CT60A4301 Tietokannat

CT60A2410 Olio-ohjelmointi

Harjoitustyö – T.I.M.O.T.E.I.

Lappeenrannan teknillinen yliopisto
Innovation and Software (IS), LUT LBM

CT60A4301 Tietokannat
CT60A2410 Olio-ohjelmointi
Kevät 2016

0438691 Elias Vähäjylkkä
elias.vahajylkka@student.lut.fi

SISÄLLYSLUETTELO

1 TEHTÄVÄNANTO

Harjoitustyössä on tehtävänä luoda Toiminnaltaan Itellaa Muistuttava Ohjelmisto, Tietokantaa Edellyttävä Integraatio (tästä eteenpäin TIMOTEI) käyttäen jo valmista SmartPost-verkostoa. Tehtävään liittyvät SmartPost-verkosto kartalle esitettynä, pakettien varastointi, paketit sekä niiden sisällöt.

Ohjelmassa on tarkoitus luoda SmartPost-olioita, jotka koostuvat automaatin paikkatiedoista sekä muista tiedoista (osoitteet/aukioloajat) joita saadaan avoimesti. Automaatteja hallinnoidaan erillisen tietokannan avulla, joka pitää varastossa tiedon automaateista ja mahdollistaa yksittäisten automaattien etsimisen. Lisäksi tietokannassa on oltava osio, joka sisältää paketteja ja paketit sisältävät erilaisia esineitä. Huomioi myös, että käyttäjän tulee olla mahdollista lisätä ja poistaa SmartPost-automaatteja / paketteja / esineitä / reittejä tietokannasta.

Paketteja on oltava kolmea erilaista, 1. 2. ja 3. luokan paketteja, joilla on erilaisia ominaisuuksia kuten koko ja painorajat. Tämän lisäksi paketeille voi keksiä vapaasti erilaisia toiminnallisuuksia.

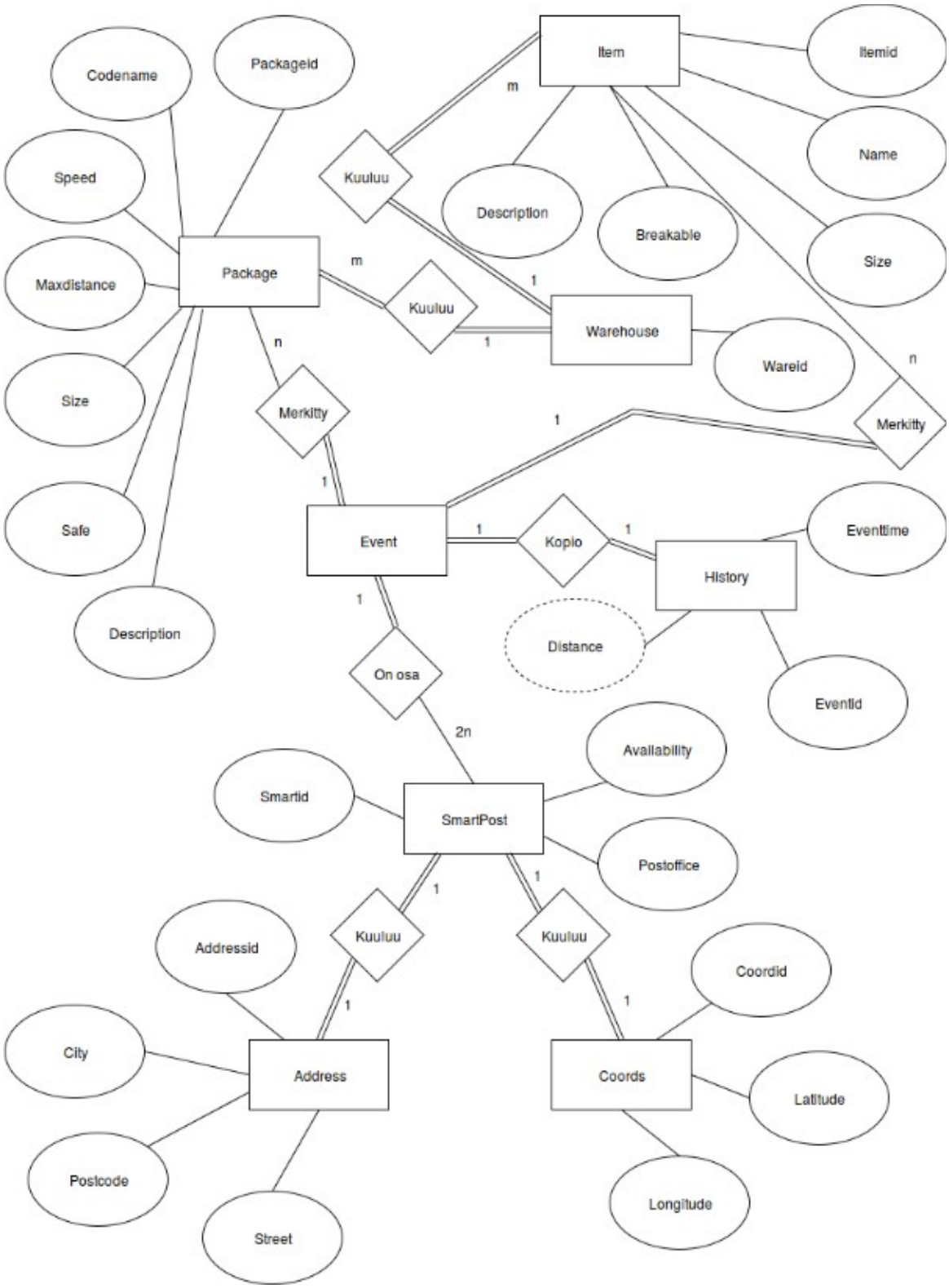
Paketit sisältävät vapaavalintaisia esineitä, joiden on sovittava pakettiin. Ohjelma huomauttaa virheellisistä sijoituksista eikä suostu lähettämään pakettia, johon esine ei mahdu. Esineet voivat olla esimerkiksi My Little Pony-figuureita, Kaunarit-DVD-paketteja, Pokémon-korttipakkoja, Mikko Alatalo-fanituotteita, Kaalimadon tavaratoimituksia tai mitä ohjelmoija itse haluaa määritellä, aihe on vapaa. Esineitä on oltava ainakin 4 erilaista ja ne määrittelevät, menevätkö ne rikki kuljetuksen aikana (== esineellä on feature, voivatko ne hajota vai eikö).

- 1. luokan paketti on kaikista nopein pakettiluokka, jonka vuoksi sitä ei voi lähettää pidemmälle kuin 150 km päähän. 1. luokan paketti on myös nopea, koska sen turvallisuudesta ei välitetä niin paljon, jolloin kaikki särkyvät esineet tulevat menemään rikki matkan aikana.

- 2. luokan paketit ovat turvakuljetuksia, jolloin ne kestävät parhaiten kaiken särkyvän tavaran kuljettamisen. Näitä paketteja on mahdollista kuljettaa jopa Lapista Helsinkiin, sillä matkan aikana käytetään useampaa kuin yhtä TIMOTEI-miestä, jolloin turvallinen kuljetus on taattu. Paketissa on kuitenkin huomattava, että jos se on liian suuri, ei särkyvä esine voi olla heilumatta, joten paketin koon on oltava pienempi kuin muilla pakettiluokilla.
- 3. luokan paketti on TIMOTEI-miehen stressinpurkupaketti. Tämä tarkoittaa sitä, että TIMOTEI-miehellä ollessa huono päivä pakettia paiskotaan seinien kautta automaatista toiseen, joten paketin sisällön on oltava myös erityisen kestävää materiaalia. Myös esineen suuri koko ja paino ovat eduksi, jolloin TIMOTEI-mies ei jaksa heittää pakettia seinälle kovin montaa kertaa. Koska paketit päätyvät kohteeseensa aina seinien kautta, on tämä hitain mahdollinen kuljetusmuoto paketille.

SmartPost-verkosto sekä pakettivarasto tulee liittää yhteen toimivalla pääluokalla, jonka avulla tietoja voidaan käyttää ja yhdistää. Pääluokka tarvitsee toimiakseen graafisen käyttöliittymän, jotta tietoa voidaan esittää järkevästi ja helposti. Käyttöliittymän kautta hallinnoidaan myös automaattien visuaalista esitystä kartalla sekä pakettien lähettämistä automaatista toiseen.

2 TIETOKANNAN KÄSITEMALLI JA EHEYSSÄÄNNÖT



Kuvassa harjoitustyössä vaaditun tietokannan käsitemalli. Tarkempi kuva on tiedostossa kasitekaavio.png, joka on osa harjoitustyön palautusta.

Kuvassa on kahdeksan yksilötyyppiä eli SQL-kannassa on saman verran tauluja.

Yksilötyypit ovat siis SmartPost, johon sisältyvät osoite ja koordinaatit erillisinä yksilöinä sekä paketti, esine, varasto ja lähetyshistoria. Lähetyshistoria sitoo tietokannan yhteen eheyden vuoksi, jonka lisäksi se pitää kirjaa tehdyistä lähetyksistä. Lähetyshistoriaan kuuluu omien muuttujien lisäksi paketti, esine ja kaksi SmartPostia, jotka ovat paketin lähetys- ja vastaanottopaikka.

Paketti ja esine ovat erillisiä yksikköjä, ne niputetaan yhteen vasta paketointivaiheessa. Tällöin molemmat tallennetaan warehouse-yksikköön saman wareid:n alle. Historiaan tulee merkintä vasta kun lähetysnappia painetaan.

Event-yksikkö sitoo tietokannan yhteen ja tallentaa lähetykset. Lähetykset luetaan eventistä suoraan historiaan, joka koostuu vakaammista muuttujista. Jos historiatiedot perustuisivat suoraan id:hen, niin pakettien, esineiden ja SmartPostien poistaminen sotkisi historiatiedot.

Eheyssäännöt koskevat lähinnä kokonaislukumuuttujia. Minkään kohteen id ei saa olla negatiivinen, jotta systeemi olisi yhtenäinen ja ohjelman silmukat toimisivat ongelmitta. Esineiden ja pakettien koon on oltava suurempi kuin 0. Paketin nopeus on oltava kokonaisluku väliltä 1-3. Arvot eivät siis kuvaa koon tai nopeuden täsmällistä arvoa, vaan sen suuruusluokkaa yksinkertaistuksen vuoksi.

3 OHJELMAN KUVAUS

Valmis TIMOTEI-ohjelma täyttää kaikki tehtävänannon vaatimukset, joten se toimii isoksi osaksi kuvauksena. SmartPost-automaatit voi hakea netistä osaksi tietokantaa, ja jos haettava automaatti on jo tietokannassa, sitä ei tallenneta. Ohjelmassa on esineitä ja paketteja, joilla on erilaisia ominaisuuksia.

Käyttäjä voi luoda omia esineitä ja laittaa niitä paketteihin. Pakettityyppejä on ohjeistuksen mukaan kolme, mutta esineitä voi tehdä itse lisää. Esineitä voi myös poistaa tietokannasta, samoin SmartPost-automaatteja. Valmiit paketit

voi lähettää SmartPostista toiseen, jolloin automaattien sijainti näkyy kartalla ja paketin kulkua voi seurata reaaliajassa kartalle piirtyvän viivan muodossa.

Kaikki tehdyt lähetykset tallennetaan lähetyshistoriaan, jossa niitä voi tarkastella. Historian voi myös tyhjentää.

Lähetetyt paketit eivät poistu tietokannasta, eli samaa pakettia voi lähettää uudelleen automaatista toiseen. Ohjelmassa onkin pyritty lähinnä simuloimaan yksittäisiä lähetyksiä. Vain rikkoutuneet paketit poistetaan tietokannasta automaattisesti. Käyttäjä voi poistaa paketteja myös käsin.

Kun lähetettäessä valitaan lähtö- ja saapumisautomaatteja, ne näkyvät kartalla erivärisinä nastoina. Jos käyttäjä muuttaa mielensä ja vaihtaa automaattia, myös nasta siirtyy. Kun lähetys tehdään ja pakettia kuvaava viiva alkaa liikkua kartalla, nastat (ja viiva) jäävät siihen missä ovat. Uusi lähetys käyttää uusia nastoja ja vanhat jäävät merkkaamaan edellisten lähetysten reittiä ja suuntaa. Koko kartan voi tyhjentää merkinnöistä milloin vain napinpainalluksella.

Ohjelman alussa käyttäjältä kysytään, haluaako hän käyttää valmista tietokantaa (johon on tallennettu edellisen kerran tilanne) vai aloittaa käytön alusta.

Käyttöliittymän ajatus on imitoida Itellan virallista ulkoasua ja ykkösluokan postilaatikkoa, eli värimaailmassa on käytetty paljon oranssia sinisellä pohjalla.

4 OHJELMALLINEN TOTEUTUS JA TOIMINNALLISUUDET

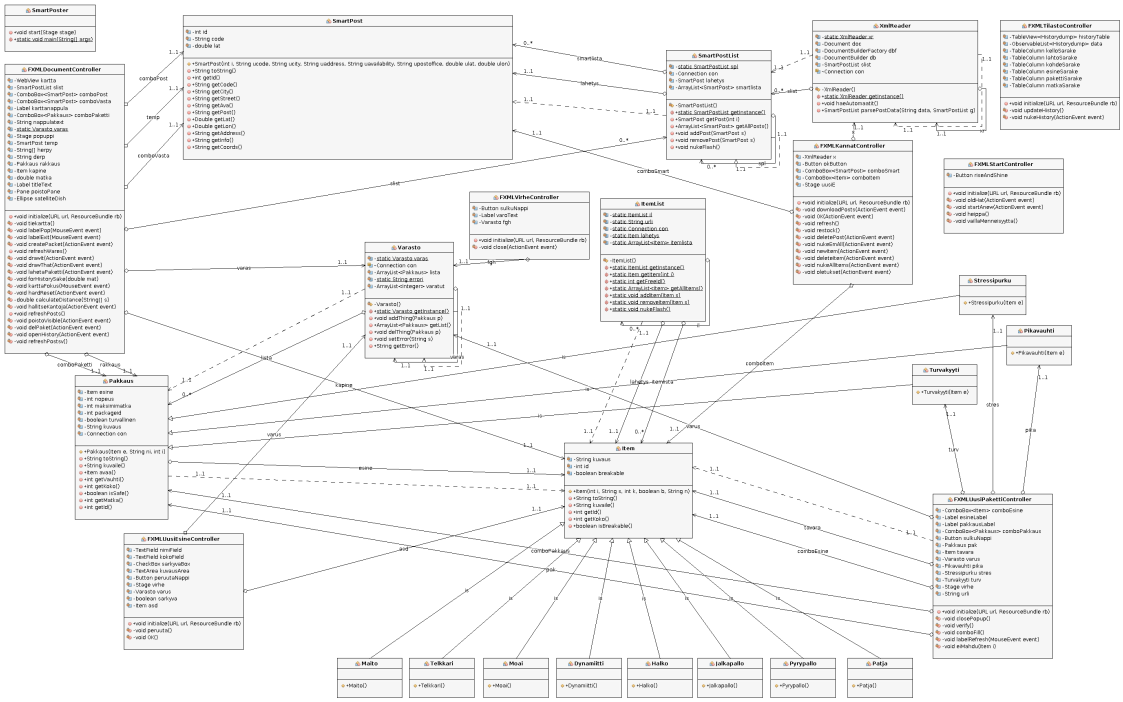
Käytännössä kaikki tiedot tallennetaan tietokantaan ja ongitaan sieltä tarvittaessa. Olioita käytetään lähinnä ohjelman sisäiseen tiedonsiirtoon, eli esimerkiksi paketin tietokannasta hakeva funktio etsii tarvittavat tiedot ja kasaa niistä olion, jonka sitten lähettää pääohjelmalle.

Ohjelma on tehty ikkunapohjaisesti, eli uutta asiaa varten tehdään uusi ikkuna. Aloitusikkunassa valitaan, halutaanko ladata vanha tilanne vai aloittaa uusi, pääikkunassa nähdään kartta ja tehdään lähetykset, paketti-ikkunassa voi luoda uuden paketin, tietokantaikkuna hallinnoi SmartPosteja ja esineitä, uusi esine -ikkunassa voi tehdä uuden esineen, historiaikkunassa voi tarkastella historiatietoja ja virheikkunaa käytetään virheiden näyttämiseen.

Ohjelman alussa tarjotaan valintaa vanhasta tai uudesta tilanteesta. Vanha tilanne -painike ei tee muuta kuin avaa pääikkunan, uusi tilanne taas poistaa ennen sitä pakettityyppejä lukuunottamatta kaikki tiedot tietokannasta. Käytetään siis koko ajan samaa tietokantaa. Pakettityypit ovat tietokannan ainoa muokkaamattomissa oleva osa.

Tehtävänannon yhteydessä annettu index.html -kartanmuokkaajatiedosto ei sisältänyt kaikkia haluamiani ominaisuuksia, joten lisäsin omia toimintoja. Tekemäni muutokset on dokumentoitu kommentteina. Lisäykset liittyvät nastojen poistamiseen kartalta ja kirjanpitoon erivärisistä nastoista.

5 LUOKKAKAAVIO



SmartUml.png -tiedostossa on ohjelman täydellinen luokkakaavio. Tässä on pienennetty kuva, jotta kompleksisuustasosta saa jonkinlaisen käsityksen, mutta yksityiskohdista ei luonnollisesti ota mitään selvää.

Kaavio on generoitu suoraan koodista ja järjestelty sitten manuaalisesti luettavuuden parantamiseksi. Osa taulukoista näyttää irrallisilta, mutta se johtuu siitä, että niiden ainoa yhteys muuhun ohjelmaan on avata uusi FXML-ikkuna ja sitä myöten uusi FXMLController. Tämä yhteys ei näy kaaviossa.

6 YHTEENVETO

Ohjelma on toteutettu ikkunaperiaatteella, joten yksi isompi ongelma oli käskynsiirto ikkunoiden välillä. Päädyin käyttämään oliotallennusta, jossa ikkuna A tallentaa tarvittavan tiedon olioon C ja avaa sitten ikkunan B. B taas lukee avautuessaan tiedot oliosta C. Tämä vaati staattisten metodien käyttöä. Ratkaisua on käytetty esimerkiksi virheilmoituksissa: varastoluokassa on String-muuttuja, joka pitää virheilmoituksen tekstin tallessa. Aina kun virheilmoitus avautuu, se lukee tekstin varastoluokasta ja käyttää sitä virheen kuvaustekstinä. Virheilmoituksen voi siis avata mistä päin ohjelmaa tahansa, jolloin haluttu virheteksti tallennetaan varastoon juuri ennen ikkunan aukaisua.

Toinen ongelma oli comboBox-pudotusvalikoiden päivitys, joka liittyy edelliseen ongelmaan. Kun esimerkiksi SmartPostit haetaan tietokantojen muokkausikkunassa, ne pitää jotenkin päivittää pääikkunan valikoihin. Yritin käyttää staattisia päivitysmetodeja ja siten staattisia comboBoxeja pääikkunassa, jolloin päivityksen voisi kutsua suoraan tietokantaikkunasta. Vaikka ratkaisu toimi IDE:ssä, valmiiksi kasattu ohjelmatiedosto ei siitä tykännyt. Ohjelma kaatui aina kun päivitystä yritettiin käyttää. Niinpä vaihdoin päivitysmetodin ajamaan itsensä aina, kun hiiren vie valikon päälle. Ratkaisu toimii, mutta aiheuttaa sen, että valikko nollautuu ja käyttäjän tekemä valinta katoaa jos vetäisee vahingossa hiirellä valikon päältä. Käyttäjän on varottava, ettei vie hiirtään valikon päälle, ellei halua tehdä uutta valintaa. Kyseessä ei ole bugi vaan ominaisuus.

Huomasin harjoitustyön teon aikana, että suunnitteluvaiheessa tehdyt virheet moninkertaistuvat helposti ellei niitä korjaa heti. Jokunen tietokannan määrittelyvaiheessa mukaan lipsahtanut virhe sinnitteli mukana lähes valmiiseen harjoitustyöhön asti, jolloin niiden korjaamisessa oli iso työ.

Esimerkkinä tästä tajusin ihan harjoitustyön loppuvaiheessa, että event-taulukko, jota käytin kirjanpitoon ja tietokannan yhteen sitomiseen, oli epäluotettava. Jos poisti tietokannasta esineen, josta oli historiamerkintöjä, merkintä ei enää toiminut, koska event-kanta viittasi sen id:hen. Kokeilin myös

tehdä pelkkään tekstiin id-viittausten sijaan pohjaavan historiataulun, joka toimi hyvin, mutta tietokanta sirpaloitui relaatiomallin vastaisesti osiin. Niinpä jouduin lopulta toteuttamaan molemmat ratkaisut yhtä aikaa. Historiatiedot tallennetaan event-taulukkoon id-viittauksina ja kopioidaan sieltä saman tien history-taulukkoon tekstimuodossa. Kun historiatietoja tarkastellaan, ne luetaan history-taulukosta.

Niinpä suunnitteluun olisi kannattanut käyttää alunperin enemmän aikaa. Ensi kerralla sitten.