Käyttäjäkokemus avoimen lähdekoodin ohjelmistossa

Aleksi Suomalainen

Pro gradu -tutkielma

ITÄ-SUOMEN YLIOPISTO

Tietojenkäsittelytieteen laitos

Tietojenkäsittelytiede

Maaliskuu 2015

ITÄ-SUOMEN YLIOPISTO, Luonnontieteiden ja metsätieteiden tiedekunta, Kuopio
Tietojenkäsittelytieteen laitos
Tietojenkäsittelytiede

Opiskelija, Aleksi Suomalainen: Käyttäjäkokemus avoimen lähdekoodin ohjelmistossa
Pro gradu -tutkielma, 74 s.
Pro gradu -tutkielman ohjaajat: FM Jaakko Parviainen, FT, DI Keijo Haataja
Maaliskuu 2015

Tiivistelmä:

Käyttäjäkokemus on muodostumassa yhä enemmissä määrin mielenkiintoiseksi tutkimusaiheeksi. Jo nyt käyttäjäkokemuksen tutkimuksessa on edistytty huomattavasti. Käyttäjäkokemus on kuitenkin havaittu erittäin monitahoiseksi tutkimusaiheeksi. Käyttöliittymien kannalta käyttäjäkokemuksen kehittäminen on ensiarvoisen tärkeää ja tieto aiheesta saattaa vaikuttaa merkitsevästi sen onnistumiseen. Käyttäjäkokemuksen suunnittelu tulee olemaan tulevaisuudessa enimmissä määrin merkittävää. Suunnittelun pääkohtia ovat mm. teorioiden tukeminen käytännöllä ja erilaisten ohjenuorien ja hyväksi havaittujen heuristiikkojen soveltaminen.

Avoin lähdekoodi on myös tehnyt jalansijaa itselleen sekä teollisuudessa, että tutkimuksessa. Sen antama merkitys monille projekteille on ollut erittäin merkittävää. Avoimen lähdekoodin lisensointi on erittäin monipuolista ja se mahdollistaa monien projektien kehittämisen avoimessa ilmapiirissä ja ohjelmistojen ja kehittäjien vapautta vaalien. Lisensointitapoja on monia ja niiden soveltaminen käytännössä ei ole aina yksinkertaista.

Nemo Mobile on yksi projekti, joka käyttää avointa lähdekoodia ja sen tuottamia komponentteja kehityskaaressaan. Glacier on Nemo Mobilen käyttöliitymän uudelleensuunnitteluprojekti, jossa on korostettu avoimuutta suunnittelussa ja toteutuksessa alusta loppuun. Vaikka suunnittelusta päättää komitea, kaikki suunnittelussa mukana olleiden henkilöiden mielipiteet otetaan huomioon lopullisessa tuotoksessa.

Avainsanat: Avoin Lähdekoodi, Käyttäjäkokemus, Projekti, Vapaat Ohjelmistot, Käyttäjäkokemuksen Suunnittelu, Käytettävyys

ACM-luokat (ACM Computing Classification System, 1998 version): A.m, H. 5.2, K. 5.1, K. 6.3

UNIVERSITY OF EASTERN FINLAND, Faculty of Science and Forestry,

Kuopio
School of Computing
Computer Science

Opiskelija, Aleksi Suomalainen: User experience in an open source software project
Master's Thesis, 74 p.
Supervisors of the Master's Thesis: M.Sc. Jaakko Parviainen, Ph.D., M.Sc. (Tech.) Keijo Haataja
March 2015

Abstract:

User experience has been formed to be increasingly interesting research topic. Even now user experience studies have made important strides. User experience has, however, been noted to be quite multifaceted topic for scholars. In terms of user interfaces, the development of user experience is of utmost importance and knowledge about it may affect its success. User experience design will thus be more significant in the future. Main headlines of design are, among others, the application of theory to practise, usage of different guidelines and the use of appropriate heuristics.

Open source has proven itself in the industry and academia alike. Its significance has been outstanding to many projects. The licensing of open source is very varied and enables many projects to be developed in open athmosphere and guarding the liberty of both software and developers. There are many ways of licensing and their application is not always simple.

Nemo Mobile is one such project, which utilizes open source and its components in its development. Glacier is Nemo Mobiles user interface redesign project, which has always epitomized openness in design and implementation, from the ground up. Even though the design is referred as "design by committee", everyone involved in it and their opinions are always noted in the final design.

Keywords: Open Source, User Experience, Project, Free Software, User Experience Design, Usability

CR Categories (ACM Computing Classification System, 1998 version): A.m, H. 5.2, K. 5.1, K. 6.3

Esipuhe

Haluan kiittää äitiäni saamastani tuesta gradun teon aikana. Lisäksi haluan kiittää Jolla Ltd:tä saamastani tuesta ja kehittäjän alennuksesta Glacier-kehitykseen saadusta Jolla-laitteesta. Viimeisimpänä mutta ei vähäisimpänä, haluan kiittää ohjaajiani Jaakko Parviaista ja Keijo Haatajaa kaikista neuvoista ja tuesta.

Aleksi Suomalainen

Lyhenneluettelo

ACM	Association for Computing Machinery; maailmanlaajuinen tietotekniikka-alan tieteellinen yhdistys
ARPANET	The Advanced Research Projects Agency Network
BSD	Berkeley Software Distribution
DOS	Disk Operating Systen
FSF	Free Software Foundation
EGL	Embedded-System Graphics Library
ES	Embedded Systems
GNU	GNU's Not Unix
GPL	General Public License
IBM	International Business Machines Corporation
MIT	Massachusetts Institute Of Technology
OBS	Open Build Service
OpenGL	(Open Graphics Library)
PACT	the Project for the Advancement of Coding Techniques
QML	Qt Meta Language
UEF	Itä-Suomen yliopisto
UUCP	Unix-to-Unix Copy
UX	User Experience

Sisältö

1 Johdanto

Tutkielmassani on selvitys käyttäjäkokemuksen tutkimuksen tilasta ja käyttäjäkokemuksen suunnittelun eräistä näkökulmista. Käyttäjäkokemuksen suunnittelu on vielä lapsenkengissään ja yhtenäistä, helposti lähestyttävää menetelmää tai luonnehdintaa siitä ei ole vielä pystytty saavuttamaan. Käyttäjäkokemus liittyy olennaisesti tässä tutkielmassa kuvailemaani Glacier-käyttöliittymään, joka on avoimen lähdekoodin ohjelmisto.

Tässä tutkielmassa on tarkoitus selvittää, mitä avoin lähdekoodi tarkoittaa ja mistä se on syntyisin, mitä avoimen lähdekoodin yhteistoiminta tarkoittaa erään projektin näkökulmasta ja mitä erilaisia lisenssejä on tarjolla avoimelle lähdekoodille. Tutkielmassa selvitetään lyhyesti myös vapaiden ohjelmistojen roolin avoimen lähdekoodin synnyssä ja nykyisessä tilassa.

Tutkielman rakenne on seuraava: luvussa 2 tutustutaan avoimen lähdekoodin historiaa, lisensointiin, Git-versionhallintaan ja erääseen avoimen lähdekoodin projektiin nimeltä Nemo Mobile. Luku 3 keskittyy selvittämään kirjallisuuden avulla, mitä käyttäjäkokemus on ja miten käyttäjäkokemusta varten voidaan suunnitella asioita. Luvussa 4 esitellään avoimeen lähdekoodiin pohjautuvan Glacier-käyttöliittymän komponentit ja kotinäkymä, joka on tutkielman käytännön osuus.

2 Avoimen lähdekoodin ohjelmistot

Avoimen lähdekoodin ohjelmistot ovat jatkuvasti suositumpia. Niiden merkitys ohjelmistoteollisuudessa on kasvanut vuosi vuodelta enemmän ja niiden käyttöaste kasvaa jatkuvasti. Avoimen lähdekoodin merkitystä ei voi näin ollen aliarvioida millään ohjelmistoalan mittarilla. Samaan tahtiin myös mobiilien käyttäjäkokemusten monimuotoisuus on kasvanut. Avoin lähdekoodi voi antaa työkaluja käyttäjäkokemusten parantamiseen ja luomisen vapauteen. Tässä tutkielmassa on tarkoitus luodata avointa lähdekoodia syvemmin sen historian ja lisenssien tarkastelun avulla.

Luvussa 2.1 kerrataan avoimen lähdekoodin syntyä. Avoimen lähdekoodin lisensseistä kerrotaan luvussa 2.2. Git-versionhallinta on erittäin tärkeää mainita modernissa avoimessa lähdekoodissa, joten siitä on erikseen kuvaus luvussa 2.3. Lopuksi luvussa 2.4 on kuvaus tutkielmassa tutkitun avoimen lähdekoodin projektista nimeltä Nemo Mobile.

2.1 Avoimen lähdekoodin historiaa

Avoimen lähdekoodin historian voidaan sanoa alkaneen 1950-luvulla kun IBM (International Business Machines) toi markkinoille sen ensimmäisen tietokonemallin, IBM 701. Tämän tietokonemallin käyttäjät perustivat yhteisen vapaaehtoisvoimin toimivan järjestön nimeltä SHARE (Varian 1997). Tämän järjestön tavoitteena oli jakaa erilaista teknistä informaatiota tietokoneista, kuten ohjelmointikieliä, käyttöjärjestelmiä (joskin niiden alkeellisia versioita) ja tietokantoja.

IBM 701 loi myös toisen järjestön jakamaan teknistä informaatiota, tämän järjestön nimi oli PACT (the Project for the Advancement of Coding Techniques). Järjestö ei ollut SHARE:n tavoin vapaaehtoisvoimin rakennettu vaan se oli yritysten välinen järjestö, johon kuuluivat mm. Lockheed, Douglas ja RAND. Näiden yritysten lisäksi myös muita puolustuksen alihankintayrityksiä kuului tähän järjestöön.

Nämä molemmat järjestöt toimivat siis avoimesti yhteisten teknisten ongelmien ratkaisemiseksi, tekemällä yhteistyötä toisten ihmisten kanssa. Nämä samat asiat olivat avainasemassa myös myöhäisemmällä ajanjaksolla, kun UNIX-järjestelmää alettiin kehittää (Weber 2004).

UNIX ja BSD

UNIX (alkuperäinen nimi UNICS) on lähtöisin AT&T Bell Laboratories nimisen yrityksen kahden kehittäjän Dennis Ritchie ja Ken Thompson luomistyöstä. UNIX:in alkuperä lähtee vuodesta 1969, jolloin Thompson kirjoitti kesälomallaan alkuperäisen UNICS-järjestelmän lähdekoodin. Tätä aikasempi alkuperä itse järjestelmälle lähti alkuun kun Multics-järjestelmä (Multiplexed Information and Computing Service) epäonnistui kehityksessä Bell Laboratoriesilla. UNIX:in filosofiassa ohjelmiston pienet osat ovat tärkeässä roolissa, eivät isot ja monimutkaiset. (Weber 2004)

UNIX saapui tienhaaraan vuonna 1973, jolloin Ritchie ja Thompson kirjoittivat tutkimusartikkelin UNIX-järjestelmästä. Tämän artikkelin johdosta UNIX:in suosio suorastaan räjähti Yhdysvalloissa. (Weber 2004)

Järjestelmää jaettiin lähdekoodimuodossa ja sen kielenä oli C-kieli, jonka Ritchie ja Thompson olivat kehittäneet juuri UNIX:ia varten. Lähdekoodin lisenssi maksoi vain muutamia satoja dollareita ja oli siten helppo lisensoida esimerkiksi yliopistoja varten. Täten UNIX levisi nopeasti vuoteen 1975 mennessä ympäri maailmaa eri oppilaitoksiin. Lähdekoodi kulki fyysisenä mediana ja ARPANET:in (The Advanced Research Projects Agency Network) yleistymisen myötä esimerkiksi UUCP-ohjelman (Unix-to-Unix Copy) avulla (Weber 2004).

Erityisessä asemassa UNIX:in suosiossa oli Berkeleyn yliopisto Yhdysvalloissa. Tämän yliopiston professori Robert Fabry asennutti järjestelmän tietojenkäsittelytieteen laitoksen koneelle, jossa se kävi läpi muutoksia. Nämä muutokset liittyivät käsitteen BSD (Berkeley Software Distribution) alkuun, jolloin Berkeleyssä tehdyt ohjelmistot jaettiin lähdekoodeineen muille oppilaitoksille ilmaiseksi. Alussa BSD-jakelussa oli pelkästään ohjelmistoja, kuten esimerkiksi Pascal-kääntäjä ja Ex-tekstieditori. BSD-jakelun yleistyessä ja kasvattaessa suosiotaan, siihen lisättiin UNIX-järjestelmän kerneli eli ydin. Nykyisin BSD tunnetaan lähinnä eri jakeluistaan, kuten FreeBSD, NetBSD tai OpenBSD. (Weber 2004)

FSF ja GNU

Kun UNIX ei riittänyt Richard Stallmanille, hän aloitti itse rakentamaan GNU-järjestelmään (GNU's Not UNIX). Hän halusi saada GNU:sta täysin vapaan ohjelmis-

ton filosofiaa noudattavan järjestelmän. (Stallman 2009)

Hän aloitti työnsä MIT:n tekoälyn laboratoriossa, jossa oli jo alusta alkaen hyvin vapaa ja avoin ilmapiiri. Tämä inspiroi Stallmania ja hän on käyttänyt näitä aikojaan MIT:ssä vertauksena FSF:n (Free Software Foundation) toiminnalle. Ohjelmistoja, sekä niiden lähdekoodia, jaettiin vapaasti sekä yrityksille että tekoälylaboratoriossa. (Stallman 2009)

Tämä ilmapiiri ei kuitenkaan säilynyt ikuisesti. 1980-luvulla yhteisö alkoi hajota koska siellä työskennelleet ihmiset palkattiin muualle ja ohjelmistot, niiden käyttö ja lähdekoodi siirtyivät salassapidettäviksi ja suljetuiksi. Tämä ajoi Stallmanin kehittämään vaihtoehtoista järjestelmää nimeltä GNU ja myös vapauttamaan ohjelmistojen kehitystä ja käyttöä keksimällä *vapaan ohjelmiston* käsitteen.(Stallman 2009)

Vapailla ohjelmistoilla tarkoitetaan sellaisia ohjelmistoja, joiden sisältämiä ohjelmia on vapaus ajaa ilman rajoituksia, vapaus jakaa kopioita ohjelmistosta sekä vapaus muokata ja jakaa ohjelmistoa vapaasti. Vapaus tässä kontekstissa tosiaan tarkoittaa vapaata, ei välttämättä ilmaista, kuten englanninkielisessä teksteissä usein korjataan. (Stallman 2009)

GNU:n esittelemä lisenssi, nimeltään GPL aloitti tänä päivänäkin kutsutun *copyleft-lisenssien* kehityskulun. Normaalit julkiset, public domain, lisenssit ovat ongelmallisia koska ne sallivat lähdekoodiltaan suljettujen versioiden jakamisen. Näistä copyleft-lisensseistä kerrotaan enemmän luvussa 2.2.(Stallman 2009)

GNU:n kehitystyö aloitettiin vuonna 1984 ja siitä seurasi myöhemmin, vuonna 1985, FSF:n julkinen toiminta. FSF:n toiminta perustuu mm. GNU:n ohjelmistojen jakamiseen fyysisillä medioilla sekä tukitoiminnalla. (Stallman 2009)

FSF:n tulevaisuuden haasteita ovat mm. ei vapaat kirjastot, suljettu laitteisto, ohjelmistopatentit ja vapaan dokumentoinnin haasteet, mm. sen puutteet. Ei-vapaat kirjastot kuten ennen esimerkiksi QT (ennen syyskuuta 2000), asettavat haasteita siksi, että kehittäjät menevät helposti niiden tarjoamien ominaisuuksien mukaan, eivät kirjaston vapauden vuoksi. Ohjelmistopatentit ovat ehkä suurin uhka koko teollisuudelle ylipäätään, mutta Stallman mainitsee esimerkiksi erilaiset tiedon tiivistämisessä tarkoitettujen algoritmien patentit. (Stallman 2009)

Linux

Alkuvuodesta 1991, Linus Torvalds ei myöskään ollut tyytyväinen saamaansa ohjelmistoon 386-tietokoneellaan. Kyseinen ohjelmisto oli nimeltään DOS (Disk Operating System) ja tämä ei tyydyttänyt Torvaldsia käytöltään. Hän halusi sen sijaan kehittää jotain UNIX-järjestelmästä soveltuvaa tietokoneelleen ja hän valitsi tätä tehtävää varten Minix-järjestelmän, jota kehitti Andrew Tanenbaum Hollannissa. Torvalds ei kuitenkaan ollut tyytyväinen tähänkään järjestelmään ja hän alkoi kehittämään omaa järjestelmäänsä nimeltä Linux. (Miettinen 2006)

Hän julkaisi aikeistaan Internetin minix-sähköpostilistalle ja sai heti osakseen paljon huomiota eri puolilta maailmaa. Hän julkaisi sittemmin Linuxin lähdekoodin Helsingin yliopiston palvelimella ja ilmoitti kiinnostuineille ihmisille tästä asiasta sähköpostitse. Tämä muodosti heti tärkeimmän kanavan Linuxin kehitykselle, nimittäin sähköpostilistan, joka on toiminnassa yhä tänäkin päivänä. (Miettinen 2006)

Linuxin kehitys perustuu ennen kaikkea Linus Torvaldsin ylimmäiseen päätäntävaltaan, mutta hänet ympäröi ns. kehä ohjelmoijia jotka hyväksyvät muutoksia Linuxytimen koodiin. Versionhallintajärjestelmät olivat myös tärkeässä roolissa Linux-kehityksessä ja se on muuttunut vuosien myötä monta kertaa järjestelmästä toiseen. Nykyinen lähdekoodin versionhallintajärjestelmä on Git, jonka toiminnoista ja teoriasta kerrotaan enemmän luvussa 2.3.

Linuxista tuli eräs kaikkien aikojen menestyneimmistä avoimen lähdekoodin projekteista ja se on sitä yhä.

2000-luvun avoin lähdekoodi

2000-luvulla ohjelmistoyhtiöt alkoivat löytää avoimen lähdekoodin mahdollisuudet. Haluttiin pois nimenomaan vapaiden ohjelmistojen terminologiasta ja alettiin kutsua ohjelmistoja avoimen lähdekoodin ohjelmistoiksi. Tähän vaikutti osaltaan Netscape-selaimen lähdekoodin julkistus vuonna 1998. Uutta suuntaa haluttiin kutsua nimellä "Open Source"(avoin lähdekoodi). Tällä tehtiin suoraan pesäeroa vapaista ohjelmistoista, jonka ideologia oli liiketoiminnan mielestä liian rajoittunut (Raymond 1998).

Liike-elämä otti avoimen lähdekoodin omakseen ja monet yhtiöit alkoivat noudattaa

tätä toimintatapaa julkistamalla tuotteistaan lähdekoodia ja luomalla yhteisön tuotteidensa kehityksen ympärille. Samalla monet yhtiöt hyödyntivät omaa henkilökuntaansa ohjelmistojen avoimessa kehityksessä. Tästä hyvänä esimerkkinä on Red Hat, joka tuotteisti oman Linux-jakelunsa ja myy sitä ja siihen liittyviä palveluita. Red Hat oli myös 2000-luvun ensimmäinen avoimen lähdekoodin yritys joka ylitti miljardin dollarin liikevaihdon.

Lernerin ja Tirolen (2002) mukaan avoimen lähdekoodin ohjelmistojen luomisen motivaationa ovat usein tietotekniikan ongelmat. Tämän huomasivat mm. Perl- ja Sendmail-ohjelmistojen kehittäjät.

Android

Nemo Mobilen tulevaisuus on Android-laitteiden laitteistojen sovitusten kehittämisessä. Tämä sovitus on tehnyt mahdolliseksi esimerkiksi Glacierin kotinäkymän käytön Android-laitteissa, sellaisen ohjelmistokirjaston kuin *libhybris* (luku 2.4) avulla.

Android syntyi vuonna 2003 kun yritys nimeltä Android Inc. aloitti Android-käyttöjärjestelmän kehittämisen. Yritys jatkoi Android-käyttöjärjestelmän kehitystä 22 kuukautta, mutta se ajautui rahoitusvaikeuksiin. Käyttöjärjestelmän tarkoitus oli aluksi tarjota alusta digikameroille, mutta se suunnitelma vaihtui pian mobiililaitteiden alustan luomiseen. (Markoff 2007)

Kaikki muuttui kun Google osti Android Inc:n vuonna 2005. Google osti yrityksen sen tarjoaman tuotteen potentiaalin perusteella. Andy Rubin kertoi yhtiölle, että Android käyttäisi informaatiota käyttäjän sijainnista ja mieltymyksistä palveluiden tuottamisessa. Android oli erittäin salaisesti kehitetty ennen kuin Google osti sen itselleen. (Elgin 2005)

Android näki päivänvalon vuonna 2007 marraskuussa ensimmäisellä SDK-julkaisulla Androidin versiosta 0.5. Tämä versio oli nimeltään "m3-rc20a", tarkoittaen kolmatta virstanpylvästä kehityksessä. Google jatkoi Androidin kehittämistä, julkaisten vielä kaksi virstanpylväs-versiota, kunnes se ilmoitti yhteistyöstä amerikkalaisen T-Mobilen kanssa ja uuden G1-laitteen lanseerauksesta. Tämä ilmoitus aloitti myös Androidin version 1.0 julkaisun. (Amadeo 2014)

Versio 1.0 oli tärkeä virstanpylväs Androidin kehityksessä. Siinä lanseerattiin myös

Android Market-sovelluskauppa, joka oli erittäin merkittävä Androidin tulevaisuuden kannalta. Android perustuu Linux-ytimeen ja sitä kehitetään Googlen toimesta avoimesti, mutta useat laitevalmistajat ovat kehittäneet siitä suljetun version. Android on myös muutenkin avoimempi kuin esimerkiksi Applen iOS, koska käyttäjä voi asentaa oman versionsa esimerkiksi sähköpostiohjelmistosta. (Amadeo 2014)

2.2 Avoimen lähdekoodin lisensointi

Avoimen lähdekoodin lisensointi on hyvin monimuotoista ja myös monipuolista. Erilaisia lisenssejä on olemassa todella paljon ja ei ole välttämättä helppoa valita omalle projektilleen sopivaa lisenssiä. Tässä tutkielmassa kuvataan suosituimmat viisi lisenssiä, jossa molemmat versiot GPL-lisenssistä, jotka on kerätty monista eri sivustoista ja projekteista (Black Duck Software 2014). Luvussa 2.2.1 kuvataan suosituinta lisenssiä, nimeltään GPL. BSD-lisenssi on myös yksi suosituimmista lisensseistä, se kuvataan luvussa 2.2.2. Salliva avoimen lähdekoodin lisenssi nimeltä Apache kuvataan luvussa 2.2.3. MIT-lisenssi kuvataan luvussa 2.2.4.

Avoimen lähdekoodin määritelmässä (OSI 2013) kerrotaan seuraavat asiat avoimen lähdekoodin ohjelmistojen kriteereistä:

- Vapaa levitettävyys: Lisenssi ei saa estää mitään tahoa myymästä tai antamasta ohjelmistoa osana suurempaa jakelua, jossa ohjelmistoja esiintyy eri lähteissä. Lisenssi ei edellytä rojalteja tai muita maksuja myyntiä varten.
- Lähdekoodi: Ohjelman mukana tulee olla sen lähdekoodi ja sen levitys tulee sallia sekä lähdekoodina, että käännetyssä muodossa. Jos jollain ohjelmiston levitysmuodolla lähdekoodia ei tule mukana, on sen käsiin saamisen oltava mahdollisimman yksinkertaista, esimerkiksi käyttämällä Internetiä ja lataamalla lähdekoodi ilmaiseksi. Tahallaan sekoitetun lähdekoodin käyttäminen on kiellettyä. Ohjelmiston käännöksen välivaiheina saadut artefaktit kuten esikääntäjän tiedostot eivät kelpaa lähdekoodiksi.
- Muutokset: Lisenssin on sallittava muutokset ja uudet teokset lähdekoodista ja sallittava niiden jakaminen saman lisenssin vaikutuksella kuten myös alkuperäisen ohjelmiston lisenssillä.

- Tekijän lähdekoodin integriteetti: Lisenssi voi rajoittaa lähdekoodin jakamista muokatussa muodossa vain jos lisenssi sallii patch-tiedostojen käytön ohjelmiston lähdekoodin muokkauksessa. Lisenssin on erikseen sallittava ohjelmiston levityksen muokatun lähdekoodin perusteella. Lisenssi voi vaatia muokattuja versioita toimittamaan ohjelmistoa eri nimellä tai versionumerolla kuin alkuperäistä ohjelmistoa.

- Ei syrjintää yksilöitä tai ryhmiä kohtaan: Lisenssi ei saa syrjiä ketään tiettyä henkilöä tai henkilöryhmää.

- Ei syrjintää eri toimialoja kohtaan: Lisenssi ei saa rajoittaa ketään käyttämästä ohjelmaa tietyllä toimialalla. Lisenssi ei voi esimerkiksi kieltää käyttöä liiketoimminnassa tai geenitutkimuksessa.

- Lisenssin jakelu: Ohjelman oikeuksien tulee koskea kaikkia, joille ohjelmisto on levitetty, ilman, että heidän tarvitsee erikseen hankkia uutta lisenssiä.

- Lisenssi ei saa olla tuotekohtainen: Ohjelmaan liitetyt oikeudet eivät saa riippua ohjelmiston sisällytyksestä tiettyyn jakeluun. Jos ohjelma otetaan jakelusta pois, niin sen oikeudet ovat samat kuin jakelun mukana ollessaan.

- Lisenssi ei saa rajoittaa muita ohjelmistoja: Lisenssi ei saa asettaa rajoituksia toisille ohjelmistoille, jotka levitetään lisensoidun ohjelmiston mukana. Esimerkiksi lisenssi ei saa vaatia, että muut samalla medialla levitetyt ohjelmistot ovat avointa lähdekoodia.

- Lisenssin on oltava teknologisesti neutraali: Yksikään lisenssin hankinta ei saa edellyttää tiettyä teknologiaa tai liittymää.

Verrattuna FSF:n lisenssien listaan, OSI:n määrittelemät kriteerit ovat paljon sallivampia. On huomattava, että ohjelmistojen vapaus taataan ainoastaan FSF:n GPL-yhteensopivien lisenssien avulla. OSI:n kriteerien mukaiset lisenssit eivät näitä vapauksia välttämättä tarjoa. Kun lisenssi on GPL-yhteensopiva, tarkoitetaan sillä sellaisia ohjelmistoja, joiden lähdekoodi on lisensoitu GPL-lisenssillä, mutta sisältää myös muuta lähdekoodia jotka on lisensoitu muilla lisensseillä. (GNU 2009c)

2.2.1 GPL

GPL syntyi Richard Stallmanin kehittämänä 1980-luvulla kun hän halusi saada aikaan erilaisen lisenssin, josta ei tarvinnut maksaa mutta silti siihen tuli sisältyä lisenssille ominaisia piirteitä. Hän kutsuu GPL:ää nimellä copyleft-lisenssi. Sen tunnuspiirteenä voi mainita sen vapauttavan ominaisuuden: ohjelmistoa voi käyttää vapaasti ja levittää vapaasti, kunhan itse lisenssi on sisällytetty myös muokattujen ohjelmistojen mukana. GPL ei myöskään anna lisätä muita rajoituksia ohjelmiston lisenssiin, kuten esimerkiksi lähdekoodin sulkemista. Lisenssi siis rajoittaa vapauksien poistamista ohjelmistosta (Weber 2004). Tässä tutkielmassa käsitellään lisenssin versioista versioita 2 ja 3.

Stallman halusi GPL-lisenssillä kertoa neljästä vapaudesta jotka koskevat vapaita ohjelmistoja GPL-lisenssin alla:

1. Vapaus käyttää ohjelmistoa mihin tahansa tarkoitukseen
2. Vapaus tutkia ohjelman toimintaa lähdekoodista ja muokata tätä lähdekoodia vapaasti
3. Vapaus levittää kopioita ohjelmasta, ilmaiseksi tai maksua vastaan
4. Vapaus parantaa ohjelmaa ja vapaus levittää tästä parannetusta ohjelmistosta kopioita (Stallman 2009)

Nämä neljä vapautta takaavat tekijänoikeudettomuutta paremmin ohjelmiston vapauden. Pelkästään vapaasti levitettävä lähdekoodi on ns. vapaata riistaa, jolloin kuka tahansa voi ottaa lähdekoodin, muokata sitä ja julkaista pelkästään binäärimuotoinen sovellus koodin pohjalta ja hyötyä näin rahallisesti ja vapautta rajoittavasti. Tämänkaltainen käytös haittaa ohjelmistojen vapautta. (Stallman 2009)

Lisenssin vapautta edistävä vaikutus ulottuu myös muihin mahdollisiin ohjelmiston komponentteihin. Tällä tarkoitetaan sitä, että jos osa ohjelmistosta julkaistaan vapaana ohjelmiston GPL-lisenssin avulla, täytyy myös muiden komponenttien olla vapaita eikä muita sulkevia lisenssejä sallita. (Stallman 2009)

GPL-lisenssille on määritelty myös muutama sisarlisenssi, AGPL ja LGPL. AGPL (GNU Affero General Public License) on sellainen lisenssi, joka sallii ohjelman lähdekoodin hakemisen sellaisesta ohjelmasta jota ajetaan verkon ylitse. LGPL on sellainen

lisenssi, joka koskee enimmäkseen ohjelmien kirjastoja, jolloin kirjasto saa linkittää itsensä osaksi suljetun ohjelman suoritusobjektia.

GPL-lisenssillä on ollut versio 1, mutta se on suurimmaksi osaksi korvannut version 2 lisenssi. Lisenssin versio 1 on yleisesti vanhentunut, joten lisenssin versiota 2 suositellaan käytettäväksi. Tämä versio on näistä lisenssin versioista luultavasti kaikkein käytetyin. Lisenssin versio 2 julkaistiin vuonna 1991 ja sen päivitys, versio 3, vuonna 2007. (Välimäki 2001, Välimäki 2007)

Lisenssin versiossa 2 ehdot kopioimiselle, levittämiselle ja muuttamiselle ovat mm. ohjelman suorittamisen salliminen, sanatarkkojen kopioiden levitys, lisenssinsaajan oikeus saada pyydettäessä maksu ohjelman kopioinnista, ohjelman, sekä sen kopion, muuttamiselle ja ohjelman levitykselle objektikoodina. (Välimäki 2001)

GPL-lisenssin versiossa 3 tulivat uudistukset, jotka rajoittavat ohjelmistojen muokkausta laitteistotasolla. Näistä esimerkkinä FSF käyttää Tivo-laitetta, jossa suoritetaan monia GPL-lisenssin alaisia ohjelmistoja, mutta näiden ohjelmistojen muokkausta on rajoitettu laitteistotasolla. (Gnu 2009d)

2.2.2 BSD-lisenssi

BSD-lisenssi on lähtökohdiltaan aivan erilainen kuin GPL-lisenssi, se tarjoaa hyvin rennon lisenssin ohjelmistolle. BSD-lisenssejä on tarjolla 2 erilaista, 3-lauseinen ja 4-lauseinen. Sen eri versiot voivat olla yhteensopivia GPL-lisenssin kanssa. Alkuperäinen 4-lauseinen lisenssi ei kuitenkaan ole yhteensopiva GPL-lisenssin kanssa (GNU 2009b). Sitä vastoin 3-lauseinen lisenssi on yhteensopiva GPL-lisenssin kanssa.

Seuraavaksi kuvataan BSD-lisenssiä. Uudelleenjakelu tai käyttö niin lähdekoodi kuin binäärimuodossa, muokkauksien kanssa tai ilman, on sallittu sillä ehdolla, että seuraavat ehdot täyttyvät:

1. Lähdekoodin jakelujen on säilytettävä yllä oleva tekijänoikeuden lauseke, nämä listatut ehdot ja seuraava vastuuvapauslauseke.
2. Binäärimuotojen jakelun on tuotettava yllä oleva tekijänoikeuden lauseke, nämä listatut ehdot ja seuraava vastuuvapauslauseke ja/tai muut materiaalit jakelussa.

3. Omistajan nimeä ei saa käyttää suosittelemaan tästä ohjelmistosta johdettuja tuotteita ilman kirjallista suostumusta.

BSD-lisenssin 3-lauseinen versio on hyvin salliva lisenssi, mutta se on yhteensopiva GPL-lisenssin kanssa. BSD-lisenssi on lähtöisin Californian yliopistosta. (GNU 2009b)

2.2.3 Apache-lisenssi

Apache-lisenssi on ASF:n (Apache Software Foundation) hyväksymä ja yleisesti käyttämä lisenssi. Sen viimeisin versio, 2.0, tuli voimaan vuonna 2004. (Apache Software Foundation 2012) Se on yleisesti käytetty lisenssi esimerkiksi Googlen Android-käyttöjärjestelmän lähdekoodeissa (Android 2014). Lisenssin versio 2.0 julkaistiin siihen tarkoitukseen, että useimmin kysyttyjä kysymyksiä versiosta 1.1 saataisiin karsittua ja sallia lisenssin olevan uudelleen käytettävä missä tahansa projektissa, sallia lisenssin sisältyvän viittauksella, sen sijaan että lisenssin tulisi olla jokaisen tiedoston ylätunnisteessa. (Apache Software Foundation 2012)

Apache-lisenssi antaa tekijälleen tekijänoikeuslisenssin, maailmanlaajuisesti, ei yksinoikeudellisesti, maksuttoman, rojaltivapaan ja peruuttamattoman. Tämä lisenssi antaa lisäksi vapauden tuottaa uudelleen, valmistaa johdannaisia, julkisesti näyttää, alilisensoida ja jakaa lisenssin kohdetta. Apache-lisenssi antaa myös lisenssin kohteelle patenttilisenssin.

Jakaminen Apache-lisenssin alla onnistuu, niin kauan kuin seuraavat ehdot täyttyvät:

1. Lisenssin kohteena oleva teos tai johdannainen sisältää kopion lisenssistä
2. Kaikki muutetut tiedostot näyttävät selvästi muutoksen tekijän
3. Lisenssin kohteena olevan johdannaisen täytyy sisällyttää kaikki tekijänoikeus-, patentti-, tavaramerkki- ja muokkausilmoitukset työn lähdemateriaalissa
4. Jos lisenssin kohteena oleva teos sisällyttää NOTICE-tekstitiedoston jakelussaan, silloin jokaisen johdannaisen on sisällytettävä tämä sama tiedosto. Tiedostossa tulee käydä ilmi muut teokseen liittyvät tekijänoikeudelliset ilmoitukset

Tekijä voi antaa myös oman tekijänoikeuden omille muutoksilleen ja voi tarjota lisäehtoja tai kokonaan erilaiset ehdot käytölle, johdannaisille tai jakelulle muutoksista tai sellaisista johdannaisista kokonaisuudessaan, kunhan tekijä itse suostuu Apache-lisenssin ehtoihin käytölle, kopioinnille ja jakelulle työstä. (Apache Software Foundation 2004)

Kaikki muut panokset työhön (esim. ohjelmistoa muokkaava toinen henkilö) sisältyvät Apache-lisenssin ehtoihin. Lisenssi kieltää tavaramerkkien käyttöä tai lisenssinantajan nimeä, ellei se ole järkevää ja tavanomaista kuvailla teoksen alkuperää tai NOTICE-tiedoston kokoamista varten. Apache-lisenssi ei myöskään anna teokselle mitään takuuta ja vastuu mahdollisista haitoista on käyttäjällä itsellään. Teokselle voi kuitenkin omalla vastuulla määrätä takuun tai vastuuta mahdollisista ongelmista.

2.2.4 MIT-lisenssi

MIT-lisenssi syntyi MIT:ssä (Massachusetts Institute Of Technology) ja se on yksi vapaiden ohjelmistojen lisensseistä. MIT-lisenssi on salliva, joten sen avulla voidaan ohjelmistoa jakaa myös suljetun ohjelmiston sisällä.

MIT-lisenssi antaa luvan saada kopio ohjelmistosta maksuttomasti, muokata ohjelmistoa rajoittamattomasti, mukaan lukien rajoituksetta oikeus käyttää, kopioida, muokata, yhdistää, julkaista, jakaa, uudelleenlisensoida ja/tai myydä ohjelmiston kopioita ja sallia henkilöiden edellämainituilla teoilla, edellyttäen että MIT-lisenssi sisällytetään ohjelmistoon tai osiin sitä.

2.3 Git-versionhallinta

Git on avoimen lähdekoodin versionhallintajärjestelmä. Sen tarkoitus on tarjota hajautetun mallin versionhallintajärjestelmä, joka tarkoittaa sitä, että toisin kuin esimerkiksi SVN, kaikki sisältö replikoidaan aina eri käyttäjien välillä riippumatta keskuspalvelimesta. Git kehitettiin alunperin Linux-ytimen kehityksen tueksi, mutta se on ollut suosittu myös muussa versionhallinnassa avoimen lähdekoodin projekteissa. Tässä luvussa kuvataan Git-versionhallintaa pitäen silmällä myöhempien lukujen prosessien kuvausta. (Chacon 2014)

Git syntyi vuonna 2005 kun silloinen versionhallintajärjestelmä Linux-ytimelle, BitKeeper, lopetti yhteistyön ytimen kehittäjien ja BitKeeperin organisaation kanssa. Linus Torvalds halusi tilalle jotain samankaltaista kuin BitKeeper. Tällaisia ominaisuuksia olivat mm. nopeus, yksinkertainen rakenne, tuki epälineaariselle kehitykselle, tuki isojen projektien tehokkaaseen käyttöön ja hajautettu rakenne. (Chacon 2014)

Git-versionhallinnan perusteet

Toisin kuin muut versionhallintajärjestelmät, Git ei tallenna yleisesti muutoksia tiedostoihin vaan tallentaa ne sen sijaan kaikkien tiedostojen tilannekuvina. Esimerkiksi kun tiedostoa luodaan, siihen tallennetaan tilannekuva. Kun tiedostoa myöhemmin muokataan, tehdään siitä taas oma tilannekuvansa, johon voi kuulua muitakin tiedostoja. (Chacon 2014)

Gitin toiminnan ytimessä ovat sen kolme eri tilaa: työhakemisto, lavastusalue ja itse hakemisto eli tietovarasto. Tiedostot ovat aina jossakin näistä tiloista, joko pysyvästi muutettuina (*committed*), muutettuina (*modified*) tai lavastettuina (*staged*). (Chacon 2014)

Kun tiedosto ei ole osa Git:n tietokantaa, on se jäljittämätön. Tiedoston saa osaksi Git:n tietokantaa komennolla *add*. Tällöin se lisätään muutettujen tiedostojen tilaan. Tämän lisäksi tiedosto on lavastettu pysyvää muutosta varten, jollei siihen ole tehty muutosta muutettuihin tiedostoihin lisäyksen jälkeen. Jos tiedostoa on muutettu, täytyy nämäkin muutokset lisätä komennolla add. Tämän jälkeen tiedostot ovat valmiita pysyvän muutoksen tekemiseen. Pysyvä muutos saadaan aikaan komennolla *commit*. Pysyvään muutokseen lisätään tässä vaiheessa viesti, jonka sisältö on vapaavalintainen. On yleisesti suosittu tapa tehdä pysyvän muutoksen viesti tiedostoihin tehtyjen muutosten tarkoituksesta. (Chacon 2014)

Kun ajatellaan Git:n toimintaa, voidaan sen ajatella esittävän tiedostojärjestelmää. Kun pysyvä muutos tehdään, tarkoittaa se sitä, että tehdään tilannekuva sen hetkisistä muutoksista tiedostoihin ja tallennetaan SHA-1-tiiviste viitteeksi tästä tilannekuvasta. Näitä viitteitä käytetään Git:ssä jokaisen operaation tuloksena syntyvissä objekteissa, näitä ovat esimerkiksi pysyvät muutokset, haarat ja tagit. Tällä tarkistussummalla varmistetaan tiedostojen eheys ja viitataan kaikkiin Git:n objekteihin. (Chacon 2014)

Git:n muut objektit, kuten haarat ja tagit ovat myös tarkistussummattuja ja niiden viittaukset ovat myös SHA-1 muotoa. Ne viittaavat aina tiettyyn pysyvään muutokseen. Tagilla (tag) tarkoitetaan merkkiä pysyvien muutosten historiassa, jonka avulla kehittäjä voi esimerkiksi viitata ohjelmistonsa lähdekoodin tiettyyn versioon. Haarat (branch) ovat kuten tagit, mutta ne voivat viitata pysyvien muutosten sarjaan. Haaran päätä kutsutaan nimellä *head* ja oletushaaraa nimellä *master*. Kun haaraan lisätään pysyviä muutoksia, haaran pää siirtyy aina viimeisimpään pysyvään muutokseen. Haaraa vaihdettaessa haaran pää siirtyy aina vaihdetun haaran viimeisimpään pysyvään muutokseen. (Chacon 2014)

Toisin kuin muissa versionhallintaohjelmistoissa, Git käyttää haaroja vain merkkeinä muutoksiin, luomatta itse kokonaista tiedostopuuta uudestaan. Tämän takia haarat ovat Git-versionhallinnassa tärkeitä ja käteviä käyttää. Kun jotain haaraa on jatkettu halutulla määrällä pysyviä muutoksia, voidaan se liittää takaisin oletushaaraan. Tämä on yleisesti käytetty tapa hallita eri haarojen muutoksia. Haaran liittämisellä tarkoitetaan sellaista operaatiota, joka luo uuden pysyvän muutoksen haarojen välisten muutosten liitokselle. Liitoksessa haarojen väliset muutokset yhdistetään ja mahdolliset muutosten väliset ristiriidat tulevat näkyviin. Ristiriidat voi korjata käsin ja sen jälkeen jatketaan tekemällä liitoksen pysyvä muutos. (Chacon 2014)

Yksi Git:n keskeisin ominaisuus on sen toiminta verkon ylitse. Verkon ylitse tapahtuvia operaatioita ovat esimerkiksi lähdekoodivaraston kloonaus ja pysyvien muutosten "työntäminen"palvelimella olevaan lähdekoodivarastoon. Kloonauksella tarkoitetaan sellaista operaatiot, jossa lähdekoodivaraston koko sisältö ja sen muutoshistoria siirretään paikasta toiseen. Tämä voi tapahtua paikallisesti tai verkon ylitse. Kun kloonaus on suoritettu, on operaation suorittajalla käytössään kaikki lähdekoodivaraston sisältämät pysyvät muutokset, haarat ja tagit. (Chacon 2014)

2.4 Avoin lähdekoodi: Case Nemo Mobile

Tässä luvussa tarkastellaan erästä avoimen lähdekoodin Linux-jakelua nimeltä Nemo Mobile. Ohjelmisto toimii pohjana tässä tutkielmassa myöhemmin esiteltävässä käyttöliittymässä nimeltä Glacier. Tässä luvussa kuvataan myös Nemo Mobilen kehityksessä käytettyä prosessia tarkasti Git-versionhallinnan termien avulla.

Mer

Mer on avoimen lähdekoodin ja vapaan ohjelmiston projekti, joka on keskittynyt tarjoamaan ohjelmistokokoelman GNU/Linux-pinolle ja olemaan optimoitu erityisesti sulautetun järjestelmän laitteille. (Mer Project FAQ 2012) Mer-projektia hallitaan meritokratian avulla, jossa komponenttien vastuuhenkilöiden tehtävät on jaettu sen mukaan, minkä ohjelmiston komponentteja vastuuhenkilö on ennen hallinnut onnistuneesti. (Mer Project Governance 2012)

Nemo Mobile

Nemo Mobile on eräs Linux-jakelu, jonka tarkoitus on tarjota vapaan ohjelmiston ja avoimen lähdekoodin mukainen jakelu mobiililaitteille ja sulautetun järjestelmän laitteille, kuten esim. Raspberry Pi. Mobiililaitteista kehityksessä käytetään Nokian N9- ja N950-laitteita, joille on jo tehty laitteistotason tuki. Mutta kuten mainitsen luvussa 2.4, laitteisto ei saata tulevaisuudessa rajoittua pelkästään näihin kahteen laitteeseen.

Nemo Mobile käyttää Mer-projektia ytimenään ja rakentuu sen ympärille tarjoten yhteisölle alustan kehittää väliohjelmistoa ja käyttöliittymää. Aluksi Nemo perustui MeeGo-projektin yhteisön versioon, mutta MeeGo-projektin loputtua se siirtyi käyttämään QML-pohjaista lipstick-käyttöliittymää. (Nemo 2014)

Nemo Mobilen kehitysprosessi

Nemo Mobile käyttää lähdekoodinsa säilytykseen Github-palvelua. Sen tarjoamat lähdekoodin tietovarastot Git-versionhallinnan avulla ovat yhteisön mielestä parhaimmat. Kehitysprosessi on melko yksinkertainen kun Git-versionhallinnan termistö on tuttua, termistöä käydään läpi luvussa 2.3.

Jokainen ohjelmisto on yksi Git-varasto, jossa lähdekoodi sijaitsee. Ohjelmistojen laaja kokoelma on nähtävissä osoitteessa `https://github.com/nemomobile`. Ohjelmiston lähdekoodi on yleisesti kaikkien nähtävissä ja paketoitu yhdeksi paketiksi OBS-palvelussa osoitteessa, esimerkiksi `https://build.merproject.org/package/show/nemo:devel:mw/qt-components`.

Kun avoimen lähdekoodin kehittäjä haluaa tehdä muutoksia johonkin lähdekoodiva-

rastoon, hänen on ensiksi tehtävä erityinen haarautumaprojekti (*fork*), joka on Github-palvelun erityinen ominaisuus. Kun haarautuma on tehty onnistuneesti voi kehitystyö alkaa.

Kehitystyö aloitetaan kloonaamalla lähdekoodivarasto haaraprojektista. Kloonauksessa varaston sisältö tuodaan kehittäjän levylle. Kun kloonaus on suoritettu onnistuneesti niin lähdekoodin muuttaminen voi alkaa.

Kun tarpeelliset muutokset on saatu tehtyä, niin Git-versionhallinnassa täytyy tehdä erityinen pysyvä muutos, jonka viesti on etuliitetty hakasuluilla ja sisältää tiedon tehdyistä muutoksista, esim. *[projekti] Muutos virtuaalinäppäimistössä*. Kun pysyvä muutos on tehty ja *työnnetty* kehittäjän haarautumaprojektiin Github-palveluun, voidaan jatkaa työn tulosten esittelyä tekemällä erityinen *vetopyyntö* (pull request).

Vetopyyntö on eräs Github-palvelun ominaisuus, jolla kehittäjät voivat tuoda muutoksiansa julki ja keskusteltavaksi kaikkien muiden kehittäjien välille. Täten muutokset voidaan helposti arvioida ja mahdolliset korjaukset voidaan sitten ehdottaa kommentoimalla vetopyyntöä web-rajapinnan avulla. Vetopyynnössä halutaan aina yhdistää kaksi haaraa toisiinsa, yksi kehittäjän lähdekoodivarastosta ja toinen pääkomponentin lähdekoodivarastosta.

Kun vetopyyntö on hyväksytty ja haarautumaprojektin pysyvät muutokset on yhdistetty osaksi päälähdekoodivarastoa, voi ohjelmiston uudelleenrakentaminen alkaa. Tämän tekee yleensä päälähdekoodivaraston vastuuhenkilö ja se tapahtuu tekemällä erityinen Git-versionhallinnan tutuksi tekemä *tag*. Tag täytyy nimetä versioksi, kuten 1.2.3 ja tämän version täytyy olla järjestykseltään suurempi kuin edellinen päälähdekoodivaraston versio.

Tagin luominen käynnistää taustaprosessin OBS-palvelussa, jossa lähdekoodivaraston sisältö otetaan palvelun sisälle ja lähdekoodi käännetään uusimmilla muutoksilla uudestaan.

Libhybris

Ohjelmiston kehitys alkaa yleensä lähdekoodin muodostamisesta. Kun ohjelmistoa halutaan suorittaa, tulee se kääntää tietyn kielen kääntäjällä tai tulkilla suoritettavaksi ohjelmistoksi. Jos ohjelma käännetään esimerkiksi C-kielellä, tulee käännöksen keski-

kohdassa tehdä *linkitys*, jolla tarkoitetaan sitä, että kääntäjä luo objektitiedostot lähdekoodista ja sitten linkittää nämä tiedostot suoritettavaksi ohjelmaksi. Joskus suoritettava ohjelmisto tarvitsee linkitykseensä symboleita toisista kirjastoista, jotka sijaitsevat käyttöjärjestelmän kirjastohakemistossa. Kirjastot tällöin linkitetään dynaamisesti ohjelmistoon. Linux-käyttöjärjestelmissä tästä huolehtii libdl-kirjasto.

Kirjaston nimi tulee kreikan sanasta *hubris*, joka tarkoittaa äärimmäistä ylpeyttä tai röyhkeyttä. Se tarkoittaa myös todellisuudesta irtautuvaa taitojen yliarviointia. Munk kertoo, että hän kirjoitti Android-käyttöjärjestelmän linkkerin glibc-ympäristöön. Libhybris pyrkii ratkaisemaan ongelman nykypäivän mobiiliteollisuuden laitteiston toteutuksessa, ongelman jossa laitteiston tarjoajat tuovat markkinoille vain Android-pohjaisia toteutuksia laitteistolle. Libhybris pyrkii hyödyntämään näitä toteutuksia, tarjoten mahdollisuuden ajaa muitakin käyttöjärjestelmiä Android-pohjaisissa laitteissa, kuin Android itse. (Munk 2013a)

Kun tehdään ohjelmiston mukautusta GNU:n glibc-kirjastolla sopivaksi, siihen tarvitaan dynaamisen linkityksen apukeinoja libdl-kirjastosta. Libhybris on tällainen ohjelmisto, jonka tehtävänä on käyttää glibc-kirjaston symboleita hyväksi Android-pohjaisissa laitteiston mukautuksissa, johon sisältyy kirjastokutsuja esimerkiksi OpenGL-kirjastoon. Libhybris tekee dynaamisessa linkityksessä version sekä Androidin bionic-kirjastolle, että GNU:n glibc-kirjastolle ja avaa täten molemmille kirjastoille sijaa ajonaikaisen ohjelman prosessissa. Siten voidaan esimerkiksi suorittaa Android-järjestelmän ajureiden avulla erilaisia käskyjä esimerkiksi ikkunoiden piirtämiseen Wayland-näyttöprotokollalla. (Munk 2013a)

Wayland-näyttöprotokollaa alettiin kehittämään vuonna 2012 kun Kristian Høgsberg ilmoitti Waylandin ja Weston-ohjelman julkaisusta. Osa libhybriksen mahdollisuuksista perustuu Wayland-protokollaan ja sen käyttämään laitteistopohjaiseen graafiseen kiihdytykseen, jotka perustuvat EGL (Embedded-system Graphics Library)- ja OpenGL (Open Graphics Library) ES (Embedded Systems) rajapintoihin. Se miten libhybris käyttää Waylandia, perustuu Android-järjestelmien käyttämään EGL-toteutukseen. Android käyttää ikkunatyyppinään tyyppiä ANativewindow, joka on kahva graafiseen puskuriin, joka sitten annetaan piirrettäväksi joko SurfaceFlingerillä tai libhybriksen Wayland-toteutukselle. (Munk 2013a)

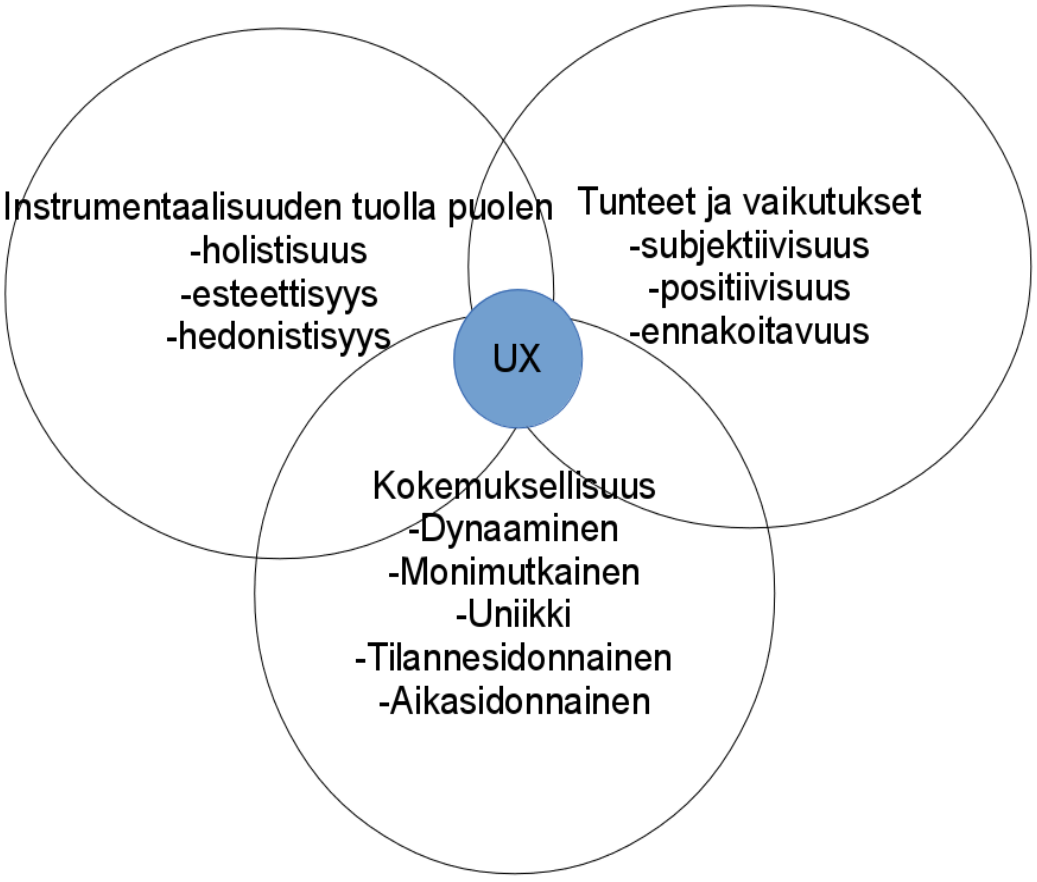
Tämä toteutus on avannut uusia polkuja Nemo Mobilen tulevaisuudelle, koska nyt on mahdollista käyttää olemassa olevia Android-laitteita Nemo Mobile-

käyttöjärjestelmän suorittamiseen. Olen itse saanut sen toimimaan jo useissa laitteissa ilman erityisiä ponnistuksia itse käyttöjärjestelmän komponenttien toimimattomuuden kanssa.

3 Käyttäjäkokemus

Käyttäjäkokemus on ollut yleisesti pelkkä muotisanonta, mutta sen taustalla oleva tutkimus on osoittanut todellisuuden aivan toiseksi (Rousi 2013, Law 2009). Termille on saatu määritelmä ISO-standardissa 9241-210 (ISO 2009). Määritelmä kuuluu näin: "Henkilön havainnot ja vasteet, jotka ovat seurausta tuotteen, järjestelmän tai palvelun käytöstä ja/tai ennakoidusta käytöstä". Tämän lisäksi Suomen kielestä löytyy termille myös toinen suomennos, käyttökokemus, jota ei tässä tutkielmassa kuitenkaan käytetä.

Käyttäjäkokemus on eräänlainen ilmentymä holistisesta näkökulmasta ihmisen ja tietokoneen vuorovaikutuksessa. Käyttäjäkokemukseen sisältyy esimerkiksi hedonistisyys, tunteet ja kokemuksellisuus (Hassenzahl 2006a), kuten kuvassa 1 näkyy kuinka käyttäjäkokemus on näiden kaikkien ominaisuuksien yhdistelmä.



Kuva 1: Käyttäjäkokemuksen puolet

Luvussa 3.1 tutkitaan käyttäjäkokemuksen eri puolia ja sen vaikutusta tutkielmassa myöhemmin esitettävään käyttöliittymään (luku 4). Luvussa 3.2 tutkitaan muutamia

käyttäjäkokemuksen suunnitteluun liittyviä teorioita.

3.1 Käyttäjäkokemuksen eri puolet

Tämä standardoitu määrittely (ISO 2009) kertoo käyttäjäkokemuksesta paljon olennaista kuten sen, että myös ennen itse fyysistä kokemusta saatu kokemus on osa käyttäjäkokemusta. Standardissa huomioidaan käyttäjäkokemuksen holistisempi näkökulma, johon sisältyy: "kaikki käyttäjien tunteet, uskomukset, mieltymykset, fyysiset ja psyykkiset vasteet, käyttäytymiset ja aikaansaannokset, jotka ilmenevät ennen käyttöä, käytön aikana ja käytön jälkeen."Juurikin tämä holistisuus aiheuttaa käyttäjäkokemuksen vaikean määrittelemisen, vaikka se onkin tässä tutkielmassa edellä mainittu. Jo sanana käyttäjäkokemus viittaa kokemuksen käsitteeseen ja kokemus itsessään on yksityinen ja subjektiivinen.

Käyttäjäkokemukselle on kuvattu monia muitakin määritelmiä (UX 2013). Tämä kertoo käyttäjäkokemuksen vaikeasta luonteesta ja sen holistisuudesta. Hassenzahl (2006a) kuvaa käyttäjäkokemusta seurauksena käyttäjän sisäisestä tilasta mm. ennakkoluulot, odotukset, tarpeet, motivaatio ja mielentila, suunnitellun järjestelmän ominaisuuksista esim. monimutkaisuus, tarkoitus, käytettävyys ja toiminnot ja käytön kontekstista mm. käytön vapaaehtoisuus, sosiaalinen tilanne. Law et al. (2009) kertovat myös haastattelujen pohjalta tehdyssä tutkimuksessa eri näkökulmia käyttäjäkokemukseen.

Käyttäjäkokemuksen suunnittelu on yksi modernin ihmisen ja tietokoneen vuorovaikutuksen suunnitelun haasteista. Hassenzahl (2006a) esitti, että voi olla haastavaa kehittää tuotteita kokemuksen pohjalle, sillä kokemukset ovat yksityisiä ja kokemuksen aikana muodostuneet tunteet ovat myös yksityisiä. On mainittava, että kauniit tuotteet vetoavat enemmän ihmisiin, kuin rumat tuotteet (Hassenzahl 2006b).

Kun puhutaan tunteista, viittaukset tutkimuksen osalta viittaavat usein affektiiviseen laskentaan. Käyttäjäkokemus ei kuitenkaan suoraan liity affektiiviseen laskentaan vaan käyttäjäkokemus tarjoa ihmisnäkökulman kun taas affektiivinen laskenta näkökulman tunteisiin koneiden näkökulmasta.

Nykyinen tutkimus on identifioinut kolme pääkohtaa käyttäjäkokemuksen tutkimuksessa (Roto et al, 2011): käyttäjäkokemus ilmiönä, käyttäjäkokemus tutkimuskohteena

ja käyttäjäkokemus käytännössä.

Kun tarkastellaan käyttäjäkokemusta ilmiönä, voidaan sanoa käyttäjäkokemuksen olevan kokemuksen alajoukko yleisesti. Käyttäjäkokemus on paljon yksityiskohtaisempi, sillä se liittyy aina jonkin järjestelmän käyttämisen kokemukseen. Käyttäjäkokemus sisältää myös kaikki kohtaamiset järjestelmän kanssa: ei vain aktiivisia kokemuksia vaan myös passiiviset kohtaamiset ovat osa käyttäjäkokemusta. Kuten aiemmin jo todettiin, käyttäjäkokemus liittyy aikaisempiin kokemuksiin tuotteesta ja sen käytön odotuksesta (Roto et al, 2011) ja voidaankin sanoa, että käyttäjäkokemus alkaa jollain tietyllä kokemuksella, esimerkiksi videon katsomisella Internetistä, jossa vaikkapa käyttöliittymää kuvataan.

Tutkimuskohteena käyttäjäkokemus on vielä vailla yhteistä perusmallia, joka sopisi kaikille sitä tutkiville tutkijoille (Hassenzahl 2008). Tutkimus osoittaa, että eri malleja kehitellään jatkuvasti ja näiden mallien käyttökelpoisuutta on selvitetty jo tutkimuksessa (Law et al. 2013). Kuinka käyttäjäkokemusta sitten tulisi tutkia? Nykytiede tuntee hyvinkin paljon menetelmiä (Vermeeren et al. 2010), joista suurin osa sisältää tutkimusta laboratoriossa ja kentällä. Myös erilaisia kvalitatiivisia menetelmiä on noussut esille (Rousi 2013, Law et al. 2009). Näistä esimerkiksi skaalaristikko (repertory grid) on toiminut tutkimuksessa hyvin. Skaalaristikossa erilaisia objekteja pyritään luokittelemaan adjektiivien vastakohdilla. Muita menetelmiä ovat esimerkiksi etnografiset tutkimukset ja haastattelut. Kvantitatiivisten menetelmien käyttö on myös suosittua, kuten esimerkiksi lomakepohjaiset tutkimukset. Täytyy huomata, että pelkän kvantitatiivisen tiedon tuottaminen käyttäjäkokemuksesta voi olla tutkimusten tulosten kannalta haitallista (Law et al. 2013).

Käyttäjäkokemuksen kokonaisuudessa on otettava huomioon sen aikapainotteisuus. Käyttäjäkokemus voi alkaa ennenkuin käyttäjä on edes ollut kosketuksissa laitteeseen tai järjestelmään: tämä voi johtua esimerkiksi mainostuksesta tai vastaavista ilmiöistä. Käyttäjäkokemus siis alkaa jo ennen käyttöä. Käytön aikana ja erityisesti ensimmäisellä käyttökerralla käyttäjäkokemus on ns. voimakkaimmillaan: käytön jälkeen sen kokeminen harvenee tasaisesti kun käyttökerrat vähenevät. (Roto et al, 2011) Pitkäaikaista käyttäjäkokemuksen tutkimusta varten on tehty erilaisia menetelmiä, käytetyimpänä on käyttäjäkokemuksen käyrä, jonka kehittivät Sari Kujala et al. Tällainen käyrä syntyy käyttäjäkokemuksen tutkimuksessa siitä, kuinka tuote on käyttäjien käytössä muuttunut ajan myötä. Myös muita ulottuvuuksia voidaan kuvata tällaisilla käyrillä, kuten esimerkiksi sitä miten tuotteen helppokäyttöisyys on muuttunut ajan kuluessa.

(Kujala et al 2011)

Battarbee ja Koskinen (2005) selvittivät tutkimuksissaan myös käyttäjäkokemuksen sosiaalista puolta, kanssakokemuksen avulla. Kanssakokemus on hyvin suuri osa käyttäjäkokemusta, sillä täytyyhän ihmisten kokea omat kokemuksensa toisten ihmisten kanssa. Battarbee ja Koskinen kuvaavat kolme eri sosiaalista tyyppiä käyttäjäkokemukselle: nostavat kokemukset, vastavuoroiset kokemukset ja hylkäävät sekä välinpitämättömät kokemukset. Nostavilla kokemuksilla tarkoitetaan sellaisia kokemuksia, jotka nostetaan kokemusten virrasta toisten koettavaksi, esim. henkilö voi kuvata jotain päivän aikana tapahtunutta ja haluaa jakaa sen muiden kanssa. Vastavuoroiset kokemukset syntyvät silloin kun vastaavaa on koettu toisen ihmisen toimesta ja ihminen jakaa tämän kokemuksen toisen kanssa. Hylkäävät ja välinpitämättömät kokemukset alkavat kokemuksen hylkäyksestä tai sen huomiotta jättämisestä. Ihmiset siis kokevat yhdessä hyvin samankaltaisilla tavoilla. (Battarbee et al 2005)

Käyttäjäkokemus on siis hyvin monitahoinen tutkimuskohde. Sen lisäksi, että kokemukset voivat olla helposti kategorisoitavissa sen tarpeen mukaan, mitä ne täyttävät (Hassenzahl et al. 2010), on myös huomattava miten ne ovat olemukseltaan aina henkilökohtaisia. Tämän lisäksi on huomattava, että kulttuuri vaikuttaa olennaisesti kokemuksen syntymiseen ja muotoutumiseen ajan kanssa. Kulttuurisia tekijöitä käyttäjäkokemuksessa on tutkittu mm. Marcuksen (2006) toimesta.

3.2 Käyttäjäkokemuksen suunnittelu

Tässä luvussa on tarkoitus kuvata erilaisia ohjenuoria käyttäjäkokemuksen suunnittelulle. Niitä ovat esimerkiksi erilaiset heuristiikat sekä käyttäjäkokemukselle, että käytettävyydelle, joka on osa käyttäjäkokemusta. Lisäksi esitellään Marc Hassenzahlin (2010) mallin käyttäjäkokemukselle ja myös tähän malliin pohjautuvan affordanssiteoriaan pohjautuvan mallin (Pucillo 2014).

Nykyinen suuntaus käyttäjäkokemuksen saralla kertoo pitkälti uraa uurtavasta tutkimuksesta ja teollisuuden haasteista. Tässä luvussa pyritään kuvailemaan käyttäjäkokemuksen suunnittelua ja sen haasteita.

Marc Hassenzahl (2013) on visioinut kokemuksellisesta suunnittelusta kolme eri lähtökohtaa:

1. Miksi? Miksi on ensimmäinen lähtökohta käyttäjäkokemuksen suunnittelussa. Esimerkiksi tekstiviesti ei ole pelkästään tekstiviesti riippuen asiayhteydestä vaan esimerkiksi rakkauskirje tietyssä kontekstissa.

2. Miten? Miten on ensimmäinen kysymys kun suunnittelija antaa toiminnallisuuden tuotteelle, esteettisesti mielyttävällä tavalla. Hassenzahl antaa esimerkiksi puhelinsoiton, joka voidaan tehdä monella eri tavalla.

3. Mitä? Mitä tarkoittaa sitä mitä vuorovaikutuksellisella tuotteella voi tehdä.

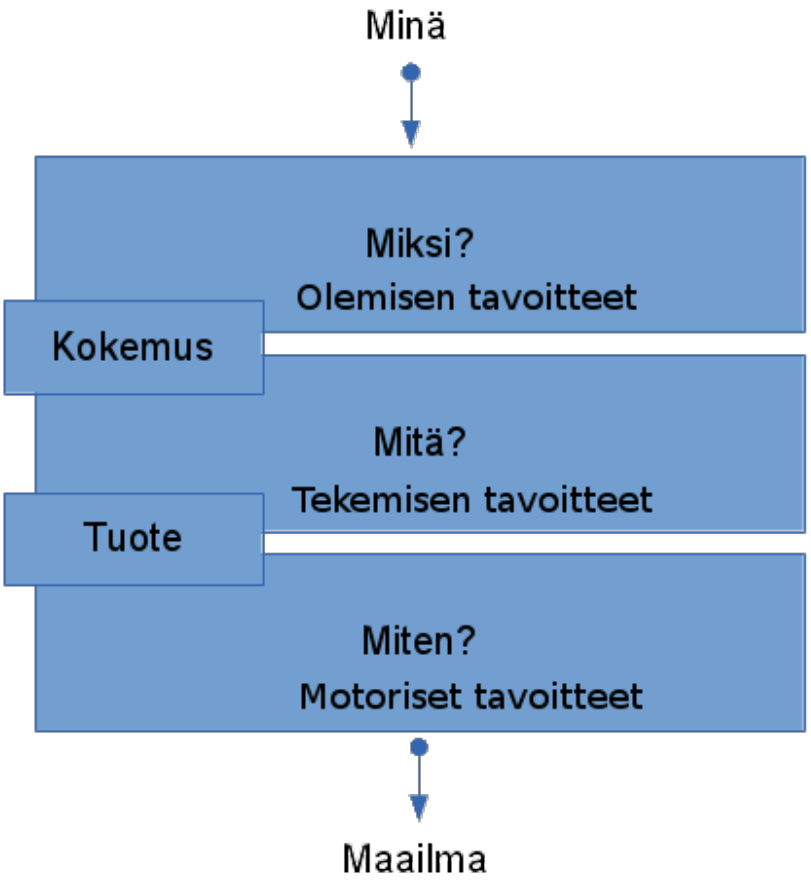
Käyttäjäkokemuksen suunnittelussa edellä mainitut kolme kysymystä on esitetty myös kuvassa 2. (Hassenzahl 2010) Tässä kuvassa ilmenee selkeästi ihmisen minän suhde tuotteeseen, kuvailtuna tuotteen kautta. Tuotteen Miksi?-ominaisuus synnyttää kokemuksen, Mitä?-ominaisuus kuvailee sen mitä tuotteella voi tehdä ja Miten?-ominaisuus lopullisen vuorovaikutteisen puolen. Lisäksi tälle tuotteelle on olemassa erilaisia tavoitteita, kuten motoriset tavoitteet, tekemisen tavoitteet ja olemisen tavoitteet.

Olemisen tavoitteissa pyritään saamaan aikaan sellainen tavoite, joka edellyttää esimerkiksi kokemusta tuotteesta. (Hassenzahl 2010) Pelkkä tekstiviesti muuntautuukin rakkaus-kirjeeksi, puhelin laitteeksi jolla voi pitää yhteyttä kaukana olevaan rakkaaseen ihmiseen. Olemisen tavoitteisiin liittyy lisäksi ajatuksia nimeomaan olemisesta. "Ole kyvykäs"tai "ole autonominen"esimerkkeinä. Tekemisen tavoitteissa on kyse sellaisista tavoitteista joita ihminen tarvitsee tehdäkseen tietyn asian. Lähimpänä maailmaa ovat motoriset tavoitteet, joihin pääsemiseksi tarvitsee vain olla vuorovaikutuksessa tuotteen tietyn elementin, ts. kosketusnäytön kanssa. Motoriset tavoitteet ovat siten yksinkertaisimpia ymmärtää, mutta niiden väärin tekeminen voi johtaa muiden tavoitteiden pääsemättömyyteen.

Käyttäjäkokemuksen suunnittelussa on luotu avuksi seuraavat 10 heuristista suunnittelun pääkohtaa (mukaillen Arhippainen 2013):

1. **Varmista käytettävyys.** Käyttäjät kokevat käytettävyyden: on siksi tärkeää varmistaa, että suunniteltava palvelu tai tuote on käytettävä

2. **Tarjoa käyttäjälle hyötyjä, jotka vastaavat käyttäjän arvoja.** Tuotteen tai palvelun hyödyt vaikuttavat käyttäjäkokemukseen. Koettu hyöty antaa anteeksi tuotteen mahdolliset virheet.



Kuva 2: Kokemuksen suunnittelu, mukaelma Hassenzahl (2013)

3. **Ylitä käyttäjän odotukset.** Käyttäjän odotukset voivat olla negatiivisia ilman syytä. Odotukset voivat kehkeytyä edellisistä kokemuksista tai esimerkiksi huhuista, joten nämä odotukset eivät välttämättä vastaa tuotetta. Tuotteen pitäisi saada käyttäjän huomio positiivisella tavalla ja antaa käyttäjän aloittaa tuotteen käyttö ja sitten ylittää hänen odotuksensa esimerkiksi helppoudella, millä tahansa ominaisuudella hän haluaakin.

4. **Kunnioita käyttäjää.** Tiedosta kohderyhmät. Käyttäjän taustalla on suuri vaikutus siihen miten hän kokee tuotteen tai palvelun. Käyttäjän tarpeiden lisäksi suunnittelijoiden pitäisi ymmärtää käyttäjän arvot, aikaisemmat kokemukset, käyttäjätyypit, taidot, rajoitteet, yms. Mitä paremmin tuote tai palvelu vastaa käyttäjän maailmaa, sitä parempia kokemuksia käyttäjä saa.

5. **Suunnittele tuote tai palvelu sopimaan tarkoitettuihin asiayhteyksiin**. Tuotetta tai palvelua käytetään aina tietyssä asiayhteydessä: käyttäjä käyttää tuotetta fyysisessä ominaisuudessa, toisen ihmisen seurassa tai yksin erilaisten kulttuu-

risten taipumusten ja elämäntapojen vaikuttaessa. Kaikki asiayhteyden tekijät vaikuttavat käyttäjäkokemukseen.

6. **Tarjoa monia tapoja vuorovaikuttaa, jätä vaihtoehto käyttäjälle.** Ihmiset ovat erilaisia ja haluavat erilaisia tapoja vuorovaikuttaa tuotteeseen tai palveluun. On tärkeää tarjota monia erilaisia vuorovaikutuskeinoja. Tarjoa manuaalista ja muokkautuvaa kontrollia ja kosketus, ele tai ääniohjautuvuutta kun mahdollista.

7. **Kunnioita käyttäjän yksityisyyttä ja tietoturvaa.** Maailma on digitaalisesti ja teknologisesti orientoitunut. Vaikka asenteet ovat muuttumassa entistä avoimempiin teknologisiin ratkaisuihin, ihmiset ovat edelleen huolissaan yksityisyydestään ja tietoturvasta. Käyttäjäkokemus riippuu aina epävarmuudesta siihen, kuinka luotettava palvelu on yksityisyyden ja tietoturvan kannalta.

8. **Tue käyttäjän aktiviteettia, älä pakota.** Kaikki palvelut tulisi tehdä tuettavuuden näkökulmasta, esim. miten palvelu tukee käyttäjän päivittäisiä toimintoja ja jokapäiväistä elämää. Palvelu ei saa pakottaa käyttäjää, koska se vaikuttaa negatiivisesti käyttäjäkokemukseen.

9. **Pyri täydelliseen visuaaliseen suunnitteluun.** Käyttäjäkokemuksen kannalta visuaalisilla näkökohdilla on kaksi merkitystä. Ensiksi visuaalinen suunnittelu voi parantaa käytettävyyttä, tekemällä käyttöliittymästä ymmärrettävämpi, yhtenäinen ja ohjaava. Toiseksi käyttöliittymästä on tehtävä esteettisesti miellyttävä suunnittelemalla visuaalisia näkökulmia. Lisäksi visuaalisen suunnittelun valinnat, esimerkiksi värien käyttö, voivat vaikuttaa käyttäjäkokemukseen.

10. **Anna yllätyslahja.** Ihmiset haluavat enemmän. Käytettävyys ei ole tarpeeksi. "Linkkuveitsi-puhelin"ei ole tarpeeksi. Käyttäjät haluavat jotain lisäarvoa, jotka tekevät heidät onnelliseksi, ylittämällä odotuksia ja parantamalla ja lisäämällä käyttäjän kokemuksia. Kokemuksen laajuus ei saa vähentyä. Käyttäjäkokemus on seitsemäs aisti, jolla ihmiset aistivat teknologiaa, elämää teknologiassa.

Edellä esitellyt heuristiikat eivät ole missään muodossa lopullisia tai orjallisesti noudatettavia. Ne antavat suuntaviivoja seuraavan luvun komponenttien kuvailemiselle. Olisi liian yksinkertaista jos käyttäjäkokemuksen suunnittelun voisi tiivistää näihin edellä mainittuihin ominaisuuksiin, mutta ne toimivat tarpeeksi hyvin ohjenuorana tulevaisuuden tutkijoille ja suunnittelijoille.

Lisäksi käyttäjäkokemuksen suunnittelussa on mahdollista käyttää esim. Nielsenin (2005) heuristiikkoja käytettävyydelle:

1. **Järjestelmän tilan näkyvyys.** Järjestelmän tulisi aina pitää käyttäjät informoituna järjestelmän tilasta, sopivalla palautteella sopivassa ajassa.

2. **Sovita maailma ja järjestelmä.** Järjestelmän tulisi puhua käyttäjien kieltä siten, että se käyttää sanoja, fraaseja ja konsepteja, jotka ovat tuttuja käyttäjälle. Seuraa oikean maailman käytäntöjä, saadaksesi tiedon näyttämään luonnolliselta ja loogiselta.

3. **Käyttäjän hallinnan vapaus.** Käyttäjät usein valitsevat järjestelmän toimintoja vahingossa ja tarvitsevat selvästi näkyvän "hätäuloskäynnin", poistuakseen virheellisestä tilasta. Tue "peru"ja "tee uudelleen-toimintoja.

4. **Johdonmukaisuus ja standardit.** Käyttäjien ei tulisi ihmetellä tarkoittavatko eri sanat, tilanteet tai toimenpiteet samaa asiaa. Tue alustan konventioita.

5. **Virheen ehkäisy.** Virheilmoituksia parempi on aina suunnitella järjestelmä ehkäisemään virheitä. Joko eliminoimalla virhetilanteet tai tarkistamalla ne voit esittää käyttäjille varmistusvaihtoehdon ennen kuin he valitsevat toiminnon.

6. **Mieluiten tunnistaminen kuin muistaminen.** Minimoi käyttäjän muistin kuormitusta tekemällä objektit, toiminnot ja valinnat näkyväksi. Käyttäjän ei tulisi muistaa aikaisemmin valittuja asioita. Järjestelmän tulisi tarjota ohjeet joko suoraan näkyville tai helposti saatavilla olevaksi.

7. **Joustavuus ja tehokkuus käytössä.** Kiihdyttimet, jotka ovat näkymättömissä noviiseille, voivat nopeuttaa toimintaa asiantuntijoille. Järjestelmän tulisi tarjota sekä kokemattomille, että kokeneille käyttäjille toimintoja. Käyttäjille tulisi olla sallittua räätälöidä usein käytettyjä toimintoja itsenäisesti.

8. **Esteettinen ja minimalistinen suunnittelu.** Dialogit eivät saisi näyttää tarpeetonta informaatiota. Jokainen visuaalinen elementti dialogissa kilpailee tarpeellisen tiedon kanssa ja vähentää suhteellista näkyvyyttä.

9. **Auta käyttäjiä tunnistamaan, diagnosoimaan ja palautumaan virheistä.** Virheviestit tulisi ilmaista pelkästään kielellä, ei virhekoodilla, sekä ilmoittaa tarkasti virheen ongelma ja rakentavasti ehdottaa ratkaisua.

10. **Ohjeet ja dokumentaatio.** Vaikka on parempi, että järjestelmää voi käyttää ilman dokumentaatiota, on parempi tarjota ohjeita ja dokumentaatiota järjestelmästä. Kaiken sellaisen informaation tulisi olla helppohakuista, keskittyä käyttäjän tehtävien listaan ja listata konkreettiset toimet toiminnon suorittamiseksi ja ei olisi liian isokokoinen.

Käyttäjäkokemuksen suunnittelun uusimpia suuntauksia on ollut ns. affordanssiteorian käyttäminen.(Pucillo 2014) Affordansilla tarkoitetaan jonkin objektin sallimien toimintojen kirjoa, esimerkiksi tuoli tuottaa (*affordoi*) istumisen funktion. Käyttäjäkokemuksen suunnittelussa affordanssiteoriaa käytetään mallina, jossa kuvaillaan käyttäjäkokemuksen kolme tasoa ja niiden affordanssit käytössä, kokemisessa, vaikutuksessa ja manipulaatiossa. Kokemuksen affordanssissa, käyttäjäkokemuksen tasolla Miksi?, kohdennetaan huomio olemisen tarkoitukseen, esim. olla lähempänä etäistä henkilöä: onnistuu kun oikea käyttömalli on tiedossa, kuten tekstiviestin kirjoittaminen. Käytön affordansissa keskitytään Miten?- kysymykseen Hassenzahlin mallissa (Hassenzahl 2013) eli kuinka asian voi tehdä. Tätä varten tarvitaan mentaalisia malleja ja käytön suunnittelua: käyttäjä voi esimerkiksi suunnitella toimintonsa sen mukaan, mitä hän aikoo puhelimella tehdä. Vaikutusten affordanssi ei erityisesti keskity mihinkään kysymykseen Hassenzahlin mallissa, vaan se liittyy seurauksen ja syyn vaikutuksen syntymiseen. Esimerkiksi kirjoittaminen tapahtuu vaikutuksen affordanssissa siten, että kirjaimia kirjoitetaan näytöllä tai siirretään valitsinta valikossa. Viimeisenä mainitaan manipulaation affordanssi. Tämä on kaikista primitiivisin affordanssi ja sen tarkoituksena on synnyttää motorinen efekti, kuten sormen siirto kosketusnäytöllä.

Affordanssit tarjoavat siis oman mallinsa käyttäjäkokemuksen suunnittelussa (Pucillo 2014). Kun kaikki affordanssin tasot on suunniteltu huolella, niistä voidaan tuottaa mahdollisia kokemuksia alimmalta tasolta alkaen. Kokemusten tuottamista ei kuitenkaan voida taata, vaan ne ovat olemisen tavoitteita Hassenzahlin mallissa (Hassenzahl 2010). Kokemuksen affordanssi on täten kaikkein korkein affordanssien hierarkiassa. Ne mahdollistavat kokemuksen syntymisen, lähtien käyttäjän tarpeiden tyydyttämisestä, hedonisia piirteitä käyttäen. Tekemisen tavoitteiden saavuttamiseksi on luotu käytön ja vaikutusten affordanssi. Käytön affordanssi mahdollistaa tekemisen tavoitteet vaikutusten affordanssien kanssa. Käytön affordanssi lähtee käyttäjän taidoista ja tuotteen pragmaattisten ominaisuuksien yhteen saattamisesta. Vaikutusten affordanssi ilmaisee manipulaatiosta johtuvan vaikutuksen syntymisen osaksi pragmaattista ominaisuutta. Esimerkiksi nappula voi syttyä kosketuksesta näytöllä. Manipulaation affordanssin ol-

lessa matalimmalla motoristen tavoitteiden kanssa, voivat nämä alimman tason manipulaation tuottamat kokemukset edelleen nostaa kokemuksia tuotteesta ylemmille tasoille. Yksinkertaisimpana pidetään käyttäjän ja tuotteen fyysistä yhteensopivuutta, kuten nappien painamisen mahdollisuutta.

Affordanssit ovat sopivia käyttäjjäkokemuksen suunnittelussa siksi, että ne ovat relationaalisia konstruktioita toimintojen mahdollisuuksille. Affordanssien avulla voidaan suunnitella kokemukselle itselleen, mutta se ei pakota käyttäjää kokemuksille. Kokemuksia ei voi itsessään kuitenkaan suunnitella, vaan niitä varten voidaan suunnitella, toisin sanoen suoda kokemuksia (Pucillo 2014).

4 Avoin mobiilikäyttäjäkokemus: Case Glacier

Tässä luvussa kuvataan Glacier-nimisen käyttöliittymän historiaa ja sidokset Nemo Mobile-jakeluun. Glacier-käyttöliittymä on tämän tutkielman käytännöllinen osuus. Se sisältää sen kehityksen tämänhetkiset komponentit ja käyttöliittymien määritelmät.

Komponentit ovat siinä järjestyksessä kuin ne ovat Michael Demetrioun Git-varastossa ja niiden kehitystilanne voi olla keskeneräinen, jolloin itse komponenttia ei ole vielä toteutettu. Lisäksi kuvataan muitakin kuin pelkkiä komponentteja, kuten esimerkiksi käyttöliittymän ilmoituksen tarjoavan ominaisuuden ja valintaikkunan tarjoavan ominaisuuden erilaisine vaihtoehtoineen.

Luvussa 4.1 on hieman taustaa Glacier-projektille ja sen projektin sisäisestä toiminnasta kertovaa materiaalia. Luvussa 4.2 esitellään kaikki Glacier-käyttöliittymän komponentit ja niiden kuvaukset sekä yhtymäkohdat käyttäjäkokemuksen suunnitteluun. Luvussa 4.3 tuodaan esille Glacier-käyttöliittymän kotinäkymä.

4.1 Taustaa

Glacier kehitettiin Nemo Mobilen käyttöliittymän moderniksi korvaajaksi. Glacier syntyi vuonna 2013 kesällä, jolloin Kenneth Kasilag kirjoitti Michael Demetrioun blogiin ideansa avoimen lähdekoodin ja vapaan ohjelmiston mukaisesta käyttöliittymästä Nemo Mobile-jakelulle. Glacierin päätarkoituksena on tuoda siis moderni käyttöliittymä osaksi Nemo Mobilea, korkealla kontrastilla ja hyvällä tekstin fontilla. (Kasilag 2013)

Glacier on suunniteltu kuten nimikin viittaa, virtaviivaiseksi ja nestemäiseksi käyttöliittymäksi. Sen lukkonäkymä on lähtökohtana kaikelle toiminnalle, ylös pyyhkäisemällä päästään etsintänäkymään, alaspäin pyyhkäisemällä avataan ohjelmien käynnistin, vasemmalle pyyhkäisemällä avataan tapahtumien näkymä ja oikealle pyyhkäisemällä avataan moniajonäkymä. Tällainen suunnittainen pyyhkäiseminen vahvistaa käyttäjän avaruudellista hahmotusta. Käyttöliittymän päätarkoitus on tarjota käyttäjille kerroksittainen navigointikokemus. (Kasilag 2013)

Etsintänäkymässä on etsimistoiminto, jolla voidaan etsiä sekä sovelluksista, että käyttäjän tietolähteistä tulevaa tietoa, esimerkiksi musiikkitiedostoista. Etsimistoiminto

käyttää nk. tracker-tietokantaa, joka on indeksoitu hakujen nopeuttamiseksi. (Kasilag 2013)

Tapahtumien näkymä näyttää kaikki tulevat tapahtumat ja uutiset, jotka käyttäjä on itse valinnut. Tällainen näkymä tarjoaa esimerkiksi sääennusteen, saapuneet tekstiviestit ja sähköpostit. (Kasilag 2013)

Ohjelmien käynnistin tarjoaa ohjelmat ruudukkona, jossa ohjelmat käynnistetään koskettamalla ohjelman kuvaketta. Ohjelmat on järjestetty eri kategorioihin, jotka käyttäjä voi tarvittaessa uudelleennimetä. (Kasilag 2013)

Projektissa vallitsee komiteallinen päätäntävalta, jossa suunnittelijat voivat esittää työnsä tulokset avoimessa ilmapiirissä. Komitea valitsee sitten esityksensä määrityksiksi, jonka kehittäjät sitten myöhemmin toteuttavat. Komiteaa johtaa Michael Demetriou, joka on myös Glacierin pääsuunnittelija.

Meritokraattinen järjestelmä on tuttu jo Mer-projektista, mutta Glacierissa kontribuointi on helpompaa ja yksinkertaisempaa, sillä jo pelkkä graafinen esitys komponentista otetaan huomioon koko yhteisön osalta. Toteutuksessa osapuolina, tämän tutkielman kirjoittajan lisäksi, ovat olleet Andrea Bernabei, Simonas Leleiva ja Lucien Xu. Tämän tutkielman kirjoittaja on ollut vastuussa osasta komponenttien luontia ja itse kotinäkymän kehittämisessä.

Komponenteista tämän tutkielman kirjoittaja on luonut nappirivin, liukuvalitsimen, valintaikkunan, tekstietiketit, vivun, tekstin syötön sekä käyttöympäristöön kotinäkymän, ilmoitukset ja alapalkin.

Nemomobilen projektissa käytetty Glacierin kotinäkymä toimii melkein kuten Kasilagin mainitsema konsepti. Kehitystyö on poikennut alkuperäisestä konseptista jo alusta lähtien, koska Michael Demetriou oli jo aikaisemmin tarjonnut yhteisölle uudenlaista lähestymistapaa esimerkiksi kotinäkymälle. Pyyhkäisyllä reunoista näytön voi lukita/avata. Lisäksi näytön reunat tarjoavat mahdollisuuden pyyhkäistä käynnissä oleva ohjelma pois näkyvistä, kun ohjelma on kaikista päälimmäisenä. (Demetriou 2014)

Glacierin toteutuskielinä on käytetty QML- ja C++-kieliä, joskin QML on vallitsevampi komponenttien osalta. QML tarjoaa helpon käyttöliittymien luomisen ilmaisun kielen. Se on osa Qt-ohjelmistoa ja se tarjoaa rajapinnat suoraan myös C++-kielelle.

4.2 Komponentit

Tässä luvussa on tarkoitus kuvata Glacierin kaikki käyttöliittymän komponentit. Jokainen komponentti on omalla sivullaan lukemisen helpottamiseksi. Komponenttien kuvauksen lisäksi peilataan niitä, soveltuvin osin, käyttäjäkokemuksen suunnittelun eri malleihin ja käytettävyyden heuristiikkoihin, pääasiassa Hassenzahlin ja Pucillon teorioihin, ja siihen, miten komponentit ottavat käyttäjäkokemuksen huomioon. Lisäksi kerrotaan hieman niiden toteutuksen haasteista matkan varrella ja toteutuksen nykyisestä tilasta. Komponentit on esitelty joko toteutuksen nykyisessä tilassa tai sitten määrittelyssä olevilla kuvilla. Kuvat on otettu ruutukaappauksina Jolla-laitteesta.

Luvussa 4.2.1 kuvataan Glacier-käyttöliittymän yksinkertaisin komponentti, nappi. Luvussa 4.2.2 kuvataan käyttöliittymälle omintakeinen nappirivi. Luvussa 4.2.3 puolestaan kuvataan valintaikkunatyypit käyttöliittymälle. Luvussa 4.2.5 kuvaillaan listanäkymän määrittely. Kuvailu ylätunnisteesta on luvussa 4.2.4. Luku 4.2.6 kuvailee puolestaan Glacier-käyttöliittymän näppäimistön ja sen ulkonäön. Kun taas luku 4.2.7 näyttää edistyksen indikaattorin toiminnan. Luvussa 4.2.8 kuvaillaan Glacier-käyttöliittymälle ominainen komponentti, valintarulla. Liukuvalitsin esitellään luvussa 4.2.9. Katkaisin puolestaan kuvaillaan luvussa 4.2.10. Viimeisenä esitellään komponenteista tekstikenttä luvussa 4.2.11.

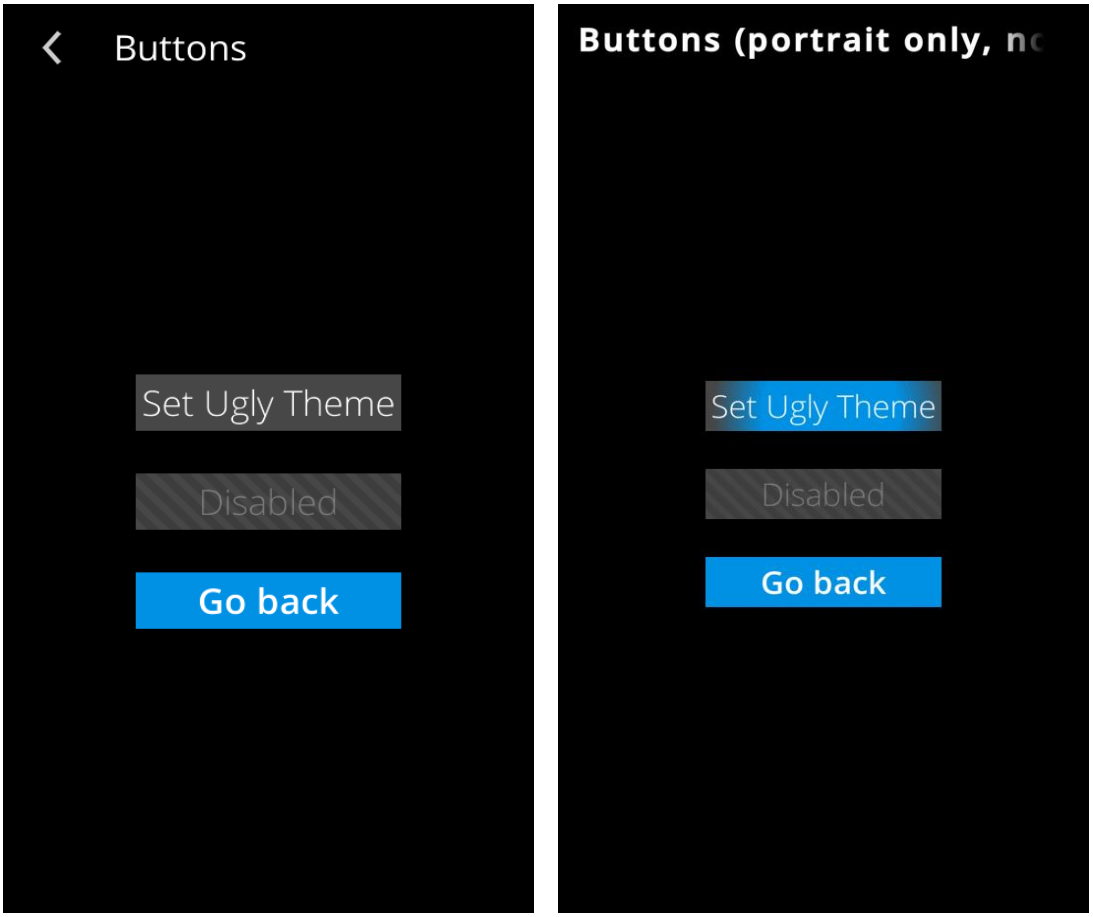
4.2.1 Nappi

Nappi on Glacier-käyttöliittymän yksinkertaisin komponentti. Sen tehtävänä on tarjota painettava kohde, joka valaistuu kun sormi on napin päällä. Tämä lisää komponentin käytettävyyttä, koska käyttäjälle näytetään hänen painamansa nappi. Kuvassa 3 näytetään perusnappi, käytöstä poistettu nappi ja tärkeä nappi. Oikeanpuoleisessa kuvassa näkyy selvästi painetun napin korostus sormen ympärillä.

Tämä efekti lisää mielestäni käyttäjän tietoisuutta tilanteesta, koska käyttäjän on vaikea olla huomaamatta painaneensa nappia kun korostus aktivoituu kosketuksesta. Se näkyy helposti sormen alta ja voi antaa täten käyttäjälle hallinnan tunteen kun hän koskettaa sitä. Käyttäjäkokemus on täten otettu huomioon kehityksen alusta lähtien, kun määrittelyssä näkyi kosketuksen aiheuttama korostuma.

Napin kolmen tason suomat kokemukset ovat haasteellisia selvittää, motoriset tavoit-

teet on ainakin otettu huomioon sekä silmälle näkyvän korostuksen, että napin painalluksen aiheuttaman välähdyksen osalta. Lisäksi motoriset tavoitteet saavutetaan myös tärkeän napin korostuksen osalta. Tekemisen tavoitteet riippuvat napin tarjoamasta toiminnallisuudesta käyttäjälle: kuvassa 3 ylimmäisen napin painallus aiheuttaa värien muuttumisen toisen teeman mukaiseksi.



Kuva 3: Erilaisia nappeja

4.2.2 Nappirivi

Kuvassa 4 esitellään kuva Glacier-käyttöliittymän nappirivistä. Komponentti on aivan erityinen Glacier-käyttöliittymälle ja sen toiminta on yksinkertainen. Käyttäjälle esitetään rivi erilaisia tekstejä, joita koskettamalla hän voi valita haluamansa. Tämä tekstin valinta näytetään käyttäjälle sinisenä laatikkona. Käyttäjä voi halutessaan raahata sinistä laatikkoa eri valintojen päälle ja näin ollen valita haluamansa option dynaamisesti.

Nappirivin toteutuksessa jouduttiin turvautumaan määrittelyn tarkkoihin ohjeisiin ja

QML-kielen erilaisiin toimintoihin, kuten mallien käyttöön erilaisten listojen luonnissa. Malli on normaali listamalli, jonka jokainen olio tuottaa nappiriviin oman nappinsa. Toteutuksen käyttöä kuvataan listauksessa 1 olevalla pätkällä QML-kieltä. Lopullisen toteutuksen mallia ei tähän komponenttiin ole vielä olemassa, koska komponentin kehitys on vielä kesken. Määrittely tarjoaa olennaiset tiedot komponentin toteutukselle, kuten sen tarvitsemat animaatiot ja erikoiset korostukset.

Kuvassa 5 näytetään toteutuksen nykyinen tilanne: siinä on valittuna vain yksi nappi nappirivistä. Lisäksi nappirivi on laitettu tässä kuvassa ainoastaan vaakasuoraan, jotta se mahtuisi komponenttien esittelyn sivulle. Toisena elementtinä kuvassa näytetään valittu nappi tekstinä. Alin elementti on käytöstä poistettu nappirivi.

Toteutuksen haasteena ovat juurikin nämä edellä mainitut hienot animaatiot, joita ei lopulliseen toteutukseen ole tehty, koska toteutuksen alussa ei ollut vielä riittävää tietotaitoa animaatioiden tekemiseen. Toteutus on lisäksi yksinkertaisin mahdollinen, sillä määrittelyssä mainitaan joko yksinkertainen toteutus tai monen valinnan toteutus. Valitsin tehtäväkseni toteuttaa vain ja ainoastaan monen valinnan toteutuksen.

Käyttäjäkokemuksen tasoilla erityisesti motoristen tavoitteiden osalta nappirivi tarjoaa paljon, koska nappirivin erityiset animaatiot auttaisivat käyttäjää näkemään tekemänsä valinnat selkeästi. Nykyisellä toteutuksella ei kuitenkaan animaatioita ole tehty, joten niiden osalta kokemus voi olla hieman köyhä.

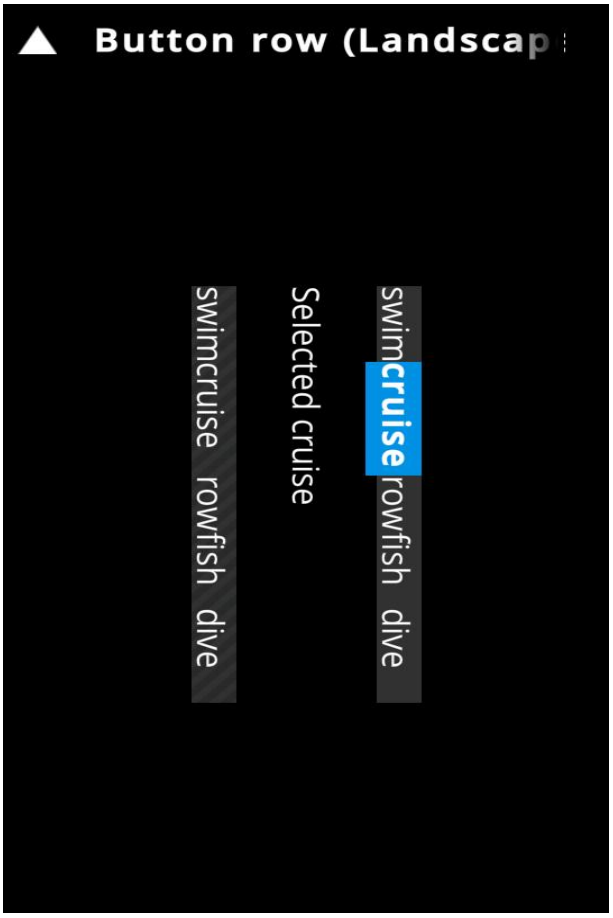
Listaus 1: Esimerkki nappirivin käytöstä QML-kielellä

```
1  ButtonRow {
2      id: row
3      model: ListModel {
4          ListElement {
5              name: "swim"
6          }
7          ListElement {
8              name: "cruise"
9          }
10         ListElement {
11             name: "row"
12         }
13         ListElement {
14             name: "fish"
15         }
16         ListElement {
17             name: "dive"
18         }
19     }
20 }
```

Listauksessa 1 kerrotaan nappirivin toteutuksesta: se käyttää tietomallinaan tavallista listamallia (rivi 3). Sen jälkeen jokainen sen listan elementti määritellään erikseen ListElement-oliolla, jossa name-attribuutti muutetaan jokaisessa oliossa. Listamalli saa nappirivin kasvamaan yhdellä napilla aina kun siihen lisätään olio. Nappirivi on kiinteän pituinen, joten montaa nappia siihen ei saa yhtä aikaa.

Nappirivin määritelmässä (kuva 4) on edellä kuvatun lisäksi näytetty mahdollisuus raahata valittua nappia sormella, helpottaen näin valintojen tekemistä. Raahaus on katkaiseva, joka tarkoittaa sitä, että raahaus sijoittuu X-akselilla olevaan komponenttiin ja raahausliike katkaistaan aina alla olevan komponentin valintaan. Kuvassa 5 näytetään nappirivin nykyinen toteutus Jolla-laitteessa, demoympäristössä.

Kuva 4: Nappirivin määritelmä

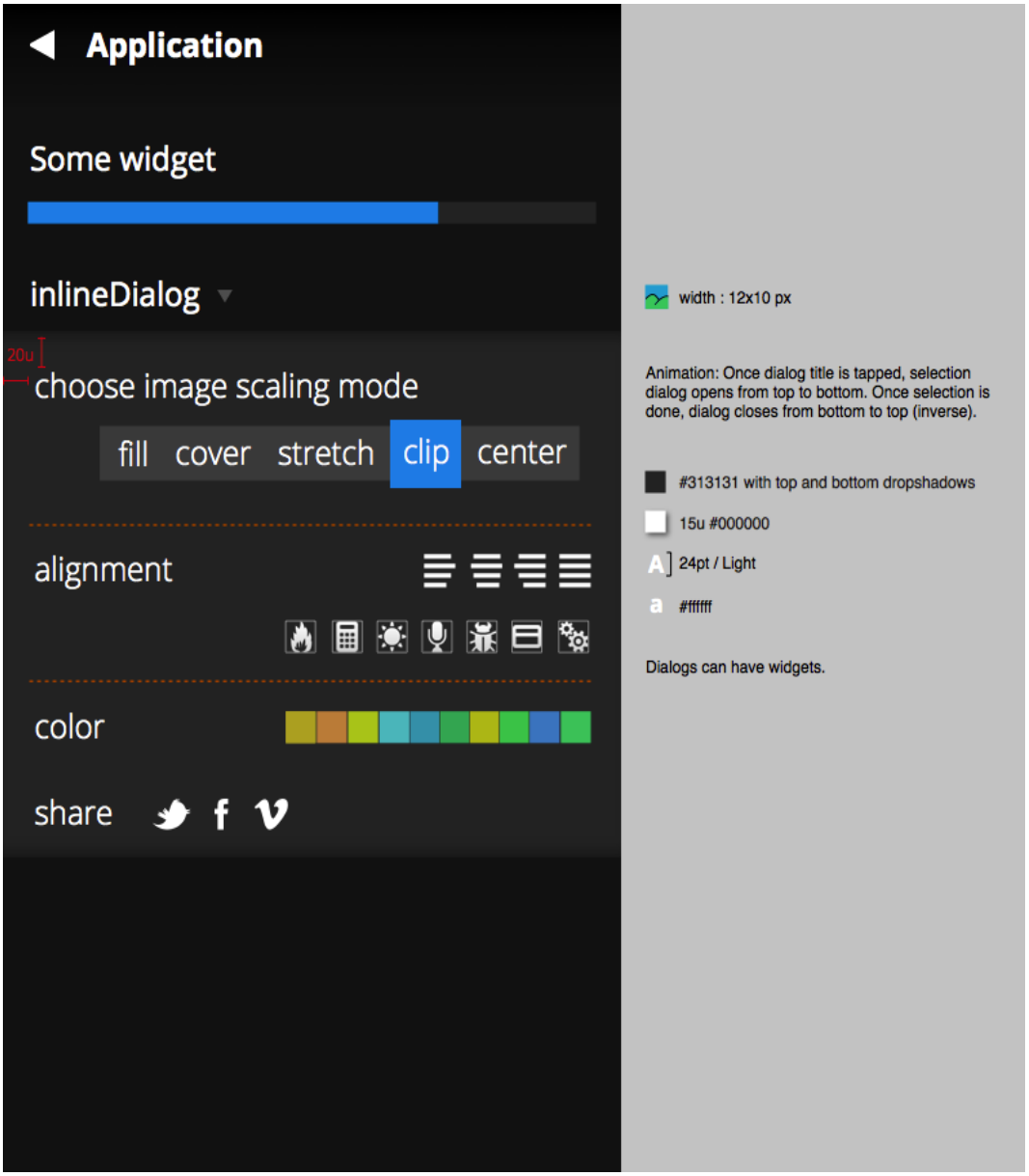


Kuva 5: Nappirivi Jolla-laitteessa

4.2.3 Valintaikkunat

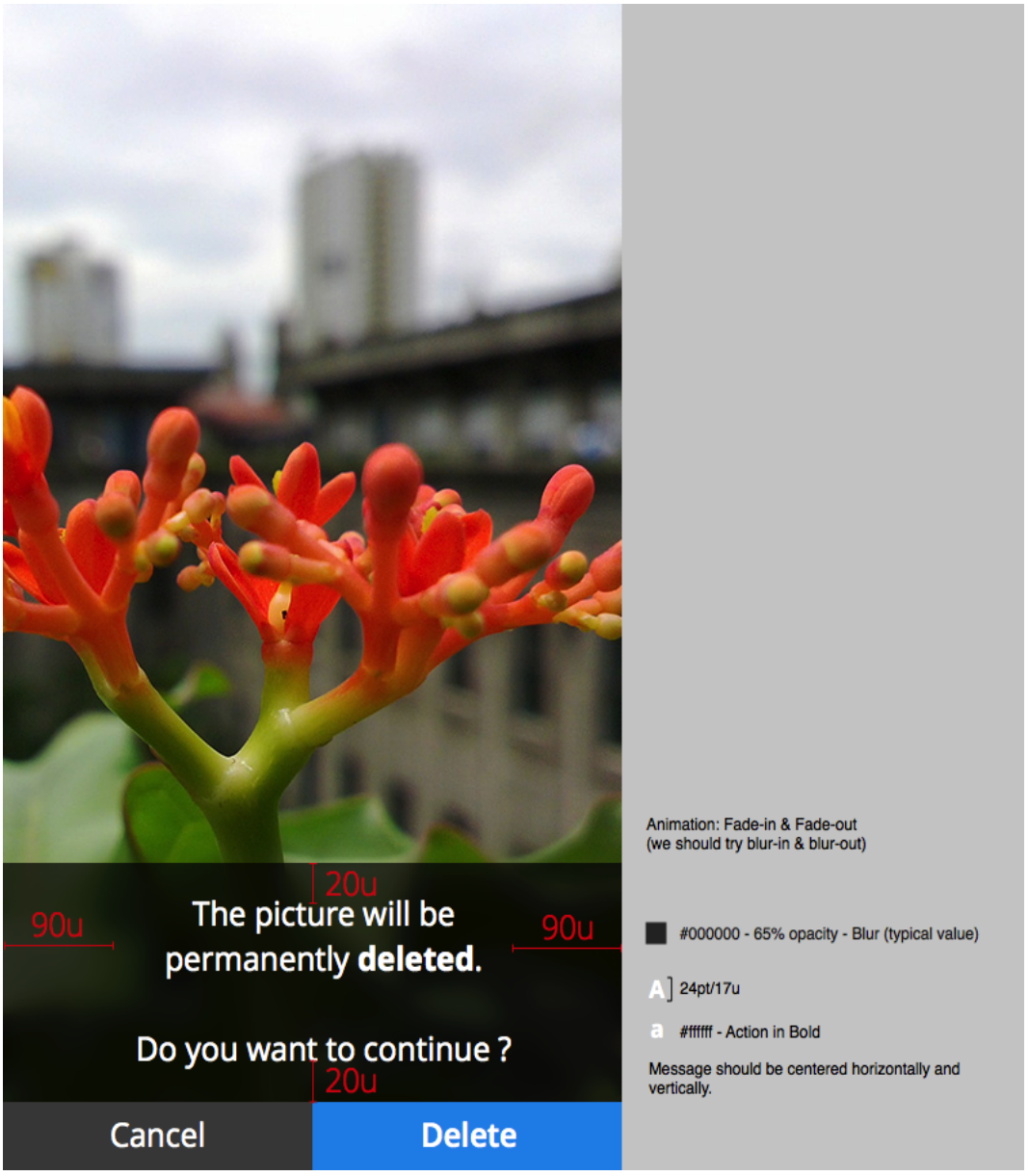
Kuvassa 6 näytetään käyttöliittymän sisälle sijoittuva valintaikkuna, jossa on erilaisia komponentteja valintojen tekemiselle. Valintaikkuna aukeaa pienestä nuolesta sen etiketin oikealla puolella ja täten näyttää käyttäjälle piilossa olevia valintoja, joita hän voi tehdä.

Tämän valintaikkunan tarkoitus on näyttää käyttäjälle ensiksi vain hänen tarvitsemansa käyttöliittymän komponentit ja sijoitettua valintaikkunaa koskettamalla hän saa näkyviin myös muita komponentteja, jotka liittyvät hänen käyttöönsä. Tämä voidaan toteuttaa esimerkiksi kuvien muokkauksessa, kuten kuvassa 6 on annettu ymmärtää. Ensiksi näytetään toiminnon peruskomponentit ja sitten valintaikkunassa kehittyneemmän tason komponentit.



Kuva 6: Sijoittuva valintaikkuna

Kuvassa 7 näytetään Glacier-käyttöliittymän normaali valintaikkuna, jossa käyttäjälle näytetään kahden valinnan valintaikkuna. Valintaikkuna on sijoitettu näkymän pohjalle ja on kaiken muun yläpuolella komponenttien suhteen.



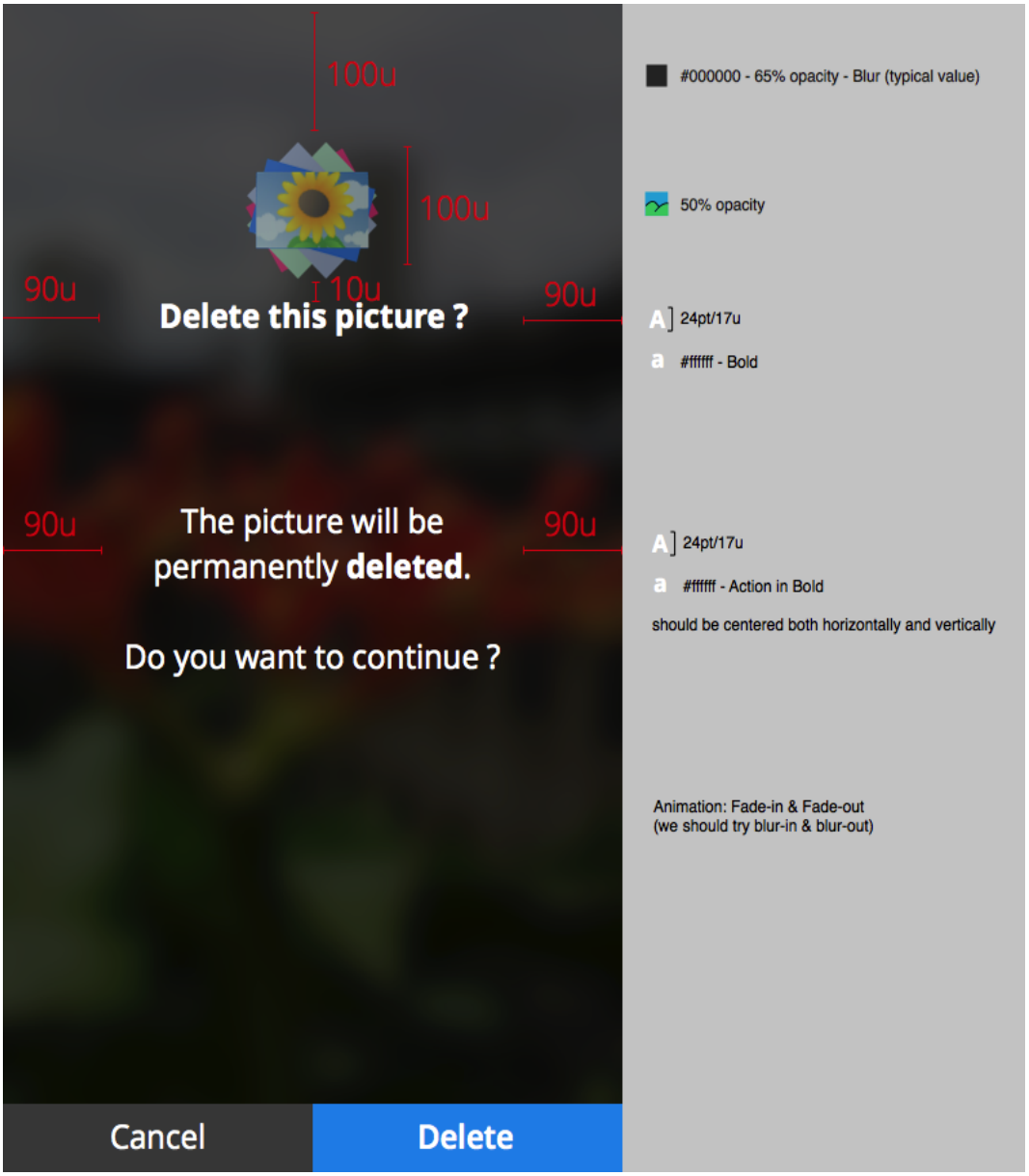
Kuva 7: Normaali valintaikkuna

Kuvassa 8 kuvaillaan koko ruudun peittävä valintaikkuna. Tämä näyttää käyttäjälle selkeästi, että hänen on nyt tehtävä jotain tärkeää, koska viesti vie koko ruudun verran tilaa. Sen näkymässä on valittava kahdesta toiminnosta yksi, joka sitten saa aikaan sen, että näkymä poistuu käyttäjän näköpiiristä.

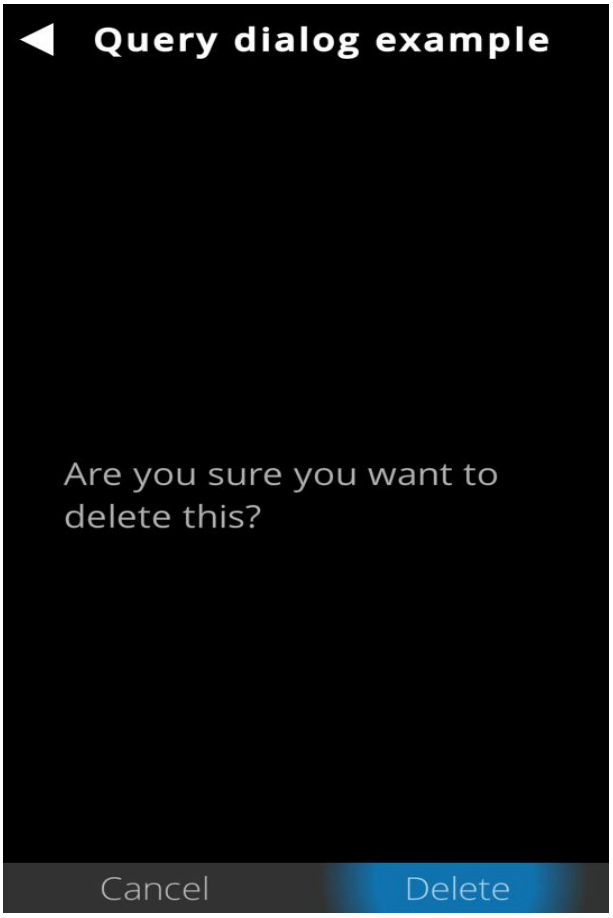
Kokemuksen kannalta tärkeät valintaikkunat ovat normaali ja koko ruudun peittävä valintaikkuna. Käyttäjälle näytetään näissä valintaikkunoissa aina jotain tärkeää hänen toimintojensa jatkumisen puolesta. Valintojen tekemisen helpottaminen on valintaikkunoiden päätavoite, olivat ne valinnat sitten kuvan poistamista tai kontaktin poistamista

puhelimesta. Kyseessä on aina tekemisen tavoite ja sen tavoitteet toteutuvat motorisesti koskettamalla valintaikkunan nappeja. Nämä tekemisen tavoitteet ovat aina tärkeitä ylemmällä tasolla käyttäjäkokemusta, varsinkin olemisen tavoitteisiin kohdistuen. Nämä olemisen tavoitteet voivat liittyä vaikkapa käyttäjän kesälomalla otettuihin kuviin, joita sitten varmistellen poistetaan puhelimesta kun muisti siinä alkaa täyttyä.

Esimerkkinä valintaikkunasta valittiin komponenttigalleriasta kokonaisen valintaikkunan tämänhetkisen toteutus Jolla-laitteessa. Kuva tästä valintaikkunasta on kuvassa 9. Kuvassa näytetään kahden valinnan napit ja valitun toiminnon teksti. Lisäksi korostin valintaa koskettamalla toisen valinnan nappia sormellani. Valintaikkuna katoaa kun valitaan jompikumpi nappi alareunasta koskettamalla sitä ja jäljelle jää teksti siitä, kumpi valinta tehtiin.



Kuva 8: Kokonainen valintaikkuna



Kuva 9: Kokonainen valintaikkuna Jolla-laitteessa

4.2.4 Ylätunniste

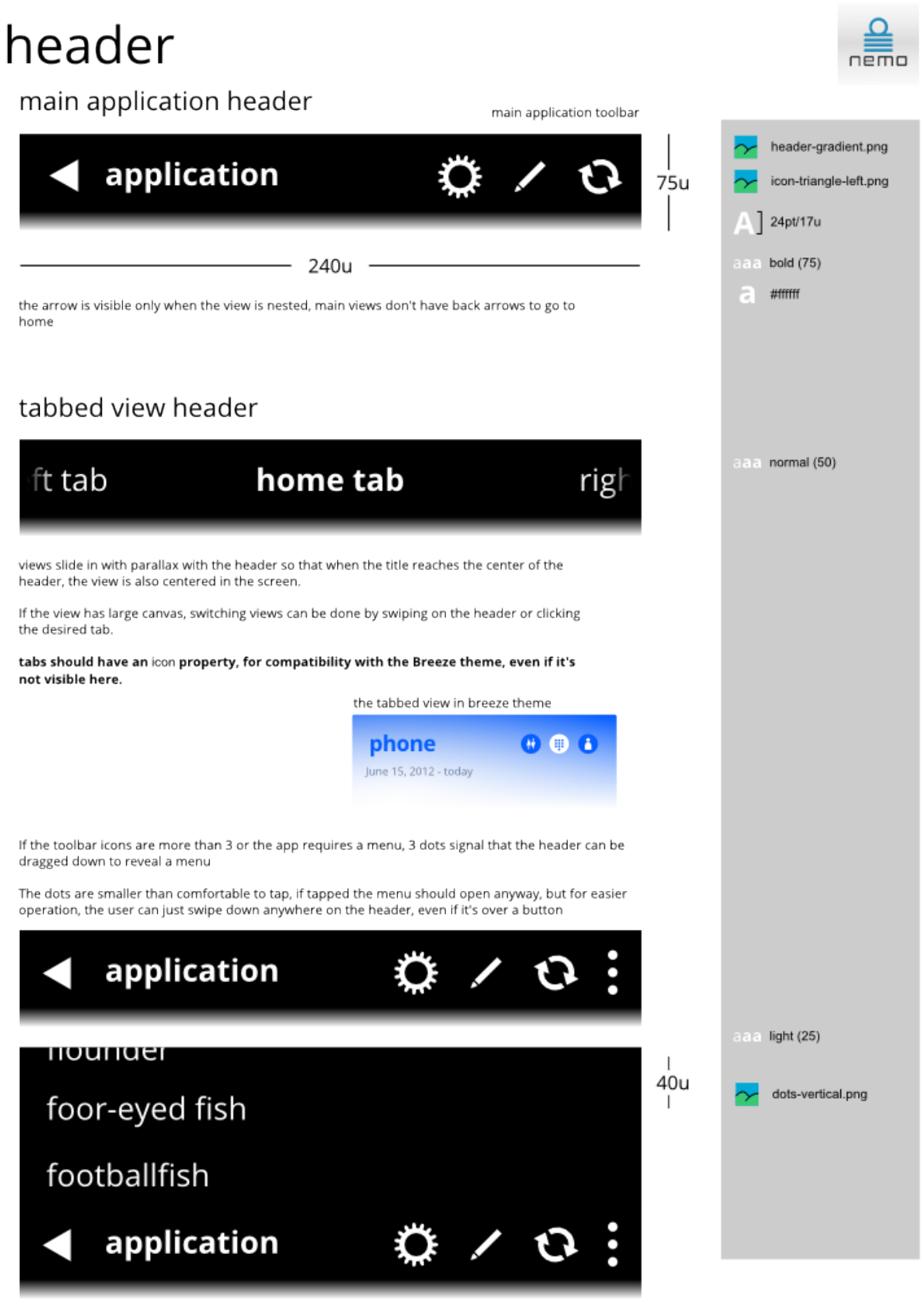
Kuvassa 10 näytetään Glacier-käyttöliittymän ikkunoiden ylätunnisteen erilaisia ominaisuuksia. Tähän ylätunnisteeseen on pakattu sisään muutama mielenkiintoinen ominaisuus, kuten se miten esimerkiksi käynnissä olevan ohjelmiston asetuksia voi muokata koskettamalla rattaan kuviota ylätunnisteessa.

Ylätunnisteen tarkoitus on myös näyttää käyttäjälle sen hetken ohjelmiston käyttämät välilehdet ja myös selkeästi se, missä välilehdellä käyttäjä tällä hetkellä on. Ylätunniste lisäksi paljastaa ohjelmiston työkaluvetimen koskettamalla kolmea pistettä. Lisäksi ylätunnistetta vetämällä voidaan työkaluvedintä laajentaa tarvittaessa askeleittain. Tämä antaa sovellukselle itselleen lisätilaa näyttää toimintoja.

Ylätunnisteen komponentteja ovat:

1. Nuoli vasemmalle, tämä tulee näkyviin vain jos näkymä on sisäkkäinen, päänäkymissä nuoli ei tule näkyville
2. Teksti, joka voi indikoida joko näytettyä toimintoa tai käynnissä olevan sovelluksen nimeä
3. Valinnainen määrä erilaisia toimintoja, jotka määritellään erikseen kuvakkein. Näiden kuvakkeiden tulisi mahdollisimman hyvin indikoida toimintoaansa.
4. Kolme pistettä. Tämän toiminnon koskettaminen avaa ylätunnisteen työkaluvetimen kokonaan yhdellä kosketuksella.

Ylätunnisteen toteutuksen teki Andrea Bernabei ja kesti hänenkin tasoisella henkilöllä useita kuukausia saada se mukautumaan määrittelyihin.



Kuva 10: Ylätunnisteen määrittely

4.2.5 Listanäkymä

Kuvassa 11 näytetään Glacier-käyttöliittymälle ominainen listanäkymä erilaisilla valinnoilla. Tutkielmassa näytetään ainoastaan keskikokoinen listanäkymä, sillä pieni ja suuri listanäkymä ovat periaatteessa samanlaisia.

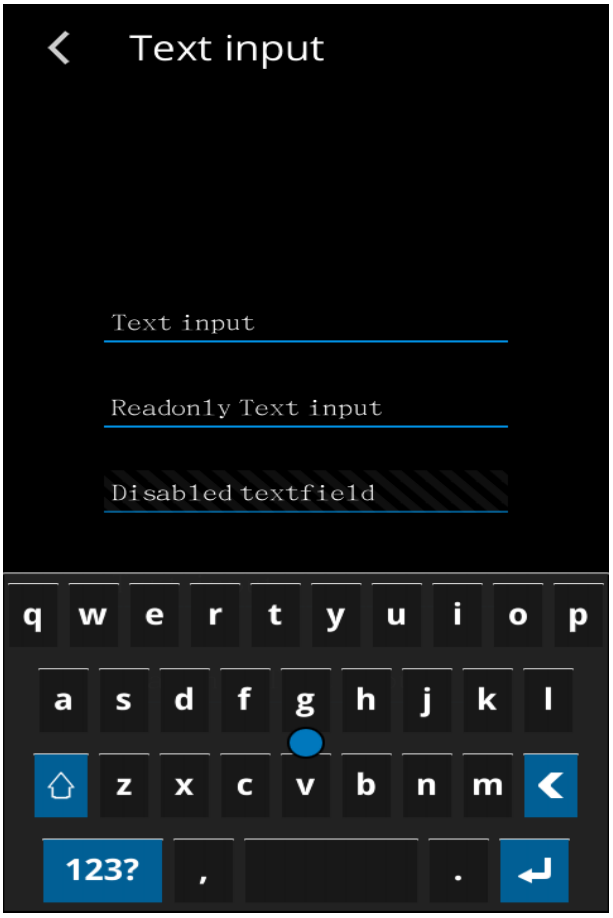
Listanäkymässä sen alkiot alkavat aina vasemmalle sijoittuvalla suurella harmaalla ympyrällä. Tämän ympyrän tarkoitus on auttaa käyttäjää valitsemaan juuri oikea alkio listanäkymästä.

Kuva 11: Listanäkymä

4.2.6 Näppäimistö

Glacier-käyttöliittymässä myös käyttäjän käyttämä näppäimistö on määritelty tarkasti. Sen on tarkoitus olla mahdollisimman käytännöllinen ja tarjoaa sekä pysty- että vaakasuuntaisen määritelmän näppäimistölle.

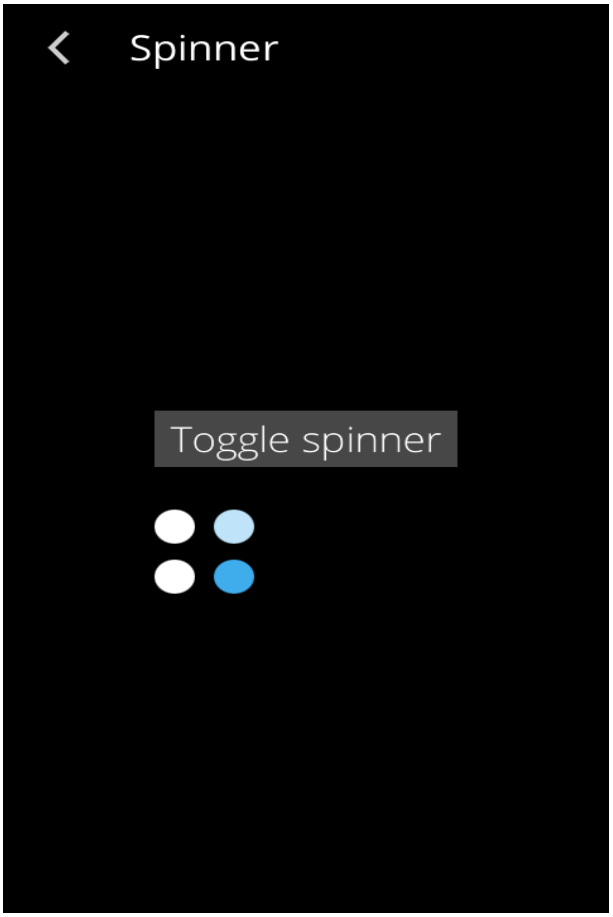
Kuvassa 12 näytetään N9-laitteessa testattua näppäimistöä, joka on jo yhteisön hyväksymä. Näppäimistön asettelu on vielä kesken, sillä vain englannin kieli on vakiona näppäimistön asettelussa.



Kuva 12: Näppäimistö

4.2.7 Edistyksen indikaattori

Edistyksen indikaattori on sellainen ilmaisin (kuva 13), jota käytetään esimerkiksi Glacier-käyttöliittymän kotinäkymässä ohjelmien suorituksen aloituksessa. Se on periaatteessa yksinkertainen animaatio, jossa ympyrät muuttavat hitaasti väriä, valkoisesta siniseksi ja takaisin valkoiseksi, alkaen vasemmalta ylhäältä, siirtyen oikealle ylös, josta sitten jatketaan oikealle alas ja lopuksi alas vasemmalle. Tämän jälkeen siirrytään taas vasemalle ylös ja animaatio jatkuu myötäpäivään edellä kuvatulla tavalla. Animaatio kestää äärettömän pitkään tai kunnes ohjelmoija lopettaa sen erikseen.



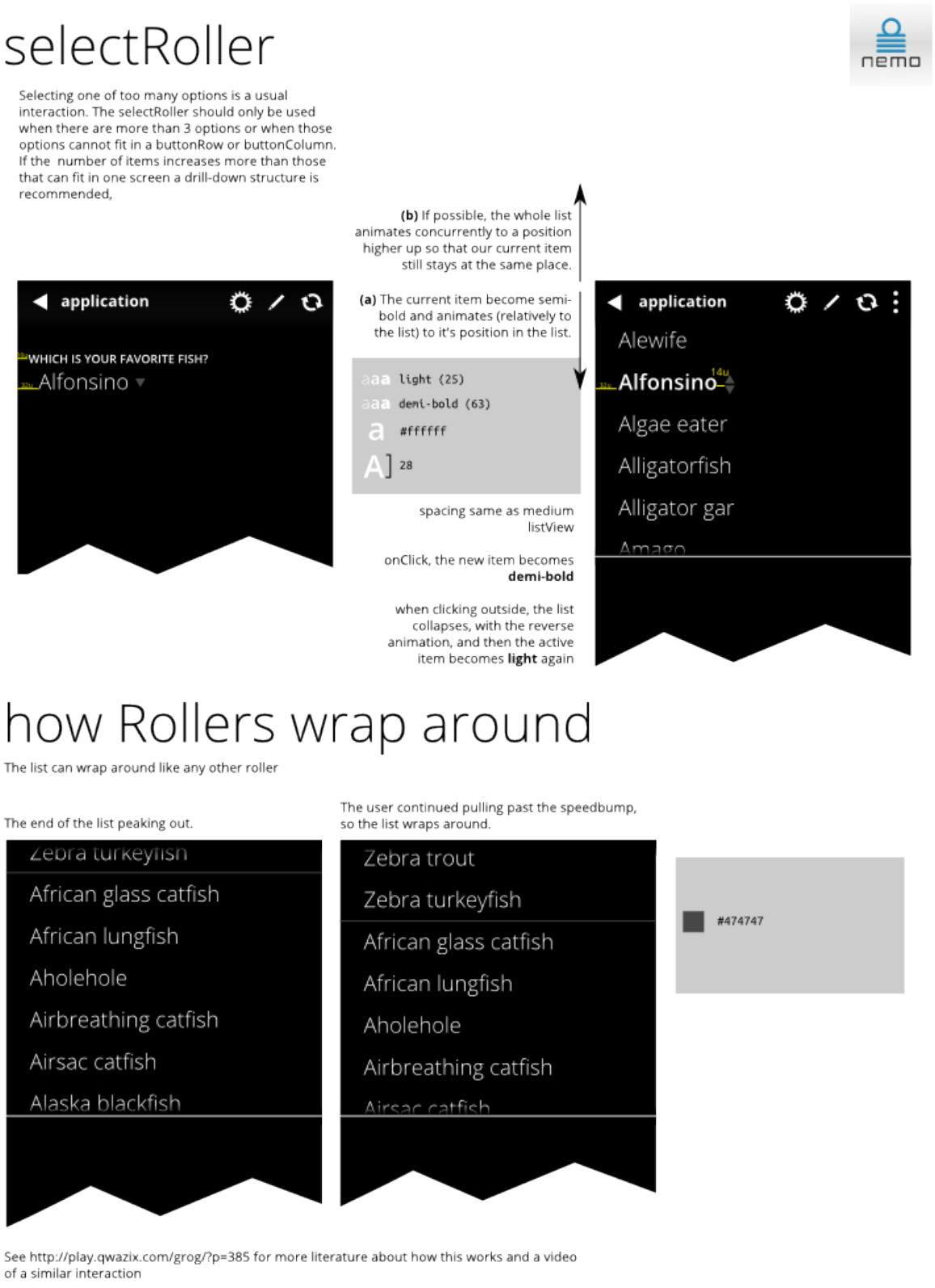
Kuva 13: Edistyksen indikaattori

4.2.8 Valintarulla

Valintarulla (kuva 14) on Glacier-käyttöliittymän korvaaja pudotusvalikolle siinä tapauksessa jos nappiriville (luku 4.2.2) ei yksinkertaisesti mahdu riittävästi elementtejä tai ne ovat liian suuria mahtuakseen sopivasti nappirivin elementteihin. Tätä varten Michael Demetriou kehitti valintarullan (eng. select roller). Tässä rullassa on monia eri valintoja kuten perinteisessä pudotusvalikossa, mutta sen sisältö "rullaa"näkyville myös listan loppuupäästä. Valintarullaa voidaan kuvitella käsillä pyöriteltävänä paperisylinterinä, jossa paperin loppuessa sylinteri jatkuu ilman katkosta näyttämään paperia.

Valintarullassa aloitetaan näyttämällä valittu elementti suuresta listasta. Sitä koskettamalla avataan suurempi lista valinnoista. Tähän siirrytään erityisellä animaatiolla, jossa lista vieritetään valitun elementin kohdalle listassa ja lisäksi valitun elementin teksti lihavoidaan.

Valintarullaa ei ole vielä sen haastavuuden takia onnistuttu toteuttamaan: kuvassa näkyy ainoastaan sen määritelmän pohjalta tehdyt kuvaukset.

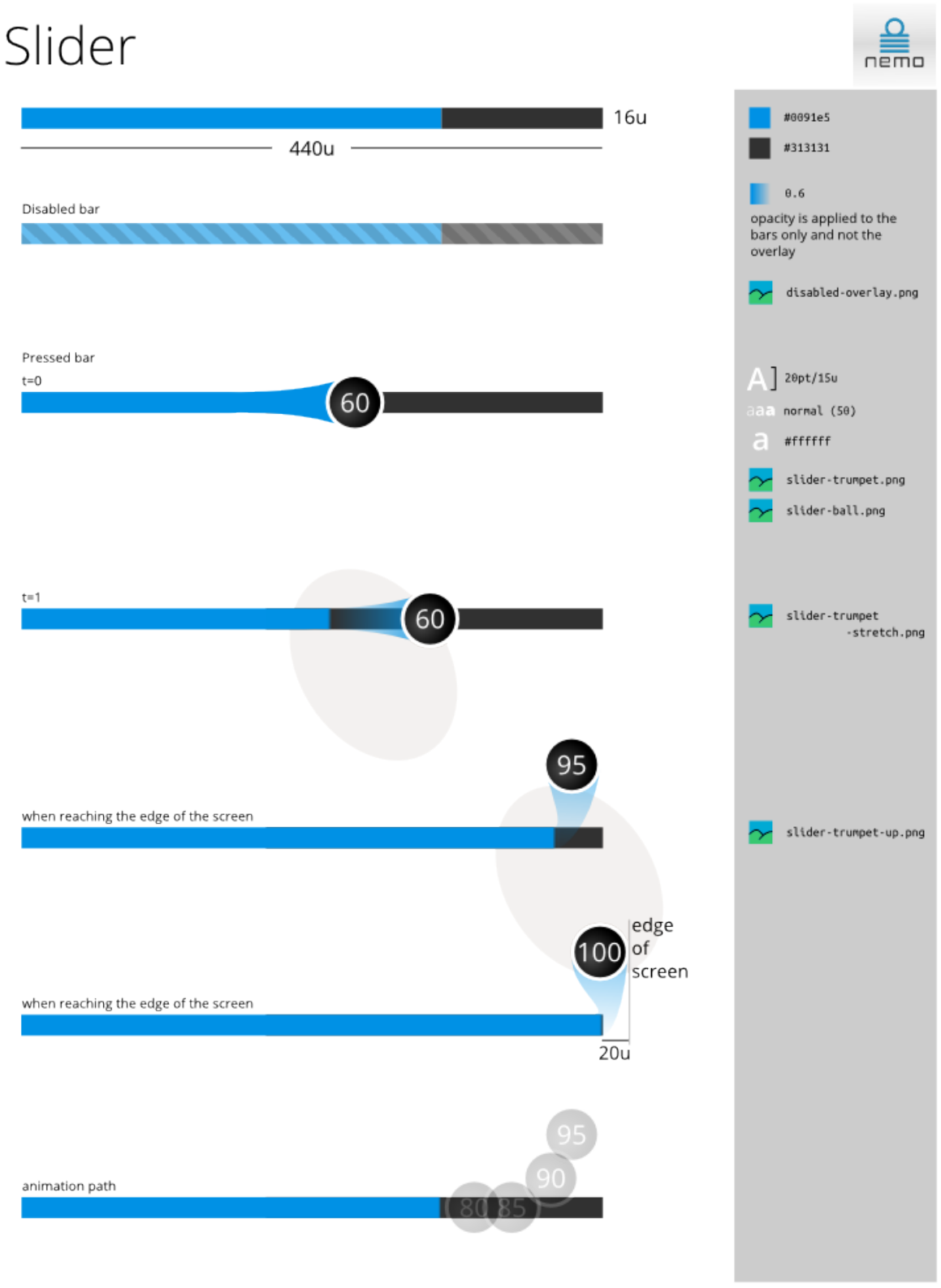


Kuva 14: Valintarullan määritelmä

4.2.9 Liukuvalitsin

Liukuvalitsin on eräs Glacier-käyttöliittymän perinteisimmistä komponenteista (kuva 15). Liukuvalitsin on onnistuttu määrittelemään haastavasti, sillä sen erityiset animaatiot käyttäjän liikuttaessa sitä kohti oikeaa reunaa ovat vaikeita saada aikaan. Kun valitsinta koskettaa sormella, sen hetkinen valitsimen arvo liukuu sormen oikealle puolelle, ikään kuin sormen alta pois. Kun lähestytään valitsimen oikeaa reunaa, valitsimen arvon näyttävä ympyrä siirtyy vasemmalle ja ylöspäin sormen yläpuolelle, kun oikea reuna on saavutettu.

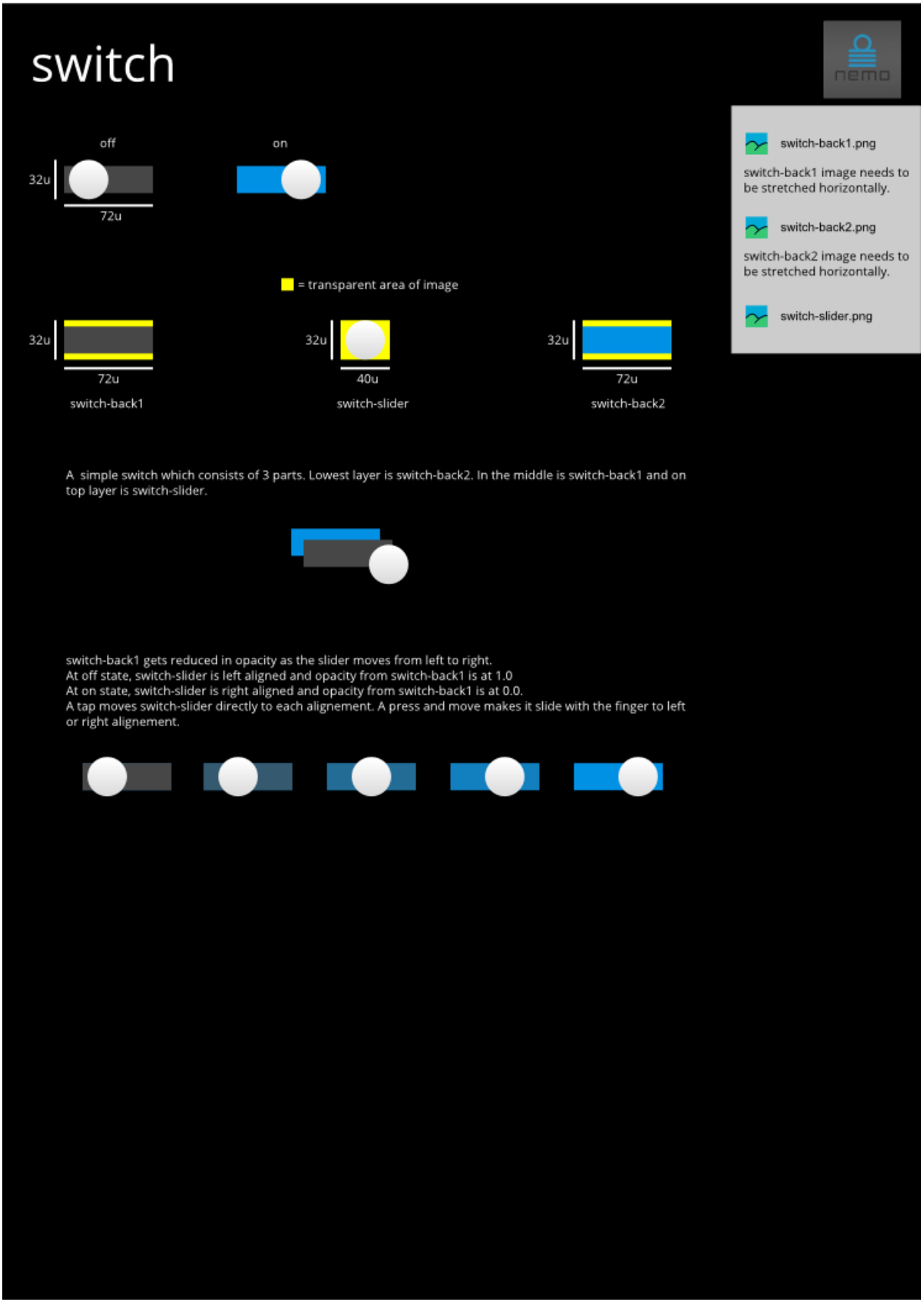
Liukuvalitsin osoittautui haastavaksi toteuttaa, määrittelyssä mainittiin painikkeen animaatio sormen alta ylös kun valitsinta liikutti tarpeeksi oikealle reunalle. Animaatiota ei tutkielman kirjoittaja saanut koskaan tehtyä, mutta sen tekemiselle on luotu pohja nykyisessä toteutuksessa. Liukuvalitsimen piirtoteknologia käyttää nyt QML kielen Canvas-oliota ja sen 2-ulotteisia piirto-ominaisuuksia.



Kuva 15: Liukuvalitsimen määritelmä

4.2.10 Katkaisin

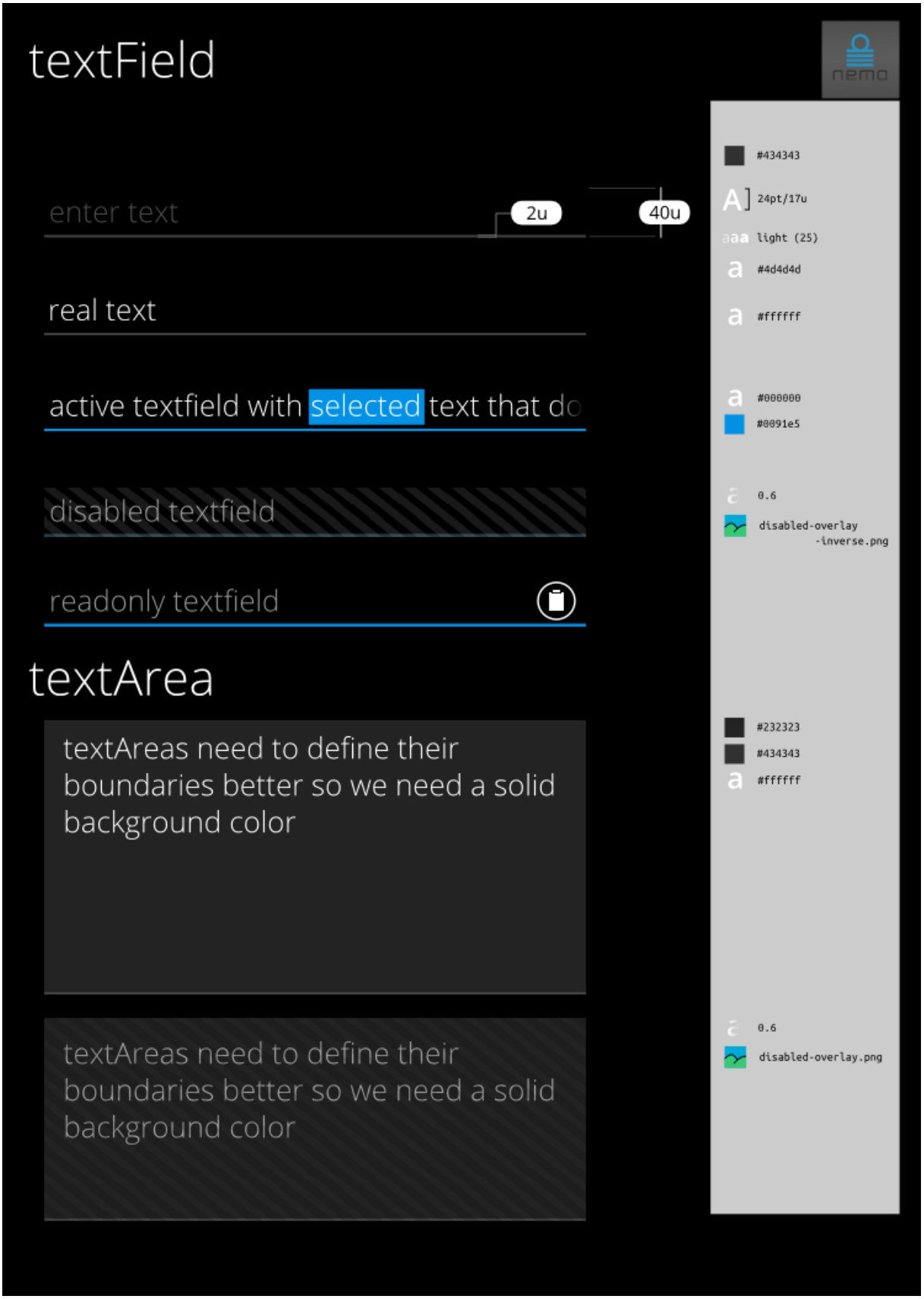
Katkaisin on tehty muistuttamaan valintaruudun ominaisuuden omaavaa komponenttia (kuva 16). Katkaisimen vasemmalla puolella tulisi olla teksti, jossa kuvataan esimerkiksi sitä, mitä ominaisuutta ollaan poistamassa käytöstä/ottamassa käyttöön. Katkaisimella on erityinen värianimaatio, joka käynnistyy katkaisinta kosketettaessa. Se aktivoituu aina vastakkaiseen väriin valintaa tehdessä tai valintaa poistaessa.



Kuva 16: Katkaisin

4.2.11 Tekstikenttä

Tekstikenttä (kuva 17) on kaikkein klassisin komponentti Glacier-käyttöliittymässä. Sen tarkoitus on vastaanottaa kirjoittajan syöttämä teksti ja näyttää se. Lisäksi kirjoittaja voi valita haluamansa palasen tekstistä itse. Lisäksi vain-luku tekstikentässä on erityinen ikoni ilmaisemaan tekstikentässä olevan tekstin kopiointia leikepöydälle.



Kuva 17: Tekstikentän määritelmä

4.3 Kotinäkymä

Tässä luvussa esitellään Glacier-käyttöliittymän kotinäkymä, kuvauksineen sen toiminnoista ja kuvineen.

Kaikkien edellä kuvailtujen komponenttien lisäksi Glacierin on tarkoitus tarjota käyttäjäkokemuksensa lähtökohdat kotinäkymästä. Tämä kotinäkymä toimii kaikilla Nemo Mobileen pohjautuvilla ohjelmistoilla, kuten esimerkiksi SailfishOS-käyttöjärjestelmällä. Kuvailu on tehty tutkielman kirjoittajan havainnoista.

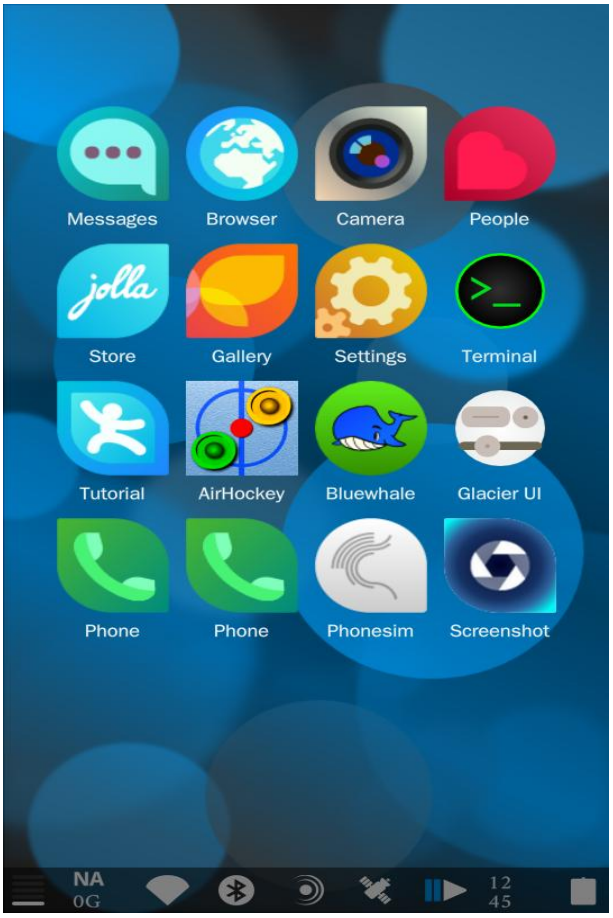
Kotinäkymälle olevat määritelmät ovat vielä kesken, mutta se ei ole hidastanut sen kehittämistä. Lähdekoodit olivat jo valmiina edellisestä kotinäkymästä nimeltä "Colorful Home"ja aloitin kehitystyöni niiden pohjalta. Kotinäkymä on toteutus lipstick-nimiselle kirjastolle, jonka tarkoitus on tarjota ohjelmoinnin rajapinnat kotinäkymille. Toinen toteutettu kotinäkymä löytyy esimerkiksi Jollan SailfishOS-käyttöjärjestelmästä.

Kotinäkymän toiminta on samankaltainen kuin Harmattan-käyttöjärjestelmässä. Siinä on kolme sivua, joista ensimmäisenä on ohjelmien käynnistyksen näkymä, toisena avoimena olevat ohjelmat ja kolmantena erilaisten tapahtumien näkymä. (Demetriou 2014b)

Ohjelmien käynnistimessä esitetään asennetut ohjelmistot ristikossa kuvakkeina, joita koskettamalla voi käynnistää halutun ohjelman. Tässä näkymässä mahdollistetaan myös erilaisten pienoisohjelmien käyttö. Ohjelmien käynnistin on näkyvissä kuvassa 18 (Demetriou 2014b)

Avoimien ohjelmien näkymässä esitellään kaikki avoimena olevat ohjelmat kahden sarakkeen ristikossa, joiden elementteinä ovat ohjelmien nykyisen tilan kuvat. Lisäksi ne vaihtuvat ohjelman suorituksen muuttaessa sen käyttöliittymää. Avointen ohjelmien näkymä mahdollistaa myös suosittujen ohjelmien kiinnittämisen näkymään pysyvästi, myös itse laitteen uudelleen käynnistyksen yhteydessä. Avoimien ohjelmien näkymä on näkyvillä kuvassa 19. (Demetriou 2014b)

Tapahtumien näkymässä käyttäjä saa näkyviin erilaisia tapahtumia, kuten esimerkiksi puhelinsoitot joihin ei ole vastattu, uudet sähköpostit ja vaikkapa Twitterin viestit. Tapahtumat ryhmitellään aina sen kontekstin mukaan, esimerkiksi kontaktin tapahtumat ryhmitellään aina kontaktin kuvan mukaan ja käyttöjärjestelmän päivitykset aina Ne-

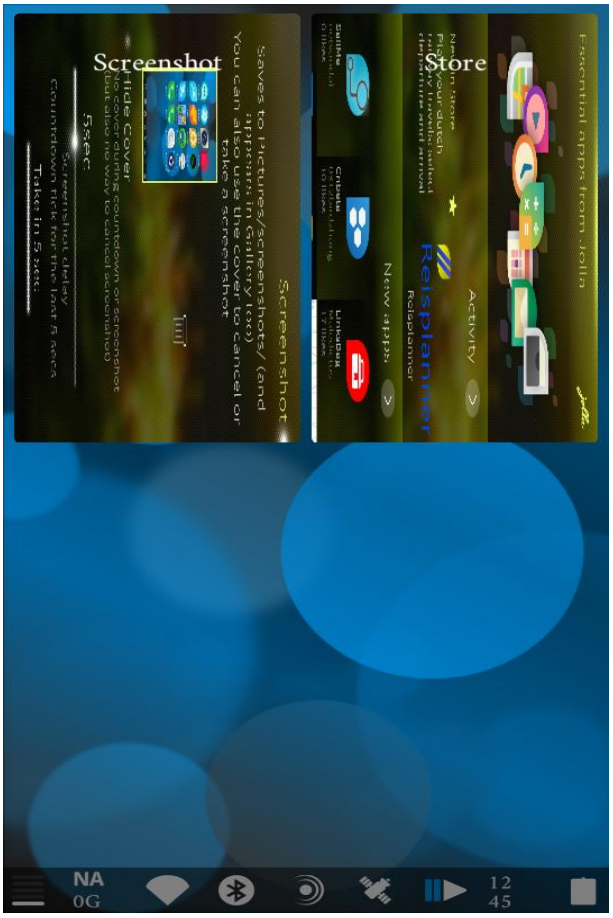


Kuva 18: Ohjelmien käynnistin kotinäkymässä

mo Mobilen kuvakkeen alle. Tapahtumien näkymä ja siinä näkyvä vastaamaton puhelu on kuvassa 20. (Demetriou 2014b)

Näkymistä muokattavissa ovat ohjelmien käynnistin ja avoimien ohjelmien näkymä. Muokattavuus aktivoidaan näissä näkymissä joko koskettamalla kuvaketta pitkään tai kuten avointen ohjelmien näkymässä, koskettamalla mistä tahansa alueelta pitkään. Ohjelmien käynnistimessä muokattavuus tarkoittaa joko kuvakkeiden siirtämistä, kuvakkeen ilmaiseman ohjelmiston poistamista tai kuvakkeen poistamista. Avointen ohjelmien näkymässä voidaan muokkaustilassa sulkea avoimena olevan ohjelma tai ne kaikki, myös ohjelman kiinnittäminen ilmaistaan kuvalla. (Demetriou 2014b)

Nykyisessä toteutuksessa on otettu huomioon yllämainitut seikat niin tarkasti kuin se tutkielman tekohetkellä on ollut mahdollista. Kotinäkymästä näkyvissä kuvissa on otettu huomioon kotinäkymän ohjelmiston versio 0.24. Se on saatavilla asennettavaksi esimerkiksi Jolla-laitteeseen osoitteesta `https://openrepos.net/content/locusf/glacier-homescreen`. Ohjelmiston versio voi muuttua vielä tutkiel-

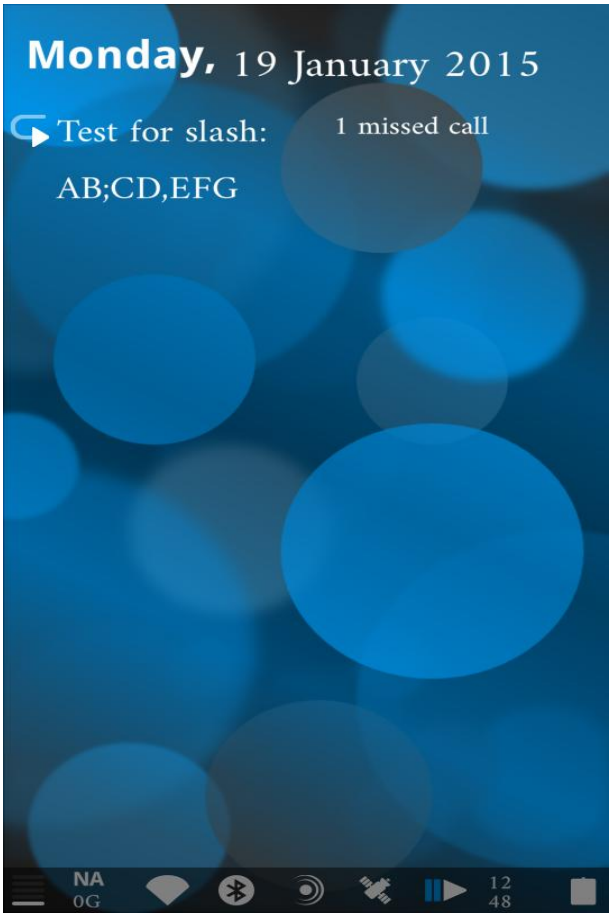


Kuva 19: Avoimien ohjelmien näkymä

man julkaisun jälkeenkin.

Kotinäkymän nykyisessä toteutuksessa on alareunassa oleva tilailmaisin, joka on toiminnaltaan erityinen Glacier-kotinäkymälle. Toiminta perustuu erityisten paneelien käyttöön, jotka aukeavat kun alareunan kuvakkeita kosketetaan tilailmaisimessa. Kosketus avaa paneelin alareunan tilailmaisimen yläpuolelle ja sen sisälle tulee tietoa sen mukaan, mitä kuvaketta kosketettiin. Tällaisessa paneelissa voi olla tietoa esimerkiksi akun nykyisestä tilasta prosentteina tai lähettyvillä olevista skannatuista WiFi-verkoista. Paneeleissa on lisäksi yhteisenä tekijänä nappi, josta paneeli on helppo sulkea.

Nykyinen toteutus tarjoaa lisäksi muutoksia avointen ohjelmien näkymään. Tässä näkymässä nähdään suorana ohjelman nykyinen suoritustilanne tai esimerkiksi videon soittamisen eteneminen. Näkymää voi lisäksi vierittää ylös ja alas, jos ohjelmia on avoinna useampia kuin kaksi ohjelmaa. Kun avointen ohjelmien näkymää kosketetaan pitkän aikaa (esimerkiksi 2 sekuntia), siihen lisätään jokaisen suoritettavan ohjelman



Kuva 20: Tapahtumien näkymä

kohdalle erityinen sulkemistoiminto, joka esitellään sinisellä pallolla, jonka sisällä on X-kirjain. Tämän lisäksi näkymään lisätään tummalle pohjalle kaksi nappia. Sulkemistoiminto voidaan lopettaa joko painamalla nappia "Done"tai sitten kaikki käynnissä olevat ohjelmat voidaan lopettaa painamalla nappia "Close all". Kun sulkemistoiminto on suoritettu, napit häviävät näkyvistä.

Yllämainittua universaalia muokkaustoimintoa ei ole toteutettu ollenkaan, vaan kuvakkeiden siirto onnistuu pelkästään ohjelmien käynnistyksen näkymässä. Avointen ohjelmien näkymässä pitkä kosketus aiheuttaa ylläolevat toimenpiteet. Muokkaustoiminto on tehtävissä yksinkertaisesti ottamalla huomioon pitkä kosketus molemmissa näkymissä.

Ohjelmien käynnistyksen näkymä tarjoaa ylös ja alas vieritettävän kehikon ohjelmien kuvakkeista, joista yhtä kuvaketta koskettamalla voidaan käynnistää haluttu ohjelma. Toteutuksessa on käynnistyksen yhteyteen tehty kuvassa 13 näkyvä edistyksen indikaattori, joka käynnistyy ohjelman käynnistyessä. Tässä näkymässä voidaan kuvak-

keita vapaasti siirrellä, kun yhtä kuvaketta kosketetaan pitkään. Tämä aktivoi siirtotoiminnon, joka voidaan lopettaa päästämällä kuvakkeen halutulla kohdalla ristikossa sormi irti näytöstä. Siirtotoiminnon ollessa käynnissä, ohjelmien käynnistyksen näkymään ilmestyy lisäksi punainen alue jossa käyttäjä voi joko poistaa ohjelman tai poistaa ohjelman kuvakkeen. Näitä toimintoja ei kuitenkaan ole toteutettu loppuun asti. Näkymän pienoisohjelmien luomisesta ei ole vielä päästy yksimielisyyteen, joten se on jätetty toteuttamatta.

Tapahtumien näkymässä on tarjolla pelkästään tekstimuotoista tietoa erilaisista tapahtumista. Teksti on näkyvissä aina sen mukaan mitä on haluttu tapahtumalla kertoa. Esimerkiksi sähköpostiviesti luo tapahtumien näkymään tiedon sähköpostin lähettäjästä ja sen lisäksi sähköpostin aiheesta. Tapahtumien näkymässä voidaan avata haluttu ohjelma kun tapahtumaa kosketetaan kerran, esimerkiksi sähköpostin saapuessa avataan sähköpostiohjelma juuri siitä viestistä, joka on näkyvissä tapahtumien näkymässä. Lisäksi toteutuksesta puuttuu tapahtuman ryhmittely erilaisten kuvakkeiden alle, toisin kuin yllä olevassa määrittelyssä on pyritty esittämään. Tämä näkymä on helposti kehitettävissä eteenpäin ja nykyinen toiminnallisuus kattaa vain perustoiminnot tässä näkymässä.

Kaikkiin näkymiin sisältyy mahdollisuus lukita laitteen näyttö vetämällä lukitusnäyttö puhelimen päälle aloittamalla vetämiseleen laitteen näytön mistä tahansa reunasta. Tämä aktivoi lukitusnäytön ja estää näin ollen kaiken vuorovaikutuksen kotinäkymään. Lukitusnäytössä on yksinkertainen taustakuva ja se tarjoaa tietoa kellonajasta ja päivämäärästä.

5 Pohdinta ja jatkotutkimusideat

Kaikenkaikkiaan tutkielmassa on perehdytty avoimen lähdekoodin ja käyttäjäkokemuksen käsitteisiin. Alkuperäiset tavoitteet on saavutettu tutkimalla molempia kirjallisuuden avulla. Glacier-käyttöliittymä osoittautui erittäin mielenkiintoiseksi tutkimuskohteeksi molempien aiheiden puolesta. Avoin lähdekoodi on hyvin lähellä tutkielman kirjoittajan sydäntä ja sen erilaisten ominaisuuksien tutkiminen oli tutkielman kirjoittajalle hyvin antoisaa. Lisäksi tutkielmassa tuotiin esille käyttäjäkokemuksen monitahoisuuden ja kokonaisvaltaisuuden uuden tutkimuksen osalta.

Avoimen lähdekoodin tutkimus on monilta osin hyvin pitkälle vietyä tietojenkäsittelytieteessä. Monia sen eri piirteitä on tutkittu laajasti, mutta tiettyjä aukkoja on vielä olemassa. Tutkielmassa luodattiin kuva avoimesta lähdekoodista sen historian, lisenssin ja nykyisen tutkimuksen avulla. Vapaiden ohjelmistojen käsite on sen sijaan sivuutettu, sillä sen tuomat rajoitteet olisivat tehneet tutkielman teosta haasteellista ja suppeaa. Nemo Mobile salli työlle käytännöllisen näkökulman avoimen lähdekoodin prosessien tutkimuksessa. Sen tuomat prosessit on kuvattu kokonaisuudessaan luvussa 2.4.

Käyttäjäkokemuksen tutkimus on sen sijaan vielä lapsenkengissään, vaikka tutkimukset ovat suuntauneetkin monille aloille, kuten esimerkiksi psykologiaan. Semiotiikka on otettu myös osaksi käyttäjäkokemuksen tutkimusta (Rousi 2013).

Tutkielman käytännön osuus painottui Glacier-käyttöliittymän tutkimiseen ja toteutukseen. Tutkielman kirjoittaja teki tätä toteutusta avoimen lähdekoodin projektin nimeltä Nemo Mobile parissa, erilaisten käyttöliittymän komponenttien ja kotinäkymän toteutuksessa. Nemo Mobilen yhteisö tuki tutkielman kirjoittajan työtä esimerkillisesti.

Tutkielmassa esitetyt tulokset kirjallisuudesta antavat monia suuntia mahdolliselle jatkotutkimukselle. Muita teorioita käyttäjäkokemuksesta voi tulla tulevaisuudessa moniakin, joiden käytännön arvon selvittäminen muiden erilaisten käyttöliittymien, joko avointen tai suljettujen, osalta voi olla hyvinkin hedelmällistä. Tutkielma antaa muutamiin teorioihin hyvän pohjan käyttäjäkokemuksen suunnittelun tutkimisessa ja käytännön esimerkkien toteutuksessa.

Muita käyttöliittymiä voi olla kuitenkin haasteellista tutkia, koska ne eivät välttämättä tarjoa yhtä avointa ja yhtenäistä suunnittelun, toteutuksen ja testaamisen päämääriä kuin Nemo Mobile. Sen tuoma vapaus avoimessa lähdekoodissa antaa tulevaisuuden

toteutuksille paljon mahdollisuuksia, mahdollisesti muissakin käyttöliittymän toteutuksen kannalta tärkeissä projekteissa, kuin mitä Glacier-käyttöliittymä tarjosi tähän tutkielmaan. Lisäksi Nemo Mobile-projekti on ollut tähän mennessä vaivatonta saattaa toimintaan monilla eri laitteistoalustoilla, kuten esimerkiksi olemassa oleviin Android-toteutuksiin.

Viitteet

Amadeo, R (2014) *The history of Android*, saatavilla osoitteesta http://arstechnica.com/gadgets/2014/06/building-android-a-40000-word-history-of-googles-mobile-os/, viitattu 6.8.2014

Android (2014), *Licenses*, saatavilla osoitteesta http://source.android.com/source/licenses.html, viitattu 19.1.2015

Apache Software Foundation (2004), *Apache License, Version 2.0*, saatavilla osoitteesta http://www.apache.org/licenses/LICENSE-2.0, viitattu 26.1.2015.

Apache Software Foundation (2012), *Licenses*, http://www.apache.org/licenses/, viitattu 19.1.2015

Arhippainen, L. (2013), *A Tutorial of Ten User Experience Heuristics*, saatavilla osoitteesta http://www.cie.fi/media-files/Profilematerial/Ten%20User%20Experience%20Heuristics%20with%20a%20Template.pdf, viitattu 23.10.2013

Battarbee, K., Koskinen, I. (2005), Co-experience: user experience as interaction, *CoDesign*, 2005, vol. 1, s.5–18

Black Duck Software (2014), Top 20 Open Source Licenses, WWW-sivusto, https://www.blackducksoftware.com/resources/data/top-20-open-source-licenses, viitattu 19.1.2015

Chacon, S. (2014), *Pro Git*, WWW-sivusto http://git-scm.com/book/fi/, viitattu 19.1.2015

Crowston, K., Wei, K., Howison, J., Wiggins, A. (2012), Free/Libre open-source software development: What we know and what we do not know, *ACM Computing Surveys (CSUR)*,44.2 (2012): 7.

Demetriou, M. (2014a) Nemomobile UX guidelines, WWW-sivusto, http://qwazix.github.io/nemomobile.github.com/

Demetriou, M. (2014b) The homescreen, WWW-sivusto, viitattu 5.8.2014, http://play.qwazix.com/grog/?p=694

Elgin, B. (2005) *Google buys Android for its mobile arsenal*, WWW-sivusto, http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal, viitattu 6.8.2014

Forlizzi, J., Battarbee, K. (2004), Understanding experience in interactive systems, *Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques*, s.261-268

GNU (2013), *What is Copyleft?*, WWW-sivusto, http://www.gnu.org/copyleft/copyleft.html, viitattu 13.11.2013

GNU (2014b), *Various Licenses and Comments about Them*, WWW-sivusto, http://www.gnu.org/licenses/license-list.html, viitattu 6.3.2014

GNU (2014c), *Frequently Asked Questions about the GNU Licenses*, WWW-sivusto, https://www.gnu.org/licenses/gpl-faq.html, viitattu 9.1.2015

GNU (2014d), *A Quick Guide to GPLv3*, WWW-sivusto, https://www.gnu.org/licenses/quick-guide-gplv3.html, viitattu 9.1.2015

Grog (2013), *Unveiling Glacier*, http://play.qwazix.com/grog/?p=344, viitattu 2.3.2014

Hassenzahl, M., Tractinsky,N. (2006a), User experience-a research agenda, *Behaviour & Information Technology*, vol. 25, s.91–97

Hassenzahl, M. (2006b), Hedonic, emotional, and experiential perpectives on product quality. *Encyclopedia of Human Computer Interaction*

Hassenzahl, M. (2008), Towards and experiential perspective on product quality, *Proceedings of the 20th International Conference of the Association Francophone d'Interaction Homme-Machine*, s. 11-15, ACM

Hassenzahl, M., Diefenbach, S., Göritz. A. (2010) Needs, affect and interactive products - facets of user experience. *Interacting with Computers*, 22(5), 353-362

Hassenzahl, M. (2010) *Experience Design: Technology for all the right reasons.* ISBN:

9781608450473.

Hassenzahl, M. (2013): User Experience and Experience Design. *In: Soegaard, Mads and Dam, Rikke Friis (eds.). "The Encyclopedia of Human-Computer Interaction, 2nd Ed.". Aarhus, Denmark: The Interaction Design Foundation*. WWW-sivusto: `http://www.interaction-design.org/encyclopedia/user_experience_and_experience_design.html`, viitattu 5.3.2014

Internation Organization for Standardization (2009), ISO FDIS 9241-210:2009. Ergonomics of human system interaction - Part 210: Human-centered design for interactive systems

Kasilag, K. (2013) Unveiling Glacier, WWW-sivusto, `http://play.qwazix.com/grog/?p=344` viitattu 9.4.2014.

Kujala, S., Roto, V., Väänänen-Vainio-Mattila K., Karapanos, E., Sinnelä A. (2011), UX Curve: A method for evaluating long-term user experience, *Interacting with Computers*, 23.5 (2011): 473-483

Law, E., Vermeeren, A., Hassenzahl, M., Blythe, M. (2007), Towards a UX manifesto, *Proceedings of the 21st British HCI Group Annual Conference on People and Computers*, ISBN: 978-1-902505-95-4, s. 205–206

Law, E., Roto, V., Hassenzahl, M., Vermeeren, A., Joke, K. (2009), Understanding, scoping and defining user experience: A survey approach, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ISBN: 978-1-60558-246-7, s. 719–728

Law, E., van Schaik, P., ja Roto, V. (2013). Attitudes towards User Experience (UX) Measurement. *International Journal of Human-Computer Studies*.

Lerner, J. ja Tirole, J. (2002), Some simple economics of open source, *The journal of industrial economics*, 50(2):197–234.

Marcus, A. (2006). Cross-cultural user-experience design. *Diagrammatic Representation and Inference* (s. 16-24). Springer Berlin Heidelberg.

Markoff, J. (2007), *I, Robot: The man behind the Google phone*,`http://www.nytimes.com/2007/11/04/technology/04google.html?`

_r=2&hp=&pagewanted=all, viitattu 6.8.2014

Mer Project (2013), *Mer Project, Main Page*, https://wiki.merproject.org/wiki/Main_Page, viitattu 9.11.2013

Mer Project FAQ (2012), *Mer Frequently Asked Questions*, https://wiki.merproject.org/wiki/FAQ, viitattu 20.8.2014

Mer Project Governance (2012), *Governance*, https://wiki.merproject.org/wiki/Governance, viitattu 20.8.2014

Miettinen R. (2006) Hajautettu luominen ja tiedon omistusoikeudet. *Tietoyhteiskunnan innovaatiopolitiikan perusteet*. Tieteessä tapahtuu 4/2006

Munk, C. (2013a) *Wayland utilizing Android gpu drivers, part 1*, WWW-sivusto, http://mer-project.blogspot.fi/2013/04/wayland-utilizing-android-gpu-drivers.html, viitattu 29.10.2014

Munk, C. (2013b) *Wayland utilizing Android gpu drivers, part 2*, wWW-sivusto, http://mer-project.blogspot.fi/2013/05/wayland-utilizing-android-gpu-drivers.html, viitattu 20.2.2015

Nakhimovsky, Y., Eckles, D., Riedelsberger, J. (2009), Mobile user experience research: challenges, methods & tools, *CHI'09 Extended abstracts on human factors in computing systems*, s. 4795–4798

Nemo (2014), *Nemo Mobile*, WWW-sivusto, https://wiki.merproject.org/wiki/Nemo, viitattu 20.8.2014

Nielsen, J. (2005) Ten Usability Heuristics, WWW-sivusto, http://www.nngroup.com/articles/ten-usability-heuristics/, viitattu 20.8.2014

Open source initiative (2013), *Open Source Definition*, WWW-sivusto, http://opensource.org/docs/osd, viitattu 17.10.2013

Pucillo, F., Cascini, G. (2014). A framework for user experience, needs and affordances. *Design Studies*, 35(2), 160-179.

Raymond, E. (1998) *Goodbye, "free software"; hello, "open source"*, WWW-sivusto,

http://www.catb.org/esr/open-source.html, viitattu 5.3.2014

Roto, V., Law, E., Vermeeren, A., Hoonhout, J. (2011), *User experience white paper*, viitattu 14.10.2013, saatavilla osoitteesta http://www.allaboutux.org/files/UX-WhitePaper.pdf

Rousi, R. (2013), *From Cute to Content - User experience from a cognitive semiotic perspective*, Väitöskirja, Jyväskylän yliopisto.

Stallman, R. M., Gay, J. (2009), *Free Software, Free Society: Selected Essays of Richard M. Stallman*, urlwww.gnu.org/philosophy/fsfs/rms-essays.pdf, viitattu 9.11.2013

User experience definitions (2013), http://www.allaboutux.org/ux-definitions, viitattu 22.10.2013

Varian, M. (1997), *VM and the VM Community: Past, Present, and Future*, haettu 21.10.2013 osoitteesta http://web.archive.org/web/20060513145315/http://www.princeton.edu/~melinda/25paper.pdf

Vermeeren, A., Law, E., Roto, V., Obrist, M., Hoonhout, J., Väänänen-Vainio-Mattila, K. (2010), *User Experience Evaluation Methods: Current State and Development Needs*, In E. Hvannberg, M. Lárusdóttir & J. Gulliksen (Eds.) Proceedings of the 6th Nordic conference on human-computer interaction 2010. New York: ACM, 521-530.

Välimäki, M. (2001) *GNU Yleinen lisenssi, versio 2*, WWW-sivusto, viitattu 12.8.2014, osoitteesta http://www.turre.com/licenses/gpl-2.0_fi.html

Välimäki, M. (2007) *GNU Yleinen lisenssi, versio 3*, WWW-sivusto, viitattu 12.8.2014, osoitteesta http://www.turre.com/licenses/gpl_fi.html

Väänänen-Vainio-Mattila, K., Roto. V., Hassenzahl, M. (2008), Towards Practical User Experience Evaluation Methods, *Proceedings of the COST294-MAUSE Workshop on Meaningful Measures: Valid Useful User Experience Measurement (VUUM)*

Weber, S. (2004), *The success of open source*, Cambridge University Press