Ville Pulli

SPOCO-RANNEKKEEN KÄYTTÖLIITTYMÄN SUUNNITTELU

SPOCO-RANNEKKEEN KÄYTTÖLIITTYMÄN SUUNNITTELU

Ville Pulli
Opinnäytetyö
Syksy 2012
Hyvinvointiteknologian ko.
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Hyvinvointiteknologian ko.

Tekijä: Ville Pulli
Opinnäytetyön nimi: SpoCo-rannekkeen käyttöliittymän suunnittelu
Työn ohjaaja: Jukka Jauhiainen
Työn valmistumislukukausi ja -vuosi: Syksy 2012 Sivumäärä: 32

Oulun seudun ammattikorkeakoulun Yrityshautomon SpoCo-projektissa rakennetaan kuntosalikäyttöön tarkoitettua rannelaitetta, joka pystyisi liikkeentunnistuksen avulla arvioimaan erilaisten liikesarjojen suoritustavan oikeellisuutta. Projektissa on mukana OAMK:n lisäksi Jyväskylän sekä Lapin yliopistot.

Tämän opinnäytetyön tavoitteena oli suunnitella rannelaitteen käyttöliittymä sekä ohjelmoida pc-ympäristössä ajettava simulaatio käyttöliittymästä, jolle voidaan suorittaa käytettävyystestejä. Varsinaisen sulautetun käyttöliittymän toteuttaminen ei kuulu tähän opinnäytetyöhön.

Pc-simulaatio käyttöliittymästä toteutettiin Java-ohjelmointikielellä Eclipse-ohjelmointiympäristössä.

Tuloksena tästä opinnäytetyöstä syntyi suunnitelma käyttöliittymälle sekä pc-ympäristössä ajettava simulaatio-ohjelma, jolle voidaan suorittaa käytettävyystestausta. Ohjelman lähdekoodi pyrittiin myös kirjoittamaan mahdollisimman helppolukuiseksi runsaan kommentoinnin kera, jos sitä halutaan käyttää esimerkiksi demomateriaalina SpoCo-projektissa.

Asiasanat: SpoCo , käyttöliittymäsuunnittelu, applet, java, eclipse

Sisällysluettelo

1 JOHDANTO

Opinnäytetyön tarkoituksena oli suunnitella käyttöliittymä kuntosalikäyttöön tarkoitetulle rannelaitteelle sekä ohjelmoida käyttöliittymästä pc:llä ajettava simulaatio. Opinnäytetyö kuuluu Oulun seudun ammattikorkeakoulun Yrityshautomon SpoCo-projektiin. SpoCo-projektissa suunnitellaan ja toteutetaan rannelaitetta, joka liikkeentunnistuksen avulla pystyy arvioimaan erilaisten liikeratojen oikeellisuutta kuntosaliharjoittelussa.

Kuntosaliharjoittelussa käytetään useita laitteita, joissa käyttäjän raajojen liikkuminen harjoituksen aikana tapahtuu kohtuullisen samaa, ennustettavaa liikerataa pitkin. Tämä tarkoittaa että yksittäinen liikkeen toisto pystytään mittaamaan gyroskooppien avulla tarkasti ja siitä voidaan kerätä dataa. Liikkeen mittaaminen gyroskoopein ilman kuntosalilaitteiden tuomaa hallittua liikerataa on erittäin vaikeaa, mutta kun tämä liikerata tiedetään, pystytään gyroskoopin antamasta datasta päättelemään tarkasti muun muassa liikkeen alku- ja loppupisteet, tai käyttäjän tekemät erheet. SpoCon avulla kokenut kuntosaliharjoittelija pystyy saamaan paljon enemmän tietoa harjoittelunsa vaikutuksista, mutta sen avulla voidaan myös opastaa aloittelijaa tekemään liikkeet oikein.

Käyttöliittymän suunnittelussa pohjana oli SpoCo-projektin tiimin listaamat toivotut ominaisuudet sekä niiden pohjalta laadittu vuokaavio. Käyttöliittymäsuunnittelun aikana tiimi myös antoi palautetta sekä parannusehdotuksia. Varsinainen sulautettu käyttöliittymä rannekkeeseen toteutettaisiin vasta myöhemmin erikseen niin sanottuun 3Generaatio-versioon.

Simulaatio-ohjelma toteutettiin java-ohjelmointikielellä. Ohjelmointiympäristönä toimi ilmainen avoimeen lähdekoodiin perustuva Eclipse.

2 TEORIA

Käyttöliittymäsuunnittelu on useiden erilaisten laitteiden, ohjelmistojen tai verkkosivustojen suunnittelua siten, että käyttäjäkokemus on parhain mahdollinen. Tarkoituksena on luoda käyttöliittymä, joka on mahdollisimman yksinkertainen ja tehokas käyttäjän näkökulmasta. Hyvä käyttöliittymä mahdollistaa käyttäjän tehtävien suorittamisen ilman että huomio kiinnittyisi varsinaisesta tehtävästä käyttöliittymän käyttöön. Käyttöliittymäsuunnittelu etenee vaiheittain; joillakin voi olla suurempi prioriteetti kuin toisilla riippuen projektin laadusta. (Holappa 2010.)

2.1 Vaatimusmäärittely

Vaatimusmäärittelyssä laaditaan lista käyttöliittymän toiminnoista ja ominaisuuksista, jotka ovat tarpeellisia projektille sekä loppukäyttäjälle. Vaatimusmäärittelyä voidaan muokata vielä projektin edetessä, esimerkiksi käyttötestauksen perusteella. Vaatimusmäärittelyssä keskitytään siihen, mitä järjestelmän pitäisi tehdä, eikä siihen, kuinka järjestelmä tulee toimimaan. (Faulkner 2000, 95.)

Käyttäjäanalyysissä analysoidaan käyttöliittymän loppukäyttäjää, joko ottamalla yhteyttä suoraan tuleviin käyttäjiin tai heidän kanssaan työskenteleviin ihmisiin. Käyttäjäanalyysissa pyritään vastaamaan mm. kysymyksiin

- mitä käyttäjä haluaisi käyttöliittymän tekevän
- miten uusi käyttöliittymä sopisi käyttäjän normaaliin rutiiniin
- kuinka teknisesti lahjakas käyttäjä on
- mitä samanlaisia järjestelmiä käyttäjällä on
- millainen käyttöliittymän ulkonäkö ja käyttökokemus kiinnostavat käyttäjää.

Potentiaaliseen käyttäjään tutustuminen muun muassa näiden kysymysten avulla auttaa luomaan järjestelmän loppukäyttäjistä yleiskuvan, jonka avulla järjestelmä pystytään suunnittelemaan mahdollisimman käyttäjäystävälliseksi.

Käyttäjät voidaan jakaa neljään eri tyyppiin perustuen siihen, kuinka he käyttävät järjestelmää. (Faulkner 2000, 47.)

Suorat ja **epäsuorat käyttäjät** ovat nimensä mukaisesti joko suoraan käyttämässä järjestelmää (esimerkiksi tekstieditori) tai epäsuoraan jonkin muun avustuksella (esimerkiksi ajanvarausjärjestelmä tai potilastietojärjestelmä, jonka varsinainen käyttäjä ei välttämättä itse istu järjestelmän edessä). **Etäkäyttäjä** käyttää muualla sijaitsevaa järjestelmää välimatkan päästä (esimerkiksi verkkopankki). **Tukikäyttäjä** auttaa toisia käyttäjiä järjestelmän käytössä. Kullakin käyttäjätyypillä on on omat erikoisvaatimuksensa, jotka tulee ottaa vaatimusmäärittelyssä huomioon.

2.2 Informaatioarkkitehtuuri

Informaatioarkkitehtuurissa laaditaan järjestys, jossa eri toiminnot ja/tai informaatio esitetään, esimerkiksi SpoCon käyttöliittymässä toimintojen sijainti joko alavalikossa tai heti perustilassa. Informaatioarkkitehtuurissa on erittäin tärkeää, että käyttäjän vaatimukset on ymmärretty oikein vaatimusmäärittelyn yhteydessä.

Informaatioarkkitehtuuri alkaa käyttäjän tehtävien, ”taskien” suunnittelulla. Nämä ovat tehtäviä, joita vaatimusmäärittelyssä analysoidun käyttäjän tulee tehdä, että hän saavuttaa tavoitteensa. Tavoite on lopputulos, jota käyttäjä haluaa. (Barnum 2011, 99.) Esimerkiksi SpoCo-rannekkeen käyttö sekuntikellona vaatii valikkorakenteessa navigoimista.

Taskien avulla käyttöliittymän toiminnot ja informaatio voidaan esittää ja järjestellä tavalla, joka helpottaa havainnollisuutta ja ymmärtämistä. Taskien suunnittelu ensimmäisenä auttaa yksinkertaistamaan ja modularisoimaan ohjelmiston rakennetta ja samalla vahvistaa käyttäjän tarpeiden oleellisuutta. (Rosson & Carroll 2002, 106, 109.) Ihmiset ovat keskittyneitä tavoitteihinsa, ja siksi taskit tulisi suunnitella siten, että ne mahdollistavat tavoitteen saavuttamisen mahdollisimman yksinkertaisella ja nopealla tavalla. (Barnum 2011, 84.)

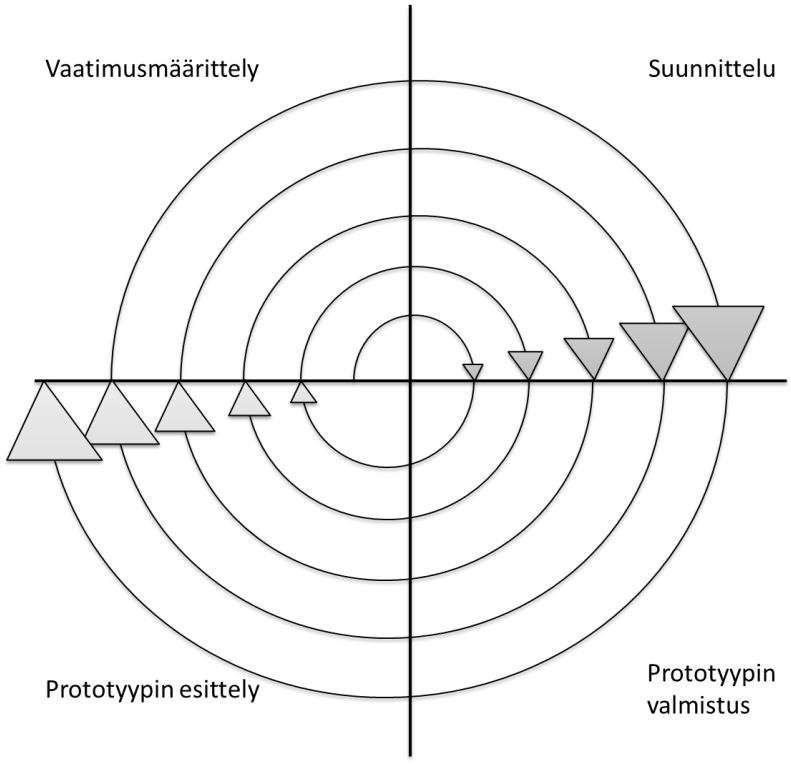
Taskien suunnittelun tavoitteena on määrittää järjestelmän toiminnallisuus, eli mitä sillä voi lopulta tehdä, mitä informaatiota se sisältää tai voi hakea, mitä operaatioita se voi tehdä tämän informaation perusteella, ja mitä tuloksia se antaa näistä operaatioista. (Rosson & Carroll 2002, 79.)

2.3 Prototyypin valmistaminen

Prototyypin valmistaminen tarkoittaa sellaisen prototyypin suunnittelua ja valmistusta, joka keskittyy käyttöliittymän käytön esittelyyn. Ulkonäköön ei panosteta enempää kuin on pakko, vaan se tapahtuu myöhemmin graafisessa suunnittelussa.

Prototyyppi on konkreettinen, muttei lopullinen, näkemys järjestelmästä. Käyttöliittymäprototyyppi on prototyyppi, joka keskittyy käytettävyyden tutkimiseen. (Rosson & Carroll 2002, 198.) Prototyyppejä voidaan valmistaa useissa projektin vaiheissa; niitä voidaan käyttää myös vaatimusmäärittelyn iterointiin. (Rosson & Carroll 2002, 8.)

Prototyyppimenetelmä (kuva 1) on ohjelmistotuotannon menetelmä, jossa ohjelman kehitys etenee spiraalimaisesti. Vaatimusmäärittelyn perusteella suunnitellaan ja valmistetaan prototyyppi, joka esitellään tilaajalle. Tilaajan palautteen mukaan iteroidaan vaatimusmäärittelyä tarvittaessa ja prototyyppiä muokataan sen mukaan. Kierros kierrokselta prototyyppi lähestyy lopullista muotoaan. (Väisänen 2010.)



KUVA 1. Prototyyppimenetelmän spiraalimalli.

2.4 Käyttötestaus

Käyttötestauksessa valitaan käyttäjäanalyysin perusteella testihenkilöitä tai jo tiedettyjä loppukäyttäjiä testaamaan prototyyppiä. Testauksen perusteella voidaan joko hyväksyä prototyypissä esitelty ratkaisu tai lähettää prototyyppi jatkokehittelyyn. Testauksen perusteella voidaan myös muokata tarpeen mukaan vaatimusmäärittelyä tai informaatioarkkitehtuuria. Testausta olisi kannattavaa jatkaa aina projektin loppuun saakka.

2.5 Graafinen suunnittelu

Graafisessa suunnittelussa laaditaan prototyypin ja käyttötestauksen pohjalta graafinen ilme käyttöliittymälle. Graafinen suunnittelu on mielekästä tehdä vasta prototyypin avulla suoritetun käyttötestauksen jälkeen, kun suunniteltu käyttöliittymä on todettu toimivaksi.

3 KÄYTTÖLIITTYMÄN SUUNNITTELU

3.1 Vaatimusmäärittely

Pohjaksi käyttöliittymän suunnittelulle annettiin laitteen valmiiksi suunniteltu ulkokuori, jossa oli näytön lisäksi neljä painiketta, kaksi vierekkäin sekä näytön ylä- että alapuolella. Vaatimuksiksi käyttöjärjestelmälle annettiin laitteen perusominaisuuksien näyttäminen järkevällä tavalla sekä valmius mahdollisille uusille ominaisuuksille SpoCo-projektin edetessä. Simulaation tarkoituksena oli toimia demoamismateriaalina sekä käytettävyystestauksen kohteena OAMK:n projektikurssissa.

Opinnäytetyön aikana SpoCo-projektissa oltiin työstämässä ensimmäistä prototyyppiä, jossa ei ole graafista käyttöliittymää. Tämän vuoksi opinnäytetyössä ei tehdä varsinaista käyttöliittymää joka käytettäisiin itse laitteessa, vaan pc-ympäristössä ajettava simulaatio. Tätä simulaatiota voidaan käyttää käytettävyystestauksessa sekä demoamisvälineenä SpoCo-projektin aikana.

SpoCo-tiimi oli jo laatinut vaatimusmäärittelyn sekä käyttäjäanalyysin käyttöliittymän suunnittelua varten. Vaatimusmäärittely oli kuitenkin edelleenkin työn alla ja uusia ominaisuuksia saatettiin lisätä vielä tämän opinnäytetyön jälkeen. Käyttöliittymä piti siksi suunnitella siten, että mahdolliset uudet ominaisuudet voitaisiin lisätä mukaan helposti.

Käyttöliittymän suunnittelun yhteydessä laadittiin käyttöliittymän logiikkaa selittävää niin sanottua wireframe-dokumenttia, jolla perusideat esiteltiin SpoCo-tiimille. Yksinkertaisten kuvien avulla käyttöliittymän perustoiminnallisuus saatiin selitettyä tehokkaasti, ja tiimi pystyi myös antamaan kehitysehdotuksia. Myöhemmin simulaatioappletin kehityksen aikana uudet ominaisuudet ja muutokset pystyttiin esittelemään simulaatioappletista suoraan.

SpoCon käyttäjä on vaatimusmäärittelyn mukaan kuntosaliharjoittelija, joka on tottunut nykyaikaisten laitteiden käyttäjä. Tämä käyttäjätyyppi on suora käyttäjä. SpoCoa myydään kuntosaleille, joten käyttäjiksi lasketaan myös kuntosalin työntekijät, jotka toimivat tukikäyttäjinä.

3.2 Informaatioarkkitehtuuri

Käyttöliittymän suunnittelu alkoi SpoCo-rannekkeen 3d-mallin pohjalta (kuva 2). 3d-malli asetti käyttöliittymälle tietyt vaatimukset; mallissa oli näytön ylä- ja alapuolelle varattu tilaa leveille painikkeille, mutta muita paikkoja painikkeille ei ollut. Alussa tuli selväksi, että SpoCon suunniteltujen ominaisuuksien käyttämiseksi tarvittiin enemmän kuin kaksi painiketta, mutta toisaalta niiden määrä tulisi pitää mahdollisimman vähäisenä.

KUVA 2. Kuvakaappaus alkuperäisestä 3d-mallista

Koska painikkeille varattu alue oli leveä, ne olivat yksinkertaisia muuttaa kukin kahdeksi vierekkäiseksi painikkeeksi. Tämän jälkeen painikkeiden logiikan suunnittelu voitiin aloittaa.

Ylemmät kaksi painiketta toimivat vahvistus- ja peruuta-painikkeina, kun taas alemmat kaksi toimivat nuolipainikkeina, joilla liikuttaisiin käyttöliittymässä ja valittaisiin haluttu toiminto (kuva 3). Valittu toiminto vahvistettaisiin sitten yläreunan vahvistus-painikkeella, tai vaihtoehtoisesti edelliseen valikkoon palattaisiin peruuta-painikkeella. Painikkeilla olisi myös vaihtoehtoisia toimintoja

tietyissä käyttöliittymän tiloissa. Esimerkiksi sekuntikello-valikossa nuolipainikkeet toimisivat kellon aloitus/pysäytys- ja nollaus-painikkeina, kun taas kellon asetuksissa niillä vaihdettaisiin tunteja ja minuutteja. Näissä tapauksissa painikkeiden erilainen toiminnallisuus selitettäisiin näytölle painikkeen ääreen ilmestyvällä selitetekstillä.

KUVA 3. Painikkeet. Kuva wireframe-dokumentista.

Käyttöliittymäsuunnittelun yhteydessä luotu wireframe-dokumentti löytyy liitteestä 1.

3.3 Käytetyt työkalut

3.3.1 Eclipse

Opinnäytetyön simulaatioappletin kehitystyökaluna käytettiin Eclipseä. Eclipse on ilmainen avoimen lähdekoodin integroitu kehitysympäristö, joka tukee mm. Java-sovellusten kehittämistä. Eclipsen käytetty versio oli Eclipse IDE for Java EE Developers, Indigo Service Release 1.
(The Eclipse Foundation 2012. http://www.eclipse.org/)

3.3.2 Paint.NET

Opinnäytetyön käyttöliittymän graafisen ilmeen suunnitteluun sekä simulaatio-ohjelman graafisten elementtien luomiseen käytettiin ilmaiseksi saatavilla olevaa Paint.NET -kuvankäsittelyohjelmaa.
(Brewster, Rick, dotPDN LLC 2012. http://www.getpaint.net/)

3.3.3 Microsoft PowerPoint 2010

Opinnäytetyön käyttöliittymän toimintaperiaatteen selittävä dokumentti luotiin Microsoftin Office-pakettiin kuuluvalla PowerPoint 2010 -ohjelmalla.
(Microsoft 2010. http://office.microsoft.com/fi-fi/products/?CTT=97)

3.3.4 Java SE 6

Java on Sun Microsystemsin kehittämä ja nykyisin Oraclen omistama laitteistoriippumaton ohjelmistoalusta. Javan syntaksi on hyvin samanlainen kuin C:llä ja C++:lla mutta sen oliomalli on paljon yksinkertaisempi, eikä se myöskään ole aivan yhtä matalan tason ohjelmointikieli kuin C tai C++. Javalla kirjoitettu lähdekoodi käännetään luokkatiedostoksi, jota voidaan ajaa millä tahansa Javan virtuaalikoneella (JVM, Java Virtual Machine). Tämän takia Javalla kirjoitettuja ohjelmia voidaan ajaa lähes millä tahansa alustalla. Java on tällä hetkellä yksi suosituimmista ohjelmointikielistä laitteistoriippumattomuutensa ansiosta, erityisesti client-server-verkkosovelluksissa.

Java on saatavilla kenelle tahansa ilmaiseksi GNU GPL -lisenssillä Oraclen verkkosivulta.
(Oracle 2012. http://www.oracle.com/technetwork/java/javase/downloads/index.html)

4 SIMULAATIOAPPLETTI

Suunnitellun käyttöliittymän toiminnallisuuden testaukseen laadittiin tietokoneella ajettava simulaatiosovellus. Sovellus luotiin Java-ohjelmointikielellä Eclipse-ohjelmointiympäristössä. Sovellus on tyypiltään niin sanottu Java-appletti, joka voidaan ajaa suoraan tietokoneen internetselaimella. Varsinainen ohjelma voi siis olla myös erillisellä palvelimella, jota voidaan ajaa internetyhteyden yli toisella tietokoneella. Appletin ajamiseen tarvitaan ainoastaan tietokoneelle asennettu Java Runtime Environment (JRE) -paketti sekä internetselain. Käytetty javan versio oli Java SE 6.

Työskentelyssä käytettiin aiemmin kuvassa 1 esitettyä prototyyppimenetelmän spiraalimallia, jossa prototyyppiä iteroidaan saadun palautteen perusteella.

4.1 Appletin toiminta

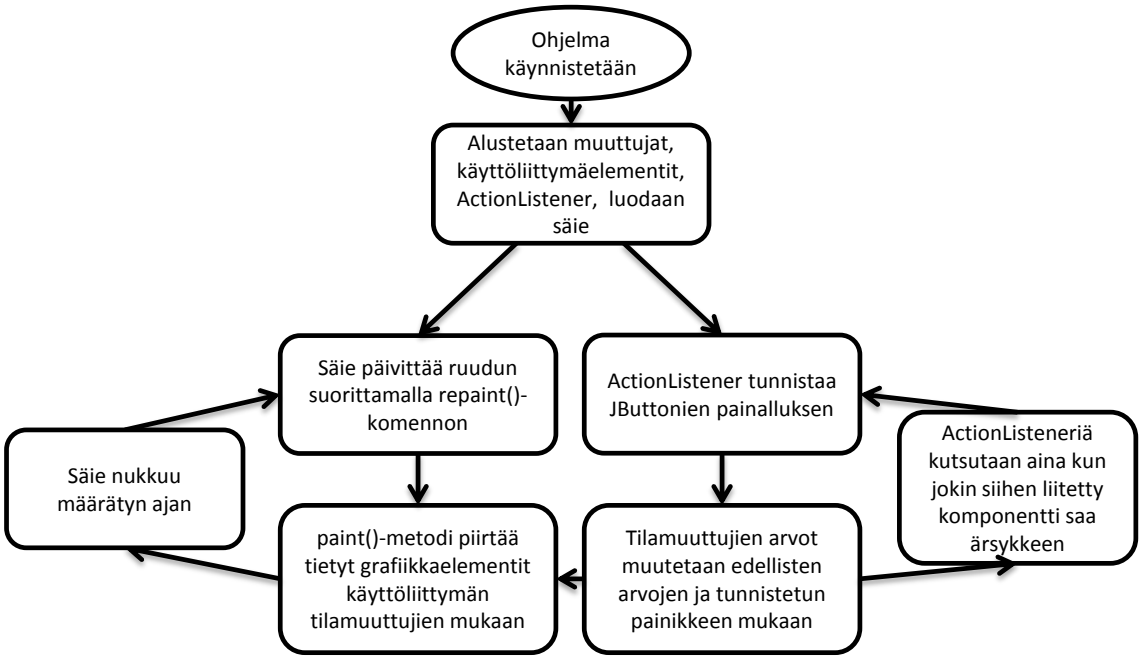
Appletti käynnistetään aukaisemalla SpocoSim.html-tiedosto verkkoselaimella. HTML-tiedostossa on tagi, joka käynnistää varsinaisen ohjelman sisältävän SpocoSim.class-tiedoston. Class-tiedosto on käännettyä ohjelmakoodia sisältävä tiedosto, joka voidaan ajaa millä tahansa Java Virtual Machinella (JVM).

Kun appletin sisältävä html-tiedosto on aukaistu, appletin koodi siirretään verkkopalvelimelta käyttäjän koneelle, jossa verkkoselain joko upottaa appletin verkkosivulle tai aukaisee sen erilliseen ikkunaan. Appletti ajetaan ns. hiekkalaatikossa, jossa se ei pääse vaikuttamaan tietokoneen muun ohjelmiston toimintaan.

Java-appletin lähdekoodi poikkeaa hieman normaalin sovelluksen lähdekoodista, sillä Javan periytymisen vuoksi perinteistä main()-metodia ei tarvitse kirjoittaa erikseen, vaan se periytyy appletille suoraan joko java.applet.Applet- tai java.applet.JApplet-luokista. Appletin oman luokan pitää ylikirjoittaa tietyt perityt metodit, joilla haluttu käyttöliittymä tai graafiset elementit alustetaan.

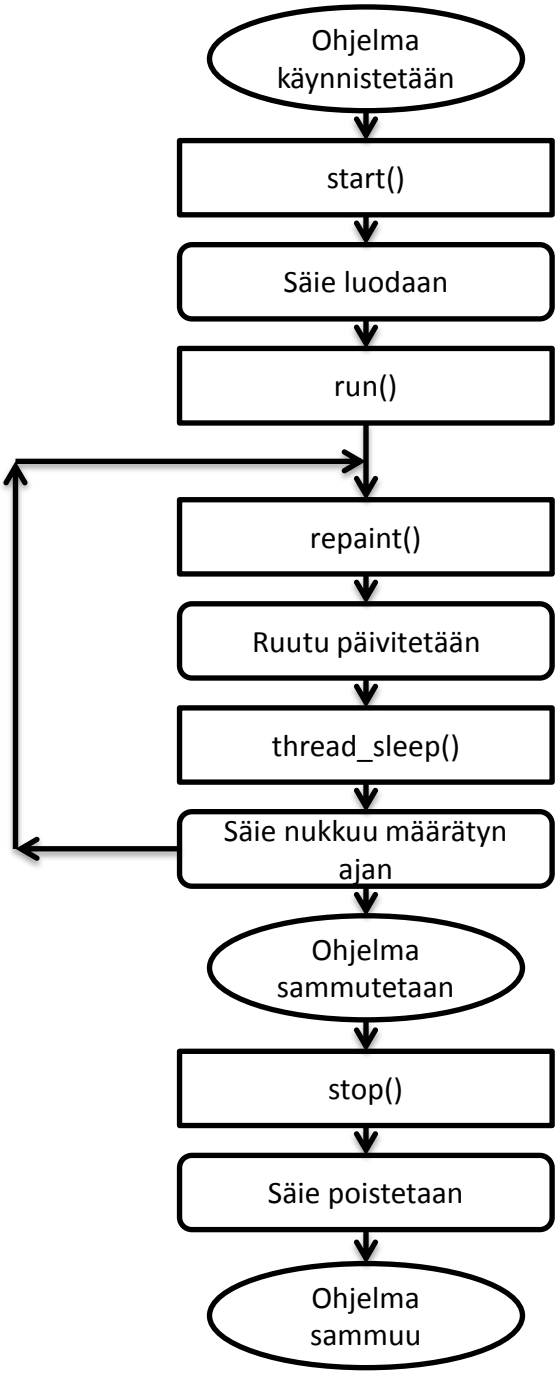
Kuvassa 4 on esitetty ohjelman toiminta. Ohjelman käynnistyessä luodaan ja alustetaan muuttujat, simulaation ohjaamiseen tarkoitetut käyttöliittymäelementit sekä ActionListener, joka kiinnitetään luotuihin käyttöliittymän nappeihin. Lisäksi luodaan ja käynnistetään uusi säie, jonka tehtävänä on ruudunpäivitys. Kun säie on luotu, se toistaa itseään while-loopissa niin kauan kunnes se pysäytetään ohjelman sammuessa. Säikeen while-loopin sisällä on kaksi komentoa, ruudun päivittämisen käynnistävä repaint()-komento, sekä kutsu metodille, joka asettaa säikeen lepotilaan määrätyksi ajaksi. Tämän ajan jälkeen säie palaa aktiiviseksi ja aloittaa while-loopin alusta.

ActionListenerin tehtävänä on tunnistaa simulaation ohjaamiseen käytettyjen JButton-elementtien painallukset. Painalluksen tunnistettaessa suoritettavassa metodissa muutetaan simuloidun käyttöliittymän tilaa ohjaavia muuttujia. Nämä muuttujat kertovat grafiikanpiirtometodille, mitkä elementit sen tulee piirtää. ActionListeneriä kutsutaan aina kun siihen liitetty komponentti saa ärsykkeen.



KUVA 4. Appletin toiminta.

Kuvassa 5 on esitetty tarkemmin ohjelmassa käytetyn säikeistyksen toiminta.



KUVA 5. Appletin säikeistyksen toiminta.

ActionListener on hyvin samankaltainen eventintunnistusmetodi kuin näppäimistöä kuunteleva KeyListener ja hiirenpainalluksia kuunteleva MouseListener. ActionListenerin avulla voidaan tunnistaa siihen liitettyjen komponenttien saamia ärsykkeitä, esimerkiksi nappikomponenttiin osuva hiirenpainallus tai enter-napin painaminen aktiivisessa tekstikentässä. ActionListener ei tunnista ärsykkeitä jotka tapahtuvat muualla kuin siihen liitettyjen komponenttien alueella.

ActionListenerin käyttö ohjelmassa vaatii kolme askelta:

1. Luokan, joka joko implementoi ActionListener-rajapinnan, tai laajentaa toista luokkaa joka on implementoinut ActionListener-rajapinnan, esimerkiksi

```
public class ListenerClass implements ActionListener{
```

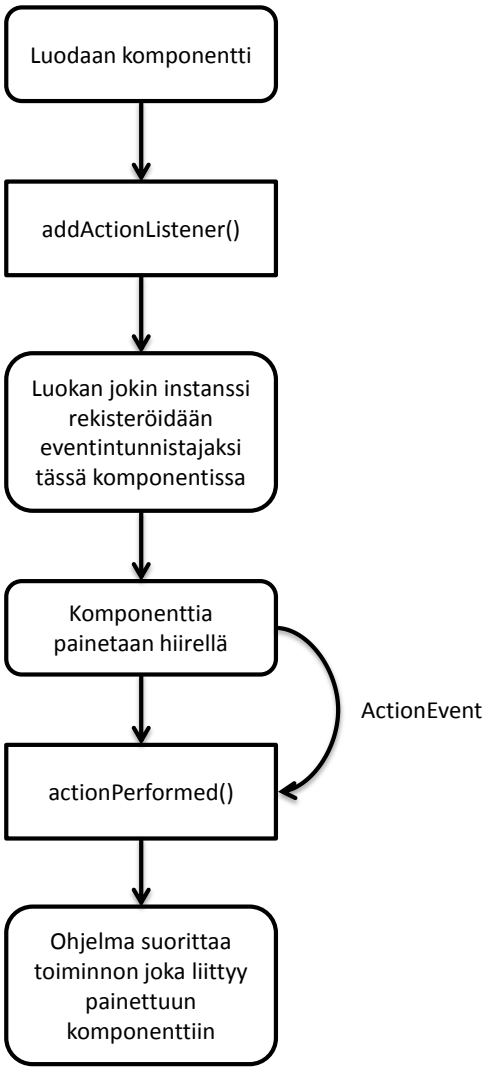
2. Kyseisen luokan jonkin instanssin rekisteröinti eventintunnistajaksi yhdessä tai useammassa komponentissa, esimerkiksi

```
listenedComponent.addActionListener(this);
```

3. actionPerformed()-metodin lisääminen koodiin, joka sisältää koodin sille mitä eventin tunnistaessa pitäisi tapahtua, esimerkiksi

```
public void actionPerformed(ActionEvent event){
```

Kuvassa 6 esitetään, kuinka actionPerformed()-metodia kutsutaan, kun siihen addActionListener()-komennolla liitetty komponentti saa käyttäjältä ärsykkeen. Tällöin metodille välitetään parametrinä ActionEvent-muuttuja, joka kertoo mikä komponentti oli kyseessä.



Kuva 6. ActionListenerin toiminta.

ActionEvent on java.lang.Object-luokasta johdettu Event-luokka, joka sisältää joukon muuttujia. Ne kertovat muun muassa ActionEventin lähteenä olleen komponentin id:n ja oliko esimerkiksi shift- tai control-nappi pohjassa kun ActionListener tunnisti painalluksen. ActionEventistä saadaan selville painettu komponentti getSource()-komennolla.

ActionListeneriä käytettiin tässä appletissa JButton-komponenttien painallusten tunnistuksessa, ja actionPerformed()-metodi on vastuussa simuloidun käyttöliittymän tilan kertovien muuttujien muuttamisesta napinpainallusten mukaan. Näistä muuttujista tärkein on view_id, joka kertoo missä tilassa laite on. Lisäksi ohjelmassa on joukko muuttujia joita käytetään muistamaan esimerkiksi valittu kieli tai aktiivisena olevan alavalikon ikoni. Näitä muuttujia käytetään paint()-metodissa määräämään mitä grafiikkaelementtejä piirretään.

4.2 Appletin rakenne

SpocoSim-appletti sisältää kaksi omaa luokkaa sekä html-tiedoston jossa appletti ajetaan.

Ensimmäinen luokka, SpoCoSim.java, sisältää appletti-tyyppisen ohjelman vaatimat metodit, säikeistyksen vaatimat metodit, käyttöliittymäelementtien luonnin ja niiden toiminnallisuuden vaatimat metodit, grafiikanpiirron sekä useita lyhyitä metodeja joita käytetään tietotyyppien muunnoksiin, kellonajan laskentaan sekä usein toistuvien graafisten elementtien piirtoon.

Toinen luokka, StopWatch.java, sisältää tietokoneen omaan kelloon perustuvan sekuntikellon, joka palauttaa kuluneen ajan millisekunteina.

Html-tiedostossa on <applet>-tagi joka käynnistää SpocoSim.class -tiedoston, eli varsinaisen käännetyn appletin. Appletin koko pikseleinä näyttölaitteella määrätään saman tagin parametrien avulla.

SpocoSim.java perii edellä mainitun java.applet.JApplet-luokan ja siihen liittyvän javax.swing-paketin, jotka mahdollistavat ketterät ja joka alustalla samalla tavalla toimivat käyttöliittymäelementit. Appletti myös käyttää java.lang.Runnable-luokkaa säikeistykseen sekä java.awt.event.ActionListener-luokkaa napinpainalluksien tunnistamiseen.

Ensiksi alustetaan tarvittavat muuttujat. Näihin kuuluvat myös käyttöliittymän napit, kuvat, säikeet, fontit, värit, ja niin edelleen normaalien int-tyyppisten numeromuuttujien lisäksi. Tärkeimpiä muuttujia ohjelman toiminnan kannalta ovat view_id sekä joukko _selection-muuttujia. Näitä käytetään muistamaan käyttöliittymän senhetkinen tila. Grafiikanpiirron kannalta view_id on erityisen tärkeä, sillä se kertoo mikä ruutu pitää milloinkin piirtää.

Muuttujien alustamisen jälkeen init()-metodissa alustetaan käyttöliittymäelementit, kuten napit sekä niiden ankkureina toimivat JPanel-elementit. Samassa metodissa myös alustetaan finDictionary-muuttuja, joka sisältää suomenkieliset käännökset käyttöliittymän tekstille.

Seuraavia neljää metodia käytetään säikeistykseen. Appletti luo säikeen, jonka tehtävänä on päivittää ruutu annetuin väliajoin. Ruutu päivitetään repaint()-komennolla, joka on appleteissa käytettävä käsky joka käynnistää paint()-metodin.

Metodi start() käynnistää säikeen. Tätä metodia kutsutaan automaattisesti ohjelman käynnistyessä kun java.lang.Runnable-luokka on käytössä.

Metodi stop() on pakollinen metodi, joka pysäyttää säikeen ja poistaa sen. Tätä metodia kutsutaan automaattisesti kun ohjelma sammutetaan.

Metodi run() sisältää while-silmukan, jota ajetaan niin kauan kun start():ssa luotu säie on olemassa. While-silmukan sisällä on edellä mainittu repaint()-komento sekä komento säikeen hetkeksi pysäyttävään thread_sleep()-metodiin.

Metodi thread_sleep() tarkastaa onko annettu säie aktiivinen, ja jos on, se asettaa säikeen lepotilaan sleep()-komennolla määrätyksi ajaksi.Lepotilassa säie ei kuluta tietokoneen laskentatehoa turhaan.

Seuraava metodi on java.awt.event.ActionListenerin actionPerformed()-metodin ylikirjoitettu versio. Metodia kutsutaan kun jokin appletin elementti saa ärsykkeen, tässä hiirenpainalluksen. Aluksi tarkastetaan onko kyseinen elementti tyyppiä JButton eli aiemmin init()-luokassa luotu nappi. Jos elementti ei ole tyyppiä JButton, metodi ei tee mitään. Jos on, tarkastetaan mikä nappi on kyseessä. Metodi asettaa sitten view_id-muuttujalle uuden arvon riippuen sen vanhasta arvosta sekä siitä mikä napinpainallus tunnistettiin. Seuraavan kerran kun run()-metodi päivittää ruudun, grafiikanpiirtometodi saa view_id:n uuden arvon, ja piirrettävä ruutu muuttuu sen mukaan.

Seuraavat neljä metodia ovat yksinkertaisia tyyppimuunnosmetodeja. Metodi convertIntToString() muuttaa numerodataa sisältävän int-muuttujan tekstidataa sisältäväksi string-muuttujaksi. Metodit convertLongToMinutes, convertLongToSeconds ja convertLongToFractions muuttavat StopWatch-luokan palauttamia millisekuntiarvoja long-tyyppisistä liukuluvuista kokonaisiksi minuuteiksi, sekunneiksi ja sekunnin sadasosiksi ja palauttavat tämän arvon int-tyyppisenä muuttujana.

Seuraavat seitsemän metodia ovat grafiikanpiirtometodeja usein toistuville elementeille, kuten tooltipeille sekä taustan piirtämiselle.

Viimeinen metodi, paint(), on appleteissa käytettävä piirtometodi. Metodi ajetaan repaint()-komennolla. Tässä appletissa metodi ensiksi piirtää kuvan erilliselle kuvamuuttujalle, ja vasta kun piirto on valmis, piirretään tämä kuvamuuttuja ruudulle. Tämä tekniikka tunnetaan niin sanottuna kaksoispuskurointina, ja se vähentää appletin grafiikan välkkymistä.

Ilman kaksoispuskurointia appletissa välähtäisi tyhjä ruutu joka kerta kun grafiikanpiirtometodi suoritetaan. Kaksoispuskuroinnin ansiosta ruudulle piirretty kuva säilyy niin kauan kunnes uusi kuva piirretään sen tilalle grafiikanpiirtometodin lopussa. Kaksoispuskurointi ei kuitenkaan eliminoi kokonaan grafiikan välkkymistä tässä appletissa, koska kuvamuuttujan piirto-operaatio grafiikkamuuttujaan saattaa kestää kauemmin kuin appletilla kestää edetä grafiikkamuuttujan piirtoon ruudulle. Tällöin välkähtää tyhjä ruutu. Tämä välkkyminen on kuitenkin paljon vähäisempää kuin ilman kaksoispuskurointia.

4.3 Käyttöliittymän rakenne

Kun SpoCo käynnistetään, se aloittaa niin sanotusta perustilasta. Perustilassa on näkyvillä kellonaika, syke sekä selitetekstit ruudun yläreunassa (kuva 7). Vasemmalla napilla edetään laitteen asetuksiin ja oikealla aloitetaan mittaustila.

KUVA 7. Perustila, jossa näkyvillä kellonaika sekä syke. Kuva simulaatioappletista.

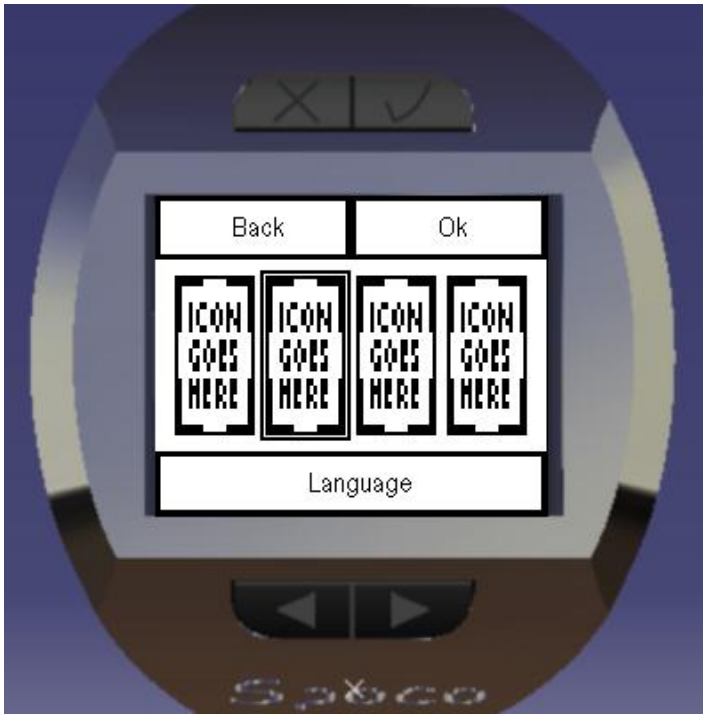
Perustila on SpoCon lepotila, kun laite on päällä mutta käyttäjä ei ole aloittanut harjoitusta.Tätä tilaa pystyttäisiin myös käyttämään normaalin sykemittarin tavoin, jos harjoituksen liikeratoja ei olisi tai ei voitaisi jostain syystä tallentaa laitteeseen. Perustilasta siirryttiin mittaustilaan selitetekstin ohjeistuksella, valittaisiin haluttu liikeohjelma ja niihin mahdollisesti liittyvät painot, tallennettaisiin liikeohjelman liikeradat laitteen muistiin jos niitä ei ole tallennettu aiemmin, ja sitten siirryttäisiin varsinaiseen mittaustilaan.

Mittaustila on tarkoitettu varsinaiseen treeniin. Mittaustila pyrittiin suunnittelemaan mahdollisimman helppolukuiseksi harjoittelun aikana (kuva 8). Syke ja palaute liikkeen oikeellisuudesta ovat käyttäjälle tärkeimmät tiedot, joten niille varattiin suurin osa näytön pinta-alasta. Toistojen määrä sekä maksimitoistojen määrä ilmoitetaan myös tässä näkymässä. Myös tästä näkymästä on mahdollista edetä laitteen asetusvalikkoon selitetekstin kertomalla painikkeella. Tämä keskeyttää väliaikaisesti mittauksen, jota voidaan jatkaa jälleen valikosta poistumalla.

KUVA 8. SpoCon mittaustila. Kuva simulaatioappletista.

Asetusvalikko on SpoCon kolmas päätila (kuva 9) perustilan ja mittaustilan lisäksi. Valikosta löytyy alavalikot kellonajan säädölle, kielen valitsemiselle, sekuntikellolle sekä laitteen sammuttamiselle. Valikkoon on mahdollista lisätä myös muita toimintoja tarvittaessa, nämä neljä valittiin valikkorakenteen toiminnallisuuden esittämiseen.

Se, että nuolipainikkeita oli ainoastaan kaksi, tuli ottaa huomioon näytöllä valikkorakenteissa. Valikkorakenteissa päädyttiin siksi horisontaaliseen liikkumiseen. Näytön leveys myös soveltui paremmin horisontaaliseen liikkumiseen, mutta näytön korkeus kuitenkin riittäisi myös ylä- ja alareunoissa oleville selitteille tarpeen mukaan. Eri toiminnot esitettiin ikonien avulla ja seliteteksti kertoi tarkemmin kustakin toiminnosta, kun sen ikoni oli nuolipainikkeilla valittuna.



KUVA 9. Käyttöliittymän logiikkaa, valikkorakenne. Kuva simulaatioappletista.

4.4 Testaus

Ohjelmaa testattiin kehityksen aikana jokaisen lisätyn ominaisuuden jälkeen, sekä ohjelman valmistuttua. Testaamalla ohjelman kehityksen aikana saatiin poistettua suurin osa ongelmista. Lopputestauksen aikana tuli ilmi joitain navigointiin liittyviä virheitä, esimerkiksi alavalikosta poistumisen yhteydessä ohjelma piirsi tietyissä olosuhteissa valitun ikonin ilmoittavan suorakulmion väärän ikonin ympärille. Lisäksi sekuntikellon mittaama aika ei vastannut todellista aikaa kun mitattiin yli minuutin. Molemmat ongelmat saatiin korjattua.

Ohjelmaan jäi yksi suuri ongelma. Ohjelman grafiikanpiirtometodi piirtää grafiikan ensin kuvamuuttujaan ja kaiken piirron valmistuttua kuvamuuttuja piirretään ruudulle. Tätä tekniikkaa kutsutaan kaksoispuskuroinniksi, ja sen tarkoituksena on vähentää ruudun välkkymistä tai repeämistä (repeämiseksi kutsutaan ilmiötä jossa ruudulle piirretään osittain vajaa uusi ruutu edellisen päälle, jolloin alaosa kuvasta on vanha ruutu, ja yläosa uusi). Kaksoispuskurointi on tarpeellinen tässä ohjelmassa, koska uuden piirto-operaatiokierroksen alkaessa java tyhjentää edellisen kierroksen tuottaman grafiikan ensimmäisen uuden ruudulle piirtävän piirtometodin yhteydessä. Tämä näkyy

käyttäjälle ärsyttävänä välkkymisenä aina kun piirtometodi käynnistyy. Kaksoispuskuroinnilla edellisen kierroksen piirtämä grafiikka säilyy niin kauan kunnes uusi kuvamuuttuja piirretään sen tilalle. Optimitilanteessa tämä eliminoisi välkkymisen kokonaan, mutta käytännössä kuvamuuttujaan piirto ei aina ehdi valmistua ennen kuin kuvamuuttujan lopullinen piirto ruudulle alkaa. Tällöin välkähtää tyhjä ruutu. Tämä välkkyminen ei kuitenkaan ole läheskään yhtä tiheää kuin ilman kaksoispuskurointia.

Välkkymisongelma voidaan korjata kokonaan poistamalla grafiikanpiirtometodin paint() alusta käsky `super.paint(graphics)`, mutta poistamalla tämä komento appletti alkaa piirtää grafiikkaa käyttöliittymäelementtien päälle. Tällöin käyttöliittymän ohjaamiseen käytetyt painikkeet eivät näy käyttäjälle ennen kuin niiden tila päivittyy.

5 YHTEENVETO

Opinnäytetyön tarkoituksena oli suunnitella käyttöliittymä kuntosalikäyttöön tarkoitetulle rannelaitteelle sekä ohjelmoida käyttöliittymästä pc:llä ajettava simulaatio. Tuloksena syntyi suunnitelma SpoCo-rannekkeen käyttöliittymälle sekä verkkoselaimessa suoritettava simulaatio-ohjelma. Simuloitu ohjelma vastaa käyttöliittymäsuunnittelun teoriassa esitettyä prototyyppivaihetta, jolle voidaan suorittaa käytettävyystestausta. Tämän vuoksi käyttöliittymän ulkonäkö on karu. Varsinaisen ulkonäön suunnittelee graafikko vasta käytettävyystestauksen jälkeen, eikä se kuulu tähän opinnäytetyöhön.

Simulaatio-ohjelman lähdekoodi kirjoitettiin mahdollisimman helppolukuiseksi runsaan kommentoinnin kera. Kommenteissa selitettiin ohjelman metodien toiminta sekä mihin mikäkin muuttuja liittyy. Ohjelmaan pystyy pienin muutoksin tuomaan oikeaa dataa, jolloin sitä voidaan käyttää demoamisvälineenä.

Suunnitelma käyttöliittymästä on esitetty wireframe-dokumentissa.

6 LÄHTEET

1. Holappa, Terhi 2010. T271003 Käyttöliittymäsuunnittelun perusteet 3 op. Opintojakson oppimateriaali syksyllä 2010. Oulu: Oulun seudun ammattikorkeakoulu, tekniikan yksikkö.

2. Holappa, Terhi 2011. T213003 Käytettävyyden arviointi 3 op. Opintojakson oppimateriaali keväällä 2011. Oulu: Oulun seudun ammattikorkeakoulu, tekniikan yksikkö.

3. Väisänen, Veijo Toivo 2010. T241003 Ohjelmistotuotanto 3 op. Opintojakson oppimateriaali syksyllä 2010. Oulu: Oulun seudun ammattikorkeakoulu, tekniikan yksikkö.

4. Faulkner, Xristine 2000. Usability Engineering. Gosport: Ashford Colour Press.

5. Rosson, Mary Beth — Carroll, John M. 2002. Usability Engineering. San Diego: Academic Press.

6. Barnum, Carol M. 2011. Usability Testing Essentials. Burlington: Elsevier Inc.

7. The Eclipse Foundation 2012. http://www.eclipse.org/. Hakupäivä 16.10.2012.

8. Brewster, Rick, dotPDN LLC 2012. http://www.getpaint.net/. Hakupäivä 16.10.2012.

9. Microsoft 2010. http://office.microsoft.com/fi-fi/products/?CTT=97. Hakupäivä 18.10.2012.

10. Oracle 2012. http://www.oracle.com/technetwork/java/javase/downloads/index.html. Hakupäivä 16.10.2012.

SpoCo-rannekkeen käyttöliittymäsuunnittelu

Wireframe

Napit

- Ylhäällä menu/cancel- ja confirm-napit
  - Perustilassa cancel/menu avaa menunäkymän, menunäkymässä peruu tehdyt muutokset ja heittää edelliselle ruudulle.
  - Confirm-nappi toimii myös on/off pitkään painettuna?
- Alhaalla nuolinapit vasen/oikea
- Näytöllä tarpeeksi tilaa ohjelaatikoille jotka kertovat tarkemmin nappien toiminnoista tarvittaessa

Perusnäkymä

- ”Perusnäkymässä” näytetään käyttäjälle syke sekä palautetta viimeisimmän liikkeen liikeradasta
  - Syke numerona, vieressä sykkivä sydänikoni
  - Palaute liikeradasta palkkeina tai jollain muulla nopeasti luettavalla tavalla (ehkä värit punainen – oranssi – keltainen – viherkeltainen – vihreä?)
- Alhaalla pienet harmaat pallot jotka kertovat käyttäjälle että nuolinappeilla voidaan siirtyä eri näkymiin
  - Väritetty pallo esittää nykyistä näkymää
  - Eivät välttämättä tarpeellisia jos kaikki info mahdutetaan yhteen näkymään, mutta perusnäkymä olisi hyvä pitää mahdollisimman helppolukuisena
  - Muilla näkymillä yksityiskohtaisempaa tietoa tms?
- Ylhäällä infolaatikko, joka kertoo mitä napit tekevät tässä näkymässä, sekä liikkeen tehdyt toistot/toistojen määrä

Menu

- Kategoriat vierekkäin, liikutaan nuolilla
  - Aktiivinen valinta ympäröidään mustalla ääriviivalla/ikoni värit vaihtuvat tms
- Alhaalla laatikko johon ilmestyy lisätietoa mitä ikonin takaa löytyy
- Ikonit ja tekstit tässä lähinnä esimerkkejä

Liikeratojen tallentaminen, ideoita

- tehtävät liikkeet valitaan erikseen pc:llä olevalla softalla
  - Sitten spoco-rannekkeen menusta liikeratojen tallennukseen
  - Menu sanoo että kiinnitä pc:hen, valittu liikeohjelma siirtyy spocoon, spoco irti pc:stä
  - Tehdään spocon käskemät liikkeet ohjaajan avustuksella, spoco tallentaa liikeradat muistiin kun ohjaaja on tyytyväinen suoritukseen
  - Kun kaikki liikkeet on tehty, spoco haluaa taas kiinni pc:hen, siirtää liikeradat talteen sinne ettei tarvitse tallennella liikeratoja joka kerta uudelleen?