LAPPEENRANNAN TEKNILLINEN YLIOPISTO
Teknistaloudellinen tiedekunta
Tietotekniikan osasto

HTML-teknologian hyödyntäminen Web-sovelluksen prototypoinnissa

Tarkastajat: Professori Jari Porras
Tutkijaopettaja Kari Heikkinen

Helsingissä 8. huhtikuuta 2010,
Marko Wallin, marko.wallin@iki.fi

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan osasto
Marko Wallin

HTML-teknologian hyödyntäminen Web-sovelluksen prototypoinnissa

Diplomityö

2010

65 sivua, 15 kuvaa, 3 taulukkoa ja 0 liitettä.

Tarkastajat: Professori Jari Porras
Tutkijaopettaja Kari Heikkinen

Hakusanat: prototypointi, Web-sovellus, käyttöliittymä, HTML-teknologia

Käyttöliittymä on rajapinta käyttäjän ja järjestelmän tarjoamien toimintojen välillä ja sen toimivuus vaikuttaa toimintojen suorittamiseen joko positiivisesti tai negatiivisesti. Täten sovelluksen suunnitteluvaiheessa on hyvä arvioida käyttöliittymän ja sen toimintojen laatua ja kokeilla ideoiden toimivuutta rakentamalla asiasta prototyyppejä. Prototypoinnilla voidaan tunnistaa ja korjata mahdolliset ongelmat jo suunnittelupöydällä.

Tämä diplomityö käsittelee Web-sovelluksen kehityksen aikana toteutettua käyttöliittymän ja sen toimintojen prototypointia. Käyttöliittymien mallintamista voidaan toteuttaa erilaisilla menetelmillä, joita työssä käydään läpi teknologisista näkökulmista eli miten prototypointimenetelmiä voidaan soveltaa projektin eri vaiheissa. Prototypoinnin apuna käytettäviin työkaluihin luodaan lyhyt katsaus esitellen yleisellä tasolla muutamia eri sovelluskategorian ohjelmistoja ja lisäksi käsitellään suunnittelumallien hyödyntämistä.

Työ osoittaa, että yleisiä prototypointimenetelmiä ja -periaatteita voidaan soveltaa Web-sovellusten prototypoinnissa. Prototypointi on hyödyllistä aloittaa luonnostelemalla ja jatkaa aikaisessa vaiheessa HTML-malleihin, joilla päästään lähelle toteutuksen teknologioita ja mallintamaan sovelluksen luonnetta, ilmettä, tuntumaa ja vuorovaikutusta. HTML-prototyypeistä voidaan jalostaa sekoitetun tarkkuuden malleja ja ne toimivat toteutuksen perustana. Jatkokehityksessä ideoita voidaan esittää useilla eri tarkkuuden tekniikoilla.

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology Management
Department of Information Technology
Marko Wallin

Using HTML technology when prototyping Web applications

Thesis for the Degree of Master of Science in Technology

2010

65 pages, 15 figures, 3 tables and 0 appendices

Examiners: Professor Jari Porras
Associate Professor Kari Heikkinen

Keywords: prototyping, Web application, user interface, HTML technology

User interfaces in every things are interfaces between the user and the functions which it is providing and thus the importance of usability and functionality can't be underestimated. With prototyping we can evaluate and examine different possibilities and usability issues in the design phases and produce better user interfaces.

This Master's Thesis is about the prototyping of Web applications' user interfaces and functionalities approaching the issue from technical point of view. The subject is dealt with reviewing different prototyping methods and how to use them in different stages of software development project. The work also goes through some prototyping tools and design patterns which are briefly examined and evaluated when prototyping Web applications.

In overall it is shown that the general prototyping methods and principles can be applied to the prototyping of Web applications' user interfaces and functionalities. It is worthwhile to start prototyping with sketching and move to HTML based models at early stages so we get quickly near the final technology and to design the characteristics, the look and feel and the interactions. HTML prototypes can be used develope mixed fidelity prototypes and are usable for the foundation for development stage. Further development issues can be prototyped with different low, mixed and high fidelity technologies.

ALKUSANAT

Tämä diplomityö on odottanut tekemistään jo useamman vuoden ajan, ja kun teksti viimein päätyy kansien väliin, päättää se yhden aikakauden elämässäni. On aika suunnata kohti uusia haasteita, mutta on lähes varmaa, että oppiminen ja opiskelu eivät tähän jää.

Haluan kiittää työni tarkastajia ja ohjaajia professori Jari Porrasta ja Kari Heikkistä asiantuntevista ja käytännöllisistä kommenteista sekä tuesta työni aikana.

Kiitän vanhempiani siitä tuesta ja kannustuksesta, mitä olen elämäni varrella ja opiskeluaikanani saanut. Opiskelukavereilleni kiitos hauskoista hetkistä opiskeluvuosien aikana ja sen jälkeen, vuodet Lappeenrannassa olivat unohtumattomia. Kiitokset myös ystäville ja työkavereille hyvistä hetkistä ja ymmärryksestä.

Helsingissä 8. huhtikuuta 2010,
Marko Wallin

Sisällysluettelo

Kuvat

Taulukot

SANASTO

AJAX Asynchronous JavaScript And XML

API Application programming interface

BSD Berkeley Software Distribution

CSS Cascading StyleSheets

GPL General Public License

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

IETF Internet Engineering Task Force

Java EE Java Platform, Enterprise Edition

JDBC Java Database Connectivity

JMS Java Message Service

JSF JavaServer Faces

JSON JavaScript Object Notation

JSP JavaServer Pages

JCP Java Community Process

JIT Just-in-time

JSR Java Specification Request

LGPL Lesser General Public License

MODFM Mockup-driven Fast-prototyping Methodology

MVC Model-View-Controller

RFC Request For Comments

RMI Remote Method Invocation

UML Unified Modeling Language

W3C World Wide Web Consortium

WARP Web Application Rapid Prototyping

WebML Web Modeling Language

XHTML eXtensible HyperText Markup Language

XML eXtensible Markup Language

1 Johdanto

Verkkosivustot ovat viime vuosina kehittyneet yksinkertaisista ja informatiivisista sivustoista enemmän ja enemmän perinteisiä työpöytäsovelluksia haastaviksi Web-sovelluksiksi, jotka sisältävä monimutkaisia toimintoja. Teknologiset innovaatiot ovat mahdollistaneet sovelluskehittäjille käytännössä vapaat kädet ja sivustojen ominaisuuksia rajoittavat nykyään lähinnä mielikuvitus ja käytettävissä olevat resurssit.

Monimutkaisten Web-palveluiden suunnitteleminen ja toteuttaminen vaatii ymmärrystä käytettävästä teknologiasta, sen mahdollisuuksista ja erilaisten toimintojen toimivuudesta käytännössä. Hyvän verkkosovelluksen suunnittelu on monimutkainen prosessi, jossa kaikkea huomionarvoista ei voida välttämättä alkuvaiheessa määritellä, suunnitella tai tietää. Sovelluksen suunnittelu ja toteutus ovatkin iteratiivista luovaa ongelmanratkaisua, jossa jo suunnitellut asiat muuttuvat ja uusia ratkaisuja pitää kehitellä. Sovelluskehityksessä muutos onkin yksi niistä asioista, joka on varmaa, ja suunnittelu, toimintojen arvioiminen ja vertailu vaatii tukea: sovelluksen prototypoimista.

Prototypoinnissa tehdään tuotteesta, tässä tapauksessa Web-sovelluksesta, erilaisia malleja, jotka kuvaavat kehitettävää sovellusta tai sen jotain osaa kuten käyttöliittymää. Mallien avulla saadaan parempi ymmärrys kehitettävästä sovelluksesta ja sen toiminnoista, sekä voidaan kokeilla miltä lopputulos voisi tuntua ja näyttää. Oikealla tavalla sovellettuna prototyypin avulla voidaan tunnistaa ja korjata mahdolliset ongelmat aikaisessa vaiheessa, jolloin sen arvo on moninkertainen käytettyyn aikaan nähden [MCP+06]. Prototyyppejä voidaan rakentaa monella tasolla ja laajuudella, ja sopivan prototypointimenetelmän valinta riippuu asiasta, jota halutaan tarkemmin arvioida.

Erilaisilla prototyypeilla päästään paremmin käsittelemään kehitettävän sovelluksen luonnetta ja yksi prototypoinnin keskittymiskohde on käyttöliittymät, niiden takana olevat toiminnot ja käytettävyys. Web-aikakausi asettaa uusia haasteita sovelluksen suunnittelulle, sillä verkkosovellusten käyttötarkoitukset ovat moninaiset ja käyttöympäristönä Internet ja Web-selain tuovat omat ominaispiirteensä sekä laadun että toiminnallisuuksien osalta. Lisäksi asioita voidaan toteuttaa hyvin erilaisilla tavoilla, eikä useille toiminnoille löydy perinteisistä työpöytäsovelluksista vastaavia jo hyväksi havaittuja toimintamalleja.

Käyttöliittymien rooli sovelluksen toiminnan kannalta jää sovelluskehitysprojekteissa usein liian vähälle huomiolle, eikä käytettävyyteen usein keskitetä riittävästi huomiota tiukan aikataulun ja budjetin painostaessa. Prototypointi onkin yksi avaintekijä interaktiivisten

systeemien suunnittelussa. Tekemällä suunnitteilla olevasta sovelluksesta prototyyppejä, voidaan kohtalaisen edullisesti ja nopeasti tutkia eri käyttöliittymä- ja toiminnallisten ratkaisujen käytettävyyttä. Prototypointimenetelmät ovatkin kehittyneet kattamaan erilaisia suunnittelutarpeita ja nykyään prototypointi on bisnesmaailmassa innovaation yksi avainelementti [BS00].

Tämä diplomityö käsittelee Web-sovelluksen kehittämistä prototypoimalla sitä määrittelyn, suunnittelun ja toteutuksen aikana käyttäen asiaa tukevia tekniikoita ja teknologioita kuten paperia, HTML:ää (Hypertext Markup Language) ja Javaa. Tavoitteena on tutkia millä tasolla ja tarkkuudella asioita kannattaa mallintaa projektin eri vaiheissa, mitä asioita pitää huomioida ja mitä hyötyjä prototypoinnilla saavutetaan.

Työ toteutetaan seuraamalla vesiputousmallia soveltavaa sovelluskehitysprojektia ja mallintamalla asioita sen aikana. Kirjallisessa osassa käsitellään lyhyesti Web-sovelluksen laatua, syvennytään prototypointimenetelmiin, mallien tarkkuuteen ja käytettäviin teknologioihin. Teorian perusteella arvioidaan eri menetelmien käyttämistä asioiden mallintamisessa projektin eri vaiheissa ja määritellään suuntaviivoja ja suosituksia.

1.1 Tausta

Sovelluskehitysprojekteissa on tullut useasti eteen, että jo suunniteltua asiaa joudutaan toteutusvaiheessa muuttamaan, tai että erilaisista ratkaisuista olisi ollut hyvä tietää jo sovelluksen suunnitteluvaiheessa. Etenkin toteutettaessa perinteisiä työpöytäsovelluksia uudelleen Web-sovelluksiksi, mutta suurin piirtein samoilla toiminnallisuuksilla ja vähäisellä suunnittelulla, olisi lopputulos voinut olla parempikin. Työpöytäsovelluksista tutut toiminnot eivät siirry samanlaisina ja samantasoisina Web-aikakaudelle, mutta projektissa käytettävissä olevan ajan puitteissa ei eri ratkaisujen toimivuutta havaita riittävän ajoissa. Haluankin tämän työn avulla selvittää, miten perinteistä jäykähköä vesiputousmallia soveltavassa projektissa pitäisi prototypointia soveltaa asioiden tutkimiseen ja selvittämiseen, jotta lopputulos olisi kaikilta osin toimiva.

Prototypoimalla sovelluksen toimintaa, voidaan käyttöliittymän toimintaa havainnollistaa, testata ja arvioida, jolloin sekä käyttäjä että sovelluksen toteuttaja puhuvat samasta asiasta. Teknologia tarjoaa erilaisia mahdollisuuksia ja rajoituksia, joiden puitteissa toimintaa on järkevää hahmotella, ja sopivan konkreettisen tason saavuttaminen käytettyyn aikaan ja saavutettuihin hyötyihin nähden ei aina ole selkeää. Uskon, että toteuttamalla

prototypointia sekä alhaisen että korkean tarkkuuden malleilla on etunsa ja se on kätevintä toteuttaa hyödyntämällä HTML:ää ja lopullisen sovelluksen teknologioita.

1.2 Tavoitteet ja rajaukset

Tässä työssä pureudutaan käyttöliittymän prototypointiin ja eri vaihtoehtojen mahdollisuuksiin ja soveltamiseen suunniteltaessa Java EE -ympäristön Web-sovellusta. Tavoitteena on selvittää, miten käyttöliittymää on käytännöllistä hahmotella, mitä prototypointimenetelmiä kannattaa soveltaa ja mitä työkaluja ja tekniikoita mallien rakentamisen apuna on hyödyllistä käyttää. Kantavana ideana on, että HTML ja Web-sovellusten teknologiat ovat yksinkertaisia ja helposti lähestyttäviä, joten asioita voidaan mallintaa nopeasti ilman erikoisohjelmia. Lisäksi tällöin prototypoinnista saadaan jäsenneltyä koodia, joka on myös helposti jalostettavissa lopullisen sovelluksen tueksi tai perustaksi.

Työn tavoitteena on saada vastaukset seuraaviin kysymyksiin:

- Miten hyödyntää käytettävissä olevia teknologioita sovelluskehitysprojektin eri vaiheissa sovelluksen prototypoinnin tukena?
- Miten HTML-teknologia soveltuu mallintamiseen ja mitä se mahdollistaa?
- Kuinka huomioida lopullisen sovelluksen teknologia ja sen hyödyntäminen?
- Millainen on käytännöllinen lähestymistapa Web-sovelluksen prototypointiin sovelluskehitysprojektissa ja sen jälkeen?

Teknologian kehitys on antanut sovelluskehittäjille suhteellisen vapaat kädet asioiden toteuttamiseen, mutta edelleen valittu toteutusteknologia asettaa omat rajaehtonsa mahdollisten ominaisuuksien ja toiminnallisuuksien suhteen, jotka on huomioitava käyttöliittymää suunniteltaessa ja toimintoja mallintaessa. Tässä työssä teknologialle asetetaan Java EE -ympäristöstä ja projektin teknologiavalinnan perusteella seuraavat reunaehdot: käytettävä teknologia on Java Server Faces (JSF) [BK06], käyttöliittymäkomponentit ovat Mojarra-referenssitoteutuksen ja sen kanssa yhteensopivia komponentteja ja asynkroniset toiminnot ovat JBoss RichFaces 3.3 -komponentteja. Sovelluksen tulee toimia vähintään Microsoft Internet Explorer 6.0 ja Mozilla Firefox 3.0 -selaimilla.

Sovelluskehityksessä on useita työvaiheita ennen käyttöliittymän ja sen toimintojen suunnittelua ja mallintamista, joiden suorittamiseen ei työssä oteta suoranaisesti kantaa. Lähtö-

oletus on, että vaatimusmäärittely, käyttötapaukset ja keskeisimmät toiminnot ovat jo selvillä, ja prototyypeissä voidaan keskittyä tutkimaan käyttöliittymään ja niiden toimintaan liittyviä asioita ja eri ratkaisujen toimivuutta. Aihetta tarkastellaan teknisestä näkökulmasta ja käsittelyn ulkopuolelle jää käyttäjäkeskeisiin asioihin syventyminen. Eli keskitytään prototypoinnin teknisiin osiin ja sivuutetaan parempaan käytettävyyteen tähtäävät asiat kuten käyttäjähaastattelut ja käyttäjätestaukset, lukuun ottamatta heuristisin menetelmin arviointia, joilla käyttöliittymän käytettävyyttä voidaan mitata. Käytettäessä sovellus ja käyttöliittymä-termejä, viitataan jatkossa Web-sovelluksiin ja niiden käyttöliittymiin ellei erikseen muuta mainita.

1.3 Työn rakenne

Diplomityö rakentuu kahdesta pääosasta: luku kaksi muodostaa työn teoreettisen osuuden ja luvut kolme ja neljä työn empiirisen osuuden.

Aiheen käsittely aloitetaan teorialla Web-sovelluksista ja niiden laadusta, luodaan katsaus prototypointiin sovelluskehityksessä ja prototypointimenetelmiin ja käsitellään menetelmien eroja, käyttökohteita ja mahdollisia työvälineitä. Lisäksi käydään lyhyesti läpi työhön liittyviä HTML- ja avustavia teknologioita. Kolmannessa luvussa käsitellään prototypointimenetelmiä ja -teknologioita Web-sovellusten näkökulmasta ja niihin liittyviä asioita. Lisäksi luodaan lyhyt katsaus eri työkalujen käyttömahdollisuuksiin. Neljäs luku sisältää sovelluskehitysprojektissa prototypointia soveltaessa havaittujen asioiden käsittelyä, luotaa mahdollisuuksia HTML-teknologian hyödyntämiseen verkkosovelluksen mallintamisessa ja tarjoaa suuntaviivoja Web-sovelluksen prototypointiin. Viidennessä luvussa pohditaan prototypointiin liittyviä asioita empiirisen osan pohjalta ja esitetään johtopäätökset. Viimeinen luku sisältää yhteenvedon.

2 Web-sovellukset ja prototypointi

Viime vuosikymmenten aikana teknologian kehitys on muuttanut verkkosivujen käyttömahdollisuuksia ja Webin tarjonta on muuttunut paljon Internetin alkuajoista. Alkujaan Web-sivut koostuivat lähinnä yhteen linkitetyistä HyperText Markup Language eli HTML-sivuista, jotka sisälsivät enemmän tai vähemmän informatiivista tekstiä ja kuvia. Tekniikan kehittyessä pelkkä HTML sai tuekseen kehitystyökaluja ja teknologioita, joilla voitiin tarjota informaation lisäksi myös monipuolisempia toiminnallisuuksia. Web-sovellukset olivat syntyneet [Pre04]. Verkkosivustot ja -sovellukset ovat kehittyneet monipuolisiksi työpöytäsovelluksia muistuttaviksi työkaluiksi ja ovat integroitu yritysten tietokantoihin ja bisnessovelluksiin.

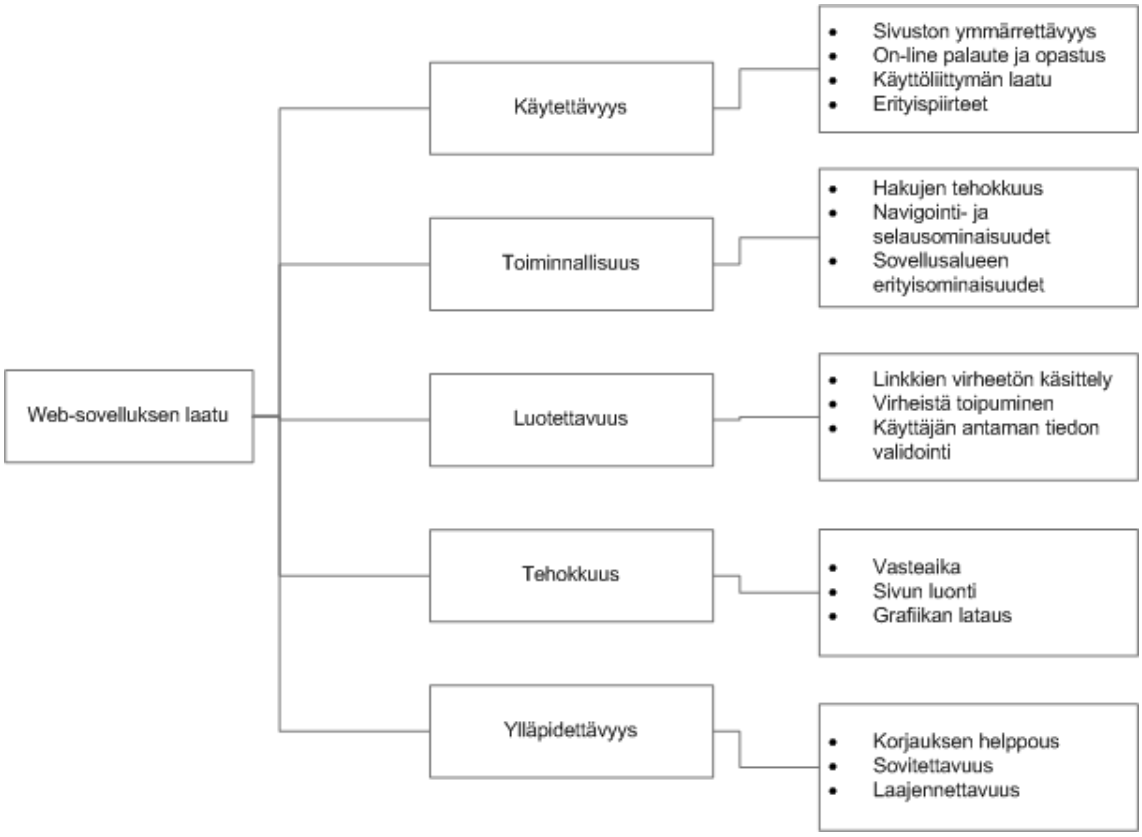
Web-sovellusten kehittyessä hienostuneiksi ja täysiverisiksi sovelluksiksi, myös ohjelmistojen painopiste on ainakin osittain siirtymässä perinteisistä työpöytäsovelluksista Web-aikakauteen, joka asettaa uusia haasteita käyttöliittymien suunnittelulle. Sovellusten käyttötarkoitukset ovat moninaiset ja käyttöympäristönä Internet ja Web-selain tuovat omat ominaispiirteensä sekä laadun että toiminnallisuuksien osalta.

Sovelluksen kannalta käyttöliittymä on vain osa kokonaisuutta, näkymä eri toimintoihin, mutta käyttäjän kannalta erittäin tärkeä osa: ilman käyttöliittymää on sovellusta vaikea käyttää ja sen toimivuus vaikuttaa myös tehtävien suorittamiseen. Tekemällä sovelluksen käyttöliittymästä ja toiminnallisuuksista malleja eli prototyyppejä, päästään ideoimaan, arvioimaan ja kokeilemaan eri ratkaisujen toimivuutta. Vanha sanonta “hyvin suunniteltu on puoliksi tehty” on usein oikeassa ja prototyyppien avulla kokeiltujen ratkaistujen avulla saadaan aikaan parempia sovelluksia.

2.1 Web-sovellusten laatu

Sovelluskehittäjän toimista suunnittelu on se, joka johtaa korkealaatuiseen sovellukseen [Pre04]. Täten on tarpeellista tunnistaa asiat, joista Web-sovelluksen laatu rakentuu, jotta voidaan määritellä prototypoinnin tavoitteet ja päämäärät. Pressman [Pre04] kuitenkin toteaa, että vaikka laatu on subjektiivista, ja jokaisella käyttäjällä on omat näkemyksensä “hyvästä” sovelluksesta, voidaan määritellä tärkeimpiä ominaisuuksia, joista laatu rakentuu. Nämä tärkeimmät ominaisuudet tarjoavat käytännöllisen perustan, jolle suunnittelija voi mallintamisen, testauksen ja sovelluksen arvioinnin rakentaa.

Web-sovelluksen laatuun liittyvät tärkeimmät ominaisuudet voidaan jakaa kuvan 1 esittämiin päätason kokonaisuuksiin, jotka jakautuvat pienempiin osa-alueisiin [OLR01]:



Kuva 1: Web-sovelluksen laatu

Viiden tärkeimmän laatuominaisuuden lisäksi listaan voidaan lisätä myös turvallisuus, tavoitettavuus, skaalautuvuus ja kehittämisen nopeus [Off02], joiden jälkeen käytössä on yhdeksän laatuominaisuutta, joiden perusteella laatua voidaan arvioida.

2.1.1 Laatuominaisuuksien huomioiminen

Web-sovelluksen laatua voidaan arvioida kuvan 1 esittämien laatuominaisuuksien perusteella. Laatu koostuu useista osa-alueista, mutta käyttöliittymien ja toiminnallisuuksien prototypoinnin kannalta tärkeimpiä ominaisuuksia ovat käytettävyys, toiminnallisuus ja tavoitettavuus. Käsittelemällä malleissa laatuun vaikuttavia asioita, päästään parempaan lopputulokseen ja ei turhaan mallinneta toiminnallisuuksia, jotka myöhemmin voivat aiheuttaa toteutuksessa ongelmia.

Käytettävyys-laatuominaisuus
Käytettävyys on kaikissa sovelluksissa tärkeä osa kokonaisuutta, jota prototypoidessa voidaan edistää. Käytettävyys muodostaa toiminnallisuuksien ja teknisten ominaisuuksien kanssa perustan sovelluksen onnistumiselle ja käyttäjän kokemalle laadulle. Käytettävyys-laatuominaisuus voidaan jakaa kuvan 1 jaoittelulla “sivuston ymmärrettävyys”, “online-palaute ja opastus”, “käyttöliittymän laatu” ja “erityispiirteet” -osa-alueisiin. Prototypoinnissa tehtävien mallien avulla voidaan arvioida käyttöliittymän toimintaa ja käytettävyyttä. Käytettävyys on korkean tason epäsuorasti mitattava laatuominaisuus, joka ilmaisee sen vaikuttavuuden, tehokkuuden ja tyytyväisyyden, jolla tietyt määritellyt käyttäjät saavuttavat määritellyt tavoitteet tietyssä ympäristössä [OLR01].

Toiminnallisuus-laatuominaisuus
Käytettävyyden lisäksi sovelluksen toiminnallisuuden laatu on tärkeä osa kokonaisuutta, johon prototypoinnissa voidaan keskittyä. Toiminnallisuus-laatuominaisuus voidaan kuvan 1 jaoittelun perusteella jakaa “hakujen tehokkuus”, “navigointi- ja selausominaisuudet” ja “sovellusalueen erityisominaisuudet” -osa-alueisiin. Toiminnallisuuden arviointi perustuu käytettävyyden tavoin paljolti käyttäjän kokemaan laatuun, jonka arviointia voidaan tukea ja kehittää muun muassa käytettävyystesteillä.

Tavoitettavuus-laatuominaisuus
Web-sovellusten oletetaan yleisesti olevan käyttäjien tavoitettavissa vuorokauden jokaisena tuntina, viikon jokaisena päivänä ympäri vuoden, mutta tavoitettavuus on muutakin kuin mahdollisuutta käyttää sovellusta vuorokauden ajasta riippumatta: niiden pitää olla tavoitettavissa erilaisilla Web-selaimilla ja käyttöjärjestelmillä [Off02]. Käyttämällä sovelluksen toteutuksessa ominaisuuksia, jotka eivät ole saatavilla eri käyttöympäristöissä, rajoitetaan mahdollista käyttäjäryhmää ja osallistutaan niin sanottuun selainsotaan. Nykyään eri selaimet eivät perustoiminnaltaan juurikaan eroa toisistaan ja tarjoavat suurin piirtein samat mahdollisuudet, mutta eroja on edelleen uusimpien määritysten tuen kattavuudessa.

2.1.2 Web-sovelluksen laadun varmentaminen

Jokaisella Internetiä ja Web-sovelluksia käyttäneellä on mielipide mikä tekee hyvän sovelluksen ja mielipiteet sekä ulkoasun että toiminnallisuuksien suhteen vaihtelevat suuresti. Toinen pitää kirkkaista väreistä, toinen yksinkertaisesta tekstistä. Toinen haluaa runsaasti tietoa, toinen tiivistettyä asiaa. Jotkut pitävät monipuolisista työkaluista tietokanto-

jen käyttämiseen, toiset haluavat tehdä sen yksinkertaisesti. “Itse asiassa, käyttäjän näkemys hyvästä (ja siitä seuraten hyväksyntä tai hylkäys) voi olla tärkeämpää, kuin mikään keskustelu teknisestä laadusta” [Pre04].

Mutta miten Web-sovelluksen laatu havaitaan? Mitä attribuutteja pitää esittää saavuttaakseen hyvyys loppukäyttäjän silmissä ja samalla säilyttää tekniset laadun ominaisuudet, jotka mahdollistavat sovelluskehittäjän korjaamaan, mukauttamaan, kehittämään ja tukemaan sovellusta pitkällä aikavälillä? Kuvassa 1 esitetyt laadun ominaisuudet muodostavat kokonaisuuden, mutta niistä käyttöliittymien näkökulmista tärkeimmät - käytettävyys ja toiminnallisuus lisättynä tavoitettavuus-laatuominaisuudella - tarjoavat käytännöllisen pohjan käyttöliittymien laadun määrittelemiseen. Myös luotettavuus, tehokkuus ja ylläpidettävyys -laatuominaisuudet ovat osaltaan tärkeitä, mutta käyttäjän näkemä laatu käyttöliittymien näkökulmasta painottuu käytettävyyden ja toiminnallisuuden muodostamaan kokonaisuuteen, kunhan luotettavuus ja tehokkuus ovat kuitenkin huomioitu.

Yleisesti ottaen, laatu koostuu hyvästä määrittelystä, suunnittelusta ja toteutuksesta, joita arvioidaan suorittamalla sarja teknisiä katselmointeja ja käyttäjätestejä, joilla arvioidaan sovelluksen eri osa-alueita. Steve Krug totetaa [Kru00], että testausta voi suorittaa hyvinkin yksinkertaisesti, eikä siihen tarvita käytettävyyslaboratorioita, paljoa aikaa tai rahaa, eikä välttämättä edes käytettävyysasiantuntijaa. Lisäksi ei ole koskaan liian aikaista näyttää suunnittelun ideoita käyttäjille tai avokonttorissa vierustoverille. Nielsenin mukaan [Nie00] suurin osa käytettävyysongelmista löydetään suorittamalla iteratiivisesti kolme testikierrosta kolmen-viiden hengen testiryhmällä ja korjaamalla havaitut ongelmat kierrosten välissä. Kokonaisuutena laadun varmentaminen ja testaus ovat aihepiiriltään laajoja, joten niitä ei tässä yhteydessä käsitellä syvemmin.

2.2 Prototypointi sovelluskehityksessä

Prototypointi tunnistetaan laajalti yhdeksi perusmenetelmäksi interaktiivisten sovelluksen ominaisuuksien tutkimiseen ja ilmaisemiseen, mutta Howde ja Hill toteavat [HH97], että termiä prototyyppi ja mitä se edustaa, on vaikea yksilöidä yksiselitteisesti, sillä asia voidaan käsittää monella eri tasolla. He tutkivat asiaa prototyypin esittämän roolin, sen “ilmeen ja tuntuman” ja toteutuksen näkökulmasta. Vastaavasti asiaa voidaan tutkia prototyypin tarkkuuden, kohderyhmän ja käyttäjien osallistuvuuden tasoilta [BS00]. Toiset pitävät prototyyppiä todisteena, että asia voidaan toteuttaa ja toiset arvostavat tarkkaa ilmeen ja tuntuman esittämistä.

Yleisellä tasolla prototyypit ovat malleja suunnittelun välituloksista ennen lopputulosta. Niitä luodaan kertomaan suunnitteluprosessista ja suunnittelussa tehdyistä päätöksistä ja ovat sekä luonnoksia että malleja eri tasoilla, kuvaten miltä asia näyttää, miten se käyttäytyy tai toimii [BS00]. Prototypoinin käyttäminen sovelluskehitysprojektissa on hyödyllistä vähintään kolmesta syystä: ei-kehittäjien on helpompi ymmärtää suunniteltavaa sovellusta, kun he näkevät sen; kokeneetkin suunnittelijat voivat tarvita suunnitellun asian visualisoimista ennen toteutusvaihetta; graafisilla suunnittelijoilla on tarvetta lisätä omat ideansa, kehittää ja kokeilla asioita ennen suunnitteluvaiheen päätöstä [BP00].

Karkealla tasolla prototypointia voi luokitella sen tarkoituksen, tarkkuuden ja käytettävien menetelmien mukaan. Erilaiset prototyypit vastaavat erilaisiin kysymyksiin, joihin malleilla haetaan vastausta tai selvennystä. Sovelluskehityksessä menestyksekäs suunnittelu pitää sisällään useiden prototypointimenetelmien, erilaisten mallien ja eri työvälineiden hyödyntämistä, joilla suunnittelija saa tukea sovelluksen rakentamiseen. Asioita ei ole mahdollista tutkia riittävällä tasolla käyttämällä vain yhtä ja tiettyä prototyyppiä, vaan suunnittelun edetessä eri asioita joudutaan arvioimaan erilaisten mallien avulla [Sze94].

Prototypoinnin teorian ja käytännön soveltamista on tutkittu vertailemalla erilaisia lähestymistapoja sovelluksen prototypointiin ja niiden toimivuuteen, ja Lichter [LSHZ93] toteaa, että ymmärrettäessä prototypointi osana kehitysprosessin kokonaisuutta, eikä vain itsenäisenä osana, saadaan parempia lopputuloksia ratkaistavaan ongelmaan. Parhaiden käytäntöjen mukaisesti prototypointi aloitetaan aikaisilla malleilla ja edetään myöhäisiin malleihin tai alkuvaiheen jälkeen syvennytään jonkin osa-alueiden mallintamiseen tarkemmin, jos aika- ja kustannusrajoitteet sallivat.

2.2.1 Prototypoinnin tarkoitus

Houde ja Hill [HH97] määrittelevät prototypoinnin vastaavan kolmeen pääkysymykseen:

1. Mikä rooli sovelluksella on käyttäjän elämässä?
2. Miltä sovelluksen pitäisi tuntua ja näyttää?
3. Miten sovellus pitäisi toteuttaa

Vastaamalla prototypoimalla näihin kolmeen pääkysymykseen, saavat kehittäjät mallinnettua sovellusta prototyyppien avulla siten, että se avustaa suunnittelussa ja lopullisen

sovelluksen toteutuksessa. Oikealla tavalla sovellettuna prototyypin avulla voidaan tunnistaa ja korjata mahdolliset ongelmat aikaisessa vaiheessa, jolloin sen arvo on moninkertainen käytettyyn aikaan nähden [MCP+06]. Lisäksi prototyyppien avulla saatujen konkreettisten tietojen pohjalta lopullisen sovelluksen kehitys on suoraviivaisempaa, kun vaikeat asiat ovat jo ratkaistu.

Haettaessa vastauksia edellä mainittuihin kolmeen kysymykseen, voidaan toteutettavia prototyyppejä luokitella niiden tarkoituksen mukaan, eli millä tavalla vastauksia etsitään: tutkiskellen, kokeillen tai kehittyen. Taulukossa 1 on esitetty prototyyppien tarkoitukset ja käyttökohteet.

Taulukko 1: Prototyyppien tarkoitus [LSHZ93]

Tutkiskeleva malli	Kokeileva malli	Kehittyvä malli
– kun ongelma ei ole selvä – pohjana alkuideat – useita lähestymistapoja – saadaan tietoa sovellusalueesta	– painotus teknisessä toteutuksessa – miten ratkaisut toimivat käytännössä – käyttäjät pääsevät kokeilemalla tarkentamaan vaatimuksiaan	– muutoksiin sopeutuva prosessi – yhteistyötä käyttäjien kanssa sovelluksen parantamiseksi – johtaa pilottijärjestelmään

2.2.2 Prototypoinnissa huomioitavat päämäärät

Prototypoinnissa huomioitavat suunnittelun päämäärät soveltuvat lähes jokaiseen sovellukseen, riippumatta sovellusalueesta tai monimutkaisuudesta [Pre04].

- **Yksinkertaisuus:** Vanha aforismi “yksinkertainen on kaunista” pätee myös sovelluksiin. On parempi pyrkiä hillittyyn ja yksinkertaiseen kokonaisuuteen, eikä hukuttaa käyttäjää grafiikalla tai kattavalla omaisuuslistalla.
- **Yhdenmukaisuus:** Sisältö pitäisi rakentaa noudattamaan samaa ilmettä sekä ulkonäöllisesti että toiminnallisesti koko sovelluksen laajuudelta. Tekstit ovat muotoiltu samalla tavalla, graafisilla elementeillä on sama ilme, graafinen ilme ei vaihdu sivuston osien välillä ja eri toiminnot käyttäytyvät samalla tavalla.

- **Yksilöllisyys, identiteetti:** Sovelluksen ulkoasu, esteettisyys ja navigointi pitää olla yhdenmukaista valitun sovellusalueen kanssa. Esimerkiksi räppäreille suunnatulla sivustolla on erilainen ilme kuin sivustolla, joka on suunniteltu bisnesihmisille.
- **Vakaus:** Valitun identiteetin perusteella annetaan usein lupaus käyttäjälle, joka odottaa vakaata sisältöä ja toimintoja, jotka ovat oleellisia käyttäjän tarpeille. Jos kyseiset elementit puuttuvat tai ovat vajavaiset, tulee sovellus todennäköisesti epäonnistumaan.
- **Navigoitavuus:** Sivustolla liikkuminen pitäisi suunnitella tavalla, joka on intuitiivistä ja ennakoitavaa. Käyttäjän pitäisi ymmärtää miten sovelluksessa liikutaan ilman navigoinnin etsimistä tai ohjeita.
- **Visuaalinen houkuttelevuus:** Kaikista sovelluskategorioista, Web-sovellukset ovat kiistämättä visuaalisimpia, dynaamisimpia ja anteeksipyytelemättä esteettisiä. Kauneus on katsojan silmässä, mutta useat suunnitelman ominaispiirteistä (ilme ja tunne, ulkoasu, värit, tekstin ja grafiikan balanssi, navigointi) vaikuttavat visuaaliseen houkuttelevuuteen.
- **Yhteensopivuus:** Sovellusta käytetään useissa eri ympäristöissä (laitteet, Internet-yhteys, käyttöjärjestelmä, selain) ja sovellus pitää suunnitella yhteensopivaksi eri ympäristöjen kanssa.

2.3 Prototypoinnin tarkkuus

Sovelluksesta tehtäviä prototyyppejä voidaan luokitella niiden tarkkuuden perusteella ja jakaa karkealla tasolla visuaalisen ilmeen, toiminnallisuuksien ja yksityiskohtien mukaan kolmeen ryhmään: aikaiset mallit (matalan tarkkuuden malli, low fidelity), myöhäiset mallit (korkean tarkkuuden malli, high fidelity) ja sekoitetut mallit (sekoitetun tarkkuuden malli, mixed fidelity). Sekoitetut mallit ovat yhdistelmä alhaisen ja korkean tarkkuuden malleista, jolloin yhdistetään molempien mallien hyviä puolia ja saadaan täten parempi ymmärrys kehitettävästä sovelluksesta [Yas07]. Mallien eroja on vertailtu taulukossa 2.

Eri tarkkuuksien mallien lisäksi prototyyppejä voidaan rakentaa myös eri tasoilla, riippuen mihin ongelmiin niillä haetaan vastauksia. Mallissa voidaan keskittyä vain horisontaaliseen tasoon, eli esimerkiksi vain käyttöliittymätasoon tai toteuttaa se vertikaalisti, eli rakentaa tietty kokonaisuus kaikilta tasoilta (käyttöliittymä, logiikka ja tietokanta), mutta jättää osa mallista pintapuoliselle tarkastelulle [LSHZ93].

Taulukko 2: Aikaisten, myöhäisten ja sekoitettujen mallien erot

Malli	Ominaispiirteet	Käyttökohteet
aikaiset mallit	– nopea kehittää – edulliset kehityskulut – ei varsinaista toiminnallisuutta – keskittyy näytön rakenteeseen – rajoitettu hyödyllisyys käytettävyystestejä ajatellen – navigaatio ja työnkulku rajoittuneita	– mahdollisuus arvioida useita konsepteja – mallin pohjalta vaikeampi toteuttaa – kätevä kommunikaation apuna – esiteltävä malli – soveltuu vaatimusten keräämiseen – ei käyttöä vaatimusanalyysin jälkeen
myöhäiset mallit	– kehitys hidasta – kallis kehittää – täydellisempi toiminnallisuus – lopullisen tuotteen näköinen – käyttäjäkeskeinen	– tehoton eri ratkaisujen arviointiin – kätevä tutkimiseen ja testaukseen – markkinoinnin ja myynnin työkalu – voidaan käyttää toimivana määritelmänä – ei sovellu vaatimusten keräämiseen – voidaan käyttää käyttäjätestaukseen
sekoitetut mallit	– nopeahko kehittää – kehityskulut kohtalaiset – osittainen toiminnallisuus – ulkonäöllisesti karkea tai tarkka – käyttäjä voi kokeilla	– tarkennetaan vaatimusta – voidaan testata osa toiminnoista – tutkitaan osaratkaisuja – osatoteutuksen määrittely mahdollista – käytettävyystestaus rajattua

2.3.1 Alhaisen tarkkuuden mallit

Alhaisen tarkkuuden mallit, eli aikaiset mallit, ovat karkeita konseptinomaisia hahmotelmia ideasta: luonnoksia staattisista sivuista. Niitä voidaan näyttää joko yksitellen tai sarjassa, esittäen jokin tarina kuvakäsikirjana, jolloin sovelluksen toiminnasta saa paremman käsityksen [Yas07]. Alhaisen tarkkuuden mallien tekniikoista luonnostelu, eli usein kynällä ja paperilla hahmottelu, on kenties yksi suosituimmista menetelmistä ideoiden esittämiseksi toisille, mutta ideoita voidaan visualisoida myös käyttämällä tietokoneavusteisia välineitä kuten kuvankäsittelyohjelmaa [Sze94]. Alhaisen tarkkuuden mallit tuovat esille sovelluksen yleisen ilmeen ja tuntuman, sekä perustoiminnot. Kehittäjän näkökulmasta ne ovat helppoja, nopeita ja edullisia toteuttaa, mutta karkeasta tarkkuudesta johtuen ne soveltuvat hyvin vain näytön rakenteen ymmärtämiseen ja tutkimiseen, mutta eivät oikein vuorovaikutukseen liittyvien asioiden selvittämiseen. Parhaiten alhaisen tarkkuuden mallit toimivat suunnittelun aikaisessa vaiheessa vaatimusten ja odotusten ymmärtämisen apuna [Yas07].

Luonnostelu

Luonnostelu on yksi yleisimmistä tekniikoista alhaisen tarkkuuden prototyyppien luomiseen. Menetelmässä usein käytetty “kynä ja paperi”-menetelmä on luonnollinen ja vähän vaivaa vaativa tekniikka, joka mahdollistaa abstraktien ideoiden siirtämisen suunnittelijan konseptista nopeasti kiinteämmälle pohjalle. Erityisen hyödylliseksi menetelmän tekee sen luovuuteen ja ajatteluun rohkaiseva luonne [Pet06]. Lisäksi luonnostelu rohkaisee useamman eri tekijän osallistumisen mallin luontiin, kun visuaalinen malli ja tekniikka on kaikille tuttua. Asiat jäävät myös riittävän epätarkoiksi, joka sallii tarkennusten tekemisen myöhemmin, estämättä luovuutta mallin kehittämisen aikana.

Rautalankamallit ovat tietokoneavusteisesti toteutettuja karkeita malleja ja usein harmaita runkomaisia diagrammeja, jotka esittävät suuriin piirtein miten navigointi ja eri elementit näytölle sijoittuvat. Graafisten työkalujen avulla malliin saadaan hahmoteltua tarkemmin ja selkeämmin halutut asiat. Rautalankamallit ovat nopeita ja edullisia rakentaa ja täten hyviä ideoiden selventämiseen, mutta jättävät “kynä ja paperi” -tekniikan tapaan vuorovaikutukseen ja toiminnalliseen puoleen liittyvät asiat etenkin monimutkaisten sovellusten osalta esittämättä.

Luonnostelun heikkouksia ovat sovelluksen käyttäytymisen kuvaaminen ja käyttöliittymän staattisuus, vaikka sovelluksen käyttäytymistä voidaan kuvata tekemällä asiasta use-

ampia piirroksia ja näyttämällä niitä “ennen ja jälkeen” toiminnon suorittamista ja avustamalla kuvausta tekstillä. Luonnokset ovatkin parhaimmillaan kuvatessaan sovelluksen ilmettä, mutta ei tuntumaa. Huolimatta luonnostelun heikkouksista, on se hyödyllinen täydentämään muita prototypointitekniikoita [Sze94].

Piirrokset vai tietokoneavusteinen malli?

Luonnosteltaessa piirtämällä toteutetut paperimallit ovat nopeita ja yksinkertaisia toteuttaa, koska tekniikka ei vaadi erityistaitoja. Vastaavasti paperiprototyypit ovat vaikeita muuttaa, kun suunnittelu edistyy ja malli voidaan joutua piirtämään usein alusta alkaen uusiksi [WTA02]. Myös mallin kehittymisen seuraaminen ja muistiinpanojen hallinta papereihin kiinnitettyjen muistilappujen ja merkintöjen lisääntyessä on hankalaa [Yas07]. Ongelmaksi muodostuu helposti myös mallien säilöminen ja myöhempi tarkastelu, sillä paperien varastointi ja niistä etsiminen ei ole kovin käytännöllistä tai kätevää.

Tietokoneella piirretyt graafiset mallit täydentävät useita paperiprototyyppien puutteita, ja graafista mallia on esimerkiksi helpompi tarvittaessa muuttaa, kun jo valmiita komponentteja voidaan leikata ja sijoittaa uudelleen. Sähköiseen malliin on lisäksi selkeämpää lisätä merkintöjä ja sen varastointi, siirtely ja myöhempi tarkastelu on helpompaa. Vastaavasti graafinen malli on työläämpi ja hitaampi toteuttaa ja voi rajoittaa luovuutta pakottamalla liialliseen tarkkuuteen. Tietokoneella tehdystä alhaisen tarkkuuden mallista voidaan jalostaa myös keskitason malli käyttämällä käyttöliittymätyökaluja tai skriptikieliä [Pet06].

Molemmilla tekniikoilla on etunsa, mutta tutkimukset osoittavat [WTA02], että paperiprototyypin ja tietokoneella tehdyn prototyypin välillä ei ole eroja käytettävyysongelmien löytämiseen tai asioiden kritisoimiseen. Toisaalta tietokonepohjainen malli on käyttäjälle kuitenkin miellyttävämpi ja halutumpi, ja täten on suositeltavaa käyttää paperiprototyyppejä vain silloin, kun tietokoneavusteinen malli ei tue suunnittelijan haluamaa ideaa tai kaikkien suunnittelutiimin jäsenten on tarve luoda prototyyppejä [STG03].

2.3.2 Korkean tarkkuuden mallit

Korkean tarkkuuden mallit, eli myöhäiset mallit, mahdollistavat käyttäjän kokeilla sovellusta, kuten se olisi lopullinen tuote. Prototyypin avulla kuvataan sovelluksen visuaalista ilmettä, interaktiivisuutta ja navigointia, sekä usein myös mahdollisia toimintoja. Käyttäjä saa mallin avulla selkeän kuvan miltä sovellus näyttää ja miten se käyttäytyy, ja voi täten

antaa ehdotuksia sen kehittämiseksi. Korkean tarkkuuden prototyypit ovatkin hyödyllisiä käytettävyystestauksen suorittamiseen suunnitteluprosessin aikana ja toimivat sekä markkinoinnin ja päättäjien että toteuttajien tukena [RSI96] [Pet06].

Sovelluksen ilmettä ja toimintoja esittävät mallit ovat usein toteutettu jollakin käyttöliittymän suunnitteluun soveltuvalla kehitysvälineellä ja skriptauskielillä, joilla saadaan nopeutettua mallin luontia. Huolimatta suunnittelua avustavista työkaluista, on korkean tarkkuuden mallin kehittäminen aikaa vievää ja kallista verrattuna alhaisen tarkkuuden malleihin [RSI96]. Koska mallit esittävät lopullisen tuotteen toimintoja, tulee jo prototyypin toteuttamisesta yksi kehitysvaihe, joka voi viedä useita viikkoja.

Lopullista tuotetta muistuttavilla malleilla on paljon hyödyllisiä ominaisuuksia, mutta myös heikkouksia, joita on vertailtu taulukossa 2. Mainittujen hitauden ja kehityskustannusten lisäksi myöhäisiä malleja on arvosteltu niiden käyttäjille ja asiakkaille antamista epärealistisista odotuksista. Toisaalta niiden avulla voidaan käytettävyystestauksessa saavutettavien hyötyjen lisäksi vakuuttaa päättäjätasolle, että suunnittelu on asianmukaista, huolellista ja lopullinen tuote on tuloillaan. Soveltuvin käyttökohde korkean tarkkuuden malleille on suunnitteluvaiheen lopussa, kun sovellusvaatimukset ovat ymmärretty ja käyttöliittymästä ja toiminnoista on päästy ymmärrykseen.

2.3.3 Sekoitetun tarkkuuden mallit

Prototyyppejä voidaan rakentaa yhdistelmällä sekä alhaisen että korkean tarkkuuden malleja, jolloin puhutaan sekoitetun tarkkuuden malleista. Käytännössä siis suunnitellaan osa prototyypistä esimerkiksi luonnostelemalla ja osa sovelluskoodina. Hyödyntämällä useampia eri tarkkuuksia samassa mallissa, on kehittäjän mahdollista keskittyä yhteen prototyypin kohtaan kerrallaan ja tutkia siihen liittyviä kysymyksiä eri tarkkuuksissa tarpeen mukaan [Pet06]. Se, mikä osa sovelluksesta toteutetaan korkealla tarkkuudella ja mikä alhaisella, riippuu mihin kysymyksiin prototyypillä ollaan hakemassa vastauksia: mikä osa sovellusvaatimuksista vaatii vielä tarkennusta.

Käytännössä sekoitetun tarkkuuden mallissa voidaan toteuttaa luonnosteltuja näyttöjä, jossa on piirrettyjä elementtejä ja kuvaelementtejä. Mallissa voi myös olla piirrosten ja kuvien lisäksi elementtejä, jotka ovat esitetty korkealla tarkkuudella. Piirretyt elementit voivat myös käyttäytyä samalla tavalla, kuin ne olisivat korkeamman tarkkuuden elementtejä [Pet06]. Jättämällä osa prototyypistä alhaiselle tarkkuudelle, suunnittelijat voivat tutkia korkean tarkkuuden elementtejä ja pitää ne kuitenkin samassa kokonaisuuudessa.

Sekoitettu malli tukee myös prototypoinnin luontaista yhteisöllisyyttä, kun mallin tekemisessä tarvitaan eri taustaisia henkilöitä. Käyttöliittymäsuunnittelijat, graafiset suunnittelijat, sovelluskehittäjät ja loppukäyttäjät osallistuvat aikaisessa vaiheessa suunnitteluprosessiin ja malli mahdollistaa eri osapuolten aktiivisen osallistumisen. Täten prototypoinnissa saadaan malliin sisällytettyä eri osapuolten ja eri näkemyksiä omaavien henkilöiden osaamista.

HTML-prototypointitekniikka

Sekoitetun tarkkuuden malleissa HTML-teknologialla saadaan nopeasti luotua lopullisen sovelluksen teknologiaa käyttäviä malleja. Vaidyanathan, Robbins ja Redmiles [VRR99] ovat tutkineet HTML-prototypointitekniikkaa näyttöpohjaisten sovellusten kuten pankkiautomaattien tapaisten sulautettujen järjestelmien suunnittelussa ja todenneet sen helpoksi tavaksi mallintaa ja arvioida niitä. Web-sovellukset ovat verrattavissa näyttöpohjaisiin sovelluksiin, sillä sovellus koostuu kohtalaisen yksinkertaisista näytöistä, joiden välillä tietoa vaihdetaan ja suoritetaan käyttäjän antamia toimintoja. Prototyyppi voidaan koostaa näytöittäin, joille hahmotellaan määrittelyiden perusteella sisältö, tarkennetaan mallia iteratiivisesti, lisätään linkit näyttöjen välillä liikkumiseen, ja jota voidaan ajaa ja arvioida Web-selaimessa.

HTML-prototypointitekniikan etuja ovat [VRR99]:

- **Työn kulku ja navigointi:** Voidaan hyödyntää sivukartta-työkaluja näyttöjen välisen riippuvuuden visualiointiin
- **Toteutus:** HTML on tekniikaltaan yksinkertaista ja toiminnot voidaan esittää sivulta toiselle vievien linkkien avulla
- **Iteratiivinen kehitys:** Muutosten tekeminen on helppoa ja nopeaa, jolloin viive mallin muutosten ja arvioijilta saadun palautteen välillä on pieni.
- **Työvälineet:** HTML-työkaluja on saatavilla sekä kaupallisia että ilmaisia, joita voidaan käyttää prototyypin luomiseen.
- **Osallistujat:** Mallia on helppo arvioida Web-selaimen kautta käyttäjän sijainnista riippumatta, joka kannustaa palautteen antamiseen.
- **Interaktiiviset osuudet:** Tarvittaessa sovelluksen toimintoihin menevää aikaa voidaan simuloida lisäämällä malliin viivettä ohjelmallisesti.

– **Konseptuaalinen yksinkertaisuus:** Käytettävä teknologia on yksinkertaista ja sisältää käytännössä vain näyttöjä ja linkkejä, jotka ovat tuttuja kaikille HTML:ää aikaisemmin käyttäneille.

2.4 Työkalut ja ohjelmistot prototypoinnin tukena

Käyttöliittymien ja toiminnallisuuksien prototypoinnissa voidaan käyttää tukena erilaisia työkaluja ja ohjelmistoja, jotka auttavat suunnittelijaa mallien rakentamisessa. Yksinkertaisimmillaan käytettävät ohjelmistot voivat olla perinteisiä yleiskäyttöisiä ohjelmistoja, tai erityisesti prototypointiin tarkoitettuja ohjelmistoja. Työkalujen avulla voidaan toteuttaa sekä alhaisen tarkkuuden että korkean tarkkuuden malleja, sekä horisontaalisesti että vertikaalisesti.

Prototypointiin soveltuvia ohjelmistoja on lukuisia ja niiden kaikkien käsitteleminen ei ole mielekästä, sillä ohjelmien soveltuvuus käsiteltävän ongelman ratkaisuun vaihtelee. Soveltuvat työkalut voidaan tarkoituksensa perusteella yleisellä tasolla jakaa “julkisivu- ja käyttöliittymätyökalut” ja “erikoisohjelmistot” -kategorioihin, joiden hyödyllisyyttä eri malleihin voidaan arvioida ohjelmien ominaispiirteiden perusteella.

2.4.1 Julkisivu- ja käyttöliittymätyökalut

Prototyyppien toteuttamisessa voidaan hyödyntää useita erilaisia työkaluja, joilla saadaan mallinnettua sovelluksen ilmettä ja toimintaa. Yksi tapa on hyödyntää eritasoisia käyttöliittymätyökaluja, joilla voidaan toteuttaa sovellusta graafisella tasolla lisäten siihen vuorovaikutusta, tai enemmän koodin tasolla vetämällä ja sijoittamalla käyttöliittymäkomponentteja haluamiin paikkoihin näytöllä ja generoimalla tästä sovelluskoodia [Sze94]. Työkalut tarjoavat mahdollisuuden esittää interaktiivisia visuaalisia konsepteja, nopeuttavat toteutusta koodigeneroinnilla ja ovat kohtalaisen yksinkertaisia käyttää.

Yleisesti ottaen, käyttöliittymätyökalut rajoittavat suunnittelijoiden mahdollisuuksia ja vaativat paljon aikaa ja vaivaa prototyypin luomiseksi [Pet06]. Täten ne eivät sovellu käytettäväksi projektin aikaisessa vaiheessa, kun erilaisia malleja pitää tutkia, mutta parhaimmillaan työkaluilla saadaan nopeasti luotua sovelluksen ilmettä ja toiminnallisuutta kuvaava malli. Toisaalta työkalujen avulla luotua prototyyppiä voi olla hankala hyödyntää myöhemmin ja generoitu koodi on yleensä sekavaa, jolloin sen käyttöarvo lopullises-

sa toteutuksessa on vähäinen [Sze94]. Vastaavasti Brooks [Bro95] väittää, että kehittäjien pitäisi aina heittää sovelluksen ensimmäinen versio pois, sillä ensimmäisellä toteutuskerralla opitaan, miten sovellus pitää seuraavalla kerralla toteuttaa. Tältä näkökannalta julkisivutyökalut ovat hyvä keino saada talteen kaikki oleellinen tieto sovellukseen liittyen, ilman sen varsinaista toteuttamista, mutta ei ole riittävästi näyttöä, että se painaisi vaakakupissa enemmän kuin uudelleen käytettävän koodin puute.

Julkisivutyökalut

Graafiseen tasoon keskittyvät niin sanotut julkisivutyökalut ovat käytännössä piirtotyökaluja, joissa piirroksiin on mahdollista lisätä vuorovaikutusta mallintavaa käyttäytymistä. Ne sisältävät useita luonnostelun hyviä puolia, lisäten niihin mahdollisuuden toteuttaa näyttöjä, jotka näyttävät ja käyttäytyvät kuten oikea sovellus, mutta ilman taustalta löytyvää sovellusta. Negatiivisina puolina julkisivutyökalut eivät tuota uudelleenkäytettävää koodia ja mallin luomiseen käytettyä panostusta ei voida hyödyntää myöhemmin. Lisäksi julkisivutyökalut voivat antaa sovelluksen mahdollisuuksista liian positiivisen vaikutelman, kuin mitä käytettävissä olevan budjetin, ajan ja teknologian puitteissa on mahdollista toteuttaa. [Sze94]

Julkisivutyökaluiksi voidaan luokitella esimerkiksi erilaiset taulukkolaskentaohjelmistot kuten Microsoft Excel ja esitysohjelmat kuten Microsoft Powerpoint, joissa toimintaa voidaan mallintaa skriptaamalla esimerkiksi Visual Basic for Applicationsin avulla. Tällaiset yleiskäyttöiset ohjelmat ovat useille tuttuja työvälineitä ja täten helposti lähestyttäviä, joten mallin luominen valmiista komponenteista rakentamalla onnistuu vähemmän teknisiltä henkilöiltäkin [LL07]. Ohjelmistot tarjoavat graafisilta ominaisuuksiltaan ja komponenteiltaan mahdollisuuden yksinkertaisten sovelluksen ulkoasua ja osittain toimintaa mallintavien prototyyppien toteuttamiseen.

Käyttöliittymätyökalut

Käyttöliittymää koodin puolelta lähestyvät työkalut ovat enemmänkin toteutus- kuin prototypointityökaluja, mutta toimivat molemmissa tarkoituksissa. Niiden vahvuus on julkisivutyökalujen tapaan mahdollisuus määritellä sovelluksen käyttöliittymä sijoittelemalla komponentteja näytölle. Lisäksi ne luovat myös ajettevaa koodia, jota voidaan hyödyntää lopullisen tuotteen toteutuksessa [Sze94]. Saavutettavien etujen vastapuolella on työkalujen rajoittuneisuus käyttöliittymän staattisten osien mallintamiseen, vaatimus toteuttaa taustalta löytyvät toiminnot ja ne pakottavat valitsemaan tietyt käyttöliittymäkomponentit. Lisäksi ohjelmista saatavaa käyttöliittymäkoodia voi olla hankala erottaa sovellukses-

ta. Käyttöliittymätyökaluina toimivat esimerkiksi erilaiset Web-ohjelmointityökalut kuten Adobe Dreamweaver ja Java-kehitystyökalut kuten Eclipse ja Sun Netbeans.

2.4.2 Erikoisohjelmistot

Prototypoinnin tueksi on kehitetty erilaisia työkaluja ja ohjelmistoja, jotka tarjoavat esimerkiksi sivukartan, kuvakäsikirjoituksen ja sivun prototyypin rakentamiseen liittyviä ominaisuuksia. Erikoisohjelmistoista mainittakoon esimerkkinä kynällä ja paperilla hahmoittelua muistuttava DENIM. Kaupallisista sovelluksista mainittakoon muun muassa rautalankamallien tekoon apua tarjoavat työkalut Axure RP Pro ja OmniGraffle Pro.

DENIM

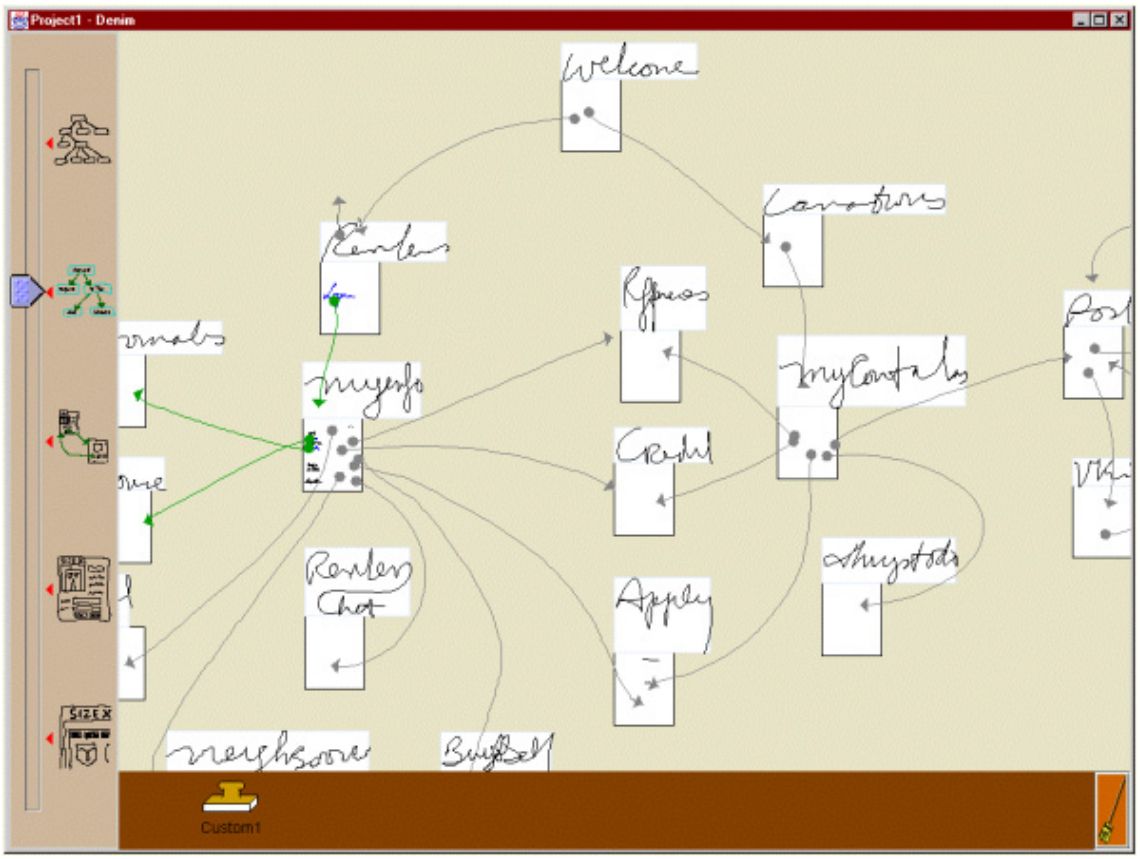
DENIM [LNHL00] on Washingtonin yliopistossa kehitetty tietokoneohjelma tukemaan sovelluskehittäjiä suunnittelun alkuvaiheen informatiivisen luonnostelun kautta, joka käytännössä tarjoaa paperintapaisen käyttöliittymän mallien luontiin. Ohjelma tarjoaa suunnittelun tueksi kynällä luonnostelua, eri tarkkuuksille kohdentamista (sivukartta, kuvakäsikirjoitus, sivu), sivujen linkittämistä kuvakäsikirjoitusta varten, mallin ajamista esittämistä ja vuorovaikutusta varten, mahdollisuuden luoda uudelleenkäytettäviä komponentteja ja viedä malli HTML-sivuiksi. Ominaisuuksiltaan DENIM soveltuu alhaisen tarkkuuden mallien luontiin ja ei tarjoa polkua korkean tarkkuuden malleihin siirtymiseen. Javalla toteutettu ja avoimen lähdekoodin Berkeley Software Distribution (BSD) -lisenssillä varustettu DENIM on saatavilla Windows, Unix ja Mac OS X -käyttöjärjestelmille.

DENIMillä luodut prototyypit muistuttavat kynällä ja paperilla luonnosteltuja malleja sekä hyvässä että pahassa kuten kuvassa 2 näkyvästä sivukartasta voi todeta. Koska nyt piirtovälineenä toimii piirtokynä tai hiiri, on piirtojälki kynää karkeampaa ja epätarkempaa. Paperimalleihin verrattuna sähköinen media tarjoaa mahdollisuuden elementtien siirtelyyn ja muokkaamiseen ja lopputuloksen dokumentointi ja säilöminen myöhempää käyttöä varten on helpompaa. Tietenkin kuten kynällä ja paperilla luonnostellessa, jää käyttöliittymän vuorovaikutuksen esittäminen ja suunnitellun mallin myöhempi hyödyntäminen vähäiseksi.

Ohjelman käyttö on suhteellisen yksinkertaista ja opasteen avulla hieman erikoinen käyttöliittymä ja muun muassa hiirieleillä toimivat komennot selviävät nopeasti, mutta tämänkin jälkeen työskentely on hieman hidasta ja monimutkaisen tuntuista. Kömpelyys osittain johtunee suurilta osin piirtovälineenä käytettävästä hiirestä ja piirtopöydällä käytet-

tynä DENIM voisi olla pätevä apuväline. Nyt osa sovelluksen kätevyydestä menee hiiren epätarkkuuden kanssa hukkaan, vaikka ohjelmisto suoristeleekin piirrettyjä viivoja.

DENIMin hyödyntäminen sähköisten luonnosten tekemisessä tarjoaa etuja "kynä ja paperi"-luonnosteluun verrattuna, mutta kaipaa käytön tueksi joko vaakata kättä tai hiirtä parempia piirtomahdollisuuksia. Kynällä ja paperilla saa edelleen luotua nopeammin ja helpommin luonnoksia, piirrosjälki on tarkempaa ja mahdollistaa pienempien ja tarkempien elementtien hahmottelun.



Kuva 2: DENIMillä hahmoteltu sivukartta [Lan08]

2.5 HTML- ja avustavat teknologiat

Verkkosivustojen ja -sovellusten taustalta löytyy useita erilaisia teknologioita, joilla tuetaan HTML-merkkausta ja saadaan aikaan sivustoja, jotka tarjoavat informaation lisäksi myös monipuolisempia toiminnallisuuksia. Sivustojen Web-selaimessa tulkitsemiseen voidaan vaikuttaa suorittamalla skriptejä kuten käyttäytymiseen vaikuttavia JavaScript-

skriptejä tai ulkonäköön vaikuttavia Cascading Style Sheets (CSS) määrityksiä. HTML:n ja skriptien tarjoamia ominaisuuksia voidaan lisäksi laajentaa selainlaajennuksilla kuten Adoben Flash- ja Flex-teknologioilla, Microsoftin Silverlight-teknologialla tai Oraclen Java Appleteilla.

2.5.1 HyperText Markup Language

HTML eli HyperText Markup Language on hallitseva merkkauskieli verkkosivustoille ja se tarjoaa keinot luoda rakenteellisia dokumentteja määrittelemällä tekstille semanttisia rakenteita kuten otsikoita, kappaleita, listoja ja linkkejä ja mahdollistaa kuvien ja muiden objektien liittämisen sivulle. Tim Berners-Leen vuonna 1991 alkuunsaattama HTML on vuosien varrella kehittynyt IETF:n (Internet Engineering Task Force) luonnosten ja RFC:den (Request For Comments) kautta vuodesta 1996 [CM00] W3C:n (World Wide Web Consortium) kehittämiksi suosituksiksi ja luonnoksiksi [Con09b].

Viimeisin spesifikaatio HTML:stä on vuonna 1999 julkaistu HTML 4.01 Recommendation ja luonnos seuraavaksi versioksi, HTML 5 Working Draft, julkaistiin tammikuussa 2008. W3C on julkaissut myös spesifikaatioita XHTML-määritykselle, joka on uudelleenmääritys HTML 4.01:lle käyttäen XML 1.0:aa. Viimeisin XHTML-suositus, XHTML 1.1, julkaistiin toukokuussa 2001. Käytännössä määritysten erot ovat käyttäjälle näkymättömiä.

W3C:n suositukset ja luonnokset määrittävät käytännössä sen, miltä verkkosivut Web-selaimissa vaikuttavat, eli kuinka selaimet piirtävät eri komponentit ruudulle ja kuinka ne käyttäytyvät. Suosituksia voi kuitenkin tulkita monella tapaa ja HTML-spesifikaation ja luonnosten toteutuksen tavassa ja kattavuudessa on eroja selainten välillä [Hamon]. Suurimpien selainten moottoreita HTML:n tulkitsemiseen käyttäjälle ovat Internet Explorerin Trident, Firefoxin Gecko, Operan Presto ja Safarin WebKit.

2.5.2 CSS-tyylit

Cascading Style Sheets (CSS) on tyyliohjekieli, jota käytetään kuvaamaan merkkauskielellä kirjoitetun dokumentin esittämisen semantiikkaa eli ilmettä ja muotoilua. Yleisimmät käyttökohteet CSS:lle ovat HTML- ja XHTML-merkkauskielillä kirjoitetut verkkosivut, mutta sitä voidaan soveltaa myös XML-dokumentteihin. Ensisijaisesti CSS on suun-

niteltu mahdollistamaan merkkauskielellä kuten HTML:llä kirjoitetun sisällön erottamisen sen esitystavasta eli dokumentin asettelusta, väreistä ja käytetyistä fonteista. Tämä parantaa sisällön saatavuutta, tarjoaa sekä joustavuutta että hallittavuutta esitystavan määrittelyyn ja mahdollistaa muun muassa saman sisällön esittämisen eri päätelaitteissa erilailla muotoiltuna tarpeen mukaan.

CSS-tyyliohjekieli on World Wide Web konsortion (W3C) määrittelemä [Conon] ja ensimmäinen CSS-suositus julkaistiin vuonna 1996. Viimeisin julkaistu suositus on CSS 2 vuodelta 1998 ja ehdokas seuraavaksi suositukseksi CSS 2.1 julkaistiin vuonna 2007. W3C:n tällä hetkellä työn alla oleva CSS 3 -määritys tulee olemaan modulaarinen ja sisältämään useita eri suosituksia.

Huolimatta W3C:n suosituksista, tulkitsevat useat selaimet määrityksiä eri tasoisesti, eivätkä välttämättä tue kaikkia määriteltyjä ominaisuuksia. CSS 1 -suositusta nykyaikaiset selaimet noudattavat asiallisesti, mutta yleisesti jo käyttöliittymän muotoiluun käytettävien CSS 2.1 -ominaisuuksien kattavuudessa on puutteita etenkin vanhemmilla Microsoft Internet Explorer -selaimilla [Koc]. CSS -suositusten toteuttamisen lisäksi eri selainten välillä on eroja myös HTML-komponenttien käyttäytymisessä tyylimuotoilujen kanssa ja tyylien määritteleminen tietylle komponentille ei aina kaikissa selaimissa onnistu.

2.5.3 Javascript ja Ajax

JavaScript on oliomainen skriptikieli, jota käytetään mahdollistamaan elementtien ohjelmallinen käsittely, ja pääasiassa sitä hyödynnetään Web-selaimeen integroituna komponenttina, jota suoritetaan verkkosivua näytettäessä. JavaScript mahdollistaa käyttöliittymän tehostamisen ja sivujen dynaamisen kehittämisen.

Alunperin Netscapen Mocha- ja LiveScript-nimillä kehittämä JavaScript on ECMAScript-standardin murre, joka esiteltiin ensimmäisen kerran Netscape Navigator -selaimen 2.0B3-versiossa vuonna 1995. Seuraavana vuonna Microsoft kehitti yhteensopivan JScript-kielen, joka julkaistiin Internet Explorer -selaimen 3.0-versiossa. Laajalti Web-selaimien tukema JavaScript 1.5 julkaistiin vuonna 2000 ja se vastaa ECMA-262 Edition 3 -standardia [Int99]. Virallisesti JavaScriptiä kehittää Mozilla Foundation.

Web-selaimet käyttävät JavaScriptin suorittamiseen omia moottoreita, jotka vaikuttavat muun muassa skriptien suorittamisen nopeuteen, eli siihen kuinka kauan esimerkiksi graafisen efektin piirtämiseen menee ja paljonko se tietokoneen prosessoria selaimen välityk-

sellä kuormittaa. Eri selaimien käyttämiä JavaScript-moottoreita ovat muun muassa Microsoft Internet Explorerin JScript, Mozilla Firefoxin TraceMonkey, Apple Safarin Nitro, Google Chromen V8 ja Operan Carackan. Modernit selaimet käyttävät Just-in-time (JIT) -tekniikkaa JavaScriptin suorittamisessa, joka kääntää JavaScriptin ajon aikana natiivikoodiksi ja täten nopeuttaa sen suorittamista. Selainten suorituskykyä JavaScriptin osalta voi testata esimerkiksi Applen WebKit -tiimin kehittämällä SunSpider -testausohjelmalla.

Sivujen dynaamista päivittämistä varten JavaScriptiä voidaan yhdistää XML:n kanssa eli käyttää niin sanottua Asynchronous JavaScript and XML eli Ajaxia. Sen avulla voidaan toteuttaa sovelluksen toimintoja lähettämällä asynkronisia pyyntöjä palvelimelle, joihin palvelin palauttaa Web-selaimelle vastauksen. Käytännössä XML-viestit voivat sisältää osia HTML-sivusta, joita muokataan viestien vaihdossa ja lopuksi päivitetään muuttunut tila sivulle. Data haetaan yleensä käyttämällä XMLHttpRequest Objectia ja vastaus voidaan saada XML:n lisäksi esimerkiksi myös JavaScript Object Notation (JSON) -formaatissa. Ajaxia käyttämällä sovelluksen toiminta saadaan muistuttamaan enemmän työpöytäsovellusta, kun koko sivua ei tarvitse lähettää kerralla, vaan voidaan päivittää vain osa sivun sisällöstä. W3C on tekemässä XMLHttpRequest Objectista standardia [Con09a].

2.6 Java EE -teknologiat

"Java Platform, Enterprise Edition", lyhyemmin Java EE, on laajalti käytetty alusta palvelinpuolen Java-ohjelmointiin, joka tarjoaa monipuoliset ominaisuudet Web-sovellusten kehittämiseen. Java EE on määritelty Java Community Process -ohjelmassa ja uusin määritys on Java EE 6 eli JSR 316 [Chi09]. Määritys pitää sisällään useita API (Application programming interface) -spesifikaatioita kuten JDBC (Java Database Connectivity), RMI (Remote Method Invocation), sähköposti, JMS (Java Message Service), Web Service ja XML ja määrittelee kuinka eri osat toimivat yhteen. Java EE pitää sisällään myös Java EE komponenteille uniikkeja spesifikaatioita kuten Enterprise JavaBeans, Connectors, Servlets, Portlets, JavaServer Pages ja useita Web Service -teknologioita.

Kokonaisuutena tämä mahdollistaaa kehittäjien toteuttaa sovelluksia, jotka ovat siirrettäviä, skaalautuvia ja toimivat yhteen vanhempien teknologioiden kanssa. Vastaavasti Java EE -sovelluspalvelimet hallitsevat transaktiot, tietoturvan, skaalautuvuuden, yhtäaikaisuuden ja sovellukset, jotka niihin on asennettu. Näin sovelluskehittäjä voi keskittyä enemmän sovelluksen logiikkaan kuin infrastruktuuriin tai eri osien integrointiin.

2.6.1 JavaServer Faces

JavaServer Faces (JSF) on Java-pohjainen Model-View-Controller- eli MVC-arkkitehtuuri, joka perustuu komponenttipohjaiseen käyttöliittymämalliin käyttäen XML-tiedostoja näkymien luomiseen. MVC- eli malli-näkymä-ohjain-ohjelmistoarkkitehtuurityylin tarkoituksena on erottaa käyttöliittymä sovellustiedosta, eli jokainen arkkitehtuurin osa vastaa omasta alueestaan sovelluksessa: Malli huolehtii järjestelmän sovellustiedon tallentamisesta, ylläpidosta ja käsittelystä; Näkymä määrittää käyttöliittymän ulkoasun ja mallin tietojen esitystavan käyttöliittymässä; Ohjain eli kontrolleri vastaanottaa käyttäjältä tulevat käskyt sekä muuttaa mallia ja näkymää vastauksena niihin.

JSF:ssä sivut käsitellään pyyntöinä, jotka FacesServlet prosessoi ladaten sopivan mallin, rakentamalla komponenttipuun, käsitellen tapahtumat ja luoden vastauksen käyttäjälle esimerkiksi HTML-sivun muodossa. Käyttöliittymäkomponenttien tila tallennetaan jokaisen pyynnön lopussa ja palautetaan näkymää uudelleen luodessa. Sivujen luontiin JSF:ssä käytetään normaalisti JSF 1.x -versioissa JavaServer Pages (JSP) -teknologiaa ja JSF 2 -versioissa kehittyneempää Facelets-teknologiaa, joka on sekä tehokkaampi että yksinkertaisempi kieli näkymän kuvaamiseen.

Java Community Process -ohjelmassa kehitettävästä JSF:stä julkaisiin vuonna 2004 ensimmäinen JSR 127 -spesifikaatio kattaen JSF:n 1.0 ja 1.1 -versiot ja tuorein JSF 2.0 -spesifikaatio JSR 314 julkaistiin vuonna 2009. JavaServer Facesista on saatavissa Sun Microsystemsin kehittämä referenssitoteutus nimeltä Mojarra ja Apache Foundationin MyFaces-toteutus. Yleisesti käytössä on vuonna 2006 julkaistua JSR 252:sta [BK06] vastaava JSF:n 1.2-versio, joka kuuluu Java EE 5 -ohjelmistokehitysalustaan.

2.6.2 RichFaces ja Ajax4jsf

RichFaces on komponenttikirjasto JSF-teknologiaan, joka mahdollistaa Ajax-toimintojen integroimisen Web-sovellukseen. Kirjastoa käytettäessä kehittäjän ei tarvitse kirjoittaa JavaScriptiä tai korvata jo olemassa olevia komponentteja Ajax-komponenteilla. RichFaces toteuttaa sivunlaajuisen Ajax-tuen perinteisen komponenttitasoisen tuen asemesta. Kehittäjä määrittelee tapahtuman, joka käynnistää Ajax-pyynnön, ja alueet sivusta, jotka päivitetään JSF:n komponenttipuussa sen jälkeen, kun Ajax-pyyntö on muuttanut palvelimella dataa käyttäjän tekemän toiminnon perusteella. Ajax-pyynnöt tehdään XMLHTTPRequest-funktioilla käyttäen viestien välittämiseen XML:ää. [JBo09]

RichFaces-kirjaston juuret löytyvät alkujaan Alexander Smirnovin Exadel-yrityksessä kehittämästä Ajax4jsf-kirjastosta, josta julkaistiin ensimmäinen versio vuonna 2006. Myöhemmin samana vuonna projekti jaettiin avoimen lähdekoodin Ajax4jsf- ja kaupalliseen RichFaces-komponenttikirjastoihin. Vuonna 2007 JBoss ja Exadel tekivät yhteistyösopimuksen ja myös RichFaces julkaistiin avoimen lähdekoodin LGPL-lisenssin (Lesser General Public License) alla. Samalla yritykset tekivät päätöksen yhdistää JBoss Ajax4jsf ja JBoss RichFaces -kirjastot RichFaces-nimen alaisuuteen. Tuorein vakaa 3.3.2.SR1-versio RichFacesista julkaistiin lokakuussa 2009 ja se tukee JSF 1.2 -versiota. Tuleva RichFaces 3.3.3-versio tuo tuen myös JSF:n 2.0 -versiolle. JBoss on nykyään osa Red Hatia.

3 Prototypointimenetelmät ja Web-sovellukset

Prototypoinnin näkökulmasta Web-sovellukset ovat vain yksi ala, joiden suunnitteluun ja kehitykseen prototypointia voidaan soveltaa. Verkkosovelluksen ja sen käyttöliittymän prototypointi seuraa suurilta osin samaa prosessia, kuin perinteisissä työpöytäsovelluksissa ja siinä voidaan hyödyntää samoja menetelmiä ja tapoja tehdä asioita. Luonteeltaan Web-sovellukset ovat kuitenkin erilaisia ja prototypoidessa on otettava huomioon asioita, jotka ovat niille ominaisia ja tuovat omat piirteensä sekä käytettäviin menetelmiin että työkaluihin.

Web-sovellusten prototypointi on pääpiirteittäin samoista lähtökohdista alkavaa ja samoja prototypointimenetelmiä hyödyntävää kuin perinteisten työpöytäsovellustenkin prototypointi: on mahdollista käyttää erilaisia menetelmiä ja lähestymistapoja, joilla haluttua asiaa mallinnetaan. Eroja sovellusten mallintamisessa kuitenkin on ja ne löytyvät sovellusten käyttöympäristöstä ja teknologioista. Web-sovellusten riippuvuus Internet-teknologioista antaa toisaalta mahdollisuuksia mallien luomiseen, mutta asettaa myös rajoitteita käytettävien menetelmien ja työkalujen käyttämiselle.

Sovellusten suunnittelun ja kehityksen tueksi on tutkittu erilaisten mallien, ohjelmointikielien ja suunnitteluympäristöjen hyödyntämistä. Esimerkiksi WEBRatio-ympäristö perustuu WebML (Web Modeling Language) -kielellä sovelluksen ulkoasun ja navigaation kuvaamiseen ja sen generoimiseen, Conallen laajentaa UML (Unified Modeling Language) -notaatiota mallintamaan Web-elementtejä ja kuvaamaan koko sovellusta UML:llä, MODFM (Mockup-driven Fast-prototyping Methodology) -menetelmä avustaa kehittäjiä tuottamaan nopeasti prototyypin ja mallitoteutuksen, ja JWeb- ja siitä jalostettu WARP (Web Application Rapid Prototyping) -ympäristö tarjoavat tukea koko sovelluksen luomiseen ja mallin ylläpitämiseen [BF04].

Kaikilla eri tutkimusten ratkaisuilla on yhteistä se, että ne koittavat yhdistää monta asiaa samaan kokonaisuuteen ja tekevät samalla prototypoinnista turhan monimutkaista, vaikka tarjoavatkin välineitä Web-sovellusten nopeampaan suunnitteluun ja kehitykseen. Web-sovellusten mallintaminen on teknologisista näkökulmista katsoen yksinkertaista ja se voidaan toteuttaa hyödyntämällä ja soveltamalla perinteisiä alhaisen, sekoitetun ja korkean tarkkuuden prototypointimenetelmiä ilman monimutkaisia prosesseja tai ympäristöjä.

3.1 Alhaisen tarkkuuden mallit Web-sovelluksen näkökulmasta

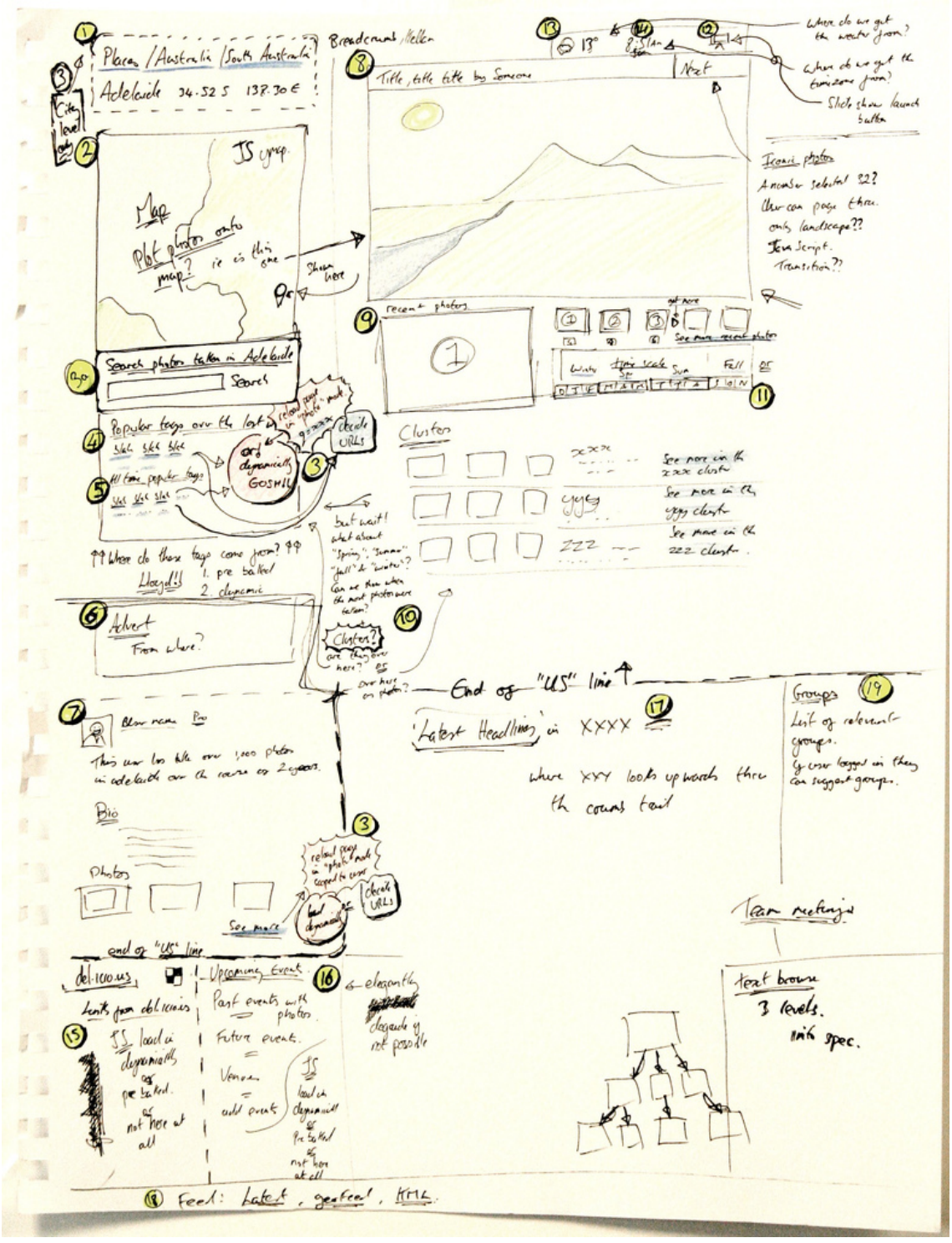
Tekniikoiltaan yksinkertaiset alhaisen tarkkuuden mallit soveltuvat hyvin myös verkkosovellusten prototypointiin, sillä niiden komponentteja ja siirtymiä eri näyttöjen välillä on yksinkertaista hahmotella. Alhaisen tarkkuuden malleja ei kannata välttämättä kovin tarkalla tasolla kuvata, sillä tarkoituksena on vain suurin piirtein kuvata eri asiat, ja käytettävissä oleva aika kannattaa panostaa erilaisten luonnosten tekemiseen. Malleja luodessa on hyvä muistaa, että asioita joita ei voida teknisesti toteuttaa, ei malliin kannata hahmotella.

Samasta sovelluksen osasta kannattaa rakentaa erilaisia luonnostelmia, sillä asioita voidaan toteuttaa monella tapaa, ja vasta kun asia on siirretty ajatuksista enemmän visuaalisempaan muotoon, voidaan arvioida sen toimivuutta. Vaikka malleilla ei saada riittävää tarkkuutta käytettävyyden tai toiminnallisuuden kunnolliseen arviointiin, saadaan sovelluksen vaatimukset ja pääominaisuudet selvitettyä, joten ne tukevat hyvin sovelluksen määrittelyä ja suunnittelua.

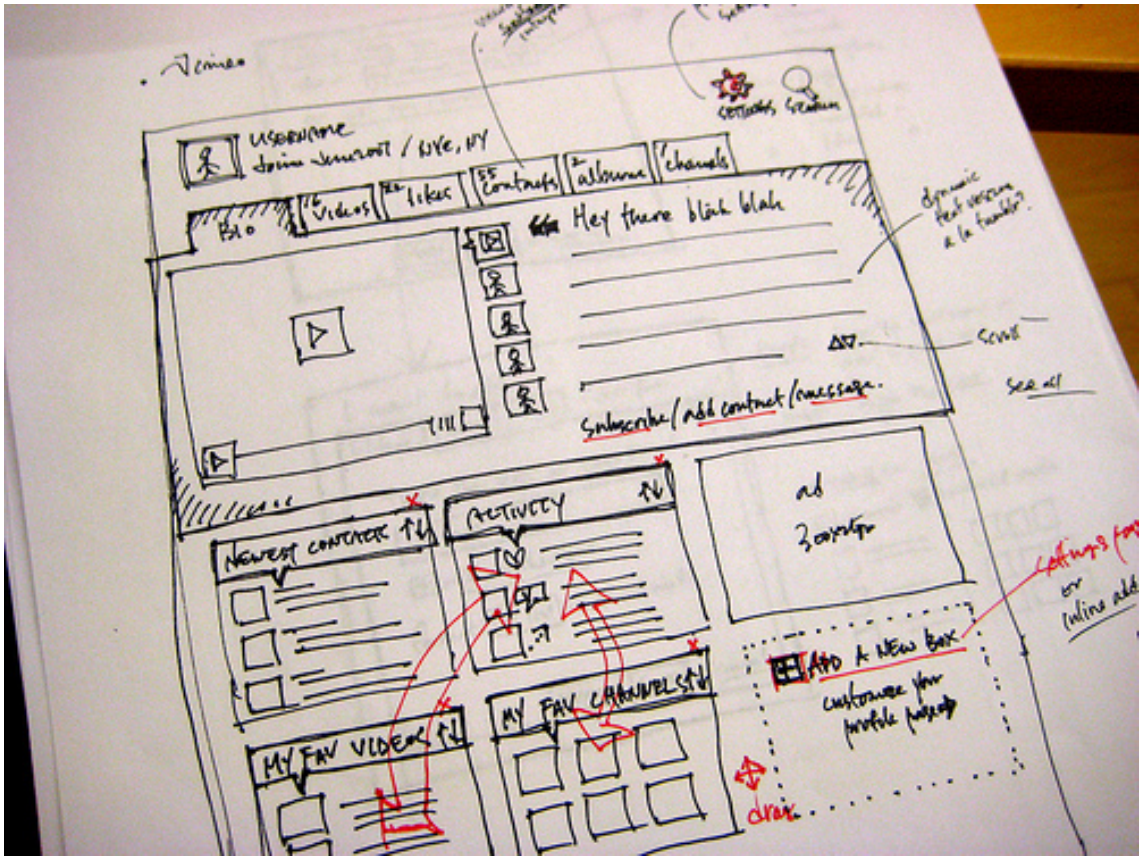
3.1.1 Luonnostelu ja paperiprototyypit

Luonnostelu on yksi kätevä menetelmä asioiden esittämiseen ja sitä voidaan hyödyntää sekä paperimallien että graafisten mallien kautta. Luonnostelu tukee sitä ajatusta, että ideoita kannattaa alkuun hahmotella karkealla tasolla, jolloin saadaan sivun rakenne luotua ja tärkeimmät elementit määriteltyä. Karkeita luonnoksia on helppo tarkentaa tai hylätä tarpeen mukaan, joten erilaisia vaihtoehtoja on vaivatonta tutkia. Esimerkiksi navigoinnin sijoittaminen, sivun palstojen määrä ja ylä- ja alapalkkien rakenne vaikuttavat suurelta osin sovelluksen ulkonäköön ja vaihtoehtoisia rakenteita on useita.

Yksinkertaisimmillaan luonnostelu käy “kynä ja paperi”-menetelmällä käyttäen mahdollisesti apuna pohjia, joissa taustalla on selainikkuna ja ruudukkoja. Paperiprototypointi on yleinen tapa toteuttaa malleja Web-sovelluksista ja muun muassa Flickr-kuvapalvelusta (http://flickr.com/) löytyy useita kuvia paperille hahmotelluista verkkosivuista ja -palveluista: kuvassa 3 näkyy Dan Cattin luonnos Flickr Place -osiolle ja kuvassa 4 Sockyung Hongin Vimeo-videopalvelun profiili-sivun idea. Kuten kuvista voi päätellä, ovat luonnosten tarkkuus, selkeys, sisältö ja tekniikka vaihtelevaa. Vaikka tekninen toteutus perustuu samaan “kynä ja paperi”-menetelmään, ei käytettävä tekniikka rajoita suunnittelijan ideointia.



Kuva 3: Dan Cattin luonnos Flickr Place -osiosta [Cat07]



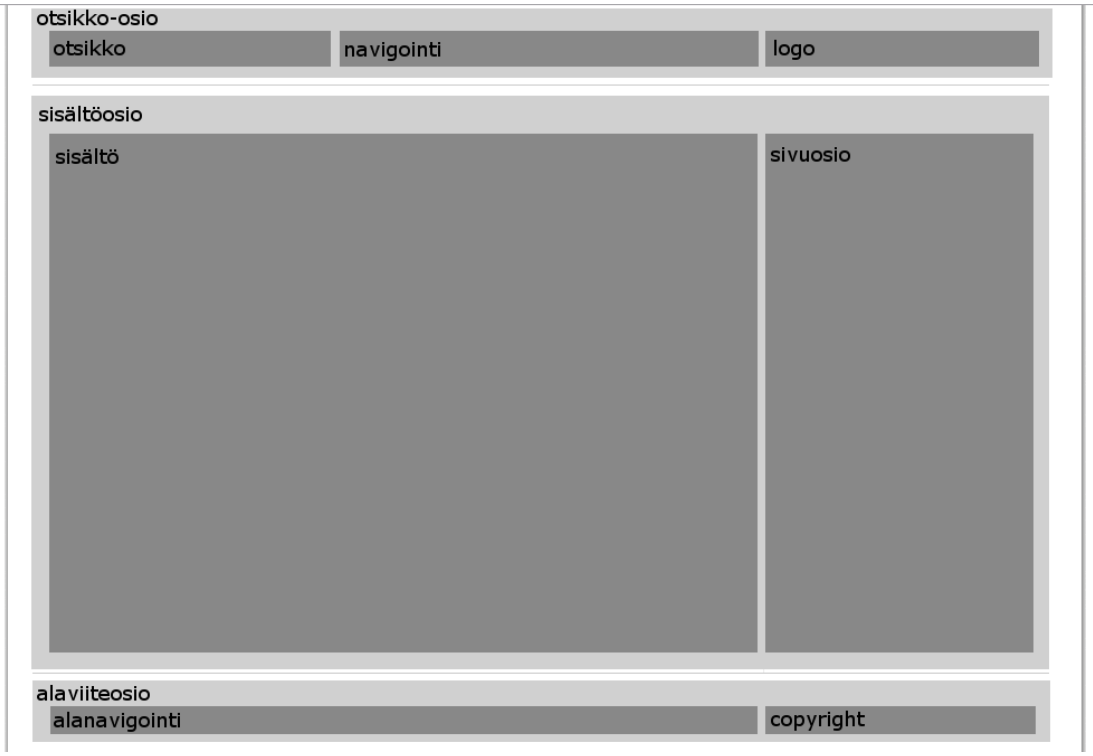
Kuva 4: Sockyung Hongin luonnos Vimeon profiili-sivusta [Hon08]

3.1.2 Luonnostelu ja rautalankamallit

Luonnostelua on mahdollista toteuttaa myös tietokoneavusteisesti tekemällä graafisia malleja, joiden luominen on paperille piirtämiseen verrattuna työläämpää, mutta jotka tarjoavat paremmat mahdollisuudet yksityiskohtien tarkentamiseen. Malleja voidaan rakentaa graafisen ilmeen kuvaamiseen tai esittämään kokonaisuutta harmaan rautalankamallin avulla. Pelkistetyn mallin avulla huomio ja mielipiteet eivät kiinnity vain graafisiin yksityiskohtiin, vaan voidaan keskittyä elementtien kokoihin, sijoitteluun ja rakenteeseen. Kuvassa 5 on esitetty sovelluksen näytön jakaminen eri osiin pelkistetyn rautalankamallin avulla. Mallista saadaan enemmän irti, kun siihen mallinnetaan tarkemmin eri asioiden sijoittuminen ja niiden koot suhteessa kokonaisuuteen.

Tietokoneavusteisia malleja voidaan toteuttaa useilla eri työvälineillä kuten Microsoft Visiolla, johon on saatavilla kolmannen osapuolen laajennuksia, jotka tuovat malleihin mahdollisuuden luoda navigoitavia kokonaisuuksia. Malleja on mahdollista toteuttaa myös ‘kynä ja paperi” -menetelmää simuloiden esimerkiksi DENIM-työkalulla, jolla lisäksi

saadaan paperinomaisesta lopputuloksesta luotua navigoitava malli. Yksinkertaisten rautalankamallien luonti onnistuu myös kuvankäsittelyohjelmalla, mutta parempaan lopputulokseen päästään käyttämällä joko julkisivutyökalua tai erikoisohjelmistoja, jolloin lopputulos on usein siistimpi ja helpommin muokattavissa.



Kuva 5: Sovelluksen eri osien sijoittuminen pelkistettynä mallina

3.2 Sekoitetun ja korkean tarkkuuden mallit Web-sovelluksen näkökulmasta

Web-sovellukset toimivat suhteellisen yksinkertaisessa käyttöympäristössä ja niiden prototypoiminen myöhäisillä, eli sekoitetun ja korkean tarkkuuden malleilla, on sekä helppoa että myöhemmin hyödyllistä. Malleja ei kuitenkaan kannata toteuttaa liikaa, jolloin niihin käytetty aika ja vaiva ei välttämättä ole saavutettujen hyötyjen arvoista. Käytettävät menetelmät ovat hyödyllistä kohdentaa lähelle sovelluksen teknologioita, eli HTML-ympäristöön tai lopulliseen teknologiaan, jolloin toteutettavia malleja voidaan mahdollisuuksien mukaan hyödyntää myös lopullisessa toteutuksessa. Esimerkiksi Java EE -ympäristön JSF-teknologiat tuottavat lopputuloksena normaaleja JavaScriptillä tehostettuja HTML-sivuja.

Verkkosovelluksen tavoin toimivalla mallilla on helpompaa kertoa eri osapuolille miten sovelluksen on ajateltu toimivan, näyttämällä suunniteltu vuorovaikutus. Toimiva malli nopeuttaa myös toteutusta ja alentaa kehityskustannuksia, kun asioiden haluttu toimivuus ja työnkulku ovat selkeästi havainnoitavissa, eivätkä kuvattuna vain dokumenteissa tai staattisina kuvina [Sta03]. Sekoitetun ja korkean tarkkuuden malleissa suunnittelija on myös pakotettu huomioimaan mitä oikeasti on mahdollista toteuttaa käyttöliittymän rakenteen ja komponenttien suhteen. Ottamalla huomioon lopullisen sovelluksen tarjoamat komponentit ja ominaisuudet, vastaa prototyyppi monilta osin paremmin todellisuutta kuin esimerkiksi luonnosteltu malli.

3.2.1 HTML-ympäristön hyödyntäminen

Web-sovelluksen teknologia perustuu HTML-merkkauskieleen ja CSS-tyyleihin, jotka ovat luonteeltaan yksinkertaisia ja niillä on kohtalaisen nopeaa luoda käyttöliittymiä. Kun sovelluksen käyttöliittymän hahmotelma on saatu luonnosteltua aikaisen mallien tekniikoilla, voidaan mallien pohjalta toteuttaa sovelluksesta HTML-malli, ellei sellaista jo aikaisemmin rakennettu. Riippuen mitä tekniikoita aikaisemmissa vaiheissa käytettiin, voidaan malliin nyt lisätä HTML-komponentteja, interaktiivisuutta ja visuaalista ilmettä, jolloin suunniteltavasta sovelluksesta saadaan parempi näkemys. HTML-prototypointitekniikalla saadaan käytännössä toteutettua sovelluksen perusrakenteet tyylimäärityksineen, joiden päälle toteutusta on myöhemmässä vaiheessa helppo rakentaa.

HTML-ympäristössä toteutettujen prototyyppien etuna on parempi vuorovaikutuksen esittäminen ja mahdollisuus käyttää niitä toimivana määrityksenä. Lopullista sovellusta lähellä olevassa mallissa on myös helppo nähdä miltä paperilla suunnitellut asiat näyttävät käytännössä ja onko toteutus mahdollista. Jos alhaisen tarkkuuden luonnoksissa malli toteutettiin vain paperilla tai rautalankamalleilla, on tällöin helppo unohtaa asioiden toteutuskelpoisuus [Dim06]. HTML-malli tarjoaa myös mahdollisuuden arvioida laatuominaisuuksia paremmin, sillä käytettävyyden osalta päästään kokeilemaan muun muassa sivuilla liikkumista ja voidaan mallintaa osaa toiminnallisuuksista.

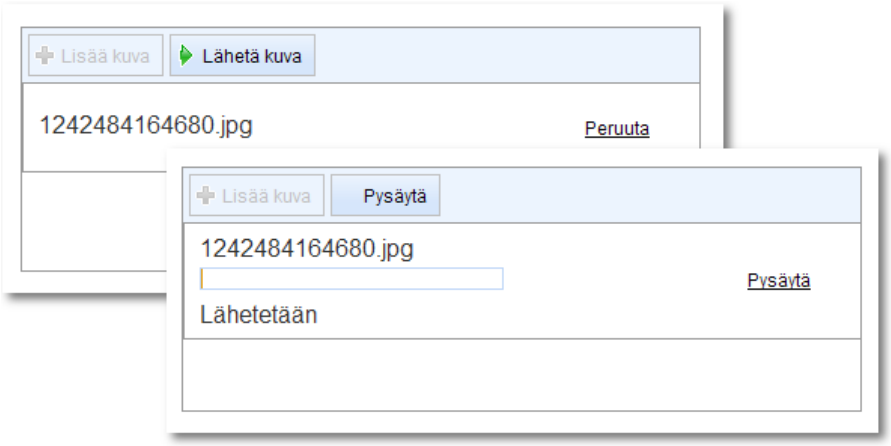
Sovelluksen vuorovaikutusta ja toimintoja kuten tiedostojen lisäämistä, tietojen tallentamista ja poistamista, työn kulkua, järjestelmän ilmoituksia ja eri toimintojen ilmenemistä voidaan mallintaa kuten ne lopullisessa sovelluksessa tulisivat toimimaan, käyttämällä JavaScriptiä ja Ajaxia. Mallin arvioinnissa on kuitenkin muistettava, että apuna käytettävät erilaiset JavaScript-kehykset kuten jQuery, MooTools ja Prototype eivät välttämättä

vastaa toiminnallisuuksiltaan ja ulkonäöltään lopullisen sovelluksen tekniikoita.

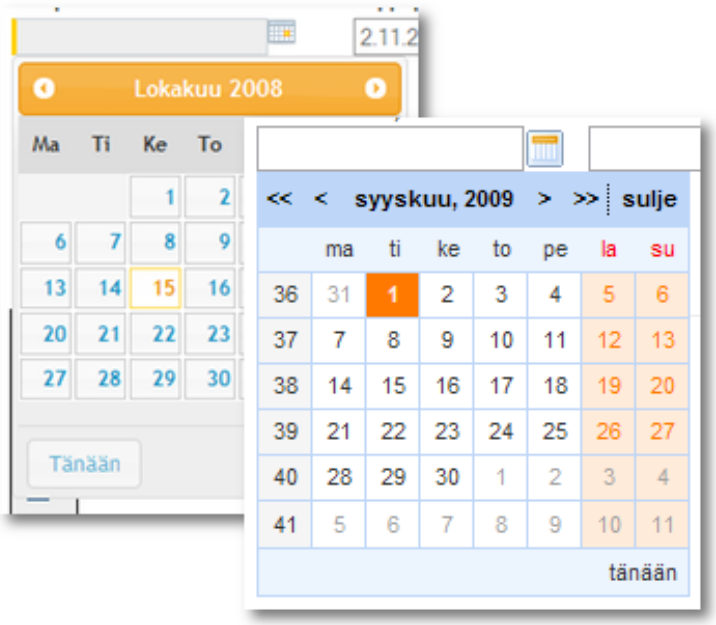
Mallinnettaessa perinteisiä sovelluksia HTML-tekniikoilla, saa arvioija selkeämmin mielikuvan, että malli on prototyyppi, eikä varsinainen systeemi. Web-sovelluksen tapauksessa HTML-malli on vastaavuudeltaan lähellä sitä, mitä oikea sovellus tulee olemaan, kun molemmissa käytetään samaa esityskerrosta, HTML-merkkausta ja -teknologioita, asioiden esittämiseen. Esiteltäessä mallia asiakkaille ja päättäjille on tärkeää painottaa, että malli on vain prototyyppi, jolloin asioista ei synny vääriä mielikuvia esimerkiksi sovelluksen toteutuksen aikataulun ja ominaisuuksien suhteen.

3.2.2 HTML-malli ja Java EE -ympäristön huomioiminen

Prototypoidessa sovellusta HTML-tekniikoilla on muistettava huomioida toteutuksessa käytettävien teknologioiden vaikutukset komponentteihin ja niiden ulkoasuun. Valitsemassamme Java EE -ympäristössä RichFaces-komponenttien mallintaminen esimerkiksi jQueryä hyödyntäen onnistuu kohtalaisen helposti, mutta sekä ulkonäöllisesti että toiminnallisesti ei päästä aivan samaan lopputulokseen. Etenkin monimutkaisten Ajax-komponenttien prototypointi ilman taustalta löytyvää varsinaista teknologiaa ja palvelinpuolen tukea on työlästä. Esimerkiksi RichFaces tarjoaa kuvassa 6 näkyvän “tiedostojen lisäys”-komponentin, jonka ulkoasua ja asynkronista toimintaa on hankala korvaavalla tekniikalla esittää ja vastaavasti kuvan 7 kalenterikomponentin, sekä validoinnin ja ehdotuslaatikon suhteen päästään aika lähelle oikeaa lopputulosta. Myös valitun teknologian rajoitteet kuten lomakkeiden toimiminen vain HTTP-protokollan POST-metodilla, kenttien validointiin ja sivun osittaiseen päivitykseen liittyvät asiat on hyvä pitää prototypoidessa mielessä.



Kuva 6: RichFacecs -kirjaston “tiedostojen lisäys”-komponentti



Kuva 7: jQuery ja RichFaces -kirjastojen kalentereissa on lähinnä ulkonäöllisiä eroja

Sovelluksessa käytettävät teknologiat eivät rajoita HTML-mallien käyttökelpoisuutta, sillä toimintaa ja ulkonäköä päästään arvioimaan riittävällä tarkkuudella, kunhan toteutettavien toimintojen suhteen muistetaan lopullisen sovelluksen ja käyttöliittymäkomponenttien mahdollisuudet ja rajoitteet. HTML-tekniikoilla rakennettua mallia voidaan hyödyntää myös lopullisessa toteutuksessa, sillä käyttöliittymän rakentaminen elementteineen ja tyylimäärityksineen on mallin avulla jo suurilta osin suoritettu. Malliin tarvitsee vain lisätä Java EE -teknologioiden määritteet ja rakentaa taustalle palvelinpuolen palvelut. Sovelluskehittäjä voi keskittyä toimintojen toteuttamiseen, kun käyttöliittymään liittyvät asiat ovat jo melkein valmiita.

3.2.3 Lopullisen sovelluksen teknologian hyödyntäminen

Lopullisen sovelluksen teknologiaa hyödyntävät prototyypit ovat yksi mahdollisuus, jolloin käyttöliittymä vastaa kaikilta osin lopullista sovellusta. Valitsemassamme Java EE -ympäristössä lopullisen sovelluksen teknologia on Java Server Faces, lisättynä RichFaces-käyttöliittymäkomponenteilla. Sovelluskehitysalustana voidaan käyttää esimerkiksi Eclipseä, joka tarjoaa mahdollisuuden myös koodigenerointiin.

Javalla toteutettu korkean tarkkuuden malli tarjoaa paremmat mahdollisuudet sovelluksen eri osa-alueiden arviointiin ja laadun varmistamiseen käytettävyyden, vuorovaikutuksen ja toiminnallisuuksien osalta. Tämä on hyödyllistä etenkin projekteissa, joissa toteutuksessa käytettävä teknologia on uutta, eikä ole varmuutta sen toiminnasta tai käyttäytymisestä. Kokonaisen sovelluksen prototypointi Javalla ei välttämättä ole järkevää ja sekoitetun mallin luominen yhdistelemällä Java EE -teknologioita ja pelkistettyä HTML:ää tai graafista esitystä voi olla käytännöllisempää. Tällöin voidaan testata sovelluksen osia, joiden toimivuudesta ja logiikasta on eniten kysymyksiä, ja joihin mallilla yritetään hakea vastauksia. Esimerkiksi voidaan toteuttaa asynkronisia toimintoja ja kuvata niiden käyttöliittymäkomponenttien toimintaa, joiden mallintaminen muuten on työlästä.

Korkean tarkkuuden mallin luominen lopullisen sovelluksen tekniikoilla on työläämpää ja hitaampaa, kuin kevyemmän HTML-tekniikalla tehdyn mallin, mutta koska valmis prototyyppi vastaa sovellusta, säästetään aikaa lopullisessa toteutuksessa. Javalla toteutettua mallia kannattaa harkita siinä vaiheessa, kun sovelluksen vaatimukset ja toiminta ovat jo hyvin selvillä. Mallin tekeminen vie resursseja ja asioiden muuttaminen on työlästä, jolloin mallin kustannukset kohoavat. On hyödyllistä miettiä, tarvitaanko korkean tarkkuuden toiminnallista mallia vai riittäisikö HTML-tekniikoilla toteutettu malli. Toisaalta kattavasti toteutettua mallia voidaan myös käyttää pilottijärjestelmänä ja kehittää sitä eteenpäin lopulliseksi sovellukseksi.

3.3 Suunnittelumallien käyttäminen prototypoinnissa

Käyttöliittymän prototypoinnin apuna voidaan hyödyntää myös ratkaisumalleja, eli design patternseja, jotka tarjoavat valmiita ratkaisuja yleisiin sovelluksista löytyviin toimintoihin. Asioita ei tarvitse, eikä kannata keksiä uudestaan, sillä useita työpöytäsovelluksista tuttuja vuorovaikutuksen toimintamalleja voidaan mahdollisesti hyödyntää myös Web-sovellusten puolella. Koska käyttäjä on tottunut tekemään asioita samalla tavalla, on toimivien mallien käyttäminen järkevää myös sovelluksen opittavuuden kannalta.

Käyttöliittymien kehittämisen avuksi löytyy useita eri suunnittelumallien lähteitä, joissa listataan sovelluksissa toteutettuja ratkaisuja. Muun muassa Tidwell [Tid05] ja Yahoo!:n suunnittelumalli-sivusto [Incon] käsittelevät käyttöliittymän vuorovaikutusmalleja sekä Web-sovellusten että työpöytäsovellusten osalta ja UI Patterns -sivusto [Tox09] keskittyy Web-sovellusten ratkaisuihin. Suunnittelumallien osalta näytetään yleensä yksi tai useampi esimerkki ja kerrotaan minkä ongelman se ratkaisee, milloin mallia pitäisi käyttää, mi-

kä ratkaisu on, miksi mallia pitäisi käyttää ja liittyykö siihen jotain erikoistapauksia. Hyvin perusteltua ja esiteltyä toimintoa on helppo arvioida ja soveltaa omaan toteutukseen. Erilaisia ratkaisuja löytyy jaoteltuna navigointiin, ulkoasuun, valintoihin, vuorovaikutukseen ja sosiaalisiin toimintoihin liittyen. Esimerkiksi monimutkaista tietoa voidaan esittää näyttämällä sekä yleiskuva että tarkempi kuva yhtä aikaa ja taulukon lajittelu voidaan toteuttaa otsikkokentistä klikkaamalla ja sarakkeita siirtämällä.

Suunnittelumalleja voidaan käyttää sekä alhaisen että korkean tarkkuuden malleissa, mutta helpointen valmiin mallin hahmottelu prototyyppiin onnistuu alhaisella tarkkuudella, kun varsinaisesta toiminnallisuudesta ei tarvitse vielä huolehtia. Tällöin pitää kuitenkin huolehtia, että valittu toiminto on mahdollista ja järkevää toteuttaa. Etenkin työpöytäsovelluksista tutut toimintamallit, kuten tiedostojen kanssa työskentelyyn ja paljon prosessointia vaativiin tehtäviin liittyvät, eivät välttämättä toimi sellaisenaan Websovelluksissa. Myös teknologiset rajoitteet vaikuttavat mallien käyttöön ja joidenkin asioiden kuten HTTP-protokollan GET-kutsuihin perustuvat toiminnot eivät valitussa Java EE -ympäristössä ole suositeltavia ja ovat työläitä toteuttaa.

Suunnittelumallien käyttämisessä pitää olla tarkkana, sillä erilaiset mallit eivät välttämättä sovellu käsiteltävään ongelmaan ja lukuisista malleista sopivan valinta voi olla hankalaa. On kuitenkin hyvä tuntea erilaisia malleja asioiden toteuttamiseen, sillä niiden avulla voidaan tutkia ja valita käsiteltävään ongelmaan parhaiten toimiva ratkaisu ja tarvittaessa soveltaa jo olemassa olevaa ratkaisua.

4 Sovelluskehitysprojekti ja prototypointi

Sovelluskehitysprojektissa eri osa-alueiden onnistuminen vaikuttaa lopputulokseen, eikä mikään osa ole varsinaisesti itsenäinen kokonaisuus: määrittely luo perustan suunnittelulle ja toteutus- ja testausvaiheita on hankala edistää ilman asianmukaisia suunnitelmia. Käyttäjän näkökulmasta sovelluksen lopputulos näkyy käytännössä käyttöliittymän kautta, joka on sen näkyvin ja konkreettisin osa ja vaikuttaa käytettävyydellään koko sovelluksen toimivuuteen. Tietenkään mikään sovellus ei ole pelkkää ulkokuorta, vaan visuaalisen ilmeen ja tuntuman taakse tarvitaan myös toimintoja. Kaikkien osa-alueiden toimiessa saumattomasti ja ilman miettimistä, on sovellus onnistunut, johon pääsemistä prototypoinnilla voidaan edistää.

Käyttöliittymän näkökulmasta prototypointi painottuu paljolti käytettävyyden, ulkoasun ja toiminnallisuuksien mallintamiseen, jotka ovat sovelluksen ominaispiirteistä selkeinten käyttäjälle näkyviä. Huomioimalla sovelluksen luonne ja prototypoinnin tarkoitus, on käytännöllistä valita lähestymistapa, joka vastaa kysymyksiin, joihin prototyypillä haetaan vastausta. Eri kysymyksiin löydetään vastaus helpommin aikaisilla malleilla ja toisiin kysymyksiin voidaan soveltaa sekoitettuja tai myöhäisiä malleja.

Karkeasti jaoteltuna käyttöliittymän ja sen toimintojen prototypoinnin voi jakaa kolmeen vaiheeseen: käyttöliittymän käsitteellisen mallin rakentaminen, uuden mallin prototyypin kehittäminen ja prototyypin arviointi. Näitä kolmea vaihetta toistetaan, kunnes haluttu lopputulos on saavutettu [Pet06]. Tässä työssä prototypointia käsitellään mallien kehittämisen ja siihen liittyvien asioiden näkökulmasta vesiputousmallin mukaan etenevässä sovelluskehitysprojektissa, jossa projektin jokainen vaihe suoritetaan portaittain edeten määrittelystä suunnitteluun, toteutukseen, testaukseen ja jatkokehitykseen.

4.1 Käyttöliittymän hahmottelu osana kehitysprojektia

Prototypoinnin hyödyntämistä tutkittiin sovelluskehitysprojektissa, jossa toteutettiin kuvien ja niihin liittyvien metadatojen tallentamiseen ja tietojen selaamiseen tarkoitettu Web-sovellus. Sovellus oli kokonaisuudessaan suhteellisen yksinkertainen ja pieni, joten prototypointia voitiin soveltaa koko sovelluksen mallintamiseen: näyttöjä oli vain muutamia ja toiminnot suhteellisen selkeitä. Sovelluksen käyttötapaukset rakennettiin vaatimusten pohjalta ja käyttöliittymän ja toiminnallisuuksien hahmottelemisessa hyödynnettiin osiltaan suunnittelumalleja.

Projekti eteni soveltaen Winston Roycen määrittelemää vesiputousmallia [Roy70] ja jokaisessa vaiheessa tehtiin sovelluksesta malli, jonka avulla tarkennettiin määrittelyyn, suunnitteluun ja toteutukseen liittyviä asioita. Prototypointia sovellettiin pääasiassa kehittäjälähtöisesti, eikä asiakas aktiivisesti osallistunut sovelluksesta tehtävien eri mallien arviointiin lukuun ottamatta projektikokouksissa mallien läpikäyntiä. Tämä ei ole ideaalitilanne, sillä prototypoinnista saadaan enemmän hyötyä, kun sitä sovelletaan ketterästi ja asiakas on aktiivisesti mukana mallien kehittämisessä. Vaikka teoria osoittaa useita hyödylliseksi todettuja tapoja tehdä asioita, on niiden soveltaminen käytännössä aina toinen maailma, ja tässäkin tapauksessa piti soveltaa niitä menetelmiä ja välineitä, joita tarjolla oli: yhdistää prototypointi vaiheittaiseen ohjelmistotuotantoprosessiin.

Alkuvaiheessa tutkittiin erilaisia mahdollisuuksia prototypoinnin suorittamiseksi menetelmien ja käytettävien työkalujen osalta ja päädyttiin keskittymään yksinkertaisiin menetelmiin mallien luomisessa, koska sovelluksen toteutuksessa käytettävä teknologia ei varsinaisesti ole monimutkaista. Eli käytännössä alhaisen tarkkuuden mallien osalta luonnostelua paperiprototyypeillä ja sekoitetun ja korkean tarkkuuden mallien osalta panostaen HTML-teknologian hyödyntämiseen. Näin prototyypeistä saataisiin mahdollisesti eniten irti myös toteutuksen näkökulmasta. Vaikka prototyyppien teko on projektissa lisäkustannus, arvioitiin siitä saatavat hyödyt suuremmiksi kuin mitä mallien tekemiseen menisi resursseja ja aikaa.

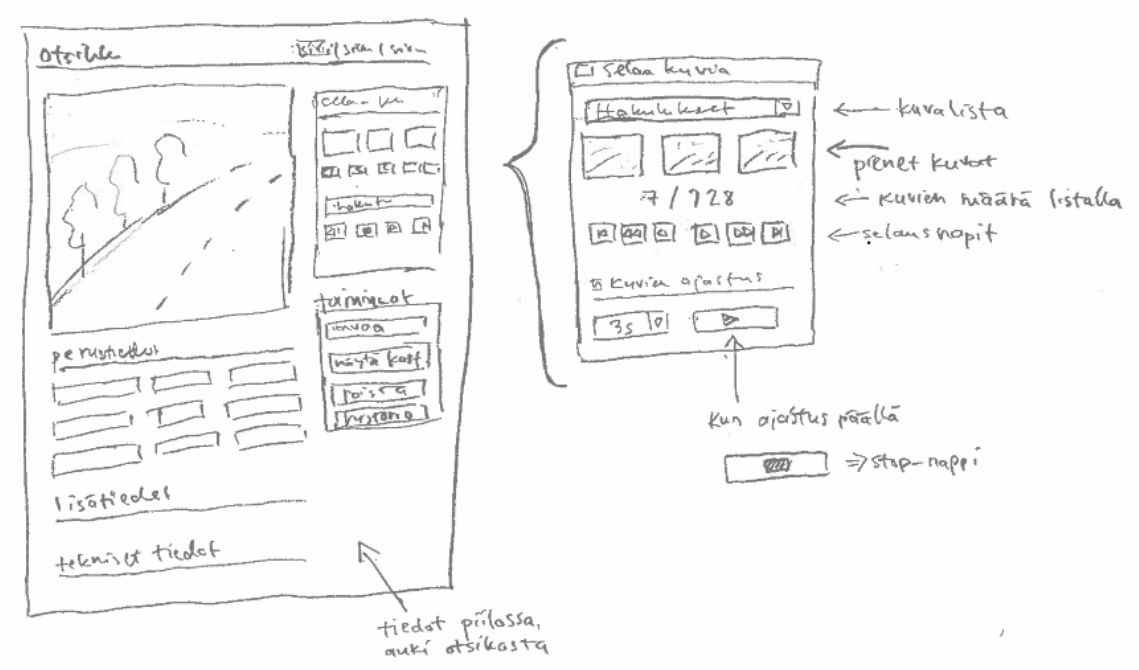
Sovelluksen mallintaminen päätettiin myös aloittaa aikaisessa vaiheessa, sillä prototypointia ei pitäisi jättää sovelluskehitysprojektin suunnitteluvaiheeseen kuten esimerkiksi vesiputousmallia soveltavassa projektityössä usein on tapana. Suunnitteluvaiheeseen päästäessä on jo tehty päätöksiä ohjelmiston toiminnoista ja rakenteesta, jolloin käyttöliittymään syntyy käytettävyysongelmia [Mäk05]. Jos käyttöliittymää päästään mallintamaan ja testaamaan mahdollisimman aikaisessa vaiheessa, päästään arvioimaan ohjelmiston hyödyllisyyttä ja käytettävyyttä hyvissä ajoin. Huonoimmassa tapauksessa käyttöliittymän arviointi jää toteutusvaiheeseen, jolloin suunnitteluvirheiden korjaaminen on vaikeampaa ja kalliimpaa.

4.1.1 Määrittelyvaihe

Määrittely- ja suunnitteluvaiheet olivat projektijohdollisesti omia vaiheitaan, mutta prototypoinnin näkökulmasta jo vaatimusten määrittelyn ja käyttötapausten kirjoittamisen yhteydessä tutkittiin eri ideoiden ja tarpeiden realisoimista luonnostelemalla sovelluksen

käyttöliittymää alhaisen tarkkuuden “kynä ja paperi” -menetelmällä ja osittain jo suunnittelemalla sovellusta. Sovelluksesta prototypoitiin muutamia näyttöjä, joiden mahdollisesta rakenteesta ja sisällöstä haluttiin saada parempi näkemys. Luonnoksia ei käsitelty projektipalavereissa asiakkaan kanssa, vaan ne toimivat vain taustatukena määrittelyä ja suunnittelua ajatellen.

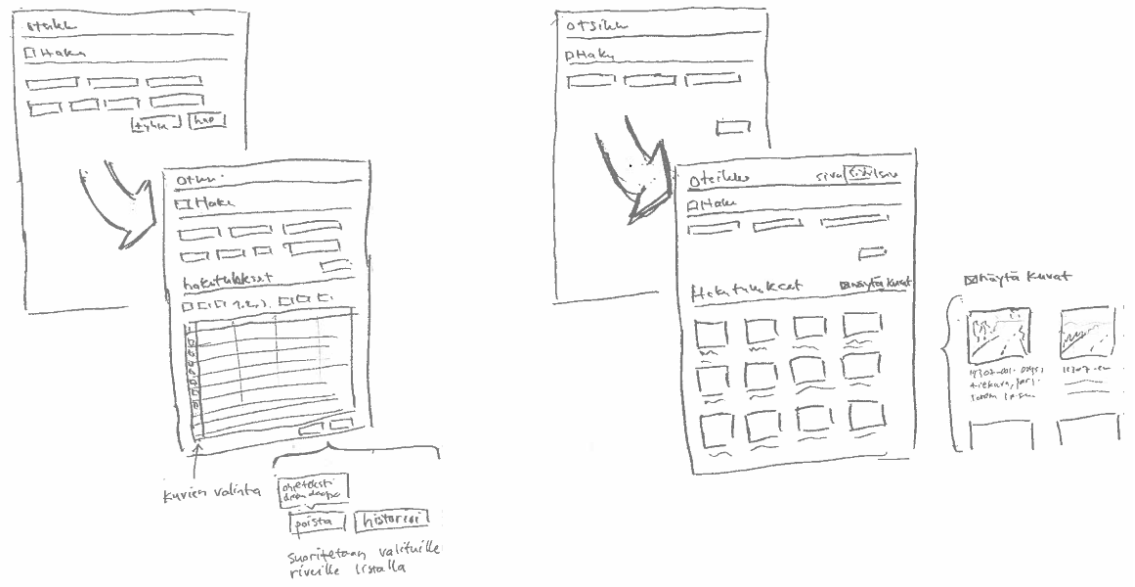
Paperiprototypointi valittiin menetelmäksi sen yksinkertaisuuden takia, jolloin eri ideoita saatiin visualisoitua nopeasti ja pienellä vaivalla. Tällöin ei käytettäisi aikaa turhaan asioiden tarkkaan mallintamiseen, jos ja kun määrittely tuo eteen uusia asioita ja tarpeet muuttuvat. Pääosin mallit olivat kuvassa 8 näkyvän “Selaus”-näytön luonnoksen tyylisiä, joissa oli piirretty karkealla tasolla näytön yleiskuva ja hieman tarkemmalla muita osia näytön rakenteesta. Malleja täydennettiin lisäksi muutamilla kommenteilla selventämään toimintoja ja rakennetta.



Kuva 8: Kuvien selaus -näytön luonnoksessa osa mallia on piirretty tarkemmin

Näytöistä ja niiden välillä siirtymisestä luonnosteltiin muutamia kuvakäsikirjamaisia malleja, joissa kuvassa 9 näkyy miten “Haku”-näytöllä sivun sisältö muuttuu “Hae kuvia”-toiminnon suorituksen jälkeen ja mitä vaihtoehtoisia tapoja esittää hakutuloksia on. Taustalla on käytetty tukena paperia, jossa on selainpohjia apuruudukon kanssa, jolloin malli suhteutuu paremmin kokonaisuuteen. Kuvassa 10 näkyy sama asia piirrettynä ilman mallipohjaa, jolloin siihen on saatu lisättyä paremmin kommentteja ja tarkennuksia yksityiskohdista.

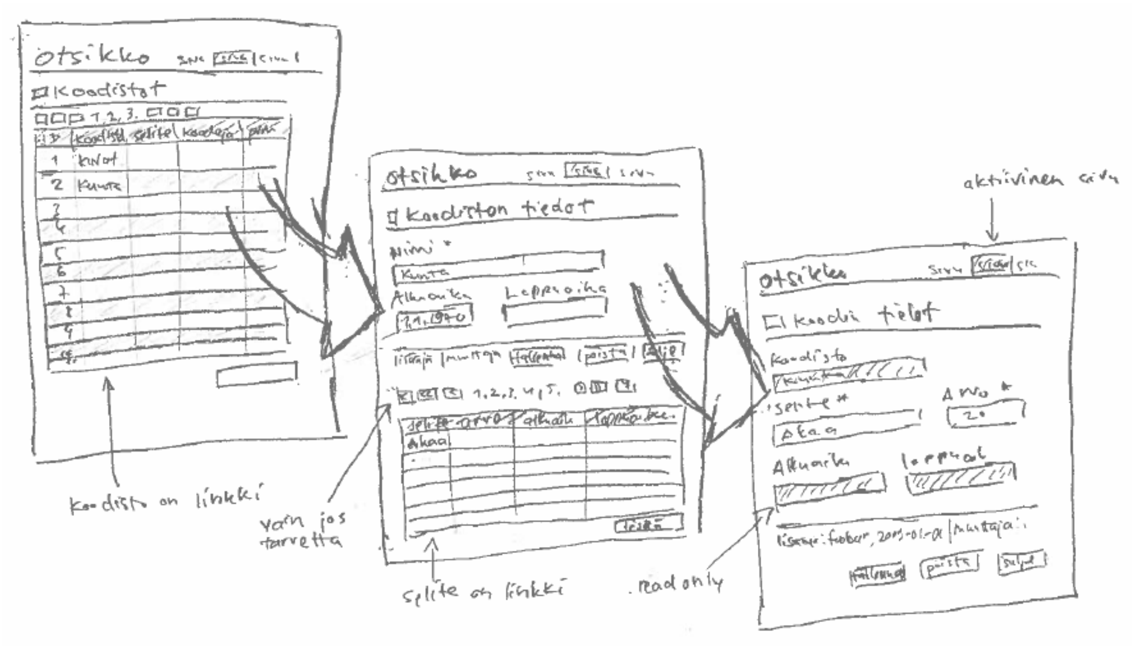
Kuva 9: “Haku”-näytön luonnos käyttäen taustalla apuna selainruudukkoa



Kuva 10: “Haku”-näyttö ja kaksi eri tapaa esittää hakutulokset

Kuvakäsikirjamaisia piirroksia hahmoteltiin myös tyhjälle paperille kuvassa 11 näkyvän “Koodisto”-osion tyyliin, jossa on kuvattu siirtymät päätasolta alatasoille ja tärkeimmät toiminnot, joita kyseisille sivuille on ajateltu. Tyhjälle paperille luonnosteltaessa ei tarvinnut rajoittua valmiiseen malliin, vaan pystyi vapaammin keskittymään ideointiin ja asioiden hahmotteluun.

Sovelluksesta toteutetut paperiprototyypit olivat luonteeltaan karkeita luonnoksia sivun rakenteesta ja mahdollisista toiminnoista. Mallit toimivat pääasiassa sekä määrittelijän että suunnittelijan, jotka tässä tapauksessa olivat sama henkilö, ideointivälineinä eri ratkaisujen ja vaihtoehtojen tutkimiseksi. “Kynä ja paperi” -menetelmän lisäksi kokeiltiin lyhyesti DENIM-ohjelman käyttämistä mallien tekemiseen, mutta se todettiin yleisesti



Kuva 11: Sovelluksen “Koodisto”-osio kuvakäsikirjamaisesti

ottaen ja etenkin hiirellä käytettynä turhan hankalaksi mallien piirtämiseen, vaikka mallien myöhempi säilöminen olisikin onnistunut sähköisesti helpommin.

Vaatimusmäärittelyn loppuvaiheessa aloitettiin mallien rakentaminen toteuttamalla hahmotellut paperimallit yksinkertaisina rautalankamaisina HTML-malleina, jolloin ideat saatiin paremmin kuvattua ja niitä oli helpompi esitellä projektikokouksissa. HTML-teknologian valinta oli loogista, sillä sovelluksen teknologia perustuu HTML-merkkaukseen, ja näin jo tehtyä työtä sovelluksen rakenteen osalta ei tarvitse heittää hukkaan. Rautalankamalleihin lisättiin määrittelyn loppuvaiheessa tyylimäärityksiä, joilla malleihin saatiin enemmän visuaalisuutta ja asioiden hahmottaminen helpottui. Jo määrittelyvaiheessa rakennettujen yksinkertaisten HTML-mallien avulla asiakas sai tekstidokumentaation tueksi kuvan sovelluksesta ja siihen ajatelluista asioista.

4.1.2 Suunnitteluvaihe

Käyttötapausten ja paperimallien pohjalta toteutettuja HTML-malleja jalostettiin suunnitteluvaiheessa tarkentamalla näyttöjen sisältämiä tietoja, toteuttamalla malleihin graafinen ilme ja mallintamalla sovelluksen toimintaa. Suunnitteluvaiheessa prototyyppejä käytiin läpi projektipalavereissa ja niitä tarkennettiin kommenttien perusteella keskittyen lähinnä eri kenttien sijaintiin ja näyttöjen sisältöön. Sovelluksen tietosisällöstä määriteltiin tar-

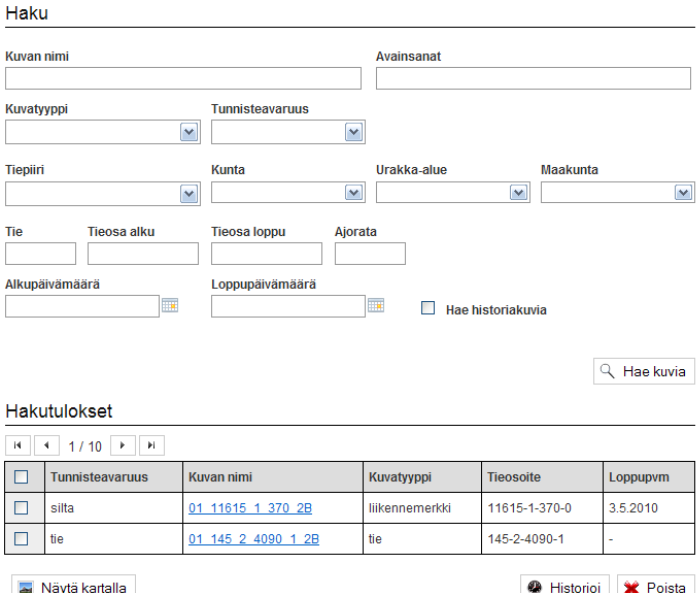
kemmin kuvista esitettävien metatietojen laajuutta ja luokittelua tärkeysasteen perusteella. Muun muassa EXIF-otsikkotietojen näyttämistä käyttöliittymässä rajattiin ja tiedot luokiteltiin kolmeen eri kategoriaan: ensisijaiset tiedot, toissijaiset tiedot ja tekniset tiedot.

Suunnitteluvaiheessa tehtiin päätös, että koska sovellus on kooltaan pieni ja käytettävä teknologia tukee lopullista toteutusta merkkauskieleltään, toteutetaan HTML-mallit jokaisesta näytöstä. Kaikkiaan sovelluksessa oli seitsemän erilaista näyttöä: haku, selaus, lisäys (2 näyttöä) ja koodistot (3 näyttöä). Tietosisällön ja käyttöliittymän rakenteen esittämisen lisäksi sovelluksen toimintoja mallinnettiin käyttämällä jQuery-kirjastoa, jolla saatiin HTML-malliin interaktiivisuutta muun muassa navigoinnin, siirtymien, järjestelmän viestien ja toimintojen simuloimisen osalta. Suunnitteluvaiheen jälkeen valmiina oli käyttöliittymän osalta sovelluksen toimintaa mallintava HTML-prototyyppi, jolla pystyi esittämään sovelluksen päätoiminnot.

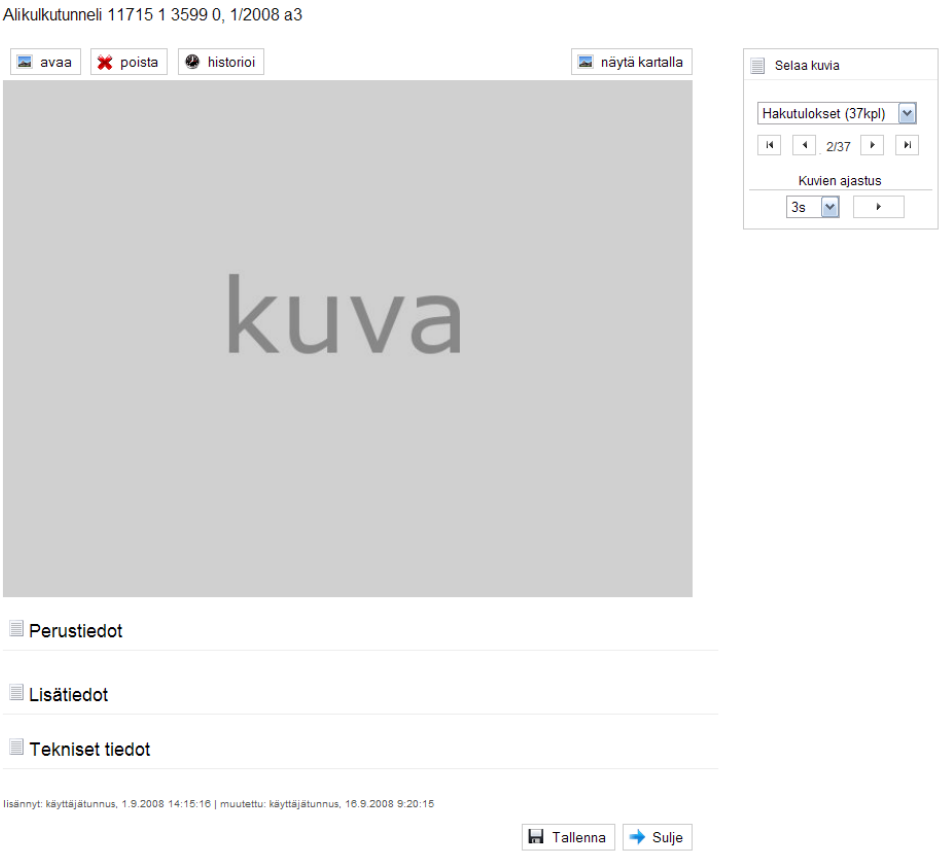
HTML-malleja ja visuaalista ilmettä toteutettaessa piti huomioida myös projektin reunaehdot käytettävien Web-selaimien suhteen, joka rajoitti muun muassa CSS-tyylimääritysten käyttämistä. Koska Internet Explorerin versio oli rajattu 6.0 -sarjasta ylöspäin, toteutettiin tyyliominaisuudet käyttäen pääasiassa CSS 1 -luokan määrityksiä, mutta osiltaan hyödyntäen CSS 2 -luokan määrityksiä siten, että ne eivät vaikuttaneet epäedullisesti sovellusta käytettäessä niitä tukemattomissa selaimissa. Lisäksi läpinäkyvyyttä sisältävien PNG-kuvien käytön osalta toteutettiin oma tyylitiedosto Internet Explorer 6.0 -sarjan selaimille, joissa kuvat oli määritelty GIF-muodossa.

Toteutettujen mallien osalta rakennettiin esimerkiksi sovelluksen “Haku”-sivusta kuvassa 12 näkyvä HTML-malli, jossa näytön alaosassa näkyvä Hakutulokset-osio oli alkuun piilotettuna ja se tuli näkyviin vasta kun käyttäjä täytti ennalta määriteltyyn hakukenttään jotain ja painoi “Hae kuvia”-nappia. Haun kestoa simuloitiin näyttämällä haun aikana ilmoitustekstiä näytön ylälaidassa muutaman sekunnin ajan. Eli “Haku”-näytön osalta malli kuvasi sovelluksen toimintaa lähes kuten lopullisessa toteutuksessakin. Vietäessä hiiren kursori Hakutulos-listauksen alta löytyvien “Historioi”- ja “Poista”-nappien päälle, sai käyttäjä opasteen nappien toiminnoista.

“Haku”-näytöltä käyttäjä pystyi siirtymään kuvassa 13 näkyvälle “Selaa”-sivulle, jossa esitettiin kuvaan liittyvät tiedot ja toiminnot. Ylälaidassa olevat toimintopainikkeet antoivat ilmoituksen toiminnon suorittamisesta ja “Avaa”-nappi aukaisi kuvan isompana ruudulle. Kuvaan liittyvät tiedot avautuivat näkyviin “Perustiedot”, “Lisätiedot” ja “Tekniset tiedot” -otsikoista klikkaamalla. Malliin ei toteutettu selaukseen liittyvää interaktiota, mutta muiden toimintojen tapaan myös sen olisi voinut JavaScriptillä mallintaa.



Kuva 12: “Haku”-näytön HTML-malli kuvasi hyvin sovelluksen aiottua toimintaa



Kuva 13: “Selaus”-näytöllä kuvaan liittyvät tiedot ovat normaalisti piilotettuina

4.1.3 Web-sovelluksen laatu ja käytettävyyden asiantuntija-arvio

Sovellettaessa prototypointia sovelluskehitysprojektin aikana, haluttiin saada aikaa parempi lopputulos sekä käytettävyyden että toiminnallisuuksien osalta, joten suunnitteluvaiheen loppupuolella toteutettiin sovellukselle käytettävyyden asiantuntija-arviointi käyttäen siinä HTML-prototyyppiä.

Asiantuntija-arvio toteutettiin heuristisin menetelmin projektin ulkopuolisen asiantuntijan toimesta. Heuristisessa arvioinnissa käyttöliittymä arvioidaan suunnittelun perussääntöjen pohjalta, ja kun varhaiselle prototyypille tehdään heuristinen arviointi, käytettävyysongelmat voidaan löytää jo aikaisessa vaiheessa, jolloin ongelmien korjaaminen on tehokasta sekä ajankäytön että kustannusten suhteen [Nie07]. Asiantuntija-arvio ei korvaa käyttäjän kanssa suoritettavia testejä, sillä se ei ota kantaa järjestelmän sopivuuteen aiottuun tehtävään eli järjestelmän toimivuuteen käyttäjän näkökulmasta.

Arvio toteutettiin palvelun käytettävyyden heikkouksien ja vahvuuksien löytämiseksi ja lähtökohdiksi palvelun jatkokehitystä varten. Sen avulla sovelluksesta tunnistettiin loppukäyttäjille hankalia osa-alueita ja saatiin suosituksia niiden korjaamiseksi. Käytettyä heuristiikkaa ei raportissa kerrottu, mutta heuristiikkana voidaan käyttää muun muassa Nielsenin esittämiä kymmentä sääntöä [Nie05].

Asiantuntija-arviossa havaitut käytettävyysongelmat olivat jaoteltu neljään eri tasoon:

1. **Vakava käytettävyysongelma:** Vaikeuttaa merkittävästi käyttöä ja/tai aiheuttaa väärinkäsityksiä. Ongelma, jota ei voi järjestelmässä välttää. Saattaa estää loppukäyttäjää käyttämästä järjestelmää tai kehottaa käyttämään sitä väärin. Suositellaan korjattavaksi.

2. **Käytettävyysongelma:** Estää tehokkaan tehtävän suorittamisen. Ongelman voi järjestelmässä välttää kiertämällä. Tekee järjestelmän käytöstä loppukäyttäjälle kohtuuttoman vaikeaa. Suositellaan korjattavaksi, jotta järjestelmästä tulisi helpompi ja nopeampi käyttää.

3. **Pieni käytettävyysongelma / kosmeettinen virhe:** Ei estä käyttäjän toimintaa, mutta hankaloittaa sitä. Ongelman voi järjestelmässä välttää kiertämällä. Aiheuttaa loppukäyttäjälle hämmennystä ja ärtymystä. Suositellaan korjattavaksi, jotta käyttäjäkokemuksesta muodostuisi positiivinen.

4. **Ei käytettävyysongelma:** Voidaan pitää kysymyksenä tai huomionarvoisena kehitysehdotuksena.

Havaitut käytettävyysongelmat jaottuivat kaikki tasoille 2 ja 3 eli asioita oli saatu hyvin ratkaistua määrittelyn ja suunnittelun aikana. Arvion avulla käsiteltiin suurimmat ongelmat liittyen Web-sovelluksen laatuominaisuuksiin ja saadut tulokset voidaan jakaa kuvassa 1 määritellyn luokittelun perusteella kolmen tärkeimmän osa-alueen suhteen.

Käytettävyys
Käytettävyyden osalta kiiteltiin online-palautteen ja opastuksen osalta muun muassa käyttäjälle annettavien ilmoituksia, Tool tip -ohjeita ja selkeitä toimintoja. Sivuston ymmärrettävyydestä arvioitiin positiivisesti ulkoasun selkeys ja minimalistisuus, mutta arvosteltiin negatiivisesti käytettyjen termien osalta, jotka kuuluivat sovelluksen erityispiirteisiin. Myös käyttöliittymän laatu sai arvostelua, sillä käytettävyysongelmiksi luokiteltiin asioita, jotka johtuivat prototyypistä: kaikki asiat eivät toimineet kuten oli ajateltu. Arvion perusteella tehtiin muutoksia muun muassa kenttien aktiivisuuden visualisoimiseksi ja muuttamalla painikkeiden järjestystä.

Toiminnallisuus
Sovelluksen toiminnallisuus arvioitiin pintapuolisesti. Koska arviointi oli musta laatikko ilman ohjeistusta prototyypin käyttöön, jäivät malliin toteutetut toiminnallisuuksiin liittyvät interaktiiviset toiminnot asiantuntijalta huomaamatta. Toiminnallisuus-laatuominaisuudesta arvioitiinkin lähinnä navigointi- ja selausominaisuuksia, jonka perusteella lisättiin taulukon sivuttamiseen vaihtonappeja ja sivunumerointia, sekä mahdollistettiin tietojen lajittelu.

Tavoitettavuus
Tavoitettavuuden osalta ei heuristinen arviointi tuonut uutta tietoa, sillä tässä tapauksessa sovelluksessa olevat ratkaisut ja sen kohderyhmän erikoisvaatimukset olivat huomioitu. Pääperiaatteena pidettiin lähtökohtaa, jossa sovellus toimii selaimesta ja käyttöliittymästä riippumatta. Myöskään eri selainversioiden tuki eri ominaisuuksille ei ollut rajoitteena, mutta yleisesti ottaen kohderyhmän käyttämä selain voi rajoittaa asioiden toteuttamista ja käytettäviä teknologioita. Jos on tarvetta käyttää erikoisempia tekniikoita asioiden toteuttamiseen, pitää tiedostaa mitä rajoituksia se käyttäjille asettaa.

Prototyypin avulla suoritettu heuristinen arviointi oli hyödyllistä, vaikka tässä tapauksessa mallia käsiteltiin ja arvioitiin kuten lopullista sovellusta, joka toisaalta antoi hyviä vinkkejä asioiden kehittämiseksi, mutta esitti käytettävyysongelmina myös asioita, jotka

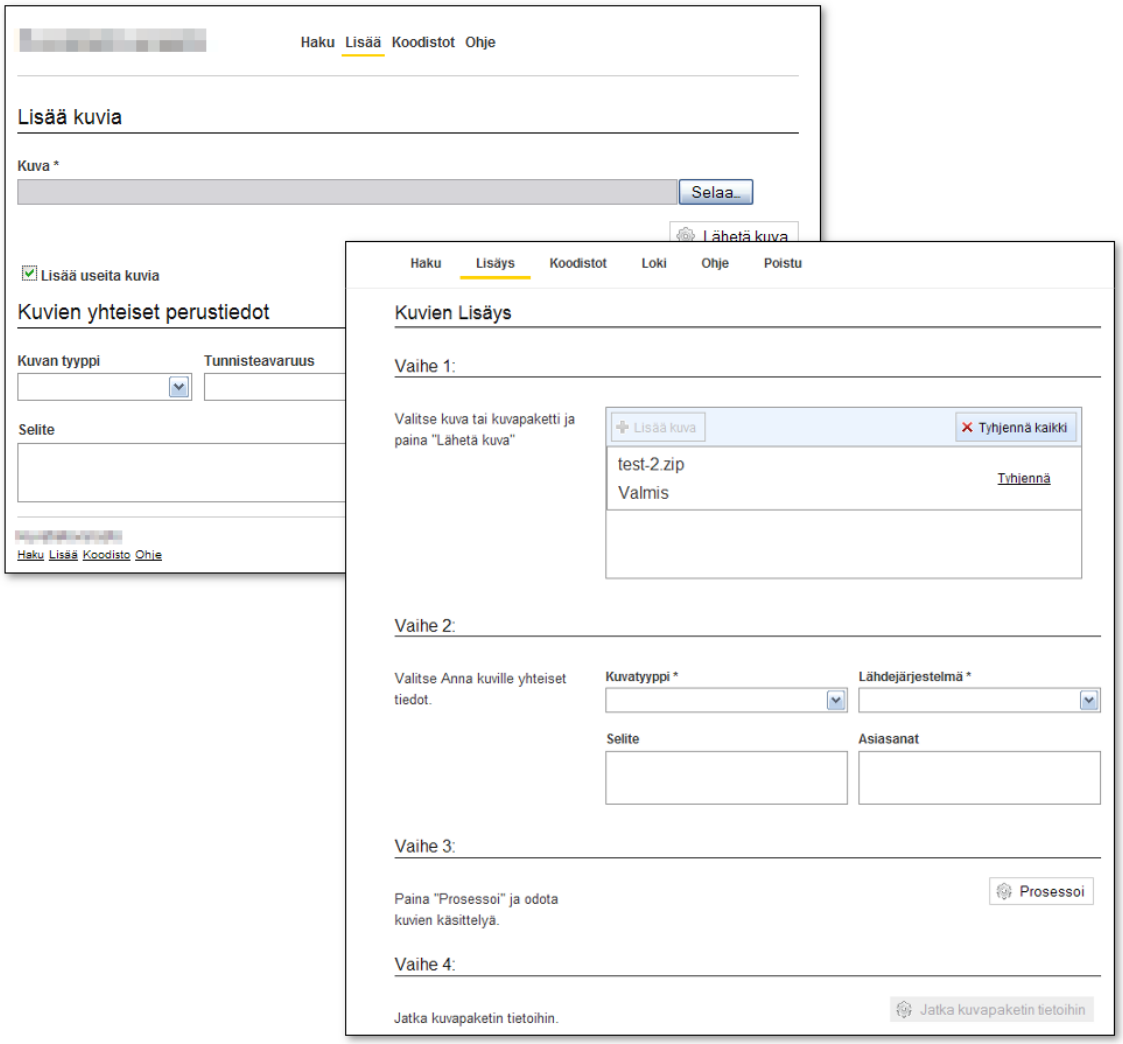
johtuivat prototyypissä olevista virheistä. Arvioinnissa onkin syytä huomioida käytettävän mallin prototyyppimäisyys, asioiden mahdollinen keskeneräisyys ja soveltaa arviointiin maalaisjärkeä. Heuristisen arvioinnin jälkeen olisi voitu jatkaa käyttäjätutkimuksilla, joilla oltaisiin päästy käsittelemään tarkemmin käytettävyyttä ja toiminallisuutta.

4.1.4 Toteutusvaihe

Toteutusvaiheessa prototypointia ei enää jatkettu, mutta jo rakennettua HTML-prototyyppiä pystyttiin hyödyntämään JSF-toteutuksen perustana. Käytännössä HTML-pohjiin vaihdettiin JSF-komponenttien tagit ja toteutettiin sovelluksen taustarakenteet, jolloin käyttöliittymän osalta saatiin nopeasti aikaan lopullista sovellusta muistuttava toteutus. Suunnitteluvaiheessa JavaScriptillä hahmoteltujen toimintojen perusteella päästiin käytännössä näkemään miten jokin toiminto oli ajateltu käyttäytyvän, joka antoi selkeämmän kuvan toiminnosta kuin vain pelkän käyttötapauksen tai ikkunakuvauksen lukeminen.

Käytäntö on opettanut, että muutos on projektissa enemmän sääntö kuin poikkeus ja muutoksenhallinta on prototypoinnissakin tarpeellista. Toteutusvaiheessa havaittiin puutteita suunnittelun osalta ja niitä jouduttiin tarkentamaan muun muassa lisänäyttöjen ja -toiminnallisuuksien osalta, mutta suunnittelumuutoksia ei viety aikaisempiin HTML-malleihin, vaan ne hahmoteltiin suoraan JSF-toteutukseen. Toteutuksen alkuvaiheessa todettiin, että JSF-teknologia ja RichFaces-komponentit ovat suhteellisen yksinkertaisia toteuttaa sovellukseen ja toisaalta toiminnoiltaan vaikeasti muuten mallinnettavissa, että suunnitteluvaiheen HTML-malleja ei olisi hyödyllistä pitää ajantasalla toteutuksen rinnalla.

Toteutusvaiheessa pystyttiin hyvin käyttämään perustana suunnitteluvaiheessa rakennettuja HTML-malleja, mutta “Lisää”-näytön ajateltua rakennetta ja toimintalogiikkaa jouduttiin toteutusvaiheessa muuttamaan eniten verrattuna muihin prototyyppeihin sovelluksen näytöistä. Suunnitteluvaiheessa ei huomioitu riittävän tarkasti lopullisessa toteutuksessa käytettävien RichFaces-komponenttien mahdollisuuksia, jotka tarjosivat paremmat ominaisuudet kuvien lisäämiseen. Kuvassa 14 näkyy muutos prototyypin ja lopullisen toteutuksen välillä: varhaisissa testeissä todettiin “lisäys”-toiminto sekavaksi ja se muutettiin vaiheittaiseksi. Uusi “Lisää”-näyttö suunniteltiin ja toteutettiin suoraan JSF-teknologialla, sillä vaikka HTML-prototyyppiin olisi voitu lisätä RichFaces-komponentti grafiikkana, arvioitiin Java-toteutukseen menevä työ hyödyllisemmäksi.



Kuva 14: “Lisää”-näyttö muuttui teknisten vaatimusten tarkentuessa

Sovelluksen toiminnan kannalta toisarvoiset ja toteutuksen aikana esille tulleet ideat ja tarpeet sovelluksen toimintojen suhteen kirjattiin ylös ja tarvittavilta osin prototypoitiin ottamalla käyttöliittymästä ruudunkaappaus ja tekemällä muutokset kuvankäsittelyohjelmalla. Näin ideat saatiin heti määriteltyä, suunniteltua ja mallinnettua, joka helpottaa niiden toteuttamista myöhemmin.

4.1.5 Prototypointi toteutuksen jälkeen

JSF- ja RichFaces -teknologiat havaittiin toteutuksen aikana hyviksi välineiksi toteuttaa sovelluksen uusista ominaisuuksista niin sanottuja “proof of concept” -toteutuksia, eli todistaa että jokin asia on mahdollista rakentaa ja samalla saada siitä prototyyppi. Toteu-

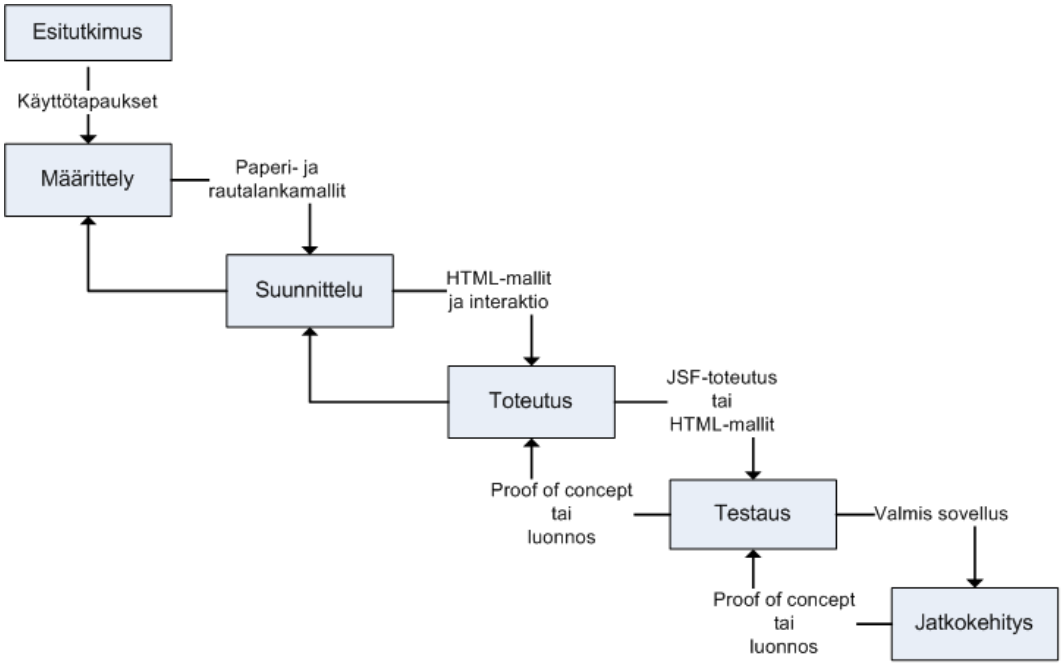
tusvaiheessa todettiin JSF-sovelluksen käyttöliittymän olevan helposti muokattavissa, jos Javan lisäksi hallussa ovat sekä HTML- että CSS-merkkauskielet, joilla mallin luominen kaikilta osa-alueiltaan onnistuu alusta loppuun asti.

Myöhemmin sovellukseen liittyvistä kehitysehdotuksista ja uusista ominaisuuksista rakennettiin sekä malleja että mallitoteutuksia tarpeen mukaan. Muun muassa pienten kuvien näyttämiseksi hakutuloslistalla toteutettiin "proof of concept"-tason malli eli todiste asian toimivuudesta, koska haluttiin varmistua sovelluksen ja tietokannan suorituskyvystä hakea useita pieniä kuvia listalle nopeasti. Mallitoteutuksena prototypoitiin myös pienempi selaus-ikkuna ja siihen liittyvät toiminnot, jotka vastasivat suunnilleen jo toteutettua näyttöä ja uuden ohjelmakoodin osuus jäi vähäiseksi. Vastaavasti graafisen mallin avulla visualisoitiin sovelluksen käyttöliittymän uudelleenjärjestely, sillä muutos oli teknologisesta näkökulmasta yksinkertainen ja selkeä. Yksinkertaiset käyttöliittymään kohdistuvat muutokset oli nopeampaa toteuttaa ruudunkaappauksella nykyisestä toteutuksesta ja muokkaamalla kuvaa kuvankäsittelyohjelmassa.

4.2 Käytännöllinen lähestymistapa Web-sovelluksen prototypointiin

Prototypointia voidaan soveltaa monella eri tapaa, eikä sen käyttämistä ole sidottu esimerkiksi projektin tiettyyn vaiheeseen, tai että juuri jotain menetelmää pitäisi käyttää esimerkiksi suunnitteluvaiheessa. On kuitenkin olemassa hyviä käytäntöjä, joiden avulla prototypoinnista saadaan eniten hyötyä ja joiden käyttäminen tukee sovelluskehitysprojektia. Useissa projekteissa on havaittu, että parhaiden käytäntöjen mukaisesti prototypointi aloitetaan alhaisen tarkkuuden malleilla ja edetään korkeamman tarkkuuden malleihin tai alkuvaiheen jälkeen syvennytään jonkin osa-alueiden mallintamiseen tarkemmin, jos aika- ja kustannusrajoitteet sallivat.

Web-sovelluksen käyttöliittymän ja toiminnallisuuksien prototypoinnin työnkulku voidaan jakaa vesiputousmallia käyttävän projektin vaiheiden mukaan viiteen osaan: määrittely, suunnittelu, toteutus, testaus ja jatkokehitys. Kuvassa 15 on esitetty ehdotus prototypointiprosessin etenemiselle ja mitä malleja on hyödyllistä toteuttaa eri vaiheissa. Jokainen vaihe saa syötteenä edellisessä vaiheessa toteutetun mallin, jota hyödynnetään prosessin edetessä ja tarkempien mallien luomisessa. Taulukosta 3 näkyy tarkemmin eri vaiheissa käytetyt prototypoinnin tarkkuudet, teknologiat ja mallit.



Kuva 15: Vesiputousmallin vaiheet ja prototypointiprosessin syötteet ja tulokset

Vaihe	Määrittely	Suunnittelu	Toteutus	Jatkokehitys
Tarkkuus	Alhainen	Sekoitettu	Sekoitettu / Korkea	Korkea / Sekoitettu / Alhainen
Teknologia	Kynä ja paperi	HTML	HTML / JSF	Graafinen / JSF
Malli	Paperi- ja rautalanka	HTML	JSF	Rautalanka / JSF

Taulukko 3: Projektin vaiheet ja menetelmien tekniikat

Määrittelyvaiheen prototypointi saa syötteenä vaatimusmäärittelyn käyttötapaukset, joiden perusteella voidaan luonnostella sovelluksen rakennetta ja sisältöä. Lopputuloksena saatavat paperi- tai rautalankamallit toimivat syötteenä suunnitteluvaiheen prototypoinnille, jossa sovelluksesta voidaan toteuttaa piirrosten pohjalta HTML-malleja. Suunnitteluvaiheessa malleihin lisätään myös interaktiivisuutta, jolloin toteutusvaiheessa malleja voidaan käyttää toimivina esimerkkeinä asioiden halutusta käyttäytymisestä. Toteutuksessa jalostetaan suunniteltuja HTML-malleja ja lisätään niiden taustalle sovelluskoodia, tässä tapauksessa JSF-komponentteja, jolloin saadaan nopeasti aikaan sekoitetun tarkkuuden prototyyppi halutun asian tutkimiseksi. Testausvaihe saa syötteenä valmiin sovelluksen tai HTML-mallit, joita voidaan käyttää sovelluksen testauksessa. Jatkokehityksessä uusia ominaisuuksia voidaan toteuttaa nopeasti myös JSF-teknologialla tekemällä niistä niin sanottuja “proof of concept”-malleja tai käsittelemällä muutokset ruudunkaappauksiin.

Projektin aikana todettiin, että voidaan edetä alhaisella tarkkuudella tehtyjen paperimallien kautta nopeasti suoraan HTML-malleihin ja limittää vesiputousmallin mukaan etenevän projektin määrittely- ja suunnitteluvaiheita. HTML-malleihin toteutettiin lisäksi aikaisessa vaiheessa myös luonteeltaan selkeä ja minimalistinen graafinen ilme. Alhaisella tarkkuudella luonnosteltaessa saatiin hahmoteltua sovelluksen näyttöjen rakenne ja sisältö, jonka perusteella HTML-merkkauksella oli helppo toteuttaa tarkemmat visuaaliset HTML-mallit. Graafisen ilmeen minimalistisuuden ansiosta se oli luontevaa toteuttaa heti ja jättää pelkistetyt rautalankamallit toteuttamatta. Graafisissa HTML-malleissa on etuna, että huomio ei kiinnity ilmeen puuttumiseen, mutta toisaalta huomio voi keskittyä pelkkään ilmeeseen, eikä sisältöön ja toimintoihin, jos sovellus on visuaalisesti rikas.

HTML-teknologioilla toteutettujen mallien osalta todettiin myös käteväksi mallintaa sovelluksen interaktiota lisäämällä malleihin JavaScriptillä hahmoteltuja toimintoja. Nyt mallin avulla voitiin myös näyttää sovelluksen aiottua toimintaa käytännössä, joka on havainnollisempaa kuin vain dokumentteihin kirjattu asia. Toteutetusta HTML-mallista oli hyötyä myös Java-toteuksen alkuvaiheessa. Koska lopullisen sovelluksen käyttöliittymä pohjautuu HTML-komponentteihin, voitiin mallia jalostaa lisäämällä siihen JSF-komponentteja ja tutkia näin tarvittavien asioiden toimintaa sekoitetun tarkkuuden mallilla. Koska suunnittelussa oli toteutettu malliin myös graafinen ilme, oli toteutuksen alussa jo rakennettuna sovelluksen käyttöliittymä tyyleineen ja siihen ei enää tarvinnut käyttää juurikaan aikaa. Toteutusvaiheessa tarkennettiin lähinnä RichFaces-komponenttien sovittamista graafiseen ilmeeseen.

Toteutusvaiheen alussa mietittiin HTML-mallin ylläpitämistä toteutuksen rinnalla, mutta todettiin kahden lähes vastaavan mallin ylläpitämisen olevan työlästä verrattuna mallista saatavaan hyötyyn ja muutosten tekemiseen menevään aikaan. JSF-teknologia todettiin kohtalaisen helpoksi käyttää eri asioiden toteuttamiseen ja useat ominaisuudet tukeutuvat valmiiden komponenttien käyttöön, joiden vieminen HTML-malliin ei olisi ollut hyödyllistä tai edes järkevästi mahdollista. Toteutuksen aikana ilmenneitä uusia ominaisuuksia ja kehitysehdotuksia oli kätevää visualisoida joko muokkaamalla sovelluksesta otettuja ruudunkaappauksia tai toteuttamalla niistä “proof of concept”-malleja JSF-teknologialla.

5 Pohdinta ja johtopäätökset

Prototypointia voi kuvailla ideoiden keräämiseksi, visualisoimiseksi ja arvioimiseksi eri osapuolten kesken ja ymmärrettäessä se osana kehitysprosessin kokonaisuutta, eikä vain itsenäisenä osana, saadaan parempia lopputuloksia ratkaistavaan ongelmaan. Periaatteessa prototypointi on vain eri työvaiheiden ketjuttamista, mutta asioiden hallitseminen ja seuraaminen on monimutkaista, etenkin jos asioista tehdään useita rinnakkaisia malleja. Web-sovelluksen käyttöliittymän prototypointia varten ei ole yhtä selkeää tapaa tehdä asioita ja useat menetelmät ja työvälineet jättävät toivomisen varaa prosessin ja kehitettävän mallin tukemisen suhteen.

Prototypoidessa herääkin kysymyksiä muun muassa kommenttien ja suunnitteluratkaisujen liittämisestä prototyyppiin ja mallien versioinnin ja säilyttämisen hoitamisesta. Sähköisten mallien osalta säilyttäminen ja jakaminen eri osapuolten kesken on helppoa, mutta paperiprototyyppien arkistointi ja kierrättäminen henkilöiltä toiselle on työläämpää. Vastaavasti paperimalleihin on helppo lisätä kommentteja ja selityksiä esimerkiksi liittämällä niihin lappuja, mutta HTML-mallin osalta kommenttien liittäminen on jo hankalampaa. Kommenttien ja selitysten liittäminen malliin on tärkeää etenkin silloin, jos asioista tehdään useita erilaisia malleja, joita verrataan toisiinsa: miksi tehdään näin ja miksi ei toisin. Rakennettujen mallien määrän kasvaessa on myös mietittävä miten niiden versionhallinta pitäisi hoitaa: mitä missäkin mallissa on tutkittu, mihin päädyttiin ja miten asia muuttuu seuraavassa mallissa?

Tuettaessa sovelluskehitysprojektin määrittelyä, suunnittelua, toteutusta ja jatkokehitystä prototypoinnilla, ovat monet eri tekniikat käyttökelpoisia asioiden mallintamiseen. Alhaisen tarkkuuden mallit soveltuvat hyvin ajatusten tarkentamiseen ja suunnan hahmotteluun, ja mitä korkeammalle tarkkuudessa mennään, sen yksityiskohtaisempia asioita mallilla voidaan tutkia, mutta samalla siihen käytettävä aika kasvaa. On tärkeää löytää tarkkuuden suhteen sopiva keskitie, jolla avoimena oleviin kysymyksiin vastauksia haetaan.

Sekoitetun tarkkuuden mallit ovat yksi kätevä tapa Web-sovelluksia mallintaessa, sillä esimerkiksi HTML- ja JSF-komponentteja sekoittamalla saadaan aikaan suhteellisen pienellä vaivalla ensiksi yhtä kohtaa tarkentava malli, jonka jälkeen sitä voidaan laajentaa iteratiivisesti, käyttää pilottijärjestelmänä ja kehittää eteenpäin lopulliseksi sovellukseksi. Toisaalta käyttökelpoisen sekoitetun HTML- ja JSF-teknologioita käyttävän mallin rakentaminen vaatii kohtalaisen paljon lopullisen sovelluksen perusrakenteiden toteuttamista, joten siirtymävaihe HTML-mallista korkeamman tarkkuuden tasolle ei välttä-

mättä suju ihan hetkessä. Malleja ei täten kannata myöskään toteuttaa liian aikaisin, jolloin riski muutoksille on suuri. Rakenteelliset ja ulkonäölliset muutokset tosin onnistuvat Web-teknologioilla suhteellisen kivuttomasti, mutta niiden tekeminen on aina työläämpää ja hitaampaa kuin kevyemmässä HTML-prototyypissa. Kannattaa myös miettiä tarvitaanko ylipäätään korkeamman tarkkuuden toiminnallista mallia vai riittääkö kattavammin HTML-tekniikoilla toteutettu malli. Erillisen HTML-mallin ylläpitäminen JSF-toteutuksen rinnalla ei ole kannattavaa, sillä tarvittaessa JSF-toteutuksesta saadaan HTML-malli, jolla voidaan hahmotella uusia toimintoja helpommin ja nopeammin, jos ne eivät varsinaisesti käsittele JSF-puolen osuutta.

5.1 Johtopäätökset

Web-sovelluksen prototypointi ei ole vaikea prosessi ja käytettävät prototypointimenetelmät ovat yksinkertaisia: projektin alussa on järkevää tehdä malleja alhaisella tarkkuudella ja projektin edetessä siirtyä korkeamman tarkkuuden malleihin ja sekoittaa eri tarkkuuksia. Ei kannata keskittyä sovelluksen mallintamiseen vain alhaisen tarkkuuden malleilla, vaan tarpeen mukaan on hyödyllistä toteuttaa myös sekoitetun ja korkean tarkkuuden malleja, eli rakentaa osa prototyypistä esimerkiksi graafisesti tai HTML-teknologioilla ja osa lopullisen sovelluksen teknologioilla. Asioita kannattaa mallintaa sillä tarkkuudella, millä asteella ideakin on. Konkreettiset ideat ovat valmiita “proof of concept”-toteutukselle, kun vastaavasti pohdinnan tasolla olevat ideat kannattaa jättää luonnosten tasolle.

Työn johdannossa asetettiin tavoitteeksi vastata neljään kysymykseen liittyen prototypoinnin suorittamiseen sovelluskehitysprojektissa ja niiden osalta päästiin seuraaviin päätelmiin:

– **Miten hyödyntää käytettävissä olevia teknologioita sovelluskehitysprojektin eri vaiheissa sovelluksen prototypoinnin tukena?**
  Web-sovelluksen käyttöliittymän prototypointi on periaatteiltaan kohtalaisen yksinkertaista ja prosessissa voidaan soveltaa työpöytäsovellusten parissa jo hyväksi havaittuja prototypointimenetelmiä ja -periaatteita, vaikka toteutettavissa malleissa painottuu käyttöympäristön eli Web-selaimen ja -teknologioiden merkitys. Yhtä ja oikeaa tapaa suorittaa prototypointia ei ole ja asioita voi aina soveltaa tarpeen mukaan.

– **Miten HTML-teknologia soveltuu mallintamiseen ja mitä se mahdollistaa?**
HTML-mallin rakentaminen jo projektin alkuvaiheessa nopeiden paperiprototyyppien jälkeen tarjoaa monipuoliset mahdollisuudet asioiden mallintamiseen. Web-sovellukset rakentuvat HTML-teknologioiden päälle, joten asioita voidaan monilta osin lähes toteuttaa ja vähintään tutkia ideoiden toteutuskelpoisuutta. Yhdistämällä malliin JavaScriptiä, saadaan kuvattua ulkoasun lisäksi myös tuntumaa ja vuorovaikutusta, mutta toimintojen simuloiminen ilman palvelinpuolen komponentteja voi olla haasteellista. Kokeiltavalla mallilla on helpompi puhua samaa kieltä eri osapuolten välillä ja näyttää miten jonkin toiminnon pitäisi käytännössä toimia. Luonnosteluun verrattuna HTML-prototyypin muuttaminen on työläämpää, mutta siitä on hyötyä myös toteutuksen kannalta. Mallia voidaan jalostaa sekoitetun tarkkuuden malliksi ja tutkia näin tarvittavien asioiden oikeaa toimintaa.

– **Kuinka huomioida lopullisen sovelluksen teknologia ja sen hyödyntäminen?**
Toteutettaessa sovellusta Java EE -ympäristössä JSF-teknologialla RichFaces-käyttöliittymäkomponentteja käytettäen, on tarpeen tunnistaa mitä mahdollisuuksia teknologiat antavat ja rajaavat. Vaikka useita käyttöliittymän toimintoja voidaan esittää korvaavilla teknologioilla, ovat esimerkiksi RichFaces-komponentit toiminnaltaan hankalia kopioida. Graafiset erot eivät ole niin merkitseviä, mutta logiikan suhteen asioiden oikeanlainen toiminta on tärkeämpää. JSF-teknologiat toimivat hyvin yhteen HTML-teknologian kanssa, joten mallin tarkentaminen JSF-toteutukseksi onnistuu asteittain. Käytettävät teknologiat eivät rajoita prototypointia, sillä toimintaa ja ulkonäköä päästään usein arvioimaan riittävällä tarkkuudella. Sekoitetun tarkkuuden mallin luominen on hyödyllistä, koska sen avulla päästään paremmin arvioimaan sovelluksen eri osa-alueita lähempänä totuutta.

– **Millainen on käytännöllinen lähestymistapa Web-sovelluksen prototypointiin sovelluskehitysprojektissa ja sen jälkeen?**
Web-sovelluksen käyttöliittymän ja toiminnallisuuksien prototypoinnin työnkulku voidaan jakaa projektin vaiheiden mukaan viiteen osaan: määrittely, suunnittelu, toteutus, testaus ja jatkokehitys. Jokainen vaihe saa syötteenä edellisessä vaiheessa toteutetun mallin, jota hyödynnetään prosessin edetessä ja tarkempien mallien luomisessa. Määrittelyvaihe saa syötteenä käyttötapaukset ja tuottaa luonnosteltuja paperi- ja rautalankamalleja. Suunnitteluvaiheessa jatketaan HTML-mallien ja sovelluksen interaktion hahmottelulla, ja toteutusvaihe saa pohjakseen HTML-mallit, joista voidaan jalostaa sekoitetun tarkkuuden malleja. Valmista sovellusta tai HTML-malleja voidaan testata ja jatkokehityksessä prototypointi onnistuu useilla eri tarkkuuden teknologioilla.

5.2 Aiheen jatkokäsittely

Voidaan sanoa, että projekteissa asiat melkein aina muuttuvat, sillä projektin edetessä ja toteutettavan asian selkeytyessä, uusia vaatimuksia ja tarpeita ilmenee. Muutostarpeita voidaan vähentää prototypoimalla, mutta toteutettujen mallien suunnitteluratkaisujen vertailu ja kommenttien käsitteleminen tiettyyn ratkaisuun liittyen on hankalaa. Prototyyppejä olisi hyvä pystyä vertailemaan ja kokeilemaan rinnakkain, jolloin mallin evoluutio näkyisi selkeästi. Lisäksi näkymässä pitäisi olla esillä kyseiseen malliin liittyvät kommentit. Paperiprototyyppien keskinäinen vertailu on tältä osin helpompaa, mutta sähköisten mallien osalta asia on vaivalloisempaa. Sähköiset mallit kuten HTML-malli tarvitsisivat tuekseen jonkin keinon, jolla prototypointiprosessin etenemisen seuraaminen ja mallien arviointi olisi mahdollista.

Kokonaisuudessaan on nähtävissä, että Web-sovellusten HTML-teknologiaa on hyödyllistä soveltaa prototypoinnissa jo aikaisessa vaiheessa, mutta prototypointiprosessi ja -menetelmät kaipaisivat silti enemmän tukea mallien elinkaaren hallintaan. Toisaalta määrittelyssä ja suunnittelussa toteutettujen mallien arvo projektin päättymisen jälkeen voi olla käytännössä olematon, joten ylimääräistä työtä mallien hallintaan on syytä välttää. Malleista voisi kuvitella olevan hyötyä myös projektin jälkeen, jos ne ovat hyvin dokumentoitu, eli eri ratkaisut näkyvät mallissa selityksin ja mahdollisesti aikaisempien ratkaisujen kanssa, jolloin tiedetään miksi päädyttiin tiettyyn lopputulokseen. Kokonaisuuden pitäisi silti pysyä kevyenä ja joustavana, jolloin prototyyppien rakentaminen ei muutu työlääksi, vaan pysyy suhteellisen yksinkertaisena ja innovaatiota edistävänä.

6 Yhteenveto

Web-sovellukset ovat viime vuosikymmenten aikana kehittyneet staattisista sivuista monimutkaisiksi sovelluksiksi, joiden tärkeänä osana on käyttöliittymän toimivuus, jonka eteen voidaan sovelluskehitysprojektissa käyttää sekä aikaa että vaivaa. Päämäärän saavuttamiseksi asioita voidaan edistää monella eri tapaa, joista yksi on käyttöliittymän ja sen toiminnallisuuksien prototypoiminen projektin eri vaiheissa. Prototypoinnin avulla voidaan tunnistaa ja korjata mahdolliset ongelmat aikaisessa vaiheessa, jolloin sen arvo on moninkertainen käytettyyn aikaan nähden. Käyttöliittymän mallintamista on mahdollista toteuttaa eri tarkkuuksilla, useilla prototypointimenetelmillä ja tekniikoilla, ja tässä työssä aihetta lähestyttiin HTML-teknologian hyödyntämisen näkökulmasta. Asiaa käsiteltiin soveltaen prototypointia vaiheittain etenevässä projektissa pohtien miten eri menetelmät ja teknologiat ovat hyödynnettävissä.

Prototypointi tunnistetaan laajalti yhdeksi perusmenetelmäksi interaktiivisten sovelluksen ominaisuuksien tutkimiseen ja ilmaisemiseen, ja parhaiden käytäntöjen mukaisesti se aloitetaan alhaisella tarkkuudella ja edetään korkeamman tarkkuuden malleihin. Alun jälkeen voidaan myös syventyä jonkin osa-alueen tarkempaan mallintamiseen, jos aika- ja kustannusrajoitteet sallivat. Eri tarkkuuden malleilla on omat käyttökohteensa ja sopiva malli riippuu käsiteltävästä ongelmasta. Alhaisen tarkkuuden mallit ovat konseptinomaisia hahmotelmia ideasta, joita voidaan toteuttaa luonnostelemalla esimerkiksi “kynä ja paperi”-tekniikalla. Luonnostelu soveltuu muun muassa näytön rakenteen ymmärtämiseen ja tutkimiseen ja sitä voidaan toteuttaa myös tietokoneavusteisesti. Korkean tarkkuuden mallit mahdollistavat käyttäjän kokeilla sovellusta kuten se olisi lopullinen tuote, eli vastaavat monilta osin valmista sovellusta, mutta niiden kehittäminen on aikaa vievää ja kallista verrattuna alhaisen tarkkuuden malleihin. Ne ovat hyödyllisiä käytettävyystestauksen suorittamiseen suunnitteluprosessin aikana ja toimivat sekä markkinoinnin ja päättäjien että toteuttajien tukena. Sekoitetun tarkkuuden malleissa yhdistetään sekä alhaisen että korkean tarkkuuden mallien hyviä puolia, jolloin käytännössä suunnitellaan osa prototyypistä esimerkiksi luonnostelmalla ja osa sovelluskoodina.

Web-sovellusten teknologiat ovat käytännössä suhteellisen yksinkertaisia ja helppokäyttöisiä sovelluksen prototypoinnissa. Vaikka teknologiat kuten HTML ja CSS ovat hyvin määriteltyjä, useat Web-selaimet kuitenkin tulkitsevat määrityksiä erilailla, eivätkä välttämättä tue kaikkia määriteltyjä ominaisuuksia, joka täytyy suunnittelussa ottaa huomioon. Web-teknologiat sopivat hyvin Java EE -ympäristön JSF-teknologiaa käyttävien sovellusten prototypointiin, koska JSF ja RichFaces tuottavat Web-selaimelle käytännössä nor-

maalia HTML:ää JavaScriptillä tehostettuna. Tämä mahdollistaa sovelluksen ilmeen ja luonteen suhteellisen luotettavan prototypoimisen HTML-teknologioilla ja mallien avulla saatujen tietojen pohjalta sovelluksen toteutus on suoraviivaisempaa.

Työn käytännön osuudessa todettiin toimivaksi malliksi edetä paperiprototyyppien kautta nopeasti suoraan HTML-malleihin ja limittää vesiputousmallin mukaan etenevän projektin määrittely- ja suunnitteluvaiheita. HTML-teknologioilla toteutettujen mallien osalta todettiin myös käteväksi mallintaa sovelluksen interaktiota käyttäjän kanssa lisäämällä niihin JavaScriptillä hahmoteltuja toimintoja. HTML-mallista oli hyötyä myös toteutuksen alkuvaiheessa, sillä sitä voitiin jalostaa sekoittamalla JSF-komponentteja ja HTML-komponentteja ja tutkia näin tarvittavien asioiden oikeaa toimintaa. Toteutuksessa havaittiin myös, että JSF-teknologioilla on suhteellisen yksinkertaisia toteuttaa asioita ja toisaalta monet asiat ovat vaikeasti muuten mallinnettavissa. Erilaisten työvälineiden käyttämistä prototypoinnin apuna ei pidetty tarpeellisena, sillä vaikka ne tarjoavat parhaimmassa tapauksessa nopean tien prototyypin luomiseen, rajoittavat ne usein suunnittelijaa pakottamalla aikaisessa vaiheessa tiettyihin teknisiin ratkaisuihin. Suunnittelumallien hyödyntäminen vastaavasti on suotavaa, sillä ne tarjoavat valmiita, hyväksi todettuja käyttökelpoisia ratkaisuja ja ideoita asioiden toteuttamiseen.

Yhteenvetona keskeisin havainto on, että Web-sovellusten käyttöliittymien prototypoinnissa voidaan soveltaa yleisiä prototypointimenetelmiä ja -periaatteita ja alhaisen tarkkuuden HTML-teknologialla toteutetut mallit ovat hyödyllisiä jo projektin aikaisessa vaiheessa. Toteutettavissa malleissa painottuu käyttöympäristön eli Web-selaimen ja -teknologioiden merkitys, joka toisaalta myös helpottaa mallien rakentamista ja prototyyppien hyödyntämistä sovelluskehitysprojektin myöhemmissä vaiheissa. Prototypointia on hyödyllistä toteuttaa sekä alhaisen, sekoitetun että korkean tarkkuuden malleilla, jolloin saadaan hyödynnettyä molempien mallien hyviä puolia. On hyödyllistä aloittaa luonnostelemalla ja siirtyä aikaisessa vaiheessa HTML-pohjaisiin malleihin, joilla päästään lähelle lopullisen toteutuksen teknologioita ja mallintamaan sovelluksen luonnetta, ilmettä, tuntumaa ja vuorovaikutusta. Toteutuksen aikana ja sen jälkeen prototypointia on hyödyllistä suorittaa monella eri tapaa riippuen tutkittavasta asiasta, mutta JSF-teknologia on joustava “proof of concept”-tyylisiin ratkaisuihin, jolloin saadaan oikea kuva myös teknisistä mahdollisuuksista.

Viitteet

[BF04] Mario Bochicchio ja Nicola Fiore. Warp: Web application rapid prototyping. Kirjassa *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, sivut 1670–1676, New York, NY, USA, 2004. ACM.

[BK06] Ed Burns ja Roger Kitain. Javaserver faces 1.2. Java Specification Request (JSR) 252, May 2006.

[BP00] Mario Bochicchio ja Roberto Paiano. Prototyping web applications. Kirjassa *SAC '00: Proceedings of the 2000 ACM symposium on Applied computing*, sivut 978–983, New York, NY, USA, 2000. ACM.

[Bro95] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison-Wesley, August 1995.

[BS00] Marion Buchenau ja Jane Fulton Suri. Experience prototyping. Kirjassa *DIS '00: Proceedings of the 3rd conference on Designing interactive systems*, sivut 424–433, New York, NY, USA, 2000. ACM.

[Cat07] Dan Catt. "places"-page, 1st draft concept sketch. [verkkodokumentti]. Julkaistu 28.10.2007. [viitattu 4.3.2010]. Saatavissa: http://www.flickr.com/photos/35468159852@N01/2072452369.

[Chi09] Roberto Chinnici. Javatm platform, enterprise edition 6. Java Specification Request (JSR) 316, December 2009.

[CM00] D. Connolly ja L. Masinter. The 'text/html' Media Type. RFC 2854 (Informational), June 2000.

[Con09a] World Wide Web Consortium. The XMLHttpRequest Object W3C Working Draft. [WWW]. World Wide Web Consortium, Päivitetty 19.10.2009. [viitattu 4.3.2010]. Saatavissa: http://www.w3.org/TR/XMLHttpRequest/.

[Con09b] World Wide Web Consortium. Html specifications. [WWW]. World Wide Web Consortium, Päivitetty 29.1.2009. [viitattu 27.7.2009]. Saatavissa: http://www.w3.org/MarkUp/.

[Conon] World Wide Web Consortium. CSS Specifications. [WWW]. World Wide Web Consortium, Julkaisupäivä tuntematon. [viitattu: 21.7.2009]. Saatavissa: http://www.w3.org/Style/CSS/#specs.

[Dim06] Garrett Dimon. Just Build It: HTML Prototyping and Agile Development. [verkkodokumentti]. Digital Web Magazine, Julkaistu 6.3.2006. [viitattu 27.7.2009]. Saatavissa: http://www.digital-web.com/articles/just_build_it_html_prototyping_and_agile_development/.

[Hamon] David Hammond. Web Browser HTML Support. [WWW]. Julkaisupäivä tuntematon. [viitattu 27.7.2009]. Saatavissa: http://www.webdevout.net/browser-support-html.

[HH97] Stephanie Houde ja Charles Hill. What Do Prototypes Prototype? *Handbook of Human-Computer Interaction 2nd Edition*, 1997.

[Hon08] Sockyung Hong. sketch: vimeo profile page idea. [verkkodokumentti]. Julkaistu 9.1.2008. [viitattu 4.3.2010]. Saatavissa: http://www.flickr.com/photos/soxiam/2182204230.

[Incon] Yahoo! Inc. Design Pattern Library. [WWW]. Yahoo! Inc., Julkaisupäivä tuntematon. [viitattu 3.8.2009]. Saatavissa: http://developer.yahoo.com/ypatterns/.

[Int99] E. C. M. A. International. *ECMA-262: ECMAScript Language Specification 3rd edition*. ECMA (European Association for Standardizing Information and Communication Systems), Geneva, Switzerland, 1999.

[JBo09] JBoss. Richfaces 3.3.x developer guide. [verkkodokumentti]. Red Hat, Julkaistu 2009. [viitattu 4.3.2010]. Saatavissa: http://docs.jboss.org/richfaces/latest_3_3_X/en/devguide/html/.

[Koc] Peter-Paul Koch. Quirksmode: CS contents and browser compatibility.

[Kru00] Steve Krug. *Don't Make Me Think!: A Common Sense Approach to Web Usability*. Que Corp., Indianapolis, IN, USA, 2000. Foreword By-Black, Roger.

[Lan08] James Landay. Denim: An informal tool for early stage web site and ui design. [WWW]. University of Washington, Päivitetty 30.10.2008. [viitattu 25.3.2010]. Saatavissa: http://dub.washington.edu/denim/.

[LL07] Liang-Cheng Lin ja Wai On Lee. UI toolkit for non-designers in the enterprise applications industry. Kirjassa *CHI '07: CHI '07 extended abstracts on Human factors in computing systems*, sivut 1795–1804, New York, NY, USA, 2007. ACM.

[LNHL00] James Lin, Mark W. Newman, Jason I. Hong, ja James A. Landay. DENIM: finding a tighter fit between tools and practice for Web site design. Kirjassa *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, sivut 510–517, New York, NY, USA, 2000. ACM.

[LSHZ93] Horst Lichter, Matthias Schneider-Hufschmidt, ja Heinz Züllighoven. Prototyping in industrial software projects - bridging the gap between theory and practice. Kirjassa *Proceedings of the 15th international conference on Software Engineering*, sivut 221–229. IEEE Computer Society Press, 1993.

[MCP+06] Michael McCurdy, Christopher Connors, Guy Pyrzak, Bob Kanefsky, ja Alonso Vera. Breaking the fidelity barrier: an examination of our current characterization of prototypes and an example of a mixed-fidelity success. Kirjassa *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, sivut 1233–1242, New York, NY, USA, 2006. ACM.

[Mäk05] Vesa-Matti Mäkinen. Ohjelmiston käyttöliittymään kohdistuvat riskit määrittelyvaiheen käyttöliittymäsuunnittelun jälkeen. Master's thesis, Helsingin yliopisto, 2005.

[Nie00] Jacob Nielsen. Jakob nielsen's alertbox: Why you only need to test with 5 users. [verkkodokumentti]. Julkaistu 19.3.2000. [viitattu 4.4.2010]. Saatavissa: http://www.useit.com/alertbox/20000319.html.

[Nie05] Jacob Nielsen. Jakob nielsen's alertbox: Ten usability heuristics. [verkkodokumentti]. Julkaistu 2005. [viitattu 21.3.2010]. Saatavissa: http://www.useit.com/papers/heuristic/heuristiclist.html.

[Nie07] Jacob Nielsen. How to conduct a heuristic evaluation. [verkkodokumentti]. Julkaistu 2007. [viitattu 21.3.2010]. Saatavissa: http://www.useit.com/papers/heuristic/heuristic_evaluation.html.

[Off02] Jeff Offutt. Quality Attributes of Web Software Applications. *IEEE Software*, 19(2):25–32, 2002.

[OLR01] Luis Olsina, Guillermo Lafuente, ja Gustavo Rossi. Specifying Quality Characteristics and Attributes for Websites. Kirjassa *Web Engineering, Software Engineering and Web Application Development*, sivut 266–278, London, UK, 2001. Springer-Verlag.

[Pet06] Jennifer Petrie. Mixed-Fidelity Prototyping of User Interfaces. Master's thesis, University of Saskatchewan, Canada, 2006.

[Pre04] Roger S. Pressman. *Software Engineering: A Practitioner's Approach, 6th edition*. McGraw-Hill College, 2004.

[Roy70] W. Royce. Managing the development of large software systems. Kirjassa *Proceedings of IEEE WESCON*, sivut 1–9, August 1970.

[RSI96] Jim Rudd, Ken Stern, ja Scott Isensee. Low vs. high-fidelity prototyping debate. *interactions*, 3(1):76–85, 1996.

[Sta03] Julie Stanford. HTML Wireframes and Prototypes: All Gain and No Pain. [verkkodokumentti]. Boxes And Arrows, Julkaistu 6.1.2003. [viitattu 10.8.2009]. Saatavissa: http://www.boxesandarrows.com/view/html_wireframes_and_prototypes_all_gain_and_no_pain.

[STG03] Reinhard Sefelin, Manfred Tscheligi, ja Verena Giller. Paper prototyping - what is it good for?: a comparison of paper- and computer-based low-fidelity prototyping. Kirjassa *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, sivut 778–779, New York, NY, USA, 2003. ACM.

[Sze94] Pedro Szekely. User Interface Prototyping: Tools and Techniques. Kirjassa *In Proceedings of INTERCHI'93*, 1994.

[Tid05] Jenifer Tidwell. *Designing Interfaces: Patterns for Effective Interaction Design*. O'Reilly Media, Inc., 2005.

[Tox09] Anders Toxboe. User Interface Design Pattern Library. [WWW]. 2009. [viitattu 3.8.2009]. Saatavissa: http://ui-patterns.com/.

[VRR99] Jaya Vaidyanathan, Jason E. Robbins, ja David F. Redmiles. Using html to create early prototypes. Kirjassa *CHI '99: CHI '99 extended abstracts on Human factors in computing systems*, sivut 232–233, New York, NY, USA, 1999. ACM.

[WTA02] M Walker, L Takayama, ja Landay J. A. High-fidelity or low-fidelity, paper or computer medium? Kirjassa *Proceedings of the Human Factors and Ergonomics Society 46th Annual Meeting*, sivut 661–665, 2002.

[Yas07] Ansar-Ul-Haque Yasar. Enhancing experience prototyping by the help of mixed-fidelity prototypes. Kirjassa *Mobility '07: Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology*, sivut 468–473, New York, NY, USA, 2007. ACM.