Ylläpitodokumentti

Biocafe

Helsinki 7.9.2006

Ohjelmistotuotantoprojekti

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Kurssi

581260 Ohjelmistotuotantoprojekti ($9 + 1$ op, 6 ov)

Projektiryhmä

Sami Laiti
Mari Vierelä
Sampsa Lappalainen
Jaakko Nyman
Teemu Kemppainen

Asiakkaat

Harri Laine
Petri Kutvonen

Johtoryhmä

Juha Taina
Kimmo Simola

Kotisivu

`http://www.cs.helsinki.fi/group/biocafe`

Versiohistoria

Versio	Päiväys	Tehdyt muutokset
0.9	3.9.2006	Ensimmäinen versio
1.0	7.9.2006	Lopullinen versio

Sisältö

1 Johdanto

Tämä dokumentti on biometriseen tunnistukseen perustuvan kahvikassajärjestelmän, Biocafen, ylläpitodokumentti. Dokumentti on tarkoitettu helpottamaan järjestelmän ylläpitovaihetta. Dokumentissa on esitelty tiettyjen järjestelmän ominaisuuksien teknisiä yksityiskohtia, sekä annettu yleisiä neuvoja järjestelmän muokkaamiseen. Dokumentissa on myös esitetty järjestelmän testausvaiheessa havaittuja toiminnallisia sekä laadullisia epäkohtia ja selkeitä virheitä, jotka on pyritty dokumentoimaan mahdollisimman kattavasti. Lopuksi on esitetty järjestelmälle mahdollisia kehitysideoita, sekä selvitetty niitä varten valmiina olevia järjestelmän osia sekä niiden toimintaa.

2 Koodin ulkoasu

Muuttujien nimeämisessä on käytetty ns. Unkarilaista notaatiota luettavuuden parantamiseksi. Lisäksi on käytetty Javassa standardiksi muodostunutta CamelCase-nimeämistä. Eli muuttujien nimien alkuun tulee kolmikirjaiminen lyhenne muuttujan tyypistä jonka jatkeeksi lisätään yhdestä tai useammasta sanasta muodostuva muuttujan nimi. Kommentointikielenä on käytetty suomea.

Muuttujien ja objektien nimet on muodostettu seuraavasti:

Tyyppi	Lyhenne
int	int
double	dbl
long	lng
boolean	bln
byte	byt
String	str

Kuva 1: Javan primitiivityypit ja String-muuttujat

Tyyppi	Lyhenne
JPanel	pnl
JLabel	lbl
JButton	btn
JTabbedPane	tbp
JTable	tbl
JTextField	txf

Kuva 2: Käyttöliittymäkomponentit

Kaikille muille tyypeille etuliite on ”obj”.

Tyyppi	Lyhenne
ResultSet	rs

Kuva 3: ResultSet-objekti

Poikkeuksena edellä mainitulle, for-loopin indeksit on yleensä nimetty yksikirjaimisina ja kirjaimet ovat i, j tai k.

Metodit ja luokat on nimettty standardiin Java-tyyliin. Eli metodin nimen ensimmäinen sana alkaa pienellä kirjaimella, muut isolla. Luokkien nimet alkavat isolla kirjaimella. Aaltosulkeet sijoitetaan samalle riville ainoastaan luokan esittelyssä.

Esimerkki

```
public class Luokka {
  public int jokinMetodi()
    {
      while( true )
        {
          doSomething();
        }
    }
}
```

3 Järjestelmän vakiot

Tietokantayhteyteen liittyvät vakiot, kuten esimerkiksi tietokannan osoite ja käyttäjätunnus, löytyvät tiedostosta "faculty.properties". Käytettävä tiedosto määritellään luokassa DBLib. DBLib-luokassa on myös asetettuna vakioiksi järjestelmän kielet sekä niiden järjestys sovelluksen sisäisissä taulukoissa. Samoin luokassa on määritelty vakioksi järjestelmän käyttäjän rekisteröitävien sormien maksimimäärä.

Käyttäjien saldon huomautusraja on asetettu luokassa Utilities. Luokassa on myös asetettu vakioiksi sovelluksissa käytettävät fontit sekä niiden koot ja järjestelmästä tulostettavan raportin CSS-tyylitiedostoon viittavat tyylinimet. MainFrame-luokassa on määritelty vakioiksi ohjelman komentoriviparametrit, joiden avulla järjestelmä voidaan käynnistää eri näkymiin, käyttötilanteen mukaan. CoffeeDialog-luokassa on määritelty se aika, kuinka kauan palautenäkymä näkyy dialogissa. Samoin on määritelty se, kuinka kauan odotetaan, ennen kuin kahvihuoneen näkymä päivitetään oletustilaan.

DBBasicProduct-luokassa on määritelty kiinteästi järjestelmän eri tuotteiden erilaisten annoskokojen määrä. FingerPrint-luokassa on määritelty kynnysarvo, joka biometrisen tunnistuksen laadun tulee ylittää, jotta voidaan varmistua verrattavien sormenjälkien samuudesta. FingerPrintDialogissa on määritelty järjestelmän käyttäjän pakollisten sormien vähimmäismäärä.

4 Monikielisyyden toteutus

Sovelluksen monikielisyys on toteutettu Javan ResourceBundle-luokan avulla. Kun sovelluksessa halutaan tietyllä kielellä tietty sana, kutsutaan Utilities-luokan getString-metodia. Metodissa ResourceBundlelle määritellään kieli sen mukaan, mikä kieli on metodin kutsuhetkellä valittuna. Kun kieli on määritelty, hakee se kieleen liittyvästä tiedostosta sillä kielellä olevan sanan, jonka avaimena on getString-metodin saama merkkijono. Tiedostoja on kolme kappaletta, yksi kullekin kielelle. Tiedostot ovat tekstitiedostoja, joten sanoja voi muokata halujen mukaan. Tiedostojen nimet ovat:

- MessagesBundle_en_GB.properties
- MessagesBundle_fi_FI.properties
- MessagesBundle_sv_FI.properties

5 GUI-näkymien muokkaaminen

Tässä osassa on esitelty sovelluksen näkymien komponenttien luokkarakenne jonka avulla mahdollisia muutostöitä käyttöliittymään kannattanee lähteä tekemään. Jokaista sovelluksen dialogia varten on olemassa oma luokkansa. Näissä luokissa määritellään dialogin komponenttien asettamiset sekä dialogin käyttäytyminen. Suunnitteludokumentissa on kuvattu tarkemmin MainFramen sekä CoffeeDialogin komponenttihierarkia. Tämän lisäksi jokaista sovelluksen taulukkoa varten on oma AbstractTableModel-luokasta peritty TableModel, joka määrittää taulukon mitat sekä ulkoasun ja käyttäytymisen. Lisäksi kahvihuonenäkymässä on PortionSelector-luokka, joka muodostaa näkymässä olevan NumPad-painikeryhmän.

6 Sormenjäljet

Sormenjälkien tunnistaminen toimii järjestelmän mukana tulevalla laitteella luotettavasti, muttei täysin varmasti. Testauksen aikana ei ole kertaakaan sattunut väärää tunnistamista, tosin aina laite ei ole tunnistanut järjestelmään kuitenkin rekisteröityä sormea. Tämän voisi luultavasti korjata varmistamalla sen, että syötetyn sormenjäljen laatu on tarpeeksi hyvä.

Sormenjälkitunnistimen käytössä tulee myös huomioida seuraavaa; Tunnistin lukee helposti sormenjäljen monta kertaa, jos sormi annetaan tunnistimelle turhan kevyesti. Reilut sormenpainallukset näyttävät estävän tämän. Asia ei ole ongelma kahvihuoneen puolella, missä peräkkäiset sormenjälkisyötteet eivät aiheuta toimia edellisen käsittelyn ollessa kesken. Sormenjälkiä rekisteröidessä voidaan joutua peruuttamaan ja uusimaan sormien rekisteröinti.

Testauksessa havaittiin että sormenjälkien tunnistaminen tulee useammalla rekisteröidyllä sormenjäljellä huomattavan hitaaksi. Pääsyy tähän on ilmeisesti se, että testeissä on käytetty vapaassa levityksessä olevaa versiota laitteen ajurista. Hitautta tuo myös se, että vertaaminen tapahtuu aina lineaarisessa ajassa suhteessa rekisteröityjen sormenjälkien määrään. Muoto, jossa sormenjälki järjestelmän sisällä esitetään, ei ole mitenkään yksikäsitteinen esitys sormenjäljestä, ja jälkien samuus ratkaistaankin algoritmisesti. Jälkiä ei voi myöskään tallentaa hajautustauluun juuri edellämainitusta syystä. Erilaiset tavutaulukot joihin jälkien piirteet tallennetaan, voivat kaikki kuitenkin esittää samaa sormenjälkeä.

6.1 Sormenjälkien tarkastaminen

Sormenjälkien samuus tarkistetaan FingerPrint-luokan equals()-metodilla, joka palauttaa true, mikäli sormenjälki Griaulen tunnistus-metodin mukaan tunnistetaan ja tunnistuksen pisteytys ylittää tietyn raja-arvon. Raja-arvo on FingerPrint-luokassa määritelty julkinen vakio THRESHOLD.

6.2 Extractor-rajapinta

Rajapinta koostuu yhdestä palvelusta jota kutsutaan kun MainFrame huomaa, että sormenjälkilukija on vastaanottanut sormenjäljen. RegisterFingersDialogissa saadusta sormenjälkikuvasta muutetaan MainFramen palvelun avulla FingerprintTemplate-luokan ilmentymä ja asetetaan se dialogin omaan tietorakenteeseen, joka palautetaan lopulta MainFramelle kun tarpeeksi monta sormenjälkeä on luettu. CoffeeDialogissa tehdään alku samoin, mutta saadun FingerprintTemplaten avulla haetaan Utilities-luokalta sormenjälkeä vastaava käyttäjä.

7 Monisäikeisyyden toteuttaminen

Sormenjälkitunnistus ja tietokantaoperaatiot suoritetaan omissa säikeissään. Tämä sinänsä välttämättömäksi huomattu ominaisuus vaatii huomattavaa huolellisuutta muokattaessa ohjelmakoodia.

Utilities-luokassa on useita synkronoituja metodeita. Tämän tarkoitus on estää tiettyjä rinnakkaisuuden mahdollistamia poikkeuksia Utilities-luokan tietorakenteissa. Jokainen tietokantaa tarvitseva metodi suoritetaan omassa säikeessään. Yhden säikeen muokatessa käyttäjien listaa (Utilities.refreshData()) toinen saattaa olla iteroimassa samaista listaa. Synkronointi estää ConcurrentModificationException-poikkeusta tapahtumasta.

Monisäikeisyys on toteutettu määrittelemällä suoraan metodien koodissa Runnable-rajapinnan toteuttavia olioita, joiden run-metodissa määritellään erillisessä säikeessä suoritettavat lauseet. Tämä Runnable-olio annetaan Thread-konsturktorille ja kyseinen uusi Thread käynnistetään.

Tapa on suoraviivainen, mutta siitä seuraa ongelmia jos tekee varomattomia muutoksia.

Javan swing-komponentit eivät ole säieturvallisia. Tämä tarkoittaa sitä, että suurin osa niiden palveluista on määritelty toimimaan vain kutsuttaessa niitä säikeestä nimeltä EventDispatchThread. Se suorittaa kaikki Java-sovelluksen piirto- ja tapahtumankuuntelu-operaatiot.

Valtaosa toteutetun järjestelmän koodista suoritetaan tapahtumankuuntelijoiden kautta tuossa säikeessä, mutta tietokantaoperaatioita ja sormenjälkitunnistusta varten määritellyt omat säikeet suorittavat erikseen. Niissä on myös graafista käyttöliittymää päivittävää koodia. Järjestelmä toimii, koska osa swing-komponenttien metodeista on määritelty säieturvallisiksi ja säikeet käyttävät juuri niitä. Haluttaessa käyttää swing-komponentteja myös tietokantasäikeistä, on varmistuttava komponenttien käytettävien palveluiden säieturvallisuudesta.

Ajanpuutteen vuoksi kahvihuoneen näkymässä on estetty rinnakkaiset osto- ja tuontitapahtumat hyvin alkeellisella ja karkealla menetelmällä. Uudet sormenjälkisyötteet ja Enter-napin painallukset (PIN-koodin syöttö) aloittavat uuden osto- tai tuontitoiminnon vain, jos kenttä identifying on ’false’. Toiminnon aluksi tuohon kenttään asetetaan ’true’ ja jatketaan. Kentän arvoksi tulee palauttaa ’false’ kaikissa tapauksissa, joissa toiminnon suoritus päättyy ja seuraava voi alkaa. Nämä tapaukset on lueteltu oheisessa taulukossa. Muuten järjestelmä jumiutuu eikä uusia toimintoja voi aloittaa.

Metodi	Rivi
`Pin-syöttökentän tapahtumankuuntelija`	`511`
`buyDrink(User)`	`611, 617`
`Deliver(User)`	`666, 672`
`reset()`	`972`
`extract(FingerprintImage)`	`1173`

Kuva 4: Ne CoffeeDialog-luokan kohdat, joissa identifying-kenttää muutetaan

Kehitysidea: Tietokantaoperaatioiden ja sormenjälkitunnistuksen säikeet olisivat selkeämpiä toteutettuna SwingWorkerien avulla. Niissä guin muokkausoperaatiot ja varsinainen aikaa vievä työ on eristetty omiin metodeihinsa. Myös säieturvattomat guin päivityslauseet voi laittaa niihin, sillä päivitys suoritetaan Event Dispatch Threadissa erityisen SwingUtilities.invokeLater(Runnable) -metodin avulla. Metodia voi käyttää myös ilman SwingWorker-luokkaa. Lue lisää Java Tutorialista otsikon "How to Use Threads"alta.

Kehitysidea: Ruma ja virhealtis kahvihuoneen rinnakkaisten operaatioiden esto tulisi muuttaa paremmaksi. Työ ei ole mitenkään yksinkertainen ja vaatinee muutoksia luokan metoditasolla asti.

8 Huomioita

8.1 Huomioita tuotteiden passivoinnista

Kahvihuonedialogissa ei voi ostaa eikä tuoda passiivisiksi määriteltyjä tuotteita. Siitä huolimatta ostettaessa ei-passiivista juomaa käyttäjän sen kanssa käytettäväksi asettamien passiivistenkin lisukkeiden kulutus kasvaa.

Lisukedialogi ei ota tuotteiden passiivisuutta mitenkään huomioon. Käyttäjien lisukeasetukset säilyvät vaikka lisukkeita tai juotavia passivoitaisiin. Passiivisia ja aktiivisia lisukkeita voi asettaa kaikille juotaville, myös passiivisille.

8.2 Huomioita luokkatestauksesta

Suurin osa järjestelmän luokista luokkatestattiin suunnittellun kattavuuden puitteissa. Osa luokista jäi kuitenkin vajavaisesti tai kokonaan testaamatta perinteisessä luokkatestausmielessä. Näistä luokista kuitenkin kaikki on mm. vaikeiden riippuvuushierarkioiden johdosta testattu järjestelmätestauksen kautta. Näitä olivat ButtonRenderer, ButtonEditor, UserView, BasicProductView, DeliverableView, CoffeeDeliverableView, MainFrame (pääohjelma), Utilities ja RegisterFingersDialog. Viimeisintä ei ollut mahdollista testata yksinään. Muiden kohdalla katsottiin, että aikaa ei ollut riittävästi ja luokat olivat osittain luonteeltaan sellaisia, että luokkatestaus ei ollut mielekäs. Esimerkiksi nk. View-luokat ovat JTable-komponenttien tietosisällön malleja, jonka metodeita ei suoraan kutsuta missään ohjelman koodissa, vaan niitä kutsuu kunkin oma JTable. Virheet View-luokissa näkyvät yleensä suoraan siinä, miten JTable piirtää itsensä ja miten se käyttäytyy. Sama koskee ButtonRenderer ja ButtonEditor-olioita. Luokkien testaushierarkian loppupäässä luokat ovat niin vahvasti toisistaan riippuvaisia, että todellinen luokkatestaus ei ollut edes mahdollista.

9 Kehitysideoita

9.1 Aikaleimojen lisääminen kirjaustapahtumiin

Kirjauksen yhteydessä suunniteltiin alunperin talletettavan aikaleima tapahtumasta. Näin raportissa olisi voitu seurata käyttöä tarkemmin esimerkiksi jakamalla viimeisen kuukauden ja kaikkien aikojen kulutus erilleen. Myös yksittäisten tapahtumien kumoaminen olisi tullut mahdolliseksi. Aikaleimoista ehdittiin toteuttaa jonkin verran koodia ja sitä löytyy lähdekoodista pois kommentoituna. Jos ominaisuus halutaan lisätä myöhemmin, noita koodinpätkiä voi tutkia, vaikka niiden toiminnasta ei olekaan mitään takeita.

9.2 Varastosaldon esittäminen

Eräänä vaatimuksena oli, että järjestelmä esittäisi arvion kahvihuoneessa olevien tuotteiden varastosaldosta. Tätä varten tehtiin tietokantaankin taulu. Taulun rakenne on seuraava:

- Viite perustuotteeseen
- Määrä

Viite perustuotteeseen kuvaa tuotetta, jonka saldoa tietty rivi taulussa kuvaa. Määrä on arvio siitä määrästä, minkä verran tuotetta järjestelmän mukaan on. Tarkoituksena on, että käyttäjä pääsee suoraan asettamaan tuon määrän jokaiselle tuotteelle, ja jokaisen kulutuksen kirjauksen yhteydessä määrää vähennettään jollain määrällä. Määrä voisi olla esimerkiksi annoskoko, mutta se ei liene täysin ongelmaton, annokset ovat kuitenkin ilmeisesti laskettu yläkanttiin.

9.3 Käytettävyysasioita

Aikatauluhaasteiden lisäksi graafisen osaamisen puute sekä Javan GUI-ohjelmoinnin hyvin rajalliset resurssimme vaikuttivat siihen, että käytettävyyden suhteen jouduttiin tekemään kompromisseja. Merkittävimmät suunnitelluista, mutta toteuttamatta jääneistä käytettävyyteen liittyvistä ominaisuuksista esitellään seuraavaksi jatkokehitysideoina.

9.3.1 Toteutus kosketusnäytön avulla

Kahvihuonesovellus suunniteltiin alun perin toteutettavaksi kosketusnäppäimistön avulla, mikä olisi ollut käytettävyyden kannalta nykyistä toteutusta parempi vaihtoehto. Tästä luovuttiin kuitenkin liian suuren riskin vuoksi. Kosketusnäytön avulla oltaisiin oltu vapaampia kahvihuonenäkymän käyttöliittymän suunnittelussa.

9.3.2 Näppäimistön ulkoasu

Nykyinen käyttöliittymä suunniteltiin sitä silmällä pitäen, että järjestelmää käytetään numeronäppäimistön ja tunnistinlaitteen avulla. Suunnitelmissa oli, että näytöllä näkyvä näppäimistö muistuttaisi olennaisesti ulkonäöltään oikeaa näppäimistöä, mutta tätä käytettävyyteen ratkaisevasti vaikuttavaa ominaisuutta ei pystytty toteuttamaan. Se olisi selkeästi helpottanut numeronäppäimistön ja näytöllä näkyvän "näppämistön"yhteyden ymmärtämistä.

9.3.3 Alasvetovalikkojen ulkoasu taulukoissa

Alasvetovalikot eri taulukoissa näyttävät tyhjiltä soluilta. Siis niistä ei käy ilmi, että ne ovat alasvetovalikkoja ennen kuin niitä painaa hiirellä.

Kehitysidea: Tämän voisi ehkä korjata pelkästään vaihtamalla Javan LookAndFeel toiseksi, mutta ainakin toteuttamalla JComboBoxille oman TableCellRenderer-luokan.

9.3.4 Palaute

Hallintanäkymässä voisi olla enemmän palautetekstejä. Nykyiset palautteet kertovat lähinnä virhetilanteista. Olisi kuitenkin viisasta tarjota ns. kuittaus myös onnistuneista toimenpiteistä, kuten tietojen tallentumisesta tietokantaan. Hallintanäkymän käyttäjätietotaulukkoon oli suunniteltu toteutettavaksi värikoodaus suurten velkojen löytämisen helpottamiseksi, mutta tätä toiminnallisuutta ei ehditty toteuttaa muualle kuin raporttiin.

Myös kahvihuonenäkymän palautteissa olisi parantamisen varaa. Dynaamisesti päivittyvät tekstit, kuten valittu tuote, kirjattu tuote ja saldotiedot, oltaisiin haluttu esittää siten, että ne erottuisivat selkeämmin staattisista teksteistä. Palautesivun saldotiedoissa olisi myös ollut hyvä käyttää samanlaista värikoodausta kuin raportissa, jotta käyttäjän olisi helppo huomata suureksi kasvaneet velkansa ja reagoida niihin.

9.3.5 Tallennusongelmat

Hallitanäkymän editoitava taulukko aiheuttaa haasteita muokattujen tietojen tallentamiseen tietokantaan ja nykytoteutukseen jäi joitakin ikäviä käytettävyysongelmia. Tiedot on tässä toteutuksessa muokattava ja vietävä tietokantaan rivi kerrallaan. Käyttäjätaulukossa useamman rivin editointi ennen tallennusta on estetty, mutta virheitä ja tietojen katoamista voi yhä tapahtua tuotesivulla tai välilehtien välillä vaihdeltaessa. Nykyinen toimintalogiikka siis ei estä käyttäjää hukkaamasta editoimiaan tietoja joko vaihtamalla ennen tallennusta välilehteä tai editoimalla tuotteet-välilehdellä molempia taulukoita ennen tallentamista.

9.3.6 Sormenjälkien tunnistaminen ja rekisteröinti

Sormenjälkien rekisteröintiin liittyvää logiikkaa jouduttiin yksinkertaistamaan alkuperäisiin suunnitelmiin nähden. Alun perin suunniteltiin erityisiä varmistustoimia, joilla taata käyttäjälle rekisteröityjen sormenjälkien olevan ovat hyvälaatuisia ja helposti tunnistettavissa jatkossa. Toimet sisälsivät jokaisen sormen lukemisen useammin kuin kerran. RegisterFingersDialog-luokan vakio TIMES_FINGER kertoo, kuinka monta kertaa dialogi haluaa lukea saman sormen. Eri kertojen tulosta vertailemalla voitaisiin varmistaa, että sormesta rekisteröidään hyvälaatuinen tietue.

9.3.7 Ohje-näkymä kahvihuoneeseen

Koska kahvihuoneen käyttöliittymän opittavuus herättää tällaisenaan epäilyksiä, olisi ollut viisasta lisätä kahvihuonenäkymään yhden napinpainalluksen taakse ohje-näkymä, mistä käyttäjä voisi ongelmatilanteessa käydä lukemassa toimintaohjeita.

9.3.8 Kahvinäkymän näyttökoko

Kahvihuonesovellus suunniteltiin käytettäväksi vähintään 1280x1024 -näyttöresoluutiolla. Tätä pienemmillä resoluutioilla pitkät tuotteiden nimet eivät välttämättä näy kokonaan niissä kohdissa missä valittu tuote lukee suurin kirjaimin.

Kehitysidea: Toteutetaan fonttikokojen asettaminen dynaamisesti resoluution mukaan.

9.3.9 Termistön tarkistaminen

Käyttöliittymässä käytetty termistö ei ole kaikilta osin aivan loppuun asti harkittua ja erityisesti englanninkieliseen versioon jäi termejä, joiden sopivuudesta ei oltu täysin varmoja.