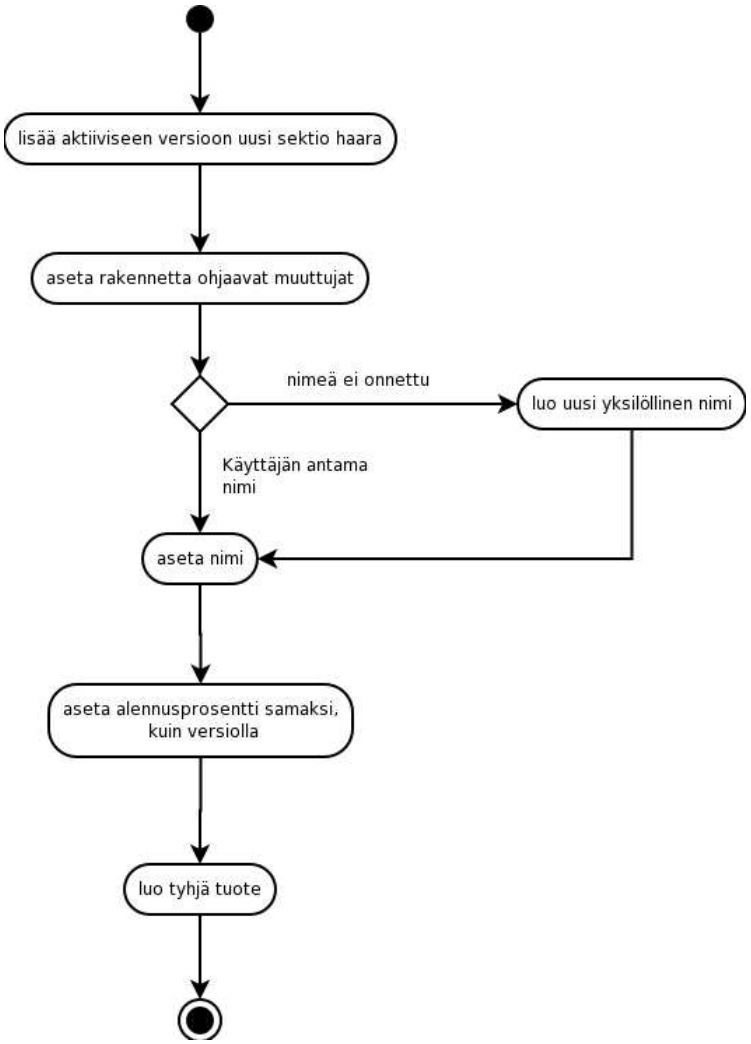
Section-luokkaan liittyvät käyttötapaukset ovat sektion luominen, tuhoaminen ja siirtäminen. Nämä on kuvattu seuraavaksi. Näihin käyttötapauksiin liittyvä Java-koodi on liitteessä 2.

Sektion luominen

Uusi sektio luodaan oletuksena järjestyksessä viimeiseksi ja sille annetaan oletusnimi, jonka perään liitetään sektion numero. Näin saadaan muotoa ”Section 3” oleva oletusnimi. Käyttäjä voi muuttaa sekä sektion nimeä että sijaintia haluamakseen, mutta kahdella sektiolla ei saa olla samaa nimeä. Jos sektiolle

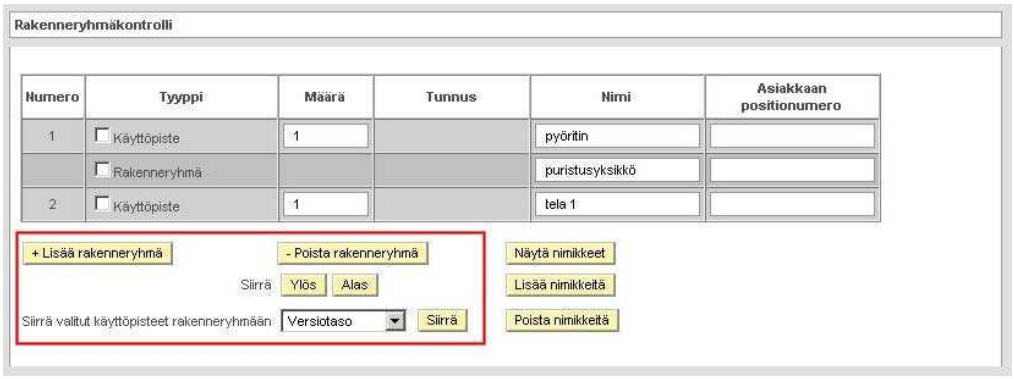
yritetään antaa jo käytössä oleva nimi, ilmoitetaan käyttäjälle virheestä.

Koska alennusprosentit periytyvät tuotehierarkiassa, täytyy uudelle sektiolle asettaa version alennusprosentti. Jokainen sektio sisältää yhden tyhjän tuotteen, joka korjaa hinnoittelussa olevan virheen. Tämä luodaan automaattisesti, kun sektio luodaan. Kuvassa 8 on aktiviteettikaavio uuden sektion luomisesta.



Kuva 8. Sektion luomisen aktiviteettikaavio

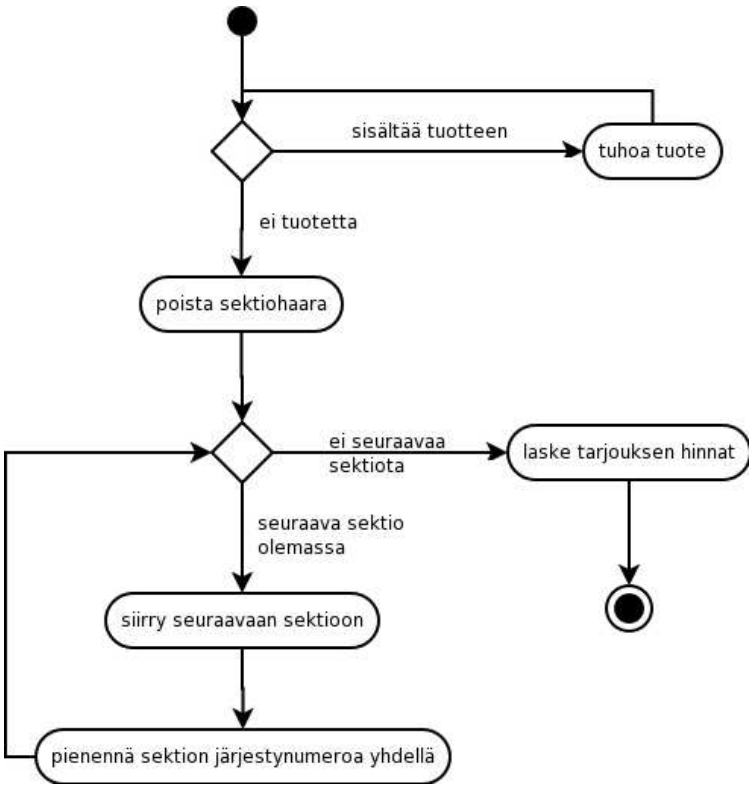
Käyttäjä voi luoda uuden sektion aktiiviseen versioon valitsemalla *lisää rakenneryhmä* -toiminnon ostoskorista tai sektioiden hallintanäkymästä. Ostoskori on kuvassa 3 ja sektioiden hallintanäkymä kuvassa 9. Tilapäissektio luodaan automaattisesti jokaiseen versioon.



Kuva 9. Sektioiden hallintanäkymä, jossa hallintapainikkeet korostettu punaisella

Sektion tuhoaminen

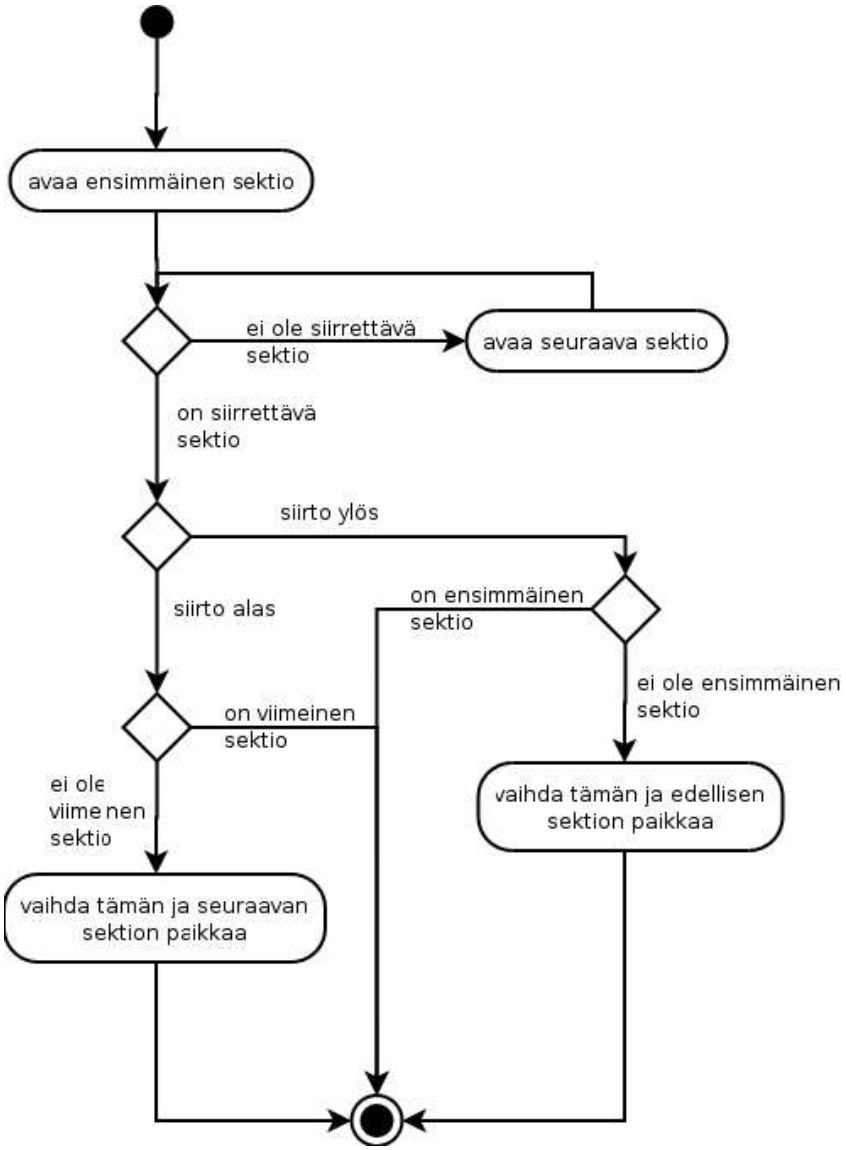
Luotuja sektioita voidaan tuhota rakenneryhmien hallintanäkymästä, joka on kuvassa 9. Käyttäjä valitsee tuhottavat sektiot ja painaa *poista rakenneryhmä* -nappia.



Kuva 10. Sektion tuhoamisen aktiviteettikaavio

Kun sektio tuhotaan, ensin poistetaan kaikki sen sisältämät tuotteet. Tämän jälkeen poistetaan kyseisen sektion haara hierarkiarakenteen versiotasolta. Järjestyksessä tuhottavaa sektiota myöhemmin olevia sektioita siirretään yksi pykälä ylöspäin. Tämä tapahtuu muuttamalla sektioon tallennettua järjestysnumeroa. Lopuksi lasketaan uudelleen tarjouksen hinnat. Kuvassa 10 on esitetty sektion tuhoaminen aktiviteettikaavion avulla.

Sektio tuhotaan käyttäjän niin halutessa ja suljettaessa tallentamaton tarjous, jolloin koko hierarkiarakenne tuhotaan.



Kuva 11. Sektion siirron aktiviteettikaavio

Sektion siirtäminen

Aktiivisen version sektioiden järjestystä voidaan vaihtaa siirtämällä sektioita ylös- tai alaspäin hierarkiarakenteessa. Tämä tapahtuu käyttäjän valitessa siirto sektioiden hallintanäkymästä tai automaattisesti, kun sektio tuhotaan.

Sektiota siirrettäessä etsitään ensin siirrettävä sektio ja sitten vaihdetaan sen paikkaa edellisen tai seuraavan sektion kanssa riippuen siirtosuunnasta. Kuvassa 11 on aktiviteettikaavio sektion siirrosta.

3.3.1.3 Tuotteiden hallinta

Tuotteiden hallinnan tarkoitus on eriyttää tuotteiden manipulointiin käytettävät metodit luokkaan ProductControl. Tämä on luokan VersionControl sisäluokka, johon pääsee käsiksi VersionControl-luokan julkisen luokkamuuttujan `productControl` kautta.

Tuotteiden hallintaan liittyvät käyttötapaukset tuhoaminen, siirtäminen ja kopioiminen on kuvattu seuraavaksi. Näihin liittyvä Java-koodi on liitteessä 3. Uuden tuotteen luonti tapahtuu omalla erityisellä toiminnallisuudella, johon ei käytetä VersionControl-luokkaa, eikä sitä näin ollen kuvata.

Tuotteen tuhoaminen

Kun tuote tuhotaan, se merkitään tuhottavaksi ja poistetaan hierarkiarakenteesta. Kun tarjous tallennetaan seuraavan kerran, poistetaan kaikki tuhottavaksi merkityt tuotteet tietokannasta. Näin tuote säilyy olemassa, jos tarjousta ei tallenneta tuotteen tuhoamisen jälkeen. Pseudokoodi tuotteen tuhoamisesta on seuraavanlainen:

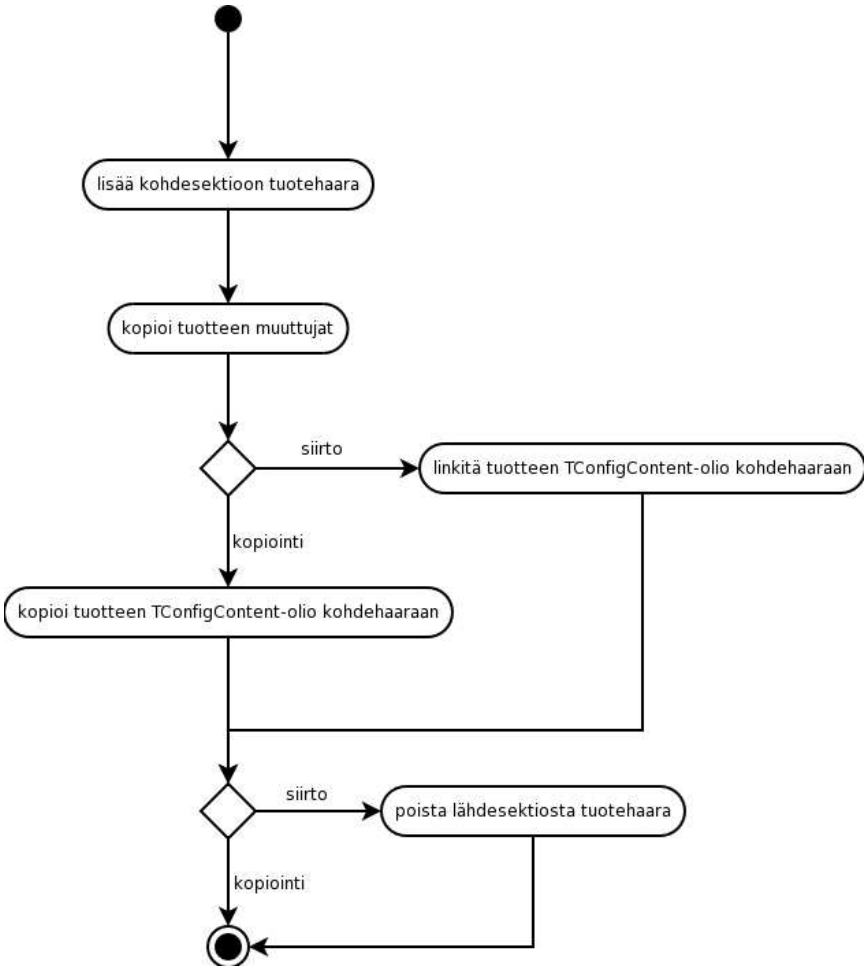
- jos tuhottavaa tuotetta ei ole olemassa
  - poistu
- lisää tuote tuhottavien tuotteiden listaan

- poista tuote hierarkiarakenteesta

Tuote tuhotaan ostoskorista valitsemalla halutut tuotteet ja painamalla *poista*-nappia. Kun sektio tuhotaan, myös sen sisältämät tuotteet tuhotaan automaattisesti.

Tuotteen siirtäminen tai kopioiminen toiseen sektioon

Tuotteita on mahdollista siirtää tai kopioida sektiosta toiseen. Tällöin kohdesektion alle luodaan uusi haara tuotetta varten ja tuotteen TConfigContent-olio siirretään tai kopioidaan uuteen haaraan. Jos tuote siirrettiin, lähdehaara tuhotaan. Kuvassa 12 on aktiviteettikaavio tuotteen siirtämisestä.



Kuva 12. Tuotteen siirtämisen ja kopioimisen aktiviteettikaavio

Kun tuote kopioidaan, sille annetaan uusi nimi. Tämä noudattaa muotoa, jossa alkuperäisen nimen eteen liitetään "Copy of" ja nimen perään lisätään numero, joka yksilöi kopion jos samasta tuotteesta tehdään useita kopioita. Esimerkiksi ”Copy of Drive point (3)”.

Tuote kopioidaan, kun käyttäjä valitsee *kopioi*-toiminnon ostoskorista. Sektioiden hallintanäkymässä voidaan siirtää valitut tuotteet eri sektion alaisuuteen. Järjestelmän sisäisesti jokainen uusi tuote luodaan ensin hierarkiarakenteen juuritasolle, josta se siirretään sille valittuun sektioon.

4 TESTAUS

Versionhallintaa testatessa keskityttiin käyttötapausten toiminnallisuuteen. Yksittäinen testitapaus koostui yleensä yhdestä käyttötapauksesta ja sen oletetusta toiminnasta.

Testaus suoritettiin noudattamalla käyttäjältä odotettua toimintaa ja tarkkailemalla ohjelman reagointia. Virhetilanteiden oikean käsittelyn toteamiseksi suoritettiin myös testejä, joissa poikettiin oletetusta toiminnasta. Ohjelman toimintaa seurattiin sekä käyttöliittymän tapahtumien että lokiin tulostuneiden viestien avulla.

Lokitulostukset ovat olleet pääasiallinen tapa paikallistaa ohjelman virhetoimintoja. JSP-koodia ei voida ajaa debugger-ohjelmalla, joten lokitulostukset muuttujien arvoista ja suoritushaaravalinnoista ovat ainoa tapa seurata käyttäjälle näkymätöntä toimintaa.

Lokitulostuksia varten tehtiin oma luokka, joka ohjaa lokitulostukset ja poikkeuksien virhetilannetulostukset omiin tulostusvirtoihinsa. Lokitulostukset voidaan tämän luokan avulla helposti kääntää pois päältä, jolloin ohjelmakoodista ei tarvitse poistaa kaikkia testauksessa käytettyjä tulostuksia, vaan ne jätetään huomiotta ohjelmaa ajettaessa. Tuotantopalvelimelle siirrettäessä riittää näin ollen lokitulostuksien kääntäminen pois päältä, niitä ei tarvitse poistaa. Poikkeusten virhetilanneviestit tulostuvat aina.

4.1 Versioiden testaus

Versioihin liittyvistä käyttötapauksista on käyty läpi version luominen, tuhoaminen ja asettaminen aktiiviseksi. Seuraavassa käydään läpi näihin liittyvät testitapaukset.

Version luominen

Esiehdot:
-Tarjous, jossa vähintään yksi versio on olemassa

Testin suoritus:
-Luodaan uusi versio painamalla *lisää versio* -nappia.

Odotetut tulokset:
-Tarjoukseen on luotu versio 2, jolla on samat tiedot, kuin versiolla 1.
-Version luonnin ajan käyttäjälle näkyy viesti, joka pyytää odottamaan.

Version tuhoaminen

Esiehdot:
-Tarjous, jossa vähintään kaksi versiota on olemassa

Testin suoritus:
-Painetaan *poista versio* -nappia perustiedot-sivulla.

Odotetut tulokset:
-Versio on poistettu.
-Poistetun version isäntäversio on asetettu aktiiviseksi
-Version poistamisen ajan käyttäjälle näkyy viesti, joka pyytää odottamaan.

Poikkeustapaukset:
-Poistettava versio on ensimmäinen versio. Tämä ei ole sallittua ja tästä näytetään käyttäjälle virheviesti.

Aktiivisen version vaihtaminen

Esiehdot:
-Vähintään kaksi versiota olemassa.

Testin toteutus:
-Perustiedot-sivulla valitaan uusi aktiivinen versio alasvetovalikosta kohdassa *aktiivinen versio*.

Odotetut tulokset:
-Valittu versio on asetettu tarjouksen aktiiviseksi versioksi.
-Käyttäjälle näkyvät nykyisen aktiivisen version tiedot.

Poikkeustapaukset:
-Jos valittu versio on sama, kuin aktiivinen versio, mitään ei tapahdu.

4.2 Sektioiden testaus

Sektioiden käyttötapaukset ovat sektion luominen, tuhoaminen ja siirtäminen. Näiden testitapaukset kuvataan seuraavaksi.

Sektion luominen

Esiehdot:
-Tarjous, jossa vähintään yksi versio olemassa.

Testin toteutus:
-Painetaan *lisää käyttöpiste* -nappia ostoskorissa.
-Annetaan sektiolle nimi ja sijainti aukeavassa näkymässä ja hyväksytään luominen.

Odotetut tulokset:
-Aktiiviseen versioon on luotu uusi sektio, jolla on annettu nimi ja sijainti.

-Luotu sektio sisältää tyhjän tuotteen, jolla korjataan hinnoittelun virhe. Käyttäjälle sektio näkyy tyhjänä.
-Suoritus palaa perustiedot-sivulle.

Poikkeustapaukset:
-Käyttäjä peruuttaa sektion luomisen, kun kysytään uuden sektion nimeä ja sijaintia. Tällöin palataan perustiedot-sivulle luomatta uutta sektiota.
-Sektiolle annettu nimi on käytössä. Tästä ilmoitetaan käyttäjälle ja pyydetään valitsemaan toinen nimi.

Sektion tuhoaminen

Esiehdot:
-Aktiivisella versiolla on vähintään yksi sektio.

Testin toteutus:
-Valitaan sektioiden hallintanäkymästä tuhottava sektion (tai sektiot) ja painetaan *poista rakenneryhmä* -nappia.

Odotetut tulokset:
-Valittu sektio (tai sektiot) on poistettu hierarkiarakenteesta. Myös kaikki tuhotun sektion sisältämät tuotteet on poistettu.
-Suoritus pysyy sektioiden hallintanäkymässä.
-Tarjouksen hinnat on laskettu uudelleen.

Poikkeustapaukset:
-Sektioiden hallintanäkymässä on valittuna tuote, kun valitaan sektion tuhoaminen. Tällöin virheellisestä valinnasta ilmoitetaan käyttäjälle ja toiminto peruutetaan.
-Yhtään sektiota ei ole valittu. Tällöin ei tehdä mitään.

Sektion siirtäminen

Esiehdot:

-Aktiivisessa versiossa on vähintään kaksi sektiota.

Testin toteutus:
-Sektioiden hallintanäkymässä valitaan siirrettävä sektio (tai sektiot) ja painetaan ylös- tai alas-nappia.

Odotetut tulokset:
-Valittu sektio (tai sektiot) on siirtynyt yhden pykälän valittuun suuntaan. Mikäli valittuna on ollut useampi sektio, niiden suhteellinen sijainti toisiinsa nähden säilyy, mikäli kaikkien sektioiden siirto onnistui.

Poikkeustapaukset:
-Sektiota ei voida siirtää valittuun suuntaan, koska se on ko. suunnassa viimeinen sektio. Tällöin sektiota ei siirretä.
-Yhtään sektiota ei ole valittu. Tällöin ei tehdä mitään.

4.3 Tuotteiden testaus

Tuotetasolla on käsitelty käyttötapaukset tuotteen tuhoaminen, siirtäminen ja kopioiminen. Seuraavaksi kuvataan kunkin käyttötapauksen testaus.

Tuotteen tuhoaminen

Esiehdot:
-Aktiivisessa versiossa on jossain sektiossa vähintään yksi tuote.

Testin toteutus:
-Ostoskorissa valitaan tuhottava tuote (tai tuotteet) ja painetaan *poista*-nappia.

Odotetut tulokset:
-Valittu tuote (tai tuotteet) on poistettu hierarkiarakenteesta.
-Tarjouksen hinnat on laskettu uudelleen.

Poikkeustapaukset:

-Yhtään tuotetta ei ole valittu. Tällöin ei tehdä mitään.

Tuotteen siirtäminen

Esiehdot:

-Aktiivisessa versiossa on jossain sektiossa vähintään yksi tuote.

-Aktiivisessa versiossa on vähintään kaksi sektiota.

Testin toteutus:

-Sektioiden hallintanäkymässä valitaan siirrettävä tuote (tai tuotteet) ja sektio, johon tuote siirretään. Siirto tehdään painamalla *siirrä*-nappia.

Odotetut tulokset:

-Valittu tuote (tai tuotteet) on siirretty valittuun sektioon.

-Tarjouksen hinnat on laskettu uudelleen.

Poikkeustapaukset:

-Sektioiden hallintanäkymässä valittu yksikkö ei ole tuote. Tällöin käyttäjälle näytetään virheilmoitus ja toiminto perutaan.

-Yhtään tuotetta ei ole valittu. Tällöin ei tehdä mitään.

Tuotteen kopioiminen

Esiehdot:

-Aktiivisessa versiossa on vähintään yksi tuote jossain sektiossa.

Testin toteutus:

-Ostoskorissa valitaan kopioitava tuote ja painetaan *kopioi*-nappia.

Odotetut tulokset:

-Aktiivisen version viimeiseen sektioon on luotu kopio valitusta tuotteesta.

-Tarjouksen hinnat on laskettu uudelleen.

-Luotu kopio on nimeltään muotoa ”Copy of [alkuperäisen tuotteen nimi] (1)”. Viimeisenä oleva järjestysnumero kertoo monesko kopio on kyseessä.

Poikkeustapaukset:
-Yhtään tuotetta ei ole valittu. Tällöin ei tehdä mitään.

5 ESIMERKKI

Hinnoittelu

Numero	Nimi	Määrä	Koodi	Lopullinen myyntihinta	Lopullinen kate %	Alennus %	Yksityiskohtainen hinnoittelu
①	Kokonaishinta				20,00%	0,00000	Siirry näkymään
② 1	pyöritin	1			20,00%	0,00000	Siirry näkymään
③	VAIHDE	1	GD1PSF20		20,00%	0,00000	Siirry näkymään
	HAMMASKYTKIN	1	HK301		20,00%	0,00000	Siirry näkymään
	SOKKELOTIIVISTYS	1	CCM20LAB		0,00%	0,00000	Siirry näkymään
	SOKKELOTIIVISTYS	1	QQM20LAB		0,00%	0,00000	Siirry näkymään
④	puristusyksikkö				20,00%	0,00000	Siirry näkymään
⑤ 2	tela 1	1			20,00%	0,00000	Siirry näkymään
⑥	VAIHDE	1	GD1PSF20		20,00%	0,00000	Siirry näkymään
	HAMMASKYTKIN	1	HK301		20,00%	0,00000	Siirry näkymään
	SOKKELOTIIVISTYS	1	CCM20LAB		0,00%	0,00000	Siirry näkymään
	SOKKELOTIIVISTYS	1	QQM20LAB		0,00%	0,00000	Siirry näkymään
3	Muut ehdot	1			0,00%	0,00000	Siirry näkymään
					Laske määräalennukset	Aseta alennukset	

Kuva 13. Hinnoittelunäkymä, jossa eri hierarkiatasot numeroitu vihreällä

Kuvassa 13 on esimerkki hinnoittelunäkymästä, jossa näkyy hyvin tarjouksen hierarkiarakenne. Ensimmäisellä rivillä, valkoisella pohjalla on versiotason tiedot (1). Sen alapuolella keskiharmaalla on tilapäissektion sisältämä tuote pyöritin (2) ja sen sisältämät moduulit (3), jotka ovat vaaleanharmaalla pohjalla. Näiden alapuolella on tummanharmaalla pohjalla tarjouksen ensimmäinen sektio puristusyksikkö (4), jonka alapuolella näytetään sen sisältämät tuotteet (5) moduuleineen (6).

Tarjouksen eri hierarkiatasoja erotellaan taustaväriä vaihtamalla. Versio on aina valkoisella pohjalla, sektioit tummanharmaalla, tuotteet keskiharmaalla ja moduulit vaaleanharmaalla. Vaihtoehtoisesti eri tasoja olisi voinut erottaa toisistaan

sisennyksen avulla, mutta se olisi vienyt liikaa tilaa. Monissa näkymissä tulostetaan kullekin riville paljon tietoja tarjouksesta, eikä ylimääräinen sisennys ole mahdollista tilanpuutteen takia. Tästä syystä päädyttiin käyttämään eri värejä hierarkiarakenteen tasojen ilmaisemiseen.

6 JATKOKEHITYS

Jatkokehityksenä VersionControl-luokan toimintaa voisi selkeyttää. Tämän luokan toteuttaminen kärsi monesti muuttuneista vaatimuksista, joissa luokan alkuperäiseen käyttötarkoitukseen oli tarpeen lisätä uusia toiminnallisuuksia projektin edetessä. Tämä on johtanut tilanteeseen, jossa luokan toteutus on sekavahko ja monia toimintoja voisi tehdä selkeämmin. Alkuperäiset metodit eivät ole optimaalisia uusimpien toimintojen toteuttamiseen, mutta niiden muuttaminen ei ole mielekästä koska ne toimivat luotettavasti. Nykyinen VersionControl-luokan toteutus toimii hyvin, mutta sen käyttö vaatii ymmärtämystä sen toteuttamisessa käytetyistä ratkaisuista.

Toteutettujen luokkien muuttaminen JSP-toteutuksesta puhtaiksi Java-pohjaisiksi luokiksi voisi nopeuttaa järjestelmää. Tämä olisi kuitenkin hankalaa, koska VersionControl-luokka käyttää muita ominaisuuksia, jotka on tehty JSP-toteutuksena eikä näiden muuttaminen Java-pohjaisiksi ole kannattavaa.

7 LÄHTEET

1. i2. [www-sivu]. [viitattu 19.6.2005] Saatavissa: http://www.i2.com/
2. i2 Technologies US, Inc. i2 Intelligent Selling Solution Implementation GuideVersion 6.1.4.1. [sähköinen dokumentti.]
3. Nakari Tomi. TAS Solutions Oy. [www-sivu]. [viitattu 21.6.2005] Saatavissa: http://www.mindset.fi
4. Nakari Tomi. Metso Minerals TradeMatrix Marketplace Storefront 5.1.5 Solution Architecture. [sähköinen dokumentti.]

Uuden version luonti

```
public TSystemConfigNode newVersion() throws TSDKInternalException
  {
    TSystemConfigNode newNode = rootNode.add();
    rootNode.commit();
    cloneNode(newNode);
    // set new node to be active node
    if (activeNode != null)
    {
      setNodeVariable(activeNode, IS_ACTIVE, "false");
      activeNode.commit();
    }

    // set variables
    setNodeVariable(newNode, IS_ACTIVE, "true");
    setNodeVariable(newNode, PARENT_NODE, Integer.toString(activeVersionName));
    if (versionNames.size() > 0)
      activeVersionName = ((Integer) versionNames.getLast()).intValue() + 1;
    else
      activeVersionName = 1;
    activeVersionIndex = rootNode.getCount() - 1;
    versionNames.add(new Integer(activeVersionName));
    setNodeVariable(newNode, VERSION_NUMBER,
Integer.toString(activeVersionName));
    setNodeVariable(newNode, NODE_TYPE, TYPE_VERSION);

    // creator and creation date/time
    setNodeVariable(newNode, constants.VERSION_BY,
(String)apiUtils.getPerson(wc, userSession).getLoginName());
    setNodeVariable(newNode, constants.VERSION_DATE, ""+(new Date()).getTime());

    newNode.commit();
    activeNode = newNode;
    sectionControl = new Section(activeNode);

    return activeNode;
  }
```

Aktiivisen version vaihtaminen

```
public void setActive(String version) throws TSDKInternalException
  {
    setActive(getVersion(version));
  }
public void setActive(TSystemConfigNode node) throws TSDKInternalException
  {
    if (node == null)
      return;

    if (activeNode != null)
    {
      setNodeVariable(activeNode, IS_ACTIVE, "false");
      activeNode.commit();
    }
    setNodeVariable(node, IS_ACTIVE, "true");
    node.commit();
    activeNode = node;
    activeVersionName = (new Integer(getNodeVariable(node,
VERSION_NUMBER))).intValue();
    activeVersionIndex = versionNames.indexOf(new Integer(activeVersionName));
    sectionControl = new Section(activeNode);

    // set currency to that of active version
    String currency = getNodeVariable(activeNode, constants.CURRENCY);
  logger.log("currency == " + currency);
    if (currency != null)
    {
      TPricingParameters pricingParams = rootSysConf.getPricingParameters();
      pricingParams.setCurrency("EUR");
      pricingParams.commit();
      pricingParams.setCurrency(currency);
      pricingParams.commit();
      rootSysConf.calculatePrices(false);
    }
    // exchange rate is being set in runtime
  }
```

Aktiivisen version tuhoaminen

```
  public void deleteActive() throws TSDKInternalException, VersionException
  {
    if (activeVersionIndex < 0)
      return;
    if (activeVersionName == 1)
      throw new
VersionException(rsbHelper.getString("exception.deleting_original_version_not
_allowed"));
    TSystemConfigNode node = activeNode;
    // remove all assosiated sections
    for (int i=0; i < node.getCount(); i++)
    {
      try {
      sectionControl.delete(node.get(i));
      } catch (Exception e) { logger.log("deleteActive "+i+": "+e); }
    }

    int removedIndex = activeVersionIndex;
    setActive(getNodeVariable(activeNode, PARENT_NODE));
    // remove node
    versionNames.remove(removedIndex);
    rootNode.remove(removedIndex);
  }
```

Sektion luominen

```
   // adds a section
   public TSystemConfigNode add(String name) throws TSDKInternalException,
VersionException
   {
     if (node == null)
     {
        return null;
     }

     // autocreate unique name, if name == null
     if (name == null)
        name = generateUniqueName();
     for (int i=0; i < sections.size(); i++)
        if (name.equals(getNodeVariable((TSystemConfigNode)sections.get(i),
SECTION_NAME)))
          throw new
VersionException(rsbHelper.getString("exception.section_name_must_be_unique")
);

     TSystemConfigNode newNode = createNode(name);
     sections.add(newNode);

     // set discount percentage for created section
     newNode.setVariable("{Node.Price."+constants.DISCOUNT_VARIABLE+"}",
        newNode.getParent().getFormattedText(new String[]
{"{Node.Price."+constants.DISCOUNT_VARIABLE+"}"})[0]);
     createEmptyProduct(newNode);
     newNode.commit();

     return newNode;
   }
```

Sektion tuhoaminen

```
    // deletes section
    // cannot delete tmpSection by name
    public void delete(String name) throws TSDKInternalException, VersionException
    {
      delete(get(name));
    }
    public void delete(int index) throws TSDKInternalException, VersionException
    {
      if (index < 0 || index > node.getCount() - 1)
      {
       logger.log("Section.delete() index out of bounds: " + index);
       return;
      }
      delete(get(index));
    }
    public void delete(TSystemConfigNode sectNode) throws TSDKInternalException,
VersionException
    {
      String type = getNodeVariable(sectNode, NODE_TYPE);
      if (type == null || !type.equals(TYPE_SECTION))
      {
       logger.log("selected not section");
       throw new
VersionException(rsbHelper.getString("exception.selected_node_is_not_section"
));
      }

      // remove associated products
      int productCount = sectNode.getCount();
      for (int i=0; i < productCount; i++)
      {
         productControl.deleteProduct(sectNode.get(0)); // when (0) is deleted (1)
becomes new (0)
      }

      int index = sections.indexOf(sectNode);
      sectNode.getParent().remove(sectNode);
      if (index != -1)
      {
```

```
      sections.remove(index);
      for (; index < sections.size(); index++)
      {
        setNodeVariable((TSystemConfigNode)sections.get(index),
SECTION_SEQUENCE_INDEX, Integer.toString(index));
      }
     }
     rootSysConf.calculatePrices(false);
   }
```

Sektion siirtäminen

```
   public void moveUp(String[] names) throws TSDKInternalException
   {
     moveUpDown(names, -1);
   }

   public void moveDown(String[] names) throws TSDKInternalException
   {
     moveUpDown(names, 1);
   }

   private void moveUpDown(String[] names, int direction) throws
TSDKInternalException
   {
     if (direction == 0)
      return;
     // find selected section indices
     int[] indices = new int[names.length];
     for (int i=0; i < names.length; i++)
     {
      for (int j=0; j < sections.size(); j++)
      {
        if
(names[i].equals(getNodeVariable((TSystemConfigNode)sections.get(j),
SECTION_NAME)))
        {
          indices[i] = j;
          break;
```

```
        }
       }
      }

      for (int i=0; i < indices.length; i++)
      {
       // do not swap when at either end of list
       if ((indices[i] <= 0 && direction < 0) || (indices[i] >= sections.size()
- 1 && direction > 0))
         continue;
       swap(indices[i], (direction > 0)? indices[i]+1 : indices[i]-1);
      }
    }

    private void swap(int index1, int index2) throws TSDKInternalException
    {
      Object tmp = sections.get(index1);
      sections.set(index1, sections.get(index2));
      sections.set(index2, tmp);
      setNodeVariable((TSystemConfigNode)sections.get(index1),
SECTION_SEQUENCE_INDEX, Integer.toString(index1));
      setNodeVariable((TSystemConfigNode)sections.get(index2),
SECTION_SEQUENCE_INDEX, Integer.toString(index2));
    }
```

Tuotteen tuhoaminen

```
   // removes product from node tree and marks it for removal from database
   public void deleteProduct(TSystemConfigNode prodNode) throws
TSDKInternalException
   {
     if (prodNode == null)
      return;
     try
     {
      //ccFactory.deleteConfigContent(prodNode.getConfigContent().getOID());
      addToDeleteList(prodNode.getConfigContent().getOID());
     }
     catch (Exception e) { logger.logerr(e); }
     try
     {
      prodNode.getParent().remove(prodNode);
     }
     catch (Exception e) { logger.logerr(e); }
   }

   // removes products from database
   public void purgeDeletedProducts() throws TSDKInternalException
   {
     String toBeDeletedProducts = getNodeVariable(rootNode,
TO_BE_DELETED_PRODUCTS);
     if (toBeDeletedProducts == null || toBeDeletedProducts.equals(""))
        return;
     logger.log("to delete == " + toBeDeletedProducts);
      // split product id string, each array item is one product id
     String[] productIDs =
toBeDeletedProducts.split(constants.NODE_ID_SEPARATOR);
     for (int i=0; i < productIDs.length; i++)
     {
      try {
        logger.log("deleting == " + productIDs[i]);
        // deleting here
        ccFactory.deleteConfigContent(productIDs[i]);
        logger.log("delete ok");
      } catch (Exception e) { logger.logerr(e); }
     }
```

```
      setNodeVariable(rootNode, TO_BE_DELETED_PRODUCTS, "");
    }
```

Tuotteen siirtäminen

```
 // use sectionName = null to move to tmp section
 public void moveProductsToSection(String sectionName, TSystemConfigNode[]
productNodes) throws TSDKInternalException
    {
      TSystemConfigNode node = sectionControl.getTemporary();
      if (sectionName != null)
       node = sectionControl.get(sectionName);
      if (node == null || productNodes == null)
       return;

      for (int i=0; i < productNodes.length; i++)
      {
       TSystemConfigNode pn = productNodes[i];
       if (pn == null) continue;
       TSystemConfigNode newNode = node.add();
       cloneNode(pn, newNode, true);
       pn.getParent().remove(pn);
      }
    }
```

Tuotteen kopioiminen

```
    // copies given products to a section
    // returns newly created nodes
    // use sectionName = null to copy to tmp section
    public TSystemConfigNode[] copyProductsToSection(String sectionName,
TSystemConfigNode[] productNodes) throws TSDKInternalException
    {
      TSystemConfigNode node = sectionControl.getTemporary();
      if (sectionName != null)
       node = sectionControl.get(sectionName);
      if (node == null)
       return null;
```

```
      TSystemConfigNode[] newNodes = new TSystemConfigNode[productNodes.length];
      for (int i=0; i < productNodes.length; i++)
      {
       TSystemConfigNode pn = productNodes[i];
       newNodes[i] = node.add();
       cloneNode(pn, newNodes[i], false);
       String newName = "Copy of " + getProductName(newNodes[i]) + " (" +
(getGreatestCopyOfIndex() + 1) + ")";
       setProductName(newNodes[i], newName);
      }
      return newNodes;
    }
```