Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan koulutusohjelma

Kandidaatintyö

Lauri Naukkarinen

WEB-MAKROT JA KÄYTTÄJÄAUTOMAATIO WEB-SOVELLUKSESSA: PROTOTYYPIN SUUNNITTELU JA TOTEUTUS

Kandidaatintyön aihe on hyväksytty 21.2.2011

Työn tarkastaja: Tutkijaopettaja Uolevi Nikula

Työn ohjaaja: Tutkijaopettaja Uolevi Nikula

Lappeenranta 20.6.2011

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan koulutusohjelma

Lauri Naukkarinen

Web-makrot ja käyttäjäautomaatio web-sovelluksessa: prototyypin suunnittelu ja toteutus

Kandidaatintyö

20.6.2011

42 sivua, 6 kuvaa, 3 taulukkoa

Työn tarkastaja: Tutkijaopettaja Uolevi Nikula

Hakusanat: web-makro, makro, web-ohjelmointi, web-sovellus, SaaS-sovellus
Keywords: web macro, macro, web programming, web application, SaaS application

Tämän kandidaatintyön aiheena on web-sovelluksen käyttäjäautomaatio web-makrojen avulla. Työssä esitellään prototyyppitoteutus SaaS-sovellukseen kohdistetusta web-makrojärjestelmästä. Työn teoriaosa esittelee web-automaation ja web-ympäristön keskeisen teknologian. Ratkaisuosa pohjustaa ratkaisuun käytetyn lähestymistavan, esittelee työn prototyyppiratkaisun ja arvioi ratkaisun lopputulosta. Lopputuloksena havaitaan, että prototyyppiratkaisu onnistuu selvittämään makrotoiminnallisuuden toteuttamiseen liittyvät tekniset haasteet. Ratkaisun merkittävimmäksi tekniseksi haasteeksi arvioidaan selainyhteensopivuuden saavutus ja ylläpito. Ratkaisun hyödynnettävyys koko web-sovelluksen mittakaavassa todetaan kuitenkin heikoksi, koska sovelluksen liiketoimintalogiikka rajoittaa makroparadigman käyttöä.

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology Management
Degree Program in Information Technology

Lauri Naukkarinen

Web macros and user automation in web application: designing and implementing a prototype

Bachelor's Thesis

20.6.2011

42 pages, 6 figures, 3 tables

Examiners: Associate professor Uolevi Nikula

Keywords: web macro, macro, web programming, web application, SaaS application

The subject of this bachelor's thesis is web application user automation through use of web macros. The thesis presents a prototype implementation of a web macro system for a SaaS application. The theory part of the thesis introduces the key technologies of web automation and web environment. The solution part explains the selected approach, provides the implementation walk-through, and evaluates the outcome of the solution. The outcome shows that the prototype implementation succeeds in solving all the technical challenges that are directly related to implementing the web macro functionality. Web browser compatibility and maintenance is estimated to be the biggest technical challenge for the prototype. However, the feasibility of the solution in the full scope of the web application is low, because of the restrictions that the application business logic imposes on the utilization of the web macro paradigm.

ALKUSANAT

Vihdoin valmista. Tästä esimerkistä Nooralle puhtia oman gradunsa kanssa, toivottavasti saat tulokset ja graafit paperille!

Työn lopussa tuntuu, että tähän kandipaperiin mahtui vain aiheen pintaraapaisu, sillä paljon jäi vielä sanomatta. Minulle tämä oli kuitenkin ennen kaikkea erittäin opettavainen projekti, varsinkin web-standardien ja JavaScript-kielen osalta.

Joten kiitokset Risto Matikaiselle erittäin kiinnostavasta kandityöaiheesta. Tero Kaijalle kiitokset ideoista ja ajatustenvaihdosta aiheen osalta. Uolevi Nikulalle kiitokset aiheen rajauksen vinkeistä ja työn ohjauksesta.

Lappeenrannassa 20.6.2011.

Lauri Naukkarinen

SISÄLLYSLUETTELO

LYHENNELUETTELO

Ajax	Asynchronous JavaScript and XML
ASP	Active Server Pages
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	HyperText Markup Language
SaaS	Software as a Service
URL	Uniform Resource Locator
WWW	World Wide Web
XHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language
XPath	XML Path Language
XQuery	XML Query Language

1 JOHDANTO

Verkkopalvelut yleistyvät ja monipuolistuvat samalla kun käyttäjäkeskeisyys korostuu. Sisältö ei ole kaikki kaikessa, myös käytettävyys ja tiedon esitystapa ovat ratkaisevassa roolissa [1]. Käyttäjälle sovellus on lähinnä päämäärää palveleva paketti eikä kokoelma yksittäisiä toimintoja [2]. Käyttäjällä on päämäärä eli tarve, jonka hän haluaa saavuttaa mahdollisimman vaivattomasti. Niinpä käyttäjän näkökulmasta automatisaatio usein helpottaa sovelluksen suoraa käyttöä. Käyttäjien erilaiset ja yksilölliset tarpeet luovat kuitenkin haasteita web-sovelluksen kehitystyölle: mitkä toiminnot voidaan paketoida, mitkä automatisoida? Palvelut saattavat sisältää manuaalista toistoa ja turhauttavaa navigaatiota, mutta ainakin toiminnot ovat saatavilla ja käytettävissä. Ratkaisuna voidaan tarjota käyttäjäkohtainen automaatio, joka toteutetaan manuaalikäytön imitoinnin kautta. Imitaation avulla kone voi käyttää sovellusta sen luonnollisen käyttöliittymän lävitse, käyttäjää imitoiden.

Safonov et al. [3] ehdottaa imitoivan automaation toteuttamiseksi *web-makro* -konseptia. Makrot tarkoittavat ohjelmointia esimerkin kautta, ja niitä on käytetty automatisoimaan aikaa kuluttavia mutta toistettavissa olevia toimenpiteitä. Safonovin mukaan ”nauhoita ja toista” -konseptin etuna nähdään erityisesti ymmärrettävyys ja yleiskäyttöisyys. Samaa ”nauhoita ja toista” -ratkaisua voidaan soveltaa moneen eri käyttötarkoitukseen: automaation ja ajansäästön saavuttamiseen, esimerkki- ja esittelynauhoitteiden tekemiseen sekä kehityksen tukena toimivien testitapausten luomiseen eli ohjelmistotestaukseen.

1.1 Tausta

Tämä työ käsittelee *web-sovelluksen* ja erityisesti *SaaS-sovelluksen* (Software as a Service) käyttäjäautomaatiota. SaaS-palvelu tarkoittaa ohjelmistojakelumallia, jossa kokonaisuus tuodaan asiakkaan käytettäväksi verkkopalveluna [4]. Tästä kokonaisuudesta voidaan eritellä kaksi osaa: ohjelmisto ja palvelu. Termi SaaS-sovellus on silloin SaaS-palvelun tekninen ja konkreettinen ohjelmistotuote, joka välittää palvelun saataville. Tämän työn kannalta SaaS-sovelluksen merkittävimmät luonteenpiirteet ovat nopea muutos- ja julkaisutahti sekä Ajax-teknologian (Asynchronous JavaScript and XML) hyödyntäminen [1,5,6]. Ajax-teknologiaa käytetään laajentamaan web-sovelluksen käyttöliittymä ja

interaktio yhdeksi saumattomaksi kokonaisuudeksi [7]. Käyttäjäautomaation kannalta nämä luonteenpiirteet ovat suuria haasteita.

Työn tulokset toteutetaan lappeenrantalaiselle Netvisor Oy -yritykselle, jonka päätuote on sähköisen talous- ja yrityshallinnon SaaS-palvelu. Työn aihepiiri nousi esille kandidaatintyön aihetta pohtiessa yrityksen ohjelmistopäällikön aloitteesta. Aihetta lähdettiin tutkimaan ennen kaikkea erittäin kiinnostavana ja haastavana ongelmana, jonka selvityksen toivotaan tarjoavan tuotteistamismahdollisuuksia. Yrityksen tavoitteena on arvioida web-makrojen hyödynnettävyyttä yrityksen SaaS-tuotteessa.

1.2 Tavoitteet ja rajaukset

Tämän työn tavoitteena on suunnitella ja toteuttaa web-makrot toteuttava prototyyppi, johon työssä viitataan nimellä *makrokirjasto*. Työn teoriaosassa tutustutaan web-makroihin ja web-sovellusten toimintaympäristön standardeihin sekä teknologiaan, joita hyödynnetään makrokirjaston toteutuksessa. Työn ratkaisuosuudessa keskitytään prototyypin esittelyyn ja arviointiin. Prototyypin päätarkoituksena on web-makrojen mahdollisuuksien ja haasteiden esille nostaminen. Lisäksi prototyypin on tarkoitus tukea työn ratkaisun arviointia ja tulosten pohdintaa. Tavoitteena on, että työn lopputuloksena syntyy makrokirjaston ansiosta myös alustava esitys web-makrojen teknisistä mahdollisuuksista ja rajoitteista kohdeyrityksen SaaS-tuotteessa. Työn näkökulman takia ratkaisun sisältö ja tulosten pohdinta rajataan vain makrotoiminnallisuuden yksityiskohtiin.

Työn tutkimuskysymykset ovat:

1. Miten web-makrot voidaan toteuttaa kohdesovelluksessa?
2. Mitkä ovat työn ratkaisun tekniset mahdollisuudet, haasteet ja rajoitteet?

Työssä mainittuja web-maailman termejä ja teknologioita avataan ja selitetään vain hyvin pintapuolisesti. Näiden teknologioiden ja termien tarkempi käsittely on tämän työn laajuuden ulkopuolella. Näistä rajoitteista johtuen työn ratkaisun esittely olettaa, että lukija ymmärtää web-ohjelmoinnin, web-teknologian ja verkkoselainympäristön periaatteet.

1.3 Työn rakenne

Työn toinen luku esittelee makrokonseptin ja määrittelee web-makro -termin. Luvussa esitellään myös web-automaation käyttökohteet ja pääongelmat. Kolmas luku määrittää web-toimintaympäristön kontekstin ja esittelee työn ratkaisussa hyödynnetyt avainteknologiat. Neljäs luku esittelee työn kohdesovellusympäristön ja pohjustaa työn pääongelman sekä työssä valitun ratkaisutavan. Luvun kuluessa valittu ratkaisusuunnitelma esitellään ja perustellaan. Viides luku esittelee työn ratkaisun eli makrokirjaston. Luvussa käydään läpi ratkaisun pääperiaatteet ja keskeiset yksityiskohdat. Kuudes luku sisältää työn pohdintaosuuden ja ratkaisun arvioinnin. Luku esittelee työn ratkaisun hyvät ja huonot puolet sekä työn aikana tehdyt havainnot. Viimeisenä tunnistetaan ja esitellään työn tuloksena syntyneet johtopäätökset.

2 WEB-MAKROT JA KÄYTTÄJÄAUTOMAATIO

Makroja käytetään tehtäväautomaation toteuttamiseen ja tavoitelähtöiseen ongelmaratkaisuun. Kurlanderin ja Feinerin [8] mukaan makrotyökalujen tavoite on lopputuloksen saavutus nopeammin ja helpommin ilman aikaa sekä tarkkuutta vaativia välivaiheita. Tietotekniikan ”nauhoita ja toista” -konsepti on peräisin tekstiä muokkaavista komentoriviohjelmista, joissa makrot ovat osoittautuneet erittäin tehokkaiksi. Käyttöliittymät ovat perinteisesti olleet ohjelmointikielimäisiä, mikä on tehokasta mutta peruskäyttäjälle hankalaa. Tästä syystä Kurlander ja Feiner toteavat graafiseen käyttöliittymän parantavan makrojen käytettävyyttä ja hyödynnettävyyttä. Grafiikan avulla makron käytön valvonta helpottuu, ja samalla makroja voidaan käyttää mm. esityksen ja opetuksen apuvälineenä. Lisääntyvä käyttäjäautomaatio ja käytettävyyden sekä oikoteiden tavoittelu ovat olleet pitkään tavoitteita myös web-maailmassa [1,9].

Web-makro määritellään Safonov et al. [3] mukaan verkkoselaimessa tapahtuvaksi käyttäjätoimien ”nauhoita ja toista” -toiminnallisuudeksi. Käyttäjä aloittaa nauhoituksen, suorittaa joukon toimintoja ja lopulta merkitsee makron nauhoituksen päättyneeksi. Kun makro toistetaan alkuperäisestä lähtötilanteesta, päästään makron avulla nauhoitustilanteen lopputulosta vastaavaan tilanteeseen. Tälle toiminnallisuudelle ei kuitenkaan ole kirjallisuudessa vakiintunutta termiä, vaan lähteestä ja käyttötarkoituksesta riippuen vallitsevana terminä esiintyy mm. web-automaatio [2,10], web-makro [11], kumpikin tasapuolisesti [12] ja yleisesti ”nauhoitus ja toisto” [13]. Tässä työssä käytetään termiä web-makro, koska se parhaiten kuvaa työn tuloksena syntyvän järjestelmän käyttötarkoitusta.

2.1 Käyttötarkoitukset ja suurimmat haasteet

Tällä hetkellä web-automaatioratkaisuja käytetään mm. yritysten välisen tiedon integraatiossa, mashup- ja proxy-sovellusten (yhdistävät tietoa monesta lähteestä yhdeksi dynaamiseksi kokonaisuudeksi) tuottamisessa, web-sovellusten automaattisessa testauksessa sekä web-hakurobottien toteuttamisessa [10]. Näissä sovelluksissa käyttäjätoimien imitoiminen on kuitenkin vain pieni osa kokonaisratkaisua. Scaffidi et al. [11] pohtii artikkelissaan syitä web-makrojen vähäiselle käytölle ja painottaa, että vaikka

tarvittava teknologia ja esitetyt ratkaisut ovat kehittyneet vuosi vuodelta, niin silti yleispätevää ratkaisua ei ole vielä löydetty. Artikkelin mukaan web-makrojen suuria haasteita ovat web-ympäristön monipuolisuus ja nopea muutostahti.

Web-automaatioratkaisujen laatu riippuu kriittisesti niiden kyvystä tunnistaa ja toistaa käyttäjätoimien tuloksena syntyviä navigaatiopolkuja [10]. Web-makrojen yleisimpinä puutteina pidetään ratkaisun riittämättömyyttä tai yleispätevyyden puutetta. Tarkat ratkaisut ovat tiukasti sidoksissa toimintaympäristöön, mutta toisaalta yleiset ratkaisut ovat hyvin naiveja sekä virhealttiita. Haasteet korostuvat dynaamisiksi rakennetuissa verkkosivuissa, joiden käyttäjänäkymä, sisältö ja rakenne ovat vapaita muuttumaan käytön aikana. Dynaamisen sivun vastakohta on staattinen sivu: palvelin siirtää sivun asiakkaalle, eikä sivun näkymä tai sisältö muutu ennen uuden sivun hakemista. Erityisen haasteellinen osa dynaamisen sivun käsittelyä on tapahtumakehys, jota hyödyntäen sivu reagoi käyttäjän toimintaan muuttamalla sivun ulkoasua ja tietosisältöä. Tapahtumakehys tarjoaa kuitenkin mahdollisuuden käyttäjätoimien koneelliseen jäljittelyyn.

Web-makrojen kohtaamat haasteet jakautuvat karkeasti kolmeen luokkaan: *teknologiahaasteisiin, paradigmahaasteisiin ja kolmantena yleisiin verkon ja toimintaympäristön luonteesta johtuviin haasteisiin* [11]. Teknologiahaasteet riippuvat suoraan ratkaisuun käytetystä teknologiasta ja ratkaisun kohdeympäristöstä, kuten web-sivun standardien ja koodikirjastojen mahdollisuuksista ja rajoitteista. Paradigmahaasteet liittyvät web-makrojen luonnin, ylläpidon, esityksen ja muokkauksen käytettävyyteen ja ymmärrettävyyteen. Ne ovat usein seurausta makrojärjestelmien heuristisista ratkaisuista, jotka johtuvat web-ympäristön epävarmasta ja virheherkästä luonteesta. Kolmannen luokan haasteet ovat käytännössä ohjelmointikielten ja web-ympäristön yleisiä ominaisuuksia. Näitä ovat mm. viiveet ja jatkuvan kehityksen mukanaan tuoma muutos sekä sen hallinta.

2.2 Valmistuotteet

Web-automaation toteuttamiseen on tarjolla useita valmiita ratkaisuja. Ratkaisut voivat olla laajoja ja sovellusten välisiä, kuten Apple Automator, joka toimii Safari-selaimen lisäksi myös monessa muussa sovelluksessa [14]. Lähes aina ratkaisut tarjotaan kuitenkin selainlaajennoksena, kuten Firefox-lisäosa Chickenfoot [15]. Eri valmistuotteet vaikuttavat

niin samankaltaisilta, että parhaiten niiden eron ymmärtää käyttötarkoitusta tutkimalla. Perinteisin käyttötarkoitus on loppukäyttäjän skriptityökaluna toimiminen. Web-maailmassa skripti tarkoittaa verkkosivulla olevaa pienohjelmaa, jolla dokumenttia voidaan muokata tai muuten käsitellä [16]. Nämä skriptityökalut jakaantuvat käyttäjäryhmänsä ja käytettävyytensä mukaisesti edelleen tavallisten loppukäyttäjien tuotteisiin ja ohjelmointitaitoa vaativiin spesiaalityökaluihin. Jälkimmäisten käyttöliittymä on käytännössä ohjelmointikielipohjainen. Muita keskeisiä käyttötarkoituksia ovat mm. tiedonlouhinta ja web-sovellustestaus.

Chickenfoot [15] on hyvä esimerkki ohjelmointisuuntautuneesta makrotuotteesta. Se perustuu avoimeen lähdekoodiin, ja tarjoaa oman JavaScript-koodin lisäämistä ja liittämistä verkkosivuistoille. Näitä koodiskriptejä voidaan käyttää mm. esitysasun muokkaamiseen, virheiden korjaukseen ja omien toimintojen rakentamiseen. CoScripter [17] ja iMacros [18] ovat päinvastoin esimerkkejä käytettävyyttä ja ymmärrettävyyttä painottavista tuotteista. Kumpikin mainitsee käyttötarkoituksenaan toistuvien tehtävien automatisoinnin ja web-makrojen nauhoituksen. CoScripter on ilmainen selainlisäosa, jossa ”nauhoita ja toista” -makrojen ”askeleet ovat esillä ja muokattavissa ilman ohjelmointitaitoa [17].” iMacros on samanlainen mutta kaupallinen tuote, jota markkinoidaan myös web-testaukseen ja tiedonlouhintaan.

3 WEB-SOVELLUSTEN TEKNINEN TOIMINTAYMPÄRISTÖ

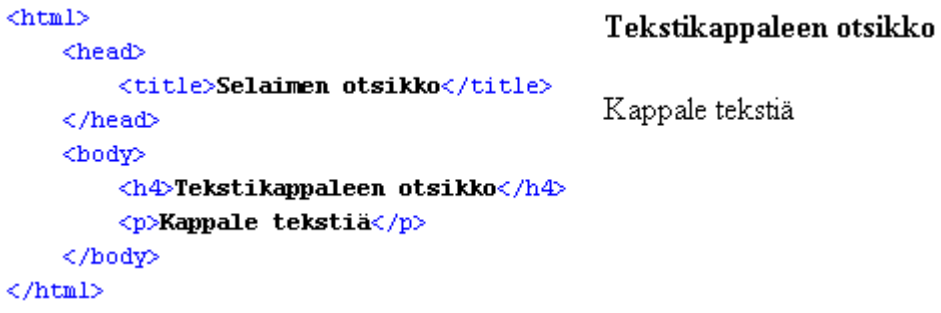
Nykyaikaiset WWW-maailman (World Wide Web) sovellukset ovat sisällön ja toiminnallisuuden osalta jo täysin työpöytäsovelluksien veroisia saumattomia ja dynaamisia kokonaisuuksia [19]. Korkealla tasolla web noudattaa kuitenkin yhä asiakas-palvelin -mallia, jossa asiakas suorittaa sivupyynnön ja saa vastaukseksi palvelimen lähettämän käyttäjänäkymän eli verkkosivun. Tämä käyttäjänäkymä eli siirrettävä tieto on perinteisesti HTML- (HyperText Markup Language) tai XHTML-dokumentti (eXtensible HyperText Markup Language), johon voidaan lisätä asiakaspuolen sovelluslogiikka skriptikielen avulla. Web-sovellusten lopullinen käyttöliittymä muodostuu käyttäjänäkymän ja verkkoselaimen yhdistelmänä.

Web-makrojen toimintaympäristö määrittyy käyttäjän verkkoselaimen toimintojen ja verkkosivun dokumentin sisällön perusteella. Tämän toimintaympäristön ymmärtämiseksi on hallittava dokumenttityypit (HTML- ja XHTML-määritys) sekä dokumenttityökalut, joista merkittävimmät tässä työssä ovat DOM-rajapinta (Document Object Model) ja XPath-hakukieli (XML Path Language). Näistä työkaluista ensimmäinen tarjoaa dokumentin rakenteen ja sisällön luettavaksi sekä muokattavaksi, ja jälkimmäinen toteuttaa lausekkeet, joilla dokumentin osiin voidaan viitata osoitemaisesti. Web-makrojen toteuttamiseksi nämä työkalut täytyy kuitenkin yhdistää verkkoselaimeen ja verkkodokumenttiin. Tämä onnistuu JavaScript-ohjelmointikielen avulla.

3.1 Web-maailman dokumenttimääritykset ja merkintäkielet

HTML on merkintäkieli, jolla voidaan kuvata WWW-ympäristön tietoresursseja eli HTML-dokumentteja [16]. Dokumentin sisältö jakaantuu otsikkotiedoiksi, esityksen merkinnöiksi sekä esitettäväksi tiedoksi. Otsikkotiedot sisältävät standardin ja dokumentin tyyppitiedot sekä myös dokumenttiin liitettyjen ulkoisten resurssien määritykset, joita ovat mm. koodi- ja tyylikirjastot. Dokumentin sisältö koostuu semanttisista rakenteista (esim. otsikot, nappulat, listat ja linkit), jotka kuvaavat tiedon tyyppiä, rakennetta ja tarkoitusta. Rakenteita kutsutaan elementeiksi. Käyttäjänäkymä syntyy verkkoselaimen avulla, kun konekielinen HTML-dokumentti muutetaan käyttäjäystävälliseen esitysmuotoon elementtejä tulkitsemalla (esimerkki kuvassa 1). Web-sovellusten käyttäjäsyötteet

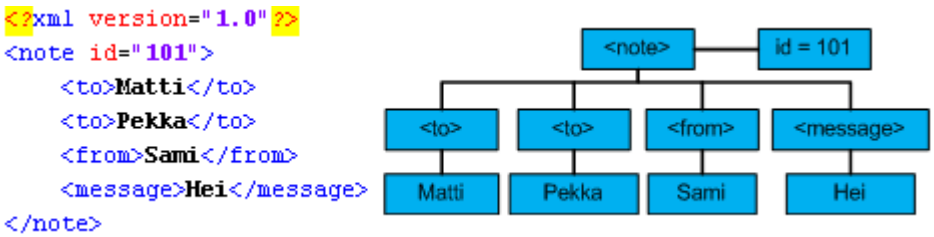
perustuvat yhä pääosin HTML:n lomake-elementtiin (form) ja sen syötekenttiin (input), esim. nappeihin, tekstikenttiin ja monivalintalistoihin. Verkkosivun ”täytetty” lomake palautetaan arvoineen syötön (submit) kautta palvelimen ja web-sovelluksen luettavaksi.



(a) (b)

Kuva 1. HTML-dokumentti: a) konekieliversio; b) käyttäjänäkymä.

XML (eXtensible Markup Language) on merkintäkieli, jonka avulla tietoon voidaan merkitä metatietoa eli tietoa tiedosta [20]. Ensisijaisesti kielen tavoitteena on edistää tiedon käytettävyyttä ja sanomien siirtoa WWW-ympäristössä. XML on tarkoitettu siirrettävän tiedon tallentamiseen, minkä standardi mahdollistaa tarjoamalla tavan määritellä ja rakentaa tietorakenteita. XML-dokumentti koostuu sisältöä merkkaavista elementeistä ja attribuuteista sekä elementtien sisään sijoitetusta tiedosta. Elementit ovat dokumentin loogisia rakenteita ja attribuutit ovat elementeille asetettuja nimettyjä arvoja. Jos XML-dokumentti ajatellaan puurakenteena niin elementit ovat puun solmuja. Silloin attribuutit ovat solmujen ominaisuuksia, ja tietoalkiot sijoitetaan puun lehdille. Esimerkki XML-dokumentista ja sitä vastaavasta puurakenteesta näkyy kuvassa 2.



(a) (b)

Kuva 2. XML-dokumentti: a) tekstimuodossa; b) puumuodossa.

Dokumenttien tiedon lukemiseen ja hakemiseen on kehitetty useita eri standardeja, joista merkittävin on XPath-kyselykielen versio 1.0 [21]. Kielen merkittävyys perustuu lähes

täydelliseen selaintukeen, jota ei ole vielä tarjolla mihinkään muuhun vastaavaan standardiin [22]. Esimerkiksi selkeästi laajempi XQuery-standardi (XML Query Language) voidaan tällä hetkellä toteuttaa vain verkkosivulle liitetyn ulkoisen kirjaston kautta [23]. Teknisesti XPath-kieli perustuu XML-dokumentin puumuotoiseen mallinnukseen [24]. Kielessä hakuehdot mallinnetaan lausekkeina, joihin voi sijoittaa puumallin rakennetietoa (osoitteita), muuttujia ja kielen funktioita sekä näiden välisiä ehtoja.

HTML-standardi yhdistyy XML-teknologian parempaan laajennettavuuteen XHTML–standardissa. Tarve erilaiselle verkkosivustandardille syntyi, kun HTML–määritys kävi riittämättömäksi monipuolisen sisällön merkkaukseen ja käsittelyyn, mikä johtui erityisesti web-sovellusten kehityksestä [25]. XHTML mahdollistaa verkkosivun käsittelyn aitona XML-dokumenttina, mikä helpottaa dokumenttien määritystä, laajentamista ja dynaamista muokkausta, koska sivuja voi luoda ja muokata suoraan XML-työkalujen avulla. Nämä kaksi verkkosivustandardia eivät ole kuitenkaan keskenään täysin yhteensopivia. Eroja esiintyy dokumentin rakenteessa ja merkintätavassa sekä myös skriptikielellä toteutetussa dokumenttikäsittelyssä [22]. Dokumenttityypin määritys tapahtuu otsikossa, mutta virheellinen sisältö voi tiputtaa selaimen lukemaan XHTML-dokumenttia HTML-muodossa [26]. Perinteisesti verkon standardeja tulkitaan hyvin anteeksiantavasti.

3.2 DOM-rajapinta dokumentti- ja tapahtumakäsittelyyn

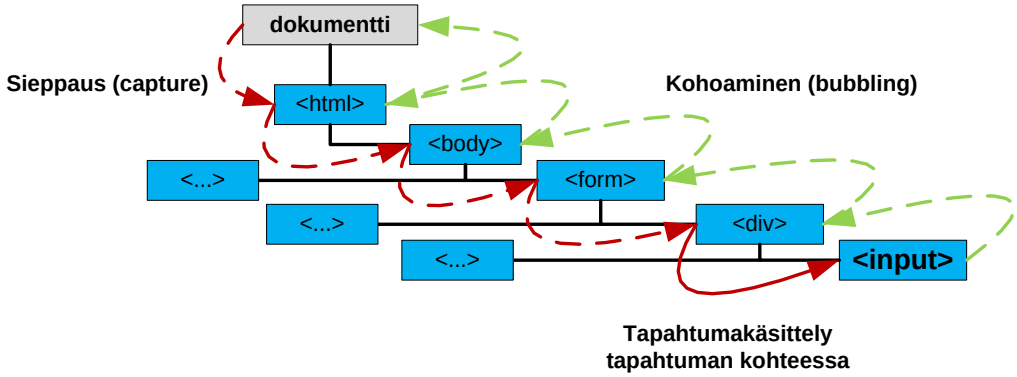
DOM (W3C DOM) on useasta standardista koostuva sovellusrajapinta dokumenttikäsittelyyn. Rajapinta määrittelee toiminnot loogisesti ja hierarkisesti rakentuville dokumenteille riippumatta ohjelmointikielestä tai dokumentin tyypistä [27]. Käytännössä DOM-rajapintaa voi soveltaa kaikkiin puurakenteen avulla kuvattaviin dokumentteihin. Rajapinnan toiminnoilla voi luoda ja rakentaa sekä lukea ja muokata dokumentin rakennetta ja sisältöä. Rajapintakuvaukseen liittyvät standardit esitetään tasokohtaisina standardiperheinä [21]. Ensimmäinen ja varhaisin taso sisältää standardiytimen dokumentin lukemiseen ja muokkaukseen. Toinen taso lisää rajapintaan standardit tyyliasetusten (jotka ovat solmujen attribuuteissa sijaitsevia CSS-tyylimäärityksiä (Cascading Style Sheets)) ja tapahtumakäsittelymallin hallinan määritykselle. Kolmas taso on vielä luonnosvaiheessa, mutta tavoitteena on tuki yhä laajemmalle kirjolle tapahtumatyyppejä sekä parannuksia dokumentin elementtien

tunnistukseen ja siirto-operaatioihin puurakenteessa [28]. Nykyaikaiset selaimet tukevat täydellisesti toista ja osittain kolmatta DOM-tasoa [29].

DOM-tapahtumakehys on kriittinen osa kaikkia web-sovelluksia, koska koko web-interaktiomalli on tapahtumapohjainen [21]. Tapahtumasuuntautuneen ohjelmointi-paradigman (event-driven programming) mukaan ohjelman suoritus ajatellaan jonona tapahtumia ja ohjelman tarkoitus on tapahtumien luonti ja näihin reagointi [30]. Tapahtumien (event) avulla dokumentti reagoi herätteisiin, joista valtaosa verkkomaailmassa on käyttäjäsyötteitä [21]. Jokainen käyttäjäsyöte (esim. kenttään kirjoitus tai napin painaminen) aiheuttaa joukon tapahtumia. Dokumentin solmuille on mahdollista rekisteröidä funktioita, jotka kuuntelevat solmulle saapuvia tapahtumia niiden tyypin (esim. ”hiiren painallus”) perusteella. Saapuva tapahtuma laukaisee solmun tapahtumatyypille rekisteröidyn funktion suorituksen. Esimerkiksi tekstirivin väriä on mahdollista vaihtaa kun hiiren osoitin on rivin päällä. Tämä tapahtuu asettamalla HTML-dokumentin rivi-elementille funktio, jonka herätteenä toimii hiiren osoittimen saapuminen ja jonka koodisisältönä on rivin värin vaihtaminen.

Tapahtuman läpikäynti on kolmiosainen prosessi, joka perustuu dokumentin hierarkiseen puurakenteeseen. Käsittelyn keskipisteessä on tapahtuman mallintava tapahtumaolio. Tapahtumakäsittelyn kolme osaa ovat sieppaus-, kohde- ja kohoamiskäsittely [31]. Käsittelykokonaisuutta voi ajatella matkana, jonka aikana tapahtumaolio ”kulkee” dokumentin juurisolmusta tapahtuman kohdesolmuun, ja sieltä takaisin dokumentin juurisolmuun. Menettelyn tarkoituksena on tapahtumareagoinnin ketjutus. Jokainen hierarkian kohdetta ylempänä oleva solmu voi siepata (capture) olion ja reagoida herätteeseen ennen kohdekäsittelyä. Vastaavasti paluumatkan aikana ylemmät solmut voivat reagoida kohoavaan (bubbling) olioon kohdekäsittelyn jälkeen. Kolmivaiheinen käsittely on esillä kuvassa 3. Tapahtumakäsittelijät voivat mm. muokata tapahtumaoliota tai pysäyttää (poistaa) olion käsittelyn. Kohoamisen aikana tapahtumaolio kutsuu vuorollaan jokaisen ”ohitetun” solmun omaa tapahtumakäsittelijää, aivan kuin jos tapahtuma olisi ollut niille alunperin osoitettu. Toisin kuin kohoaminen, sieppaus ei tapahdu automaattisesti, vaan sitä varten tapahtumakäsittelijä on rekisteröitävä eksplisiittisesti sieppaustila päällä. Tapahtumahallintaa voi pitää haasteellisena, koska

kuuntelevia funktioita voi rekisteröidä ja poistaa dynaamisesti sekä koska tapahtumat voivat aiheuttaa ketjureaktioita eli uusia tapahtumia [10].



Kuva 3. DOM-tapahtumakäsittelymalli: tapahtumaolio dokumenttipuussa [32].

3.3 JavaScript-kieli, Ajax-teknologia ja asynkroniset funktiot

JavaScript on yleisin verkkosivuilla käytetty skriptikieli [21]. Se on tulkattava ohjelmointikieli, jota hyödynnetään dokumenttisisällön dynaamisessa muokkauksessa sekä asynkronisten (tapahtuvat taustalla ja käyttäjältä piilossa) palvelupyyntöjen toteutuksessa. JavaScript-koodi ladataan sivupyynnön yhteydessä ja se suoritetaan asiakaskoneella verkkoselaimen sisällä. Koodi muuttaa staattisen verkkosivun interaktiiviseksi kokonaisuudeksi, jossa käyttäjänäkymää ja sisältöä on mahdollista muokata ilman sivupyyntöjä. Ohjelmistokehitystä JavaScript-kielellä pidetään nopeana mutta virhealttiina, joten kehitystä tehdään pääosin iteratiivisesti sekä kokeilupohjaisesti, rakentaen vähän kerrallaan ja testaten usein [33]. Kielen merkittävin haaste on standardien moninainen kirjo: on olemassa standardeja, de facto -käytäntöjä sekä selainkohtaisista ratkaisuista johtuvia poikkeuksia.

Ajax on JavaScript-kielen kautta yhdistetty teknologiajoukko, joka muuttaa verkkosivun käyttökokemusta sulavaksi ja työpöytäsovellusmaiseksi. Termin määritti Garret [7] vuonna 2005 ja alkuvaiheessa se nousi julkisuuteen Googlen verkkosovellusten esimerkin ansiosta. Ajax tuo mukanaan uuden arkkitehtuuritason, jonka tehtävänä on muokata käyttäjänäkymän ja -interaktion kulkua käyttäjän selaimessa. Taso on toteutettu JavaScript-koodilla ja se kommunikoi palvelimen kanssa hyödyntäen asynkronista (ja usein XML-muotoista) tiedonsiirtoa. Garretin mukaan Ajax-teknologian tarkoituksena on parantaa

sivuston käyttökokemusta piilottamalla verkkokommunikaatiossa tapahtuva esitys- ja siirtoviive. Lopputuloksena palvelujen käyttäjäystävällisyys kasvaa, koska tieto voidaan siirtää ja esittää saumattomasti ilman käyttäjälle näkyviä odotus- tai viivetapahtumia. Hyvä esimerkki Ajax-teknologian hyödyntämisestä on Google-haku, joka palauttaa ja muokkaa tuloksia välittömästi jokaisen kirjainsyötön jälkeen, ilman erillistä käyttäjäviivettä aiheuttavaa sivupyyntöä.

Dokumenteissa sijaitseva JavaScript-koodi voidaan jakaa synkroniseen ja asynkroniseen koodiin [21]. Synkroninen koodi tarkoittaa koodipalikkaa, joka suoritetaan rivi riviltä, alusta loppuun. Koodi muuttuu asynkroniseksi kun se rekisteröidään ajastimeen (JavaScript-kielessä setTimeout- ja setInterval-funktio), sijoitetaan kuuntelemaan tapahtumaa (DOM-tapahtumakäsittely) tai asetaan odottamaan Ajax-tiedonsiirron vastausta (JavaScript-kielen XMLHttpRequest-luokka). Tällöin kyseisen koodin suoritus siirretään asynkrooniseksi eli odottamaan valitun esiehdon täyttymistä. Näiden sisällöltään synkronisten koodipalikoiden suoritus etenee selaimessa kuitenkin jonona, koska kaikki koodi suoritetaan yhdessä säikeessä [22]. Vaikka koodia ei ajeta rinnakkaisesti, niin koodipalikoiden välinen suoritusjärjestys on (ilman eksplisiittistä hallintaa) oletettavasti täysin ennalta tuntematon. Selain valitsee ja suorittaa jonosta suorituskelpoisia palikoita eli niitä, joilla ei ole esiehtoa tai joiden esiehto on jo täyttynyt. Tapahtumakäsittelyn osalta tämä tarkoittaa sitä, että tapahtumat kulkevat läpi dokumenttipuun yksi kerrallaan ja että tapahtumakäsittelyn aikana luodut asynkroniset funktiot suoritetaan ”joskus” myöhemmin.

4 RATKAISUN POHJUSTUS JA ONGELMAN ESITTELY

Tässä työssä suunnitellaan ja toteutetaan prototyyppi web-makrot toteuttavasta *makrokirjastosta*, joka mahdollistaa käyttäjäsyötteiden *nauhoituksen* ja *toistamisen*. Työ keskittyy näiden kahden ominaisuuden toteutuksen suunnitteluun, esittelyyn ja arviointiin työn prototyyppiratkaisun avulla. Tässä luvussa käydään ensin läpi työn ratkaisun konteksti, jonka jälkeen esitellään työn suunnittelu- ja ratkaisutavat sekä perustelut näiden lähestymistapojen valinnalle.

Työn tuloksien kannalta prototyypin tehtävänä on auttaa vaatimusten hahmottamisessa ja haasteiden tunnistamisessa. Prototyyppiä kehitetään iteratiivisesti, koska oletetaan että makro-ongelma sisältää paljon haasteita, jotka tulevat esille vasta kehityksen edetessä. Haasteiden selvityksessä hyödynnetään kohdesovellustuntemusta ja suositaan sovelluskohtaisia ratkaisuja. Tämä on mahdollista, koska työn tekijä on Netvisor Oy:n ohjelmistosuunnittelija, jolla on vuoden kokemus kyseisen SaaS-palvelun kehitystyöstä. Prototyyppi toteutetaan web-sivulle liitettävänä JavaScript-kirjastona. Kirjaston liittäminen osaksi kohdesovellusta ja nauhoitettujen käyttäjäsyötteiden pysyvä tallennus on kuitenkin rajattu pois tämän työn laajuudesta.

4.1 Kohdesovelluksen esittely ja makrokirjaston tarve

Makrokirjaston kohdesovellus on yritys- ja taloushallinnon SaaS-palvelu *Netvisor*, joka sisältää mm. kirjanpidon, reskontran, henkilöstöhallinnan ja asiakkuudenhallinnan toimintoja. Palvelun teknistä osaa eli SaaS-sovellusta ajetaan yhdistelmänä Microsoftin ASP- (Active Server Pages) ja ASP.NET-teknologiaa. Tämän työn kannalta oleellista on kuitenkin vain sovelluksen asiakkaalle näkyvä osuus, eli asiakkaan verkkoselaimeen siirtyvä tieto ja sen tekninen toteutus, koska makrokirjasto käyttää web-sovellusta verkkosivun ilmentämän rajapinnan kautta. Yrityksen palvelussa web-sovelluksen näkymät toimitetaan XHTML-dokumenttina, joihin sisältyy JavaScript-koodia ja CSS-tyylimäärityksiä sekä Ajax-tiedonsiirtoa.

Toimeksiantajan kannalta työn ensisijaisena tarkoituksena on määrittää makrokirjaston mahdollisuuksia ja hyödynnettävyyttä. Makrokirjaston tarkkaa käyttötarkoitusta ei ole

vielä määritetty, mutta yksi mahdollinen käyttötapaus on esittely- ja opastustyökalu. Yrityksen ohjelmistopäällikön mukaan tällaista työkalua voisi käyttää nauhoittamaan demoja, joita voidaan myöhemmin toistaa esittely- ja opastustarkoituksessa (esim. myynnin ja käyttötuen apuna). Tässä käyttötapauksessa makrojärjestelmän keskeisiä etuja olisivat helppo ja vaivaton nauhoittaminen sekä muutoksien yli kestävät tallenteet. Makronauhoitteet heijastaisivat SaaS-palvelun tuotantoympäristön nykyistä tilaa ja ulkoasua, eivätkä ne olisi sidottuja nauhoitushetken ulkoasuun tai sisältöön. Lisäksi nauhoitteet voisi tarvittaessa siirtää saataville järjestelmän läpi, mistä esimerkkinä tapaus, jossa tukihenkilö nauhoittaa tukipyynnön vastauksen makromuotoiseksi tallenteeksi.

Työ lähtee siitä oletuksesta, että sopivaa valmistuotetta ei ole olemassa. Valmistuotteet oletetaan asiakaspainotteisiksi, koska ne keskittyvät automatisoimaan käyttäjän selaimessa tapahtuvia toimenpiteitä ilman integraatiomahdollisuuksia. Työn ratkaisun yhtenä tavoitteena on saumattoman sovellusintegraation saavutus: makronauhoitteet pyritään tallentamaan ja hallinnoimaan web-sovelluksen kautta. Tästä syystä valmisratkaisun hyödyntäminen on vaikeaa. Makrokirjaston kehityksen toivotaan myös paljastavan makrotoiminallisuuden mahdollisuuksia ja haasteita, jolloin valmisratkaisu epäilemättä rajoittaisi ratkaisun kehityssuuntaa ja hyödyntämismahdollisuuksia.

4.2 Työn pääongelma ja ratkaisun lähtökohdat

Työn ongelma kiteytyy *nauhoitus*- ja *tunnistustavan* valintaan. Nauhoitustapa määrittää kuinka käyttäjäsyötteet havaitaan ja siepataan tallennusta varten. Koska toimintaympäristönä on web-dokumentti, niin käyttäjäsyötteisiin liittyy myös tunnistusongelma: kuinka käyttäjäsyötteen kohde voidaan määrittää muutoksia kestävällä tavalla? Nauhoitusongelman ratkaisutapa on suurin makrokirjaston arkkitehtuuria määrittävä päätös, mutta lopputuloksen toimivuus riippuu kuitenkin kriittisesti myös oikean tunnistustavan valinnasta. Tämän työn kirjallisuuskatsauksen päätarkoituksena oli löytää pohja-aineistoa ja ratkaisuvaihtoehtoja näihin kahteen pulmaan. Ratkaisun kannalta tärkeimmäksi lähdeartikkeliksi osoittautui Montoto et al. [10], joka esittelee korkean tason mallin web-automaation toteutukseen ja arvioi eri ratkaisuvaihtoehtoja. Työn ratkaisussa käytetty nauhoitus- ja tunnistustapa pohjautuvat artikkelin suosituksiin.

Ratkaisusuunnitelman yksityiskohdat ja ratkaisun toteutus ovat kuitenkin täysin tämän työn omaa tuotosta.

4.2.1 Käyttäjäsyötteen nauhoitustapojen vertailu ja valinta

Tässä työssä nauhoituksen tavoitteena on luoda käyttäjäinteraktiota vastaava tallenne, jonka avulla interaktio voidaan myöhemmin toistaa. Hyvän nauhoitusratkaisun avulla iteraktion lopputulos pysyy muuttumattomana ja web-sovelluksen käyttö ei merkittävästi vaikeudu. Teknisellä tasolla sovellusta käytetään käyttäjäsyötteiden avulla, jotka siirtyvät tapahtumina dokumenttipuun sisälle. Pelkkä hiiren napin painallus dokumentin alueella muodostaa monta tapahtumaa, koska dokumentti rekisteröi jo hiiren osoittimen saapumisen ja napin painalluksen kahtena erillisenä tapahtumana. Pelkän nauhoituksen lisäksi ratkaisun on kyettävä huomioimaan myös tapahtumareagointiin ja -käsittelyyn kuluva aika, koska käsittely saattaa muuttaa dokumenttipuun sisältöä ja rakennetta. Jos syötteitä tallennetaan samalla kun dokumentti on muuttuvassa tilassa, on olemassa riski, että tallenne ei ole toistokelpoinen tai aina samaan lopputulokseen päätyvä.

Nauhoituksen toteuttamiseen on kaksi vaihtoehtoista lähestymistapaa. Montoto et al. [10] nimeää nämä lähestymistavat *implisiittiseksi* ja *eksplisiittiseksi* nauhoitukseksi (taulukko 1). Implisiittisessa nauhoituksessa käyttäjää kuunnellaan dokumentin sisältä käsin. Käytännössä tämän voi toteuttaa rekisteröimällä tapahtumia kuuntelevia proxy-funktioita, jotka tallentavat tapahtumia ennen oikean tapahtumakäsittelijän kutsumista. Eksplisiittisessä nauhoituksessa kuunnellaan käyttäjän suorittamia syötteitä, jotka tallennetaan ennen syötteen suorittamista (ja tapahtumien luomista).

Taulukko 1. Nauhoituksen lähestymistapojen keskeiset erot. [10]

	Käyttäjäsyötteiden kuuntelumenetelmä	
	Implisiittinen	**Eksplisiittinen**
Mitä kuunnellaan?	Tapahtumia (events), jotka kulkevat dokumenttipuussa.	Käyttäjäsyötteitä, joita käyttäjä suorittaa (esim. klikataan hiirellä ruudun kohtaa X) .
Mitä nauhoitetaan?	Tapahtumatyyppi ja tapahtuman kohde-elementti.	Käyttäjäsyöte ja syötteen kohde-elementti.

Työn ratkaisutapaa valittaessa nähdään, että implisiittisen nauhoituksen suurin ongelma on tapahtumien valtava määrä. Jos kaikkia tapahtumia kuunnellaan dokumentin juurisolmussa (luonnollinen valinta, koska kaikki tapahtumat kulkevat tämän pisteen läpi), niin ylivoimaisesti suurin osa siepatuista tapahtumista on turhia, koska esim. hiiren liike dokumentin yli luo massiivisen määrän tapahtumia. Tässä tapauksessa jokainen hiiren osoittimen alle joutunut solmu saa oman tapahtumajoukon: ”osoitin saapui,” ”osoitin liikkui” ja ”osoitin poistui.” Toisaalta jos taas tallennettavat tapahtumat rajoitetaan vain tiettyihin solmu- ja tapahtumatyyppeihin, niin ongelmana on ratkaisun rajoittuneisuus, koska mitä tahansa tapahtumatyyppiä kuunteleva funktio voi sijaita missä tahansa solmussa. Lisäksi implisiittisessä nauhoituksessa ongelmaksi muodostuu myös dokumentin sisäisten muutosten aiheuttamat haasteet tapahtumien kohdistamisessa, koska nauhoitus tapahtuu dokumentin sisältä käsin samalla kun dokumentti on muuttuvassa tilassa ja koska tapahtumia kuuntelevat funktiot ovat alttiina muutoksille.

Työn ratkaisussa käytetään eksplisiittistä nauhoitustapaa, koska se tarjoaa implisiittistä tapaa täydellisemmän ja merkittävästi laajennuskelpoisemman ratkaisun. Nauhoitettavaa dataa on vähemmän ja data kertoo tarkasti mitä käyttäjä teki: järjestelmä on vapaa vastaamaan tähän oman laajennuskelpoisen algoritminsa mukaisesti. Eksplisiittinen ratkaisu toteutetaan omaksi kerrokseksi, joka lisätään dokumenttiin ilman alkuperäisen dokumentin sisällön tai rakenteen muokkausta. Muokkausta vaativat ratkaisut oletaan selkeästi suuremmaksi ylläpitohaasteeksi eli virhealttiimmaksi lopputulokseksi. Haasteena tässä ratkaisussa on kuitenkin käyttäjäinteraktion sulava toteutus, sillä dokumentin tapahtumien luonti ja suoritus on pysäytettävä käyttäjäsyötteiden sieppauksen ja tallennuksen ajaksi. Dokumentin tilan pysäyttämisellä voidaan kuitenkin tehokkaasti estää käyttäjää suorittamasta uutta syötettä kun edelliseen reagointi on vielä kesken, ja tämän ansiosta valvoa dokumentin tilaa sekä nauhoituksen onnistumista.

4.2.2 Käyttäjäsyötteen kohteen tunnistustavan valinta

Käyttäjäsyötteen kohteen tunnistaminen on tärkeä osa onnistunutta nauhoitusta. Syötteen kohteena olevalla solmulla on aina nauhoitushetkeen sidottu yksilöllinen osoite, mutta nauhoite täytyy voida toistaa myös myöhemmällä hetkellä. Jotta web-makrot olisivat uudelleenkäytettäviä ja muutoksenkestäviä, niin nauhoitusten tulee säilyä toistokelpoisena

dokumentin muutoksista huolimatta. Tämä ei tietenkään ole mahdollista jos syötteen kohteena ollut solmu katoaa dokumentista, mutta yleensä muutokset ovat pikemminkin ulkonäkö- tai sisältömuutoksia. Tällaisissa tapauksissa nauhoituksen alkuperäinen kohde on vielä tallella, vaikka ehkä eri paikassa ruudulla, mutta dokumentin puurakenne tai sisältö on muuttunut. Tämän työn kohdejärjestelmässä muutokset johtuvat web-sovelluksen ylläpidosta (esim. päivitykset, laajennukset ja virhekorjaukset) ja käyttäjätoimista (esim. laskujen syöttö järjestelmään, jolloin ruudulle listataan enemmän laskuja).

Työn ratkaisussa käyttäjäsyötteiden kohde tunnistetaan ja osoite tallennetaan käyttämällä XPath-lausekkeita. Ideana on rakentaa lauseke mahdollisimman epätarkasti ja väljästi, jotta muutoksien vaikutus minimoituu. XPath-lausekkeiden käytön vahvuudet ovat täydellinen selaintuki sekä lausekerakentamisen vapaus ja yksinkertaisuus. Lausekkeet voivat sisältää mielivaltaisen yhdistelmän puurakenteen relatiivista ja absoluuttista osoitetietoa sekä solmujen tyyppi- ja sisältötietoa (esim. elementin nimi, sisältö ja attribuutit sekä arvot). Lisäksi lauseketta on helppo käsitellä tallennus- ja toistovaiheessa, koska se on kaiken aikaa valmiiksi käytettävässä muodossa: aina on mahdollista kohdistaa lauseke dokumenttiin ja tarkastaa tulos. Jotta nauhoitus ja toisto onnistuvat, on tuloksena oltava aina yhden ja vain yhden lauseketta vastaavan elementin löytyminen.

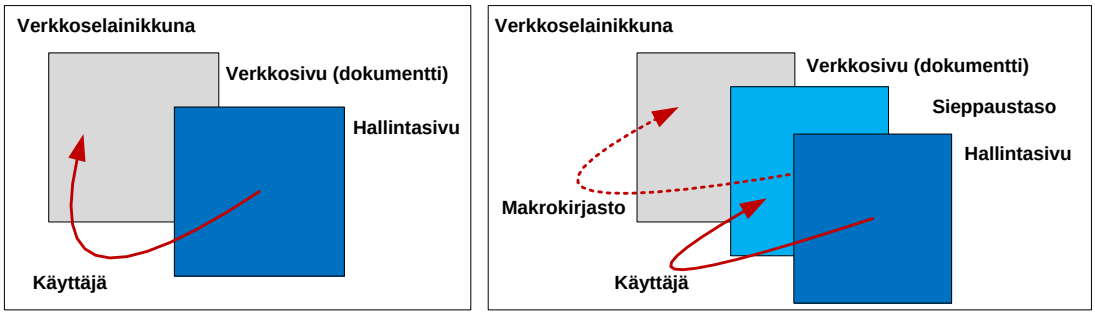
5 RATKAISUN ESITTELY

Tässä luvussa käydään läpi edellisen luvun suunnittelun pohjalta syntynyt työn ratkaisu eli web-makrojärjestelmän prototyyppitoteutus (makrokirjasto). Toteutus käydään läpi pääpiirteineen ongelma-ratkaisu -henkisen lähestymistavan avulla. Luvun päätarkoituksena on antaa käsitys makrokirjaston rakenteesta sekä esittää työssä havaitut ongelmat ja valitut ratkaisut. Makrokirjaston yksityiskohtia arvioidaan työn pohdintaosuudessa.

Ratkaisun keskeisin osa on makrokirjasto eli verkkosivuun liitettävä JavaScript-koodikirjasto, johon on koottu kaikki makronauhoituksen ja -toistamisen toimintalogiikka. Lisäksi ratkaisuun kuuluu myös erillinen makrokirjaston hallintanäkymä, joka on kehitystä ja testausta varten luotu web-sovellusliitännät korvaava surrogaatti. Ratkaisun keskeiset teknologiat ovat JavaScript-ohjelmointikieli, DOM-standardi, XPath-kieli ja XHTML-dokumenttimääritys. Vaikka ratkaisu pohjautuu standardimäärityksiin, niin tavoitteena oli kehittää Firefox-selaimella toimiva ratkaisu eikä käyttää aikaa laajemman selainyhteensopivuuden varmistamiseen. Ratkaisu toimii testatusti selaimilla Mozilla Firefox (versio 4) ja Google Chrome (versio 11). Makrokirjasto sisältää 850 riviä JavaScript-koodia, ja hyödyntää vain verkkoselaimen JavaScript-vakiorajapintaa.

5.1 Ratkaisun sisällön ja arkkitehtuurin esittely

Työn ratkaisu osaa makronauhoitteiden tallentamisen ja tallenteiden toistamisen. Kaikki toiminnallisuus on toteutettu makrokirjastossa, mutta hallintanäkymää tarvitaan nauhoituksen ja toiston aloitukseen, ohjaukseen sekä lopetukseen. Ratkaisu toteuttaa web-makron nauhoituksen eksplisiittisesti käyttäjäsyötteitä kuunnellen ja tunnistaa käyttäjäsyötteen kohteen XPath-lausekkeiden avulla. Nämä kaksi valintaa on selitetty ja perusteltu luvuissa 4.2.1 ja 4.2.2. Eksplisiittisen nauhoituksen logiikan mukaisesti käyttäjä käyttää makrokirjastoa ja makrokirjasto käyttää varsinaista verkkosivua. Makrokirjaston toiminta perustuu kolmen eri tason käyttöön samassa selainikkunassa. Ensimmäinen ja päällimmäinen näistä on hallintanäkymä ja kaikista alimpana sijaitsee varsinainen verkkosivu. Kun makron nauhoitus tai toisto on päällä, lisätään näiden tasojen väliin vielä sieppaustaso, jota käytetään web-makrot mahdollistavana rajapintana käyttäjän ja verkkosivun välillä, kuten esitetty kuvassa 4.



Kuva 4. Sivun käyttö makrot pois päältä (vasen) ja makrot päällä (oikea).

Hallintanäkymä on verkkosivu, joka sisältää kohdesivun (dokumentin URL-osoite (Uniform Resoure Locator)) valinnan ja makrokirjaston ohjaustoiminnot (”aloita tallennus”, ”aloita toistaminen” ja ”lopeta”). Käytännössä hallintanäkymä sisältää edellämainitut toiminnot ohuena palkkina ja muuten se näyttää verkkosivun isossa iframe-elementissä. Makrokirjastoa ei ole liitetty hallintasivuun, mutta se vaaditaan liitettynä kaikille sivuille, joita makrojen avulla käytetään. Lisäksi hallintasivu sisältää myös JavaScript-koodia verkkosivun makrokirjaston ja hallintasivun väliseen tiedonsiirtoon. Hallintanäkymä toimii makro-ohjauksen lisäksi myös tallenteiden väliaikaisena varastointipaikkana, koska tiedonsiirtoa ja pysyvää tallennusta web-sovellusympäristössä ei vielä tueta. Myös uudelleenohjauspyynnöt eli sivunvaihdot koordinoidaan hallintanäkymässä, koska uuden verkkosivun makrokirjastolle täytyy kertoa, että nauhoitus tai toisto on käynnissä.

Makrokirjaston toiminta ja nauhoituksen eteneminen voidaan esittää algoritmimuodossa, jonka ensimmäinen askel tapahtuu verkkosivua avattaessa. Algoritmi kuvaa järjestelmän toimintalogiikan sekä tärkeimpien välietappien sisällön ja keskenäisen aikajärjestyksen. Algoritmin kuvauksessa termeillä *verkkosivu* ja *dokumentti* viitataan makrotoiminnallisuuden kanssa käytettävään web-sovelluksen kohdesivuun. Seuraavaksi esitetään tämä algoritmi:

1. **Odota verkkosivun (dokumentin) latauksen valmistumista.**
   Uuden sivulatauksen (ja myös sivunvaihdon) yhteydessä on sivun annettava latautua valmiiksi ennen kuin makron nauhoitus tai toisto voidaan suorittaa. Jos sivulla on JavaScript-koodia, niin ladattavan dokumentin rakenne ja sisältö saattavat muuttua latauksen yhteydessä. Muutokset johtuvat synkronisen koodin, tapahtumien ja

asynkronisen koodin sisällöstä ja seurauksista. Sivulatauksen tila määritetään rekisteröimällä asynkroninen funktio, joka kuuntelee dokumentin readyState-muuttujaa ja DOMContentLoaded-tapahtumaa. Kun dokumentti todetaan valmiiksi, on makrokirjaston toimintojen suoritus mahdollista. Tämä yksinkertainen ratkaisu toimii Mozillan, Operan ja Webkit-perheen selaimessa, Internet Explorer vaatii oman monimutkaisemman käsittelyn [22].

2. Makrokirjasto on käytettävissä, odota nauhoituksen tai toiston aloitusta.

Vain makrokirjaston hallintasivun toimintopalkki näkyy käyttäjälle, web-sovellusta voi käyttää täysin normaalisti.

3. Eristä web-sovelluksen verkkosivu käyttäjästä luomalla sieppaustaso.

Web-sovelluksen verkkosivun suora käyttö estetään sieppaustason avulla, joka eristää käyttäjän sivun dokumenttisisällöstä. Nauhoituksen aikana sieppaustaso toimii käyttäjäsyötteet kaappaavana rajapintana. Eristystä tarvitaan, koska syötteen tyyppi ja kohde täytyy tallentaa ennen kuin syöte on kohdistettu (tapahtuma on luotu ja suoritettu) dokumentille. Sieppaustaso luodaan lisäämällä dokumenttiin uusi div-elementti, joka asetetaan koko dokumentti-ikkunan kokoiseksi ja nostetaan varsinaisen dokumentin päälle z-index -arvon avulla. Tähän div-elementtiin lisätään sen jälkeen iframe-elementti, jonka sisälle luodaan uusi dokumentti syötteiden sieppausta varten. Nämä lisätyt elementit asetetaan käyttäjälle näkymättömiksi, joten verkkosivun ulkoasu ei muutu. Iframe-elementin sisälle luotuun dokumenttiin rekisteröidään tapahtumakäsittelijät, jotka sieppaavat käyttäjäsyötteet ja kutsuvat makrokirjaston toimintoja. Iframe-elementin alueelle kohdistuvat syötteet aiheuttavat lokaaleja ja vain elementin sisäisen dokumentin laajuisia tapahtumia.

4. Kuuntele sieppaustason käyttäjäsyötteitä.

Käyttäjäsyöte havaitaan sieppaustason iframe-elementin tapahtumakäsittelijän avulla. Tapahtumakäsittelijälle saapuva tapahtumaolio sisältää mm. hiiren osoittimen paikan xy-koordinaatistossa.

5. **Määritä syötetoiminnon tyyppi ja syötteen kohde eli dokumentin elementti.**

   Syötetoiminnon tyyppi valitaan menu-valikon kautta, joka aukeaa käyttäjäsyötteen kohteen päälle eli hiiren osoittimen luokse. Valikon toiminnot on listattu luvussa 5.2. Käyttäjäsyötteen kohdesolmu määritetään siirtämällä sieppaustaso hetkellisesti sivuun ja hakemalla määritys käyttäjäsyötteen xy-koordinaattien kautta dokumentin elementFromPoint-funktiosta. Tämän jälkeen määritetään kohdesolmun tallennettava osoite, minkä yksityiskohdat on määritetty luvussa 5.3.

6. **Luo syötetyypin mukainen joukko tapahtumia ja lähetä tapahtumat syötteen kohdesolmuun.**

   Valitun toiminnon tyypin ja kohteen perusteella voidaan toiminto kohdistaa web-sovelluksen verkkosivuun. Syötteen tyypin perusteella luodaan joukko tapahtumia, jotka laukaistaan dokumenttipuuhun kohteenaan syötteen kohdesolmu. Tapahtumajoukon sisältö on määritetty luvussa 5.2. Käyttäjää imitoivaan lopputulokseen päästään tapahtumajoukon sisällön ja keskenäisen järjestyksen oikealla yhdistelmällä.

7. **Odota kunnes tapahtumat ja tapahtumien seuraukset on käsitelty ja verkkosivu (dokumentti) on taas muuttumattomassa tilassa.**

   Toiminnon suorituksen seurauksena syntyneiden tapahtumien käsittely odotetaan valmiiksi ennen seuraavan käyttäjäsyötteen sieppausta. Jokainen tapahtumaolio kulkee läpi dokumenttipuun yksi kerrallaan, ja kaikki käsittelyyn liittyvä synkroninen koodi suoritetaan ennen seuraavan tapahtumaolion lähetystä. Asynkroninen koodi vaatii kuitenkin erikoiskäsittelyä, joka on määritetty luvussa 5.4. Kun kaikki tapahtumat on käsitelty ja asynkroniset funktiot on suoritettu, voidaan siirtyä seuraavan käyttäjäsyötteen nauhoitukseen tai toistoon.

Algoritmissa määritetty ratkaisu ei millään tapaa rajoita sivupyyntöjen, linkkien tai lomakesyötön suorittamista. Sivunvaihtoja kuunnellaan dokumentin unload-tapahtumatyypille asetetussa tapahtumakuuntelijassa, jota selain kutsuu dokumentilta poistuttaessa. Kun sivunvaihto kohdataan, niin uuden sivun makrokirjasto asetetaan jatkamaan jo alkanutta toimintaa, mutta vasta kun dokumentin avaus on määritetty valmiiksi algoritmin kohdan 1 mukaisesti.

Ratkaisu sisältää makronauhoitteiden toiston ilman graafisia ominaisuuksia. Käyttäjä voi aloittaa toiston hallintasivun kautta, mutta toisto näkyy vain toimintojen seurauksena tapahtuvana dokumentin tilan ja ulkoasun muutoksena. Prototyyppitoteutus ei sisällä havainnollistavia lisäefektejä tai esimerkiksi hiiren osoitinta jäljittelevän kuvan liikkumista. Toimintalogiikaltaan makron toistaminen ei juuri eroa edellä kuvatusta nauhoitusalgoritmista. Toistamisen aikana käyttäjäsyötteiden vastaanotto ohitetaan, joten sieppaustasoa käytetään vain eristämään käyttäjä ja estämään tahaton sekaantuminen toiston lopputulokseen. Myös algoritmin kohta 4 ohitetaan siirtymällä suoraan syötetoiminnon tyyppi- ja kohdekäsittelyyn. Kohdesolmun XPath-osoite ja syötetoiminto on aiemmin tallennettu makrotallenteeseen, ja nyt tätä tietoa käytetään kohdesolmun määritykseen ja tapahtumajoukon luomiseen. Tapahtumien määritys ja tapahtumakäsittely toimii samoin kuin nauhoituksen aikana.

Ratkaisun kehitystavasta voidaan mainita, että vaikka ratkaisu kehitettiin iteratiivisesti, niin kehityksellä oli aina selkeä painopiste. Kehitys lähti liikkeelle alustavan osoitetunnistusalgoritmin määrityksestä, koska sen katsottiin olevan muista osista erillinen moduuli. Tämän jälkeen toteutettiin käyttäjäsyötteiden sieppauskerros, koska muuten ratkaisu ja erityisesti ratkaisun testaus ei voinut edetä pidemmälle. Tämän jälkeen luotiin alustava käyttöliittymä. Lopulta siirryttiin määrittämään käyttäjäsyötetoimintoja sekä näihin liittyviä tapahtumajoukkoja. Tässä vaiheessa valmistui alustava makrokirjastoversio makrojen nauhoitukseen ja toistamiseen. Viimeisenä käytiin ratkaisemaan asynkronisen koodin haasteita, sivunlatauksen valmistumisen määritystä ja sivusiirtymien toteutusta.

5.2 Tuetut käyttäjäsyötteet ja DOM-tapahtumamalli

Makrokirjasto toteuttaa kolme päätoimintoa, joita käyttäjä tarvitsee web-sovelluksen käyttämiseen. Nämä toiminnot määrittävät nauhoituksen aikana valittavissa olevat käyttäjäsyötteet. Nauhoituksen aikana nämä käyttäjäsyöttetyypit tallennetaan (yhdessä kohdesolmun osoitteen kanssa) ja toiston aikana ne suoritetaan. Siepatun syötteen suoritus aloitetaan kohdesolmun määrityksellä, jonka jälkeen luodaan syötetyypin mukaisesti määritetty tapahtumaolioista koostuva tapahtumajono. Nämä tapahtumat alustetaan dokumentin kontekstissa ja sen jälkeen lähetetään oikeassa järjestyksessä yksi kerrallaan dokumenttipuun läpi kohti kohdesolmua. Toiminnot ja niihin liittyvät tapahtumakartat on

listattu taulukossa 2. Tapahtumat lähetetään kutsumalla kohdesolmun dispatchEvent-funktiota, joka palauttaa arvon vasta kun tapahtumakäsittely on valmis. Kun lähetys tapahtuu iteraatiossa, voidaan olla varma, että edelliseen tapahtumaan liittyvä synkroninen koodi on suoritettu ennen seuraavan tapahtuman lähetystä. Tapahtumakartta ja tapahtumien keskenäinen järjestys on DOM-standardin mukainen [32].

Taulukko 2. Makrokirjaston toiminnot ja käyttäjäsyötteet.

	Ohjelman toiminto		
	Klikkaus	**Tekstinsyöttö**	**Osoittimen siirto**
Käyttäjäsyöte	Hiirennapin painallus tietyssä paikassa.	Kentän valinta ja tekstinsyöttö valittuun kenttään.	Hiiren osoittimen siirto tiettyyn paikkaan.
Käyttötarkoitus	Nappien, linkkien ja muiden toimintojen käyttö.	Tekstinsyöttö verkkosivun tekstikenttiin.	Saada esiin piilotettuja toimintoja, jotka odottavat hiiren osoittimen saapumista.
Tapahtumakartta ja keskenäinen järjestys	1. mouseover 2. mousemove 3. mousedown 4. focus 5. mouseup 6. click 7. mouseout 8. blur	1. mouseover 2. mousemove 3. mousedown 4. focus 5. mouseup 6. click 7. keydown 8. keypress 9. keyup 10. change 11. mouseout 12. blur	1. mouseover 2. mousemove

5.3 Algoritmi käyttäjäsyötteen kohteen määritykseen

Ratkaisun tunnistusalgoritmi perustuu XPath-lausekkeisiin. Tallennusvaiheessa algoritmi rakentaa mahdollisimman epätarkan, mutta silti kohde-elementin uniikisti määrittävän lausekkeen. Algoritmi iteroi puun solmut ja näiden solmujen attribuutit kohdesolmusta juurisolmuun. Jokaisen solmun kohdalla solmun attribuutit iteroidaan läpi ja lisätään (yksi kerrallaan) kumulatiivisesti muodostuvaan XPath-lausekkeeseen (osoite), jonka yksilöllisyys tarkastetaan jokaisen lisäyksen jälkeen. Kun yksilöllinen lauseke löytyy, iteraatiot loppuvat ja syntynyt lauseke palautetaan käyttäjäsyötteen osoitteeksi. Tunnistuksessa ei voi käyttää ruudulla näkyviä tekstejä, koska sovelluksen käyttöliittymä tukee kolmea kielivaihtoehtoa ja koska termimuutokset ovat mahdollisia.

Ideaalitapauksessa osoite voidaan määrittää ilman solmun kantaisien iteraatiota, mutta mikäli näin ei ole, iteraatio jatkuu kunnes yksilöllinen osoite löytyy. Jos uniikkia osoitetta ei löydy, niin viimeisenä keinona kohteelle määritetään relatiivinen osoite, esim. ”ensimmäisen table-solmun kolmas tr-solmu.” Osoitemääritys on esitetty algoritmina kuvassa 5. Algoritmin idea on johdettu Montoto et al. [10] artikkelista, mutta tässä työssä lisätään tärkeänä yksityiskohtana attribuuttipainotus, jota kyseinen artikkeli ei mainitse.

```
// Iteroidaan dokumenttipuun solmuja kohdesolmusta puun juurta kohden kunnes
// määritetään uniikki osoite.

solmu = kohdesolmu;
lauseke = null;

while (solmu ei ole puun juurisolmu) {

   lauseke = lauseke + solmun tyyppi; // = solmun nimi / html-tagin nimi

   if (solmulla on attribuutteja) {
       attribuutit = solmun attribuutit jonossa painokerroinjärjestyksessä;

       foreach (attribuutti in attribuutit) {
           lauseke = lauseke + attribuutti;
           if (lauseke on uniikki osoite) return lauseke;
        }
    }
   solmu = solmu.isä; // = puurakenteessa solmun kantaisä
}

return (kohdesolmun relatiivinen osoite puurakenteessa);
```

Kuva 5. Solmun XPath-lausekkeen muodostava algoritmi pseudokielellä.

Lausekemuodostuksen tärkein yksityiskohta on *attribuuttien painotus*. Kaikki solmun attribuutit eivät ole osoitemuodostuksessa samanarvoisia. Sen takia attribuutit ajetaan suodattimen läpi, joka asettaa ne painokerroinjärjestykseen. Näin painotetaan osoitteen kannalta tärkeimpien attribuuttien arvoja, jotka valitaan ensimmäisenä, ja samalla hylätään ne arvot, joiden käyttö ei edistä muutoksia kestävän lopputuloksen syntymistä. Attribuutit on luokiteltu tämän työn sovelluskontekstin mukaiseen tärkeysjärjestykseen taulukossa 3.

Ratkaisun tärkeimpinä attribuutteina pidetään id- ja name-attribuuttia, koska niiden arvoja käytetään jo valmiiksi osoitteina. Ne ovat läheisesti kytköksissä sovelluksen taustalogiikkaan, näkymämuodostukseen ja JavaScript-koodin, joten arvon vaihtuminen on erittäin epätodennäköistä. Myös solmuihin liitetyt tapahtumakäsittelijät ovat erittäin luotettavia attribuutteja, koska niiden arvot viittaavaat lähes aina JavaScript-funktioihin.

Value-attribuutti on tärkeä vain silloin kun se kuvaa toimintoa tai tietokannan tietuetta (eli valinnan kohdistusta). Näitä tapauksia ovat mm. select- ja input-elementti, mutta input-elementti vain kun sen type-attribuutin arvo ei ole button tai submit (koska silloin arvossa sijaitsee käyttöliittymän tekstiä). Lisäksi myös URL-osoitetta ja syötesolmun tyyppiä voidaan pitää luotettavana, koska ne sisältävät tyyppitietoa kohteen ja toiminnon luonteesta (esim. kohdeverkkosivu tai syötetyyppi). Merkittävää kaikkien edellisten attribuuttien osalta on myös niiden sisällön luonne: jos arvo on tallennettu makronauhoitteen elementin osoitteeseen, mutta toiston aikana arvoa ei löydy, niin nauhoitettua toimintoa ei todennäköisesti enää ole olemassa. Kyseessä on silloin kehitystyön tuloksena syntynyt sovellusmuutos. Vähäpätöisimpiä attribuutteja ovat CSS-luokat ja -tyyliasetukset. Ne määrittävät ulkoasua ja ovat erittäin muutosherkkiä, joten niiden käyttöä osoitteenmuodostuksessa tulisi välttää. Tässä ratkaisussa CSS-luokat hyväksytään, mutta CSS-tyyliasetukset hylätään.

Taulukko 3. Attribuuttien painotusjärjestys ja käyttötapa sovelluskontekstissa.

Järjestys	**Nimi**	**Arvon kuvaus**	**Käyttötapa kohdesovelluksessa**
1	id	Yksilöllinen tunnus.	Määrityksen ja käyttötavan mukaan oletettavasti yksilöllinen tunnus.
2	name	Solmuun viittaava nimi.	Usein yksilöllinen.
3	on*	Tapahtumakäsittelijä. (esim. onclick, onmouseover, yms.)	Kaikki ”on” -alkuiset attribuutit pitävät sisällään tapahtumakäsittelijöitä eli JavaScript-koodia ja -funktioita. Funktion nimi määrittää toiminnon luonteen hyvin yksiselitteisesti.
4	href	Hyperlinkkiosoite.	Linkin sisällöt ovat usein dokumentissa yksilöllisiä ja määrittävät tarkasti toiminnon luonteen.
5	value	Solmun valinnan arvo.	Tiettyjen elementtien yhteydessä liittyy oletettavasti toiminto- tai tietuetunnisteeseen.
6	type	Syötesolmujen tyyppi.	Syötesolmun tyyppi määrittää toiminnon luonteen (esim. nappi tai syötekenttä).
7	muut	Kaikki muut attribuutit, paitsi alla olevat.	Usein ulkoasumäärityksiä. Ulkoasu määritetään kuitenkin ensisijaisesti CSS-asetusten avulla.
8	class	CSS-luokat	Ulkoasua määrittäviä luokkia. Herkkiä muuttumaan.
hylätään	style	CSS-tyyliasetukset	Ulkoasua määrittäviä asetuksia. Erittäin herkkiä muuttumaan.

5.4 Asynkronisen funktion erikoiskäsittely

Tapahtumien seurauksena dokumenttiin kohdistuu asynkronisia toimintoja: ajastimeen sidottua funktiosuoritusta ja Ajax-tiedonsiirtoa. Tapahtumakäsittelyn valmistumishetken määrittämiseksi makrokirjasto valvoo suoritusta odottavia asynkronisia funktioita. Valvonnan avulla makrokirjasto tietää jokaisella hetkellä suoritusta odottavien funktioiden määrän ja pystyy keskeyttämään funktion suorituksen, jos suorituksessa kestää liian kauan. Valvonnan hallinta täytyy toteuttaa asynkronisesti, jotta myös muille funktioille tarjoutuu suoritusvuoro.

Asynkronisten funktioiden valvonta toteutetaan kolmella toimenpiteellä. Kaksi ensimmäistä ovat setTimeout- ja setInterval-funktion korvaus makrokirjaston omilla proxy-funktioilla. Nämä kaksi JavaScript-kielen funktiota on määritetty selainikkunaa mallintavassa window-oliossa, joten niiden korvauksen seurauksena kaikki selainikkunan sisällä tapahtuvat funktiokutsut menevät nyt makrokirjaston omien proxy-funktioiden läpi. Näistä kahdesta proxy-funktiosta merkittävästi yksinkertaisempi on esitetty kuvassa 6. Kolmas ja viimeinen tapa asynkronisen funktiokutsun toteuttamiseen on window-oliossa määritetty XMLHttpRequest-luokka, joka mallintaa ja toteuttaa asynkronisen tiedonsiirron toiminnan. Luokan olio lähettää viestin send-funktiokutsun aikana ja jää asynkroniseen tilaan odottamaan vastausta. Vastauksen saapumisen valvonta tapahtuu korvaamalla luokan send-funktion prototyyppi makrokirjaston omalla vastineella, joka implementoi alkuperäisen kutsun ja valvontatoiminnallisuuden. Vastauksen saapuminen voidaan tarkistaa XMLHttpRequest-luokan readyState-muuttujasta.

```
// Asynkronisen funktion valvonta, tapaus setTimeout-funktio

window.setTimeout_real = window.setTimeout; // alkuperäinen säilötään
window.setTimeout = function (fn, timeout) { // oma toteutus

    var proxy = function () {
        fn(); // funktiokutsu
        monitor.removeTimer(ret); // asynkroninen funktio on suoritettu
    }
    var ret = window.setTimeout_real(proxy, timeout); // alkuperäinen ja oikea
    monitor.addTimer(ret); // asynkroninen funktio lisätään valvottavaksi
    return ret;
}
```

Kuva 6. Asynkronisen funktion valvonta: korvaus proxy-funktiolla.

6 POHDINTA JA TYÖN TULOKSET

Työn pohdinta keskittyy arvioimaan prototyyppitoteutuksen lopputulosta. Prototyyppiä kehitettiin iteratiivisesti kunnes ratkaisu tarjosi tarpeeksi laajan kokonaiskuvan ongelmaympäristöstä. Ratkaisun testaaminen kohdesovellusympäristössä ei kuitenkaan mahtunut työn mittakaavaan, joten työn konkreettisena tuloksena on web-sovelluksesta irrallinen prototyyppi, mikä osaltaan rajoittaa johtopäätösten painoarvoa. Toisena merkittävänä rajoitteena täytyy huomioida, että tämä työ käsittelee vain dokumenttipohjaisten käyttäjänäkymien automatisointia. Web-sovellukset voivat vastata käyttäjäsyötteisiin myös muilla tavoin, joista esimerkkinä mm. Flash- ja Java Applet-pohjaiset teknologiat.

Työn havainnot syntyivät makrokirjaston kehityksen ja testauksen perusteella. Makrokirjastoa testattiin testikäyttöön rakennetun sivuston avulla, jossa oli mm. tekstisyötekenttiä, interaktiivisia toimintoja, asynkronista ja synkronista JavaScript-koodia, Ajax-tiedonsiirtoa, tapahtumakuuntelijoita sekä sivusiirtymiä. Toisin sanoen, testisivustolle pyrittiin keskittämään kaikki prototyypin kohdesovelluksesta löytyvät tekniikat, ominaisuudet ja käytännöt. Testauksen ansiosta työn prototyyppipohjainen lähestymistapa voidaan todeta onnistuneeksi, koska ilman prototyyppiä työssä ei olisi päädytty samoihin lopputuloksiin. Vaikka lähes kaikki haasteet voitiin tunnistaa jo työn alussa, niin haasteiden vakavuuden keskenäinen painotus olisi ollut täysin väärä ilman testausta. Esimerkiksi asynkronisia funktioita pidettiin työn alussa suurena haasteena, mikä kuitenkin osoittautui vääräksi oletukseksi.

6.1 Ratkaisun arviointi

Työn ratkaisua arvioidaan karkeasti luvun 5 noudattamaa ositusta ja esitysjärjestystä mukaillen. Aluksi arvioidaan makrokirjaston suoria toimintoja ja lopuksi tuodaan esille ratkaisuun välillisesti kytkeytyvät ongelmat. Lopputuloksena huomataan, että makrokirjasto sisältää hyvin vähän teknisiä haasteita ja että työn lopputulos toteuttaa makrot onnistuneesti mutta hieman keskeneräisesti. Suurimmat havaitut haasteet ja varsinaiset ongelmat liittyvät makroparadigman laajamittaiseen hyödyntämiseen sekä tämän prototyypin puutteelliseen selainyhteensopivuuteen.

Työn lähestymistapa käyttäjäsyötteiden sieppaukseen (eksplisiittinen sieppaustaso) on ratkaisuna toimiva ja erittäin laajennuskelpoinen. Prototyyppi tukee kolmea eri toimintotyyppiä, ja prototyypin käyttö vaatii toiminnon valinnan, mutta nämä seikat eivät perustu teknisiin rajoitteisiin. Ratkaisua voidaan kehittää tunnistamaan kohde-elementti ja automatisoimaan toiminnon valinta kohteen tyypin avulla. On mahdollista, että käyttäjäsyöte tunnistetaan tekstikentän yllä kirjoitusaikeeksi ja napin päällä pelkäksi napin klikkaukseksi. Kaikki mahdollisuudet makrokirjaston täysin saumattomaan ja sulavaan nauhoituskäyttöliittymään ovat olemassa. Makrokirjaston kolmen toiminnon avulla kohdesovellusta voi kuitenkin jo nyt käyttää lähes täysimittaisesti. Ainoa puuttuva toiminto on elementtien raahaus (”drag & drop”), joka on toteutettavissa.

Makrokirjasto toteuttaa makrotallenteen toiston onnistuneesti, mutta ei vielä riittävän sulavasti ja käyttäjäystävällisesti. Tallenteiden toistamisen heikkouksena prototyypissä on ratkaisun keskeneräisyys. Toistaminen toimii, mutta se täytyy siirtää graafiseksi, jotta makron kulku voidaan esittää informatiivisesti. Ilman suoritusta korostavia graafisia elementtejä makroista ei ole hyötyä esim. opetus- tai esitystarkoituksessa. Toinen tallennuksen puute on säilytys ja hallinta, johon täytyy vielä kehittää tästä versiosta puuttuva pysyvä ratkaisu. Lisäksi tallentaminen perii haasteita makrokirjaston muilta osa-alueilta, koska tallenteiden laatu on täysin riippuvainen käyttäjäsyötteiden sieppauksen tarkkuudesta ja tunnistusalgoritmin toimivuudesta. Työn aikana ei ole kuitenkaan ilmennyt mitään teknisiä rajoitteita tallennuksen ja toiston jatkokehitykselle edellä mainittujen haasteiden selvittämiseksi.

Makrokirjaston osoitteenmuodostusalgoritmi on testattu toimivaksi ja se oletetaan riittäväksi ratkaisuksi tässä sovelluskontekstissa. Jos heikkouksia on, niin ne epäilemättä saadaan selville varsinaisen laajamittaisen testauksen aikana. XPath-lausekkeiden käyttö todetaan hyväksi valinnaksi, koska standardin valinta ei rajoittanut ratkaisun toteutustapaa. Ratkaisussa on mahdollista käyttää myös muita vastaavia standardeja, mutta ne vaativat tällä hetkellä vielä ulkoisen JavaScript-kirjaston, sillä suoraa selaintukea ei ole. Muita vaihtoehtoja ei ole kuitenkaan tarpeen pohtia ennen kuin ne tarjoavat XPath-lausekkeita parempia ominaisuuksia. On kuitenkin painotettava, että *käyttäjäsyötteiden osoitteenmuodostusalgoritmi saavuttaa hyviä tuloksia vain attribuuttien painotuksen ansiosta.* Monet attribuutit ovat osoitteenmuodostuksessa käyttökelvottomia. Esimerkiksi

jos taulukon kaikilla riveillä on samat tyyliasetukset, mutta taulukossa vain yksi rivi, niin taulukon ensimmäisen rivin voisi yksilöidä tyyliasetusten perusteella. Tällainen osoite on kuitenkin käyttökelvoton kun toinen rivi lisätään. Tämän työn kuluessa huomattiin, että kohdeympäristön kehityskäytäntöjen tuntemus on tärkeä osa muutoksia kestävän osoitteen muodostusta. Työn kohdesovellus ei aseta sessionkohtaisesti sidottuja attribuutteja, mutta tämä on mahdollista jossain toisessa web-sovelluksessa. Algoritmin idea on selvästi yleistettävissä ja laajasti hyödynnettävissä, mutta yksityiskohdat ovat sovelluskohtaisia.

Makrokirjasto ratkaisee asynkronisten funktioiden haasteet onnistuneesti. Funktioiden valvonta ja seuranta toimii käyttäjälle ja web-sovellukselle näkymättömällä tavalla. Ainoat jäljelle jäävät ongelmat ovat aikaikkunoiden määritys ja virhetilanteisiin reagointi: kuinka kauan asynkronisen funktion suoritusta voi odottaa, ja miten pitää reagoida funktioon, joka ei saavuta odotettua lopputulosta? Ratkaisut näihin ongelmiin ovat kuitenkin makrokirjaston toiminnallisia määrityksiä, joihin ei ole olemassa teknisesti oikeaa vastausta. Lisäksi voidaan mainita, että asynkroniseen funktiovalvontaan sovellettu proxy-ratkaisu on yleistettävissä myös pop-up -ikkunoiden hallintaan (JavaScript-kielen prompt-, confirm- ja alert-funktio). Nauhoituksen aikana tallennettu pop-up -ikkunan vastaus täytyy esittää toiston aikana ilman varsinaisen funktion kutsua, jotta ohitetaan funktion vaatima käyttäjävastaus.

Ratkaisun suurin haaste on makroparadigman ja web-sovelluksen liiketoimintalogiikan yhteentörmäys. Sovelluksen liiketoimintalogiikka voi tehdä makrojen hyödyntämisen mahdottomaksi. Sovelluksen näkymissä tieto ja toiminnot ovat toiminnallisten määritysten ja liiketoimintalogiikan mukaan toisiinsa linkitettyjä. Esimerkiksi samaa laskua ei voi poistaa kahdesti ja samasta myyntilaskusta ei voi muodostaa montaa tositetta. Toisaalta, jos järjestelmässä luodaan monta uutta laskua ja niille asetetaan manuaalisesti tietoja, niin makroa toistaessa tämä laskut ovat jo olemassa ja luonti epäonnistuu liiketoimintalogiikan rajoitusten takia (esim. päällekkäiset arvot eivät sallittuja). Lisäksi edellisessä tapauksessa luonti epäonnistuu myös teknisten rajoitteiden takia: Nauhoitustilanteessa siirrytään luontisivulle, jossa luonnin yhteydessä tallennetaan tieto painaa linkkiä, jonka sisältö on avata lasku 10 (uuden luodun tietueen arvo tietokannassa). Kuitenkin toiston yhteydessä luodaan tietue, jolla on aivan eri arvo, joten toistaminen epäonnistuu. Web-sovellukset ovat täynnä samankaltaisia ongelmia. Vastassa on myös konkreettisia vaaroja, sillä

tuotantoympäristössä epäonnistuneilla tai heuristisissa päätöksissä vikaan menevillä makroilla voidaan tehdä konkreettisesti rahassa mitattavaa haittaa.

Makroparadigman ja liiketoimintalogiikan yhteentörmäyksen ratkaisuun voidaan tunnistaa kolme vaihtoehtoista mallia. Ensimmäisessä makrojen toiminta-alue sovelluksessa rajoitetaan. Tässä mallissa makrotoiminnallisuus keskitetään sinne, missä sitä voidaan käyttää. Toisessa ongelmaa lähdetään ratkaisemaan heuristisesti tunnistusalgoritmin kehityksen avulla. Tässä tapauksessa hylätään idea identtisestä toistotilanteesta ja ratkaisuun pyritään käyttäjäaikeen tunnistuksen avulla. Algoritmia voi kehittää ymmärtämään, että nyt halutaan käyttää sivulta löytyvää arvoa, eikä tallennettua arvoa, jollain heuristisilla raja-arvoilla. Tämän ratkaisumallin seurauksena ratkaisun tarkkuus kuitenkin heikkenee. Kolmantena mallina voidaan turvautua koko web-sovelluksen tilan pysäyttämiseen. Nauhoitushetkellä voidaan ottaa kopio tietokannan tilasta, joka palautetaan makron toistotilanteen aikana. Tämä mahdollistaa laajoja käyttötarkoituksia, mutta tämä ratkaisumalli vaatii suuria muutoksia web-sovellukseen. Ratkaisu olisi myös riippuvainen mm. tietokantamoottorin sisäisestä toteutuksesta, joten tämä malli ei ole välttämättä edes toteutuskelpoinen. Yleispätevää mallia ei välttämättä ole, ja spesifisen mallin valinta riippuu kriittisesti makrojärjestelmän käyttötarkoituksesta ja -ympäristöstä.

Makrokirjasto sisältää merkittäviä yhteensopivuushaasteita. Ratkaisun koodi on kirjoitettu suoraan Firefox-selaimella toimivaksi, mutta silti oletettavasti suuri osa siitä toimii myös Webkit- ja Opera-ympäristössä. Jos selaintuki laajennetaan koskemaan kaikkia suosittuja selaimia, niin koodirivien määrä voi karkeasti arvioituna kaksinkertaistua. Esimerkiksi sivulatauksen (valmistumisen) tilan määrittävä funktio on 60 riviä koodia, josta 39 riviä keskittyy Internet Explorer -selaimen vaatimaan erikoiskäsittelyyn. JavaScript-ympäristössä ei hyvien käytäntöjen mukaan luoteta selainten user agent -tietoon, vaan sen sijaan siihen mitä selain oikeasti tukee, mikä taas määritetään funktion ja muuttujan läsnäolon tarkistuksella. Tästä syystä koodirivit kasvavat nopeasti. Merkittävä osa kohdatuista haasteista johtuu selainten välisistä eroista DOM-rajapintatoteutuksessa (erityisesti tapahtumamallin erot), mutta myös iframe-elementin käsittelyssä todettiin olevan eroja. Toisaalta nämä ongelmat johtuvat myös prototyyppiratkaisuun päätyneistä huonoista käytännöistä. Täysi selainyhteensopivuus voi vaatia joidenkin ratkaisujen teknisen toteutuksen muokkaamista.

6.2 Johtopäätökset

Tämän työn kaksi tutkimuskysymystä olivat ”Miten web-makrot voidaan toteuttaa kohdesovelluksessa?” ja ”Mitkä ovat työn ratkaisun tekniset mahdollisuudet, haasteet ja rajoitteet?” Työn tuloksena havaittiin, että web-makrojärjestelmän toteutus on teknisesti täysin mahdollista. Työn lopputulos eli makrokirjasto on teknisesti toimiva ja laajennuskelpoinen ratkaisu makrotoiminnallisuuden toteuttamiseen. Makrokirjasto on vielä prototyyppi, mutta täysin laajennettavissa valmiiksi käyttäjäystävälliseksi tuotteeksi. Työn aikana tuli kuitenkin selväksi, että ratkaisua ei voida rakentaa ilman kompromisseja ratkaisun virheherkkyyden, täydellisyyden ja tarkkuuden välillä. Nämä ominaisuudet ovat keskenään ristiriitaisia, joten niiden keskinäinen painotus riippuu makrojärjestelmän tarkasta käyttötarkoituksesta ja käyttöympäristöstä. Työn ratkaisun perusteella voidaan esittää, että kun käyttöympäristö eli kohdesovellus otetaan huomioon, niin järjestelmän tarkkuus ja yleinen toimivuus paranevat. Ratkaisussa ei ole huomioitu tarkkaa käyttötarkoitusta, koska sitä ei ole vielä määritetty.

Ratkaisun hyödyntämisessä havaittiin kaksi merkittävää haastetta. Nämä ovat makroparadigman ja sovelluksen liiketoimintalogiikan yhteentörmäys sekä tämän ratkaisun selainkohtainen toteutus eli selainyhteensopivuuden puute. Ensimmäinen haaste on erittäin vaikea ratkaista koko järjestelmän mittakaavassa. Tästä syystä työ esittää, että ehkä aidot web-makrot eivät ole kaikissa tilanteissa paras tai edes mahdollinen ratkaisu. Erityisesti esimerkkeinä mainituissa esittely- ja opastuskäyttötapauksissa saattaa olla aiheellista tutkia muita lähestymistapoja, koska nämä tapaukset vaativat koko sovelluksen laajuisia nauhoitteita, jotka ovat erityisen alttiina järjestelmän liiketoimintasääntöjen muodostamille rajoitteille. Tämä ongelma on epäilemättä tärkeä aihe jatkotutkimukselle. Toinen haaste eli verkkoselainyhteensopivuus koskee erityisesti järjestelmän ylläpitoa. Yhteensopivuus voidaan rakentaa ja ylläpitää, mutta se todennäköisesti vaatii merkittävän osan ratkaisun jatkokehitys- ja ylläpitoresursseista. Työn kohdesovellustapauksessa tämä ei välttämättä ole liiketoiminnallisesti kannattavaa. Näiden haasteiden johdosta web-makrojen yleinen hyödynnettävyys SaaS-sovelluksessa on heikko. Hyödyntäminen vaikuttaa olevan ylläpitoresursseja kuluttavaa ja samalla vahvasti paikallisesti rajoittunutta eli heikosti yleistettävissä koko sovelluksen laajuiseksi ratkaisuksi.

7 YHTEENVETO

Tämä kandidaatintyö käsittelee web-sovelluksen käyttäjäautomaatiota web-makrojen kautta. Makrot ovat nauhoitettuja esimerkkisuorituksia, joita luodaan nauhoittamalla käyttäjäsyötteet toistokelpoiseksi tallenteeksi. Työn teoriaosa esittelee web-automaatiota ja web-makrojen toimintaympäristöä eli työn kannalta keskeistä web-teknologiaa. Työn ratkaisuosa pohjustaa, esittelee ja arvioi työssä toteutetun prototyypin, joka on SaaS-sovellukseen kohdistettu makrojärjestelmä. Ratkaisun lähtökohdat pohjautuvat Montoto et al. [10] artikkeliin, mutta yksityiskohdat ja toteutus ovat täysin tämän työn omaa tuotosta.

Työ esittelee toimivan prototyyppiratkaisun, mutta tunnustaa tarpeen ratkaisun lisätestaukselle varsinaisessa kohdesovellusympäristössä. Työn ratkaisussa käyttäjä eristetään web-sovelluksesta sieppaustasolla, joka tunnistaa, tallentaa ja suorittaa käyttäjäsyötteet. Tunnistuskäsittely sisältää käyttäjäsyötteiden paikannuksen, syötetyypin tunnistamisen sekä dokumenttiosoitteen määrityksen. Siepatut käyttäjäsyötteet siirretään sieppaustasolta web-sovelluksen toiminnoiksi dokumenttitapahtumamallin kautta. Sieppauskerros on käyttäjälle näkymätön, joten web-sovelluksen käyttöliittymä ja käyttökokemus pysyvät pääosin muuttumattomina. Teknisesti työn ratkaisu perustuu JavaScript-, XPath- ja DOM-teknologiaan.

Työn tuloksena havaittiin, että web-makrojen toteutus on teknisesti täysin mahdollista. Työn ratkaisun merkittävin tekninen haaste on selainyhteensopivuus, jonka arvioidaan vievän suuri osa jatkokehitys- ja ylläpitoresursseista. Ratkaisu kohtaa kuitenkin lukuisia toiminnallisia haasteita. Lopputuloksena ratkaisun hyödynnettävyys koko SaaS-sovelluksen mittakaavassa on heikko, koska web-sovelluksen liiketoimintalogiikka rajoittaa makroparadigman hyödyntämistä.

LÄHTEET

[1] Murugesan, S. 2007. Understanding Web 2.0. *IT Professional.* IEEE Computer Society. Volume 9, Issue 4. July-August 2007. pp. 34-41.

[2] Bolin, M., Webber, M., Rha, P., Wilson, T., Miller, R. C. 2005. Automation and customization of rendered web pages. *Proceedings of the 18th annual ACM symposium on User interface software and technology (UIST '05).* Association for Computing Machinery. October 23–27, 2005. Seattle, Washington, USA. pp. 163-172.

[3] Safonov, A., Konstan, J. A., Carlis, J. V. 1999. Towards Web Macros: a Model and a Prototype System for Automating Common Tasks on the Web. *Proceedings of the 5th Conference on Human Factors and the Web.* June 3, 1999. Gaithersburg, Maryland, USA. Saatavilla: http://zing.ncsl.nist.gov/hfweb/proceedings/safonov/ [Viitattu 21.2.2011]

[4] SIIA 2001. Software as a Service: Strategic Backgrounder. Software & Information Industry Association. Julkaistu helmikuussa 2001. Saatavilla: http://www.siia.net/estore/pubs/SSB-01.pdf [Viitattu 20.5.2011]

[5] Barry, C., Lang, M. 2001. A Survey of Multimedia and Web Development Techniques and Methodology Usage. *IEEE Multimedia.* IEEE Computer Society. Volume 8, Issue 2. April-June 2001. pp. 52-60.

[6] Ravi, J., Yu, Z., Shi, W. 2009. A Survey on Dynamic Web Content Generation and Delivery Techniques. *Journal of Network and Computer Applications.* Elsevier B.V. Volume 32, Issue 5. September 2009. pp. 943-960.

[7] Garret, J. J. 2005. Ajax: A New Approach to Web Applications. Julkaistu 18.2.2005. Saatavilla: http://www.adaptivepath.com/ideas/essays/archives/000385.php [Viitattu 6.2.2011]

[8] Kurlander, D., Feiner, S. 1992. A History-based Macro by Example System. *Proceedings of the 5th annual ACM symposium on User interface software and technology (UIST '92).* Association for Computing Machinery. November 15-18, 1992. pp. 99-106.

[9] Krulwich, B. 1997. Automating the Internet: Agents as User Surrogates. *IEEE Internet Computing.* IEEE Computer Society. Volume 1, Issue 4. pp. 34-38.

[10] Montoto, P., Pan, A., Raposo, J., Bellas, F., Lopez, J. 2011. Automated Browsing in Ajax Websites. *Data & Knowledge Engineering.* Elsevier B.V. Volume 70, Issue 3. March 2011. pp. 269-283.

[11] Scaffidi, C., Cypher, A., Elbaum, S., Koesnandar, A., Myers, B. 2008. Using Scenario-based Requirements to Direct Research on Web Macro Tools. *Journal of Visual Languages & Computing.* Elsevier B.V. Volume 19, Issue 4. August 2008. pp. 485-498.

[12] Hupp, D., Miller, R. C. 2007. Smart Bookmarks: Automatic Retroactive Macro Recording on the Web. *Proceedings of the 20th annual ACM symposium on User interface software and technology (UIST '07).* October 7-10, 2007. Newport, Rhode Island, USA. pp 81-90.

[13] Faaborg, A., Lieberman, H. 2006. A Goal-Oriented Web Browser. *Proceedings of the SIGCHI conference on Human Factors in computing systems (CHI '06).* Association for Computing Machinery. 24-27 April 2006, Montréal, Québec, Canada. pp. 751-760.

[14] Automator. Your Personal Automation Assistant. MAC OS X. Julkaisuaika tuntematon. Saatavilla: http://www.macosxautomation.com/automator/ [Viitattu 9.3.2011]

[15] Chickenfoot. Chickenfoot for Firefox: rewrite the web. FAQ. Massachusetts Institute of Technology. Julkaisuaika tuntematon. Saatavilla: http://groups.csail.mit.edu/uid/chickenfoot/faq.html [Viitattu 9.3.2011]

[16] W3C 1999. HTML 4.01 Specification W3C Recommendation. World Wide Web Consortium. Julkaistu 24.12.1999. Saatavilla: http://www.w3.org/TR/1999/REC-html401-19991224/ [Viitattu 6.2.2011]

[17] CoScripter. Simplifying web-based Processes. Julkaisuaika tuntematon. Saatavilla: http://coscripter.researchlabs.ibm.com/coscripter/ [Viitattu 9.3.2011]

[18] iMacros. Browser Automation and Web Scripting with iMacros. iOpus. Julkaisuaika tuntematon. Saatavilla: http://www.iopus.com/imacros/web-scripting.htm [Viitattu 9.3.2011]

[19] Hämäläinen, A., Ikonen, J. 2011. Luentokalvot. Opintojakso: CT30A3101 Web-ohjelmointi. Lukuvuosi 2010-2011. Lappeenrannan teknillinen yliopisto,

Teknistaloudellinen tiedekunta, Tietotekniikan koulutusohjelma.

[20] W3C 2008. Extensible Markup Language (XML) 1.0 (Fifth Edition) W3C Recommendation. World Wide Web Consortium. Julkaistu 26.11.2008. Saatavilla: http://www.w3.org/TR/REC-xml/ [Viitattu 14.2.2011]

[21] Flanagan, D. 2006. JavaScript: The Definitive Guide. Fifth Edition. O'Reilly Media, Inc.

[22] Resig, J., Bibeault, B. 2011. Secrets of the JavaScript Ninja. Manning Early Access Program, Version 4. Manning Publications.

[23] W3C 2011. W3C XML Query (XQuery): XML Query Implementations. World Wide Web Consortium. Revision: 1.191. Julkaistu 21.1.2011. Saatavilla: http://www.w3.org/XML/Query/ [Viitattu 15.2.2011]

[24] W3C 1999. XML Path Language (XPath) Version 1.0 WC3 Recommendation. World Wide Web Consortium. Julkaistu 16.11.1999. Saatavilla: http://www.w3.org/TR/xpath/ [Viitattu 15.2.2011]

[25] W3C 2002. XHTML 1.0: The Extensible HyperText Markup Language (Second Edition) W3C Recommendation. World Wide Web Consortium. Julkaistu 26.1.2000, muokattu 1.8.2002. Saatavilla: http://www.w3.org/TR/xhtml1 [Viitattu 6.2.2011]

[26] Stachowiak, M. 2006. Understanding HTML, XML and XHTML. Surfin' Safari. Julkaistu 20.9.2006. Saatavilla: http://www.webkit.org/blog/68/understanding-html-xml-and-xhtml/ [Viitattu 20.2.2011]

[27] W3C 1998. Document Object Model (DOM) Level 1 Specification Version 1.0 W3C Recommendation. World Wide Web Consortium. Julkaistu 1.10.1998. Saatavilla: http://www.w3.org/TR/REC-DOM-Level-1/ [Viitattu 17.2.2011]

[28] W3C 2004. W3C Document Object Model (DOM) Requirements. W3C Working Group Note. World Wide Web Consortium. Julkaistu 26.2.2004. Saatavilla: http://www.w3.org/TR/2004/NOTE-DOM-Requirements-20040226/ [Viitattu 17.2.2011]

[29] Koch, P.-P. 2009. Event compatibility tables. Quirksmode. Julkaistu 22.2.2009. Saatavilla: http://www.quirksmode.org/dom/events/index.html [Viitattu 26.2.2011]

[30] Fowler, M. 2006. Event Collaboration. Martin Fowler. Julkaistu 19.6.2006. Saatavilla: http://www.martinfowler.com/eaaDev/EventCollaboration.html [Viitattu 19.2.2011]

[31] W3C 2000. Document Object Model (DOM) Level 2 Events Specification Version 1.0 W3C Recommendation. World Wide Web Consortium. Julkaistu 13.11.2000. Saatavilla: http://www.w3.org/TR/DOM-Level-2-Events/ [Viitattu 19.2.2011]

[32] W3C 2010. Document Object Model (DOM) Level 3 Events Specification. W3C Working Draft. World Wide Web Consortium. Julkaistu 7.9.2010. Saatavilla: http://www.w3.org/TR/DOM-Level-3-Events/ [Viitattu 16.5.2011]

[33] Kienle, H.M. 2010. It's About Time to Take JavaScript (More) Seriously. *IEEE Software.* IEEE Computer Society. Volume 27, Issue 3. April 2010. pp. 60-62.