Juha Halme

Palvelurajapintojen käyttö mobiiliselaimesta

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Mediatekniikan koulutusohjelma
Insinöörityö
25.5.2012

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Juha Halme Palvelurajapintojen käyttö mobiiliselaimesta 50 sivua 25.5.2012
Tutkinto	insinööri (AMK)
Koulutusohjelma	mediatekniikka
Suuntautumisvaihtoehto	digitaalinen media
Ohjaajat	yliopettaja Kari Salo yliopettaja Petri Vesikivi

Mobiililaitteiden ja niille suunnattujen ohjelmien määrä on kasvanut viime vuosina. Markkinoilla on useita eri laitevalmistajia ja käyttöjärjestelmiä, joita palveluntarjoajien tulisi tukea kattaakseen koko asiakaskunnan. Samojen ohjelmien toteuttaminen ja ylläpito useille eri käyttöjärjestelmille on kuitenkin kallista, eikä yrityksillä välttämättä ole tarvittavaa tietotaitoa toteuttaa ohjelmia edes muutamalle suosituimmista alustoista.

Selainpohjaiset sovellukset ratkaisevat suurimman osan pirstaloituneista markkinoista johtuvista ongelmista. Niitä suoritetaan Internet-selaimessa, joka on kaikissa nykyaikaisissa mobiililaitteissa. Selainpohjaisilla sovelluksilla ei kuitenkaan ole mahdollisuutta käyttää kaikkia käyttöjärjestelmän tarjoamia palvelurajapintoja, vaikka HTML5 parantaakin tilannetta hieman.

Insinöörityössä tutkittiin ja toteutettiin kolme erilaista ratkaisumallia, joilla Internet-selaimella suoritettava selainpohjainen sovellus voi käyttää käyttöjärjestelmän palvelurajapintoja. Käytännössä työssä toteutettiin ohjelmallinen silta selaimen skriptiympäristön ja palvelurajapintojen välille hyödyntäen NPAPI-selainlisäosia, WebSocket-teknologiaa ja WebView-ohjelmistokomponenttia.

Kaikista ratkaisumalleista saatiin toteutettua toimivat ohjelmat, jotka ratkaisevat alkuperäisen ongelman. NPAPI-selainlisäosaan ja WebSocket-teknologiaan perustuvilla ratkaisumalleilla ei kuitenkaan tavoiteta koko asiakaskuntaa, koska kaikki markkinoiden laitteista eivät tue niitä. WebView-ohjelmistokomponenttiin perustuva ratkaisumalli puolestaan ei täyttänyt alkuperäistä vaatimusta, koska se vaatii erillisen ohjelman tavallisen Internet-selaimen sijaan.

Insinöörityössä tutustuttiin myös tarkemmin Internet-selainten toimintaan, koska ne toimivat selainpohjaisten sovellusten suoritusympäristönä.

Avainsanat	Web App, NPAPI, HTML5, WebSocket, selainmoottori

Metropolia

Abstract

Author Title Number of Pages Date	Juha Halme Utilizing service APIs from mobile web browser 50 pages 25 May 2012
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructors	Kari Salo, Principal Lecturer Petri Vesikivi, Principal Lecturer

The amount of mobile devices and software developed for them has increased in the recent years. There are multiple different manufacturers and operating systems, which should all be covered by the developers in order to get the maximum target audience. However, developing and maintaining multiple versions of the same application is expensive and companies might not have the required skill set to build software even for the most popular platforms.

Web applications bring a solution to the difficulties caused by fragmentation of the market. They are ran in an Internet browser, which can nowadays be found in every modern mobile device. Web applications do not have the means to utilize all the service APIs provided by the operating system, even though the new HTML5 specification brings some light to the situation.

The goal of the thesis was to study and implement three different solutions which let web applications use the service APIs provided by the operating system. The three different bridges between web applications and service APIs were built with NPAPI browser plug-in, WebSocket server and WebView software component.

All three different solutions were built successfully and they provided an answer to the original problem. The solutions based on NPAPI browser plug-in and WebSocket technology can not be used on every mobile device on the market due to incompatibilities of the used technologies. The WebView based solution, however, did not meet the original specifications as the web applications have to be used through their own application instead of a standard Internet browser.

The thesis also studies the structure of Internet browsers and how they work, as they provide the environment in which the web applications are ran.

Keywords	Web App, NPAPI, HTML5, WebSocket, browser engine, HTML rendering engine

Sisällys

1 Johdanto

Insinöörityön tavoitteena on tutkia, suunnitella ja toteuttaa eri mobiilialustoille ratkaisuja, joiden avulla Internet-palvelimella sijaitseva web-sovellus pystyy hyödyntämään asiakasympäristön tarjoamia palvelurajapintoja. Palvelurajapinnoilla tarkoitetaan käyttöjärjestelmän tarjoamia palveluita, joita ohjelmistot tarvitsevat toimiakseen. Tiedostojärjestelmä on yksi käyttöjärjestelmän tarjoama palvelurajapinta.

Erilaisten mobiilialustojen määrän kasvaessa palveluiden ja sovellusten tarjoaminen mahdollisimman suurelle kohdeyleisölle vaatii palveluntarjoajalta entistä enemmän resursseja. Eräs ratkaisu ongelmaan on siirtyä yksittäisistä tietyille käyttöjärjestelmille suunnatuista ohjelmista selainpohjaisiin sovelluksiin. Selainpohjaisella sovelluksella tarkoitetaan sovellusta, joka sijaitsee Internet-palvelimella, mutta itse sovelluksen suorittamiseen käytetään pääasiassa käyttäjän Internet-selainta ja sen skriptiympäristöä. Mikä tahansa laite, jossa on Internet-selain, pystyy teoriassa käyttämään selainpohjaisia sovelluksia.

Mobiiliselaimilla käytetyllä selainpohjaisilla sovelluksilla ei kuitenkaan voida hyödyntää laitteiden palvelurajapintoja kokonaisuudessaan. HTML5-standardin myötä selainpohjaisilla sovelluksilla on entistä enemmän toiminnallisuuksia, joita aikaisemmin ainoastaan natiivit sovellukset ovat voineet hyödyntää. HTML5-standardilla on kuitenkin omat rajoitteensa, eikä se kata tai osaa hyödyntää kaikkia nykyaikaisten mobiiliympäristöjen tarjoamia palvelurajapintoja. Näiden puuttuvien palveluiden hyödyntämiseksi tarvitaan ohjelmallisesti toteutettu silta Internet-selaimen skriptiympäristön ja palvelurajapintojen välille.

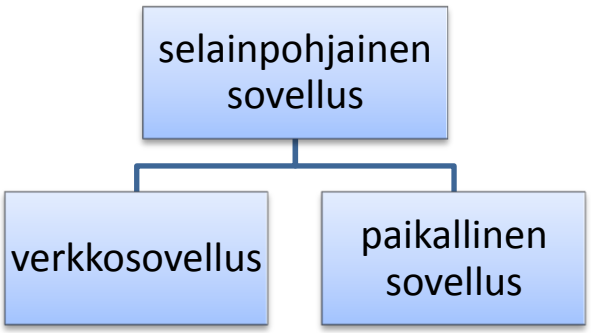
Verkkosovellusten määrän kasvaessa Internet-selaimista ja niiden selainmoottoreista on tullut todella merkittäviä sovellusten suoritusympäristöjä. Insinöörityössä tutustuttiin tarkemmin muutamaan yleisimmistä selainmoottoreista ja siihen, miten ne toimivat ladatessaan Internet-sivuja ja suorittaessaan niiden sisältämiä skriptejä.

Insinöörityössä toteutetaan kolme erilaista ratkaisumallia ohjelmallisen sillan luomiseksi. Ratkaisut perustuvat NPAPI-selainlisäosaan, WebSocket-palvelimeen asiakasympäristössä ja WebView-ohjelmistokomponenttiin.

2 Selainpohjaiset sovellukset

2.1 Selainpohjaisten sovellusten rakenne

Selainpohjaisia sovelluksia suoritetaan ja käytetään Internet-selaimen avulla. Verkkosovellukset ovat selainpohjaisia sovelluksia, ja ne sijaitsevat tavallisesti verkossa, joko julkisessa Internetissä tai yksityisessä verkossa. Internet-selaimella voidaan kuitenkin näyttää myös paikallisessa tiedostojärjestelmässä sijaitsevia selainpohjaisia sovelluksia. Kuvan 1 kaavio esittää selainpohjaisten sovellusten suhdetta toisiinsa.



Kuvio 1. Selainpohjaisten sovellusten termien hierarkia.

Tavallisista verkkosivuista poiketen selainpohjaisissa sovelluksissa suoritetaan interaktiivisia toimintoja paikallisesti Internet-selaimessa, ja niillä on mahdollisuus tallentaa tilansa [1]. Määritelmän mukaan suurin osa nykyaikaisista verkkosivustoista voidaan laskea verkkosovelluksiksi, koska erilaiset skriptikielet ja muun muassa evästeiden tallennus ovat niissä yleisesti käytössä. Tässä insinöörityössä käsitellään kuitenkin selainpohjaisia sovelluksia, jotka on suunniteltu mobiililaitteet mielessä ja joissa suurin osa toiminnallisuudesta suoritetaan paikallisesti Internet-selaimessa. Pyyntöjä palvelimelle tehdään ainoastaan tarvittaessa. Englanniksi tämänkaltaisia sovelluksia kutsutaan termillä Web App [2, s. 1].

Selainpohjaiset sovellukset muodostuvat samoista teknologioista ja elementeistä kuin perinteiset verkkosivut. Niiden rakenne kuvataan HyperText Markup Languagella (HTML), ja elementtien tyylien määrittelemiseen käytetään Cascading Style Sheets -teknologiaa (CSS). Selainpohjaisten sovellusten interaktiivisuudesta vastaavat Internet-selaimen tukemat skriptikielet. Käytetyistä skriptikielistä ECMAScript-standardiin perustuva JavaScript on suosituin.

2.2 Selainpohjaisten sovellusten edut tavallisiin sovelluksiin nähden

Laaja tuki eri ympäristöissä

Suuri koko ja kova kilpailu ovat tehneet matkapuhelinmarkkinoista hyvin sirpaloituneet eri käyttöjärjestelmien ja laitteiden suhteen [3]. Taulukkoon 1 on koottu vuonna 2011 myytyjen älypuhelinkäyttöjärjestelmien markkinaosuudet. Matkapuhelinvalmistajat voivat valmistaa älypuhelimia kolmella tai useammallakin eri käyttöjärjestelmällä. Tämän lisäksi samaa käyttöjärjestelmää käyttävät älypuhelimet voivat olla ominaisuuksiltaan hyvinkin erilaisia. Myös yksittäisistä käyttöjärjestelmistä voi olla useita eri versioita käytössä. Markkinoiden sirpaloitumisen vuoksi palveluntarjoajien on hyvin vaikeata ja kallista saada mahdollisimman suuri käyttäjäkunta palveluilleen toteuttamalla omat versiot kaikille eri matkapuhelinkäyttöjärjestelmille [4].

Taulukko 1. Älypuhelinkäyttöjärjestelmien markkinaosuudet vuonna 2011 [3].

Käyttöjärjestelmä	**Toimitukset (miljoonaa kpl)**	**Markkinaosuus**	**Vuosittainen kasvu**
Android	237,7	48,8 %	244 %
iOS	93,1	19,1 %	96 %
Symbian	80,1	16,4 %	-29,1 %
BlackBerry	51,4	10,5 %	5,0 %
Bada	13,2	2,7 %	183,1 %
Windows Phone	6,8	1,4 %	-43,4 %
Muut	5,4	1,1 %	14,4 %

Kaikissa älypuhelimissa ja myös useimmista peruspuhelimissa on kuitenkin Internet-selain, jossa voidaan suorittaa selainpohjaisia sovelluksia. Toteuttamalla selainpohjai-

sen sovelluksen palveluntarjoaja pystyy teoriassa kattamaan suurimman mahdollisen asiakaskunnan käyttämät laitteet. Tämä näkyy myös sovellusten kehittäjille suunnatussa kyselyssä, jonka mukaan mielenkiinto HTML5-sovelluksien kehitystä kohtaan on nousussa [5].

Nopeasti päivitettävissä

Verkkosovellus sijaitsee usein palveluntarjoajan omalla Internet-palvelimella, jonka sisältöä se pääsee muokkaamaan välittömästi. Näin ollen mikäli verkkosovelluksessa havaitaan virhe, se voidaan korjata heti. Verkkosovellukset päivittyvät nopeammin myös loppukäyttäjien laitteisiin. Aina kun verkkosivua käytetään, Internet-selain hakee uusimman version sovelluksesta palvelimelta eikä käyttäjä välttämättä edes huomaa päivitysoperaatiota.

Tavalliset sovellukset puolestaan sijaitsevat usein jonkun eri palveluntarjoajan tekemässä kauppa- ja latauspalvelussa. Joidenkin tällaisten palveluiden ylläpitäjät tarkistavat sovellukset ja niiden päivitykset, ennen kuin käyttäjille annetaan mahdollisuus ladata niitä. [6.]

Helposti ja nopeasti toteutettavissa

Koska selainpohjaiset sovellukset perustuvat vanhoihin ja tunnettuihin teknologioihin, niille on jo valmiiksi paljon osaavia tekijöitä. Selainpohjaisten sovellusten kehitysprosessi on myös usein yksinkertaisempi kuin natiivisovellusten. Riittää, että sovelluksen laittaa paikkaan, josta sen voi avata Internet-selaimella, ja se on heti valmiina testattavaksi. Natiivisovelluksia puolestaan joudutaan usein ensiksi kääntämään ja paketoimaan tiedostoiksi, jotka voidaan asentaa suoritusympäristöön, ennen kuin niitä voidaan testata. [2, s. 2.]

2.3 Mobiiliympäristön rajoitteet

Vaikka mobiililaitteet ovatkin viime vuosina kehittyneet ja joissakin matkapuhelinmalleissa on moniydinprosessoreita, ei niiden laskentateho ole vielä työpöytätietokoneiden tasolla. Suorituskyvyn lisäksi ongelmia tuottaa virrankulutus ja siitä seuraava akun loppuminen. Toisaalta nykyaikaiset mobiiliselaimet voivat käyttää samoja selainmoottoreita kuin työpöytäselaimet [7]. Tämän ansiosta mobiiliympäristöjen ja työpöytäympäris-

töjen verkkosovellusten erot ovat hämärtyneet sen suhteen, mitä niillä on teoriassa mahdollista tehdä.

Mobiiliympäristössä verkkosovellusten tulee ottaa huomioon mahdollisesti rajoittunut tai rajoitettu Internet-yhteys. Käyttäjällä voi olla palvelusopimus, jossa tietoliikennettä on rajoitettu joko nopeudeltaan tai siirtomääriltään [8]. Käyttäjä voi myös olla väliaikaisesti katvealueella ilman Internet-yhtettä, jolloin yhteyttä palvelimeen ei saada. Langattomissa mobiiliverkoissa on myös usein pidemmät vasteajat, mikä näkyy käyttäjälle hitautena jokaista palvelinpyyntöä tehtäessä [9, s. 11]. Nämä ongelmakohdat voidaan korjata pitämällä pyyntöjen määrä ja niiden sisältö palvelimelle mahdollisimman pieninä.

Mobiiliverkkosovellusten käyttöliittymäsuunnittelussa tulee pitää mielessä laitteiden mahdollisesti pienet näyttökoot. Nykyään kosketusnäytöt ovat yleistyneet, mikä saattaa tuoda haasteita, jos yrittää toteuttaa perinteistä verkkosivua sormilla käytettäväksi tarkan hiiren sijaan.

2.4 HTML5-merkintäkieli ja CSS3-tyylimäärittelyt

HTML5 ja CSS3 ovat World Wide Web Consortiumin (W3C) ylläpitämiä standardeja [10; 11]. Molemmat ovat vielä kehitysvaiheessa, mutta monet niiden esittelemät ominaisuudet ovat jo tuettuina uusimmissa Internet-selaimissa. Vanhempien selainten huonon tuen ja mahdollisesti eri ominaisuuksia tukevien uusien selainten vuoksi joidenkin HTML5- ja CSS3-ominaisuuksien käyttäminen voi olla vielä vaikeata [4, s. 15].

Uudet graafiset ominaisuudet

HTML on ollut alusta asti varsin staattinen ja rajoittunut piirto- ja animaatio-ominaisuuksiltaan. HTML5 ja CSS3 tuovat kuitenkin mukanaan monia uudistuksia tapoihin, joilla verkkosivuilla voidaan esittää graafista sisältöä.

Canvas on HTML5:n mukana tullut uusi HTML-elementti. Canvas-elementti itsessään on ainoastaan tyhjä suorakaiteen muotoinen alue. HTML5 kuitenkin määrittelee rajapinnan, jonka avulla skriptikielillä voidaan piirtää canvas-elementtiin ja manipuloida sen sisältöä kuvapistetasolla. Aikaisemmin vastaavien ominaisuuksien toteuttamiseksi on

pitänyt turvautua kolmansien osapuolten tarjoamiin selainlisäosiin, kuten Adoben Flash tai Microsoftin Silverlight. [12, s. 135.]

Canvas-elementti tukee erilaisia piirtorajapintoja [10]. HTML5:n määrittelemä piirtorajapinta on 2D-piirtorajapinta. Tämän lisäksi canvas-elementti tukee Khronosin määrittelemää WebGL-piirtorajapintaa. Se on OpenGL ES 2.0 -standardista johdettu rajapinta, jonka avulla canvas-elementtiin voidaan piirtää 3D-grafiikkaa [13].

Canvas-elementin piirto-ominaisuuksien lisäksi HTML-sivulle voidaan luoda erilaisia siirtymäanimaatioita CSS3:n uusien ominaisuuksien avulla. Siirtymäanimaatioilla tarkoitetaan yksittäisten tyyliattribuuttien muuttamista ennalta määrätyn aikajanan perusteella. [14.]

Video- ja audio-ominaisuudet

Multimedian käyttö Internetissä on lisääntynyt paljon viime vuosina. Internetyhteyksien ja pakkausmenetelmien jatkuva kehitys on mahdollistanut multimedian käytön myös mobiililaitteilla. Ennen HTML5:n kehitystä palveluntarjoajilla ei ole kuitenkaan ollut mahdollisuutta tarjota multimediasisältöään minkään avoimen standardin avulla, ja tehtävään on käytetty kolmansien osapuolten kehittämiä selainlisäosia. [12, s. 97.]

HTML5 esittelee kaksi uutta mediaelementtiä: video- ja audio-elementit. Uusien elementtien avulla voidaan esittää käyttäjälle video- ja audiosisältöä käyttäen avoimiin standardeihin perustuvia teknologioita [12, s. 97]. Vaikka itse elementit ovat tuettuina useissa selaimissa, ei niiden käyttö ole vielä täysin ongelmatonta. Ongelmakohdaksi ovat muodostuneet koodekit eli koodaajat tai koodinpurkajat, jotka pakkaavat ja purkavat kuva- tai äänisignaalia. Tätä kirjoitettaessa uudet mediaelementit tukevat useita erilaisia koodekkeja, mutta millekään koodekille ei löydy tukea kaikista Internetselaimista [12, s. 99].

HTML5 määrittelee rajapinnan, jonka avulla voidaan hallita mediaelementeissä toistettavia multimediatiedostoja [10].

Web storage

Web storage on HTML5:n määrittelemä rajapinta, jonka avulla skriptikielet voivat tallentaa tietoja asiakasohjelman muistiin. Rajapinta määrittelee kaksi samankaltaista mekanismia, joilla on erilaiset käyttötarkoitukset:

- localStorage: tallennetut tiedot ovat kaikkien selainikkunoiden ja -välilehtien käytettävissä. Tiedot säilyvät, vaikka selainikkuna tai -välilehti suljetaan.
- sessionStorage: tallennetut tiedot tallennetaan sivun window-olioon, mikä tekee tiedoista selainikkuna- ja -välilehtikohtaisia.

Tiedot tallennetaan asiakasohjelman käyttämään muistiin yksinkertaisina tietueina, joilla on avain ja arvo. [2, s. 79.]

Offline Application Cache

Offline Application Cache on HTML5:n ominaisuus, jonka avulla käyttäjät voivat käyttää verkkosovelluksia ilman Internet-yhteyttä palvelimeen, jolla verkkosovellus sijaitsee. Käytännössä ominaisuus toimii siten, että käyttäjän Internet-selain lataa ja tallentaa kaikki verkkosovelluksen käyttämät tiedostot. Seuraavilla kerroilla, kun käyttäjä pyytää Internet-selaintaan hakemaan sivun, selain tunnistaa käyttäjän antaman URL:n ja lataa sovelluksen omasta muististaan palvelimen sijaan. [15, s. 193.]

Ominaisuuden toiminta perustuu verkkosovelluksen tekijän määrittelemään tiedostoon, jossa on lueteltu kaikki verkkosovelluksen käyttämät resurssit, joita tarvitaan verkkosovelluksen käyttämiseen. Tämän luettelon perusteella Internet-selain osaa ladata tarvittavat tiedostot ja hakea ne uudelleen palvelimelta ainoastaan tarvittaessa. [15, s. 193.]

WebSocket

WebSocket on teknologia, joka mahdollistaa kahdensuuntaisen yhteyden palvelimen ja asiakasohjelman välille käyttäen yhtä TCP-pistoketta. Se on suunniteltu käytettäväksi Internet-palvelimissa ja -selaimissa, mutta sitä voidaan käyttää muissakin ympäristöissä. WebSocket-teknologia muodostuu kahdesta osasta: WebSocket API:sta ja WebSocket-protokollasta. WebSocket API on W3C:n määrittelemä rajapinta, joka Internet-selainten tulee toteuttaa. WebSocket API:n avulla selainten tukemat skriptikielet voivat

muodostaa ja hallita WebSocket-yhteyksiä. WebSocket-protokolla on IETF:n standardoima protokolla, joka määrittää säännöt, joilla WebSocketia tukevat palvelimet ja asiakasohjelmat keskustelevat keskenään. WebSocket-protokolla on rakennettu HTTP-protokollan päälle, jotta olemassa oleva infrastruktuuri, kuten välityspalvelimet, toimisivat sen kanssa ilman muutoksia. [12, s. 241.]

HTTP-protokolla on pyyntö/vastaus-pohjainen protokolla. Sen määrittelyiden mukaan asiakasohjelma muodostaa yhteyden palvelimeen, jolle asiakasohjelma voi lähettää pyyntöjä. Palvelin puolestaan hyväksyy asiakasohjelman yhteyden ja lähettää takaisin vastauksen asiakasohjelman pyyntöön. Standardissa HTTP-mallissa palvelin ei voi aloittaa yhteyttä asiakasohjelman kanssa eikä se voi lähettää asiakasohjelmalle tietoja ilman pyyntöä. WebSocket-teknologian avulla palvelin voi yhteyden muodostamisen jälkeen lähettää asiakasohjelmalle tietoa reaaliajassa ja milloin tahansa ilman erillistä asiakasohjelman pyyntöä [16].

Web workers

JavaScriptiä suoritetaan aina vain yhdessä säikeessä. Jos suoritettava skripti on raskas ja sen suoritus vie paljon aikaa, joutuvat sivun muut skriptit ja käyttäjä odottamaan sen suorituksen ajan. Web workers -teknologian avulla voidaan toteuttaa ohjelmia, jotka suorittavat useampaa tehtävää samanaikaisesti. Tämä mahdollistaa raskaiden operaatioiden siirtämisen suoritettavaksi taustalla, jolloin muu sovellus säilyy toiminnallisena. [15, s. 221–222.]

2.5 Puuttuvat ominaisuudet

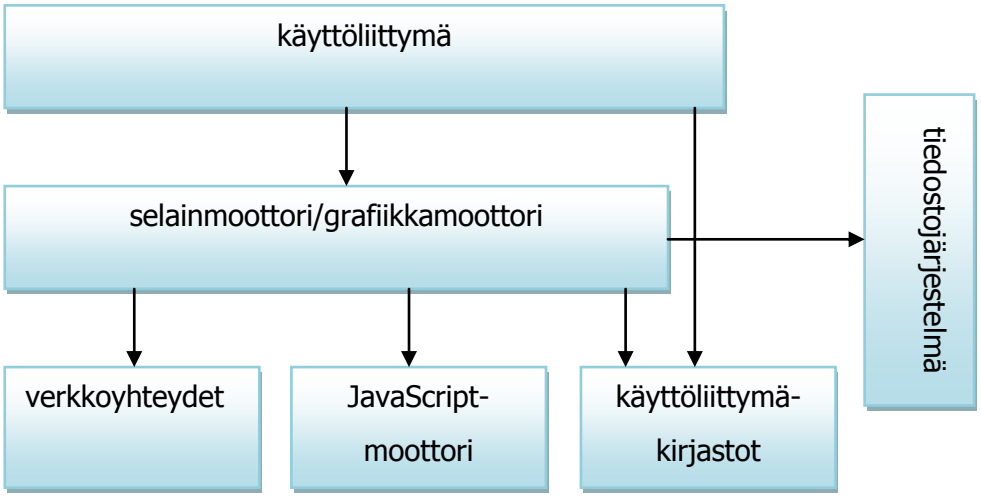
Vaikka HTML5 ja CSS3 tuovat monia uudistuksia selainpohjaisten sovellusten toiminnallisuuteen, ne eivät silti yllä natiivisovellusten tasolle palvelurajapintojen suhteen. Pääasiassa selainpohjaiset sovellukset jäävät laitteistotukensa puolesta kauaksi natiivisovelluksista. Esimerkiksi monista puhelinmalleista löytyvien liikeantureiden käytölle ei ole mitään standardien mukaista ratkaisua verkkosovelluksille. Toisaalta myös ohjelmistopuolella verkkosovelluksilla on rajoittuneempi tuki erilaisille rajapinnoille, jotka ovat natiivisovellusten käytettävissä. Esimerkiksi natiivisovelluksilla on usein mahdollisuus lukea ja muokata puhelimen osoitekirjaa siihen tarkoitetun rajapinnan avulla. Selainpohjaisille sovelluksille on vasta suunnitteilla rajapinta osoitekirjan lukemiseen [17].

3 Selainmoottorit

3.1 Selainmoottoreiden rakenne

Internet-selain voidaan jakaa kahteen osaan: käyttöliittymään ja selainmoottoriin. Käyttöliittymän avulla käyttäjä voi pyytää selainmoottoria hakemaan haluamansa Internet-sivun näytettäväksi. Selaimesta riippuen tämä voi tapahtua monin eri tavoin. Useimpien selaimien käyttöliittymässä on tekstikenttä, johon käyttäjä voi syöttää haettavan sivun osoitteen. Monet selaimet antavat myös mahdollisuuden tallentaa ennalta määriteltyjä suosikkeja. Selainmoottorin tehtävä on hakea halutun osoitteen sisällöt ja muodostaa siitä standardien mukaisesti malli, jonka se tulostaa selaimen käyttöliittymän määrittelemälle alueelle. [18.]

Tarkasteltaessa Internet-selaimen rakennetta tarkemmin, jakautuu selainmoottori useampiin pienempiin komponentteihin. Kuvan 2 mukainen esimerkkirakenne on saatu, kun on tutkittu olemassa olevien Internet-selainten rakenteita.



Kuvio 2. Internet-selaimen komponentit ja niiden riippuvuudet [18, s. 5].

Referenssikuvauksessa selainmoottori käyttää erillistä komponenttia verkkoyhteyksien hallintaan. Tämän komponentin vastuulla on toteuttaa tiedonsiirtoprotokollat, kuten HTTP ja FTP. Usein selainmoottoreihin on integroitu erillinen JavaScript-moottori, jonka tehtävänä on tulkata ja suorittaa Internet-sivujen mahdollisesti sisältämät JavaScript-koodit. Selainmoottorin grafiikkamoottori ja käyttöliittymä käyttävät molemmat käyttöjärjestelmän tarjoamia kirjastoja käyttöliittymäkomponenttien piirtämiseen. Tiedosto-

järjestelmää käytetään tallentamaan monia selausistuntoon liittyviä resursseja, kuten evästeet ja välimuistit. [18, s. 6.]

Vaikka Internet-selaimia on olemassa useita, vain muutamat niistä ovat saavuttaneet merkittävän suosion. Varteenotettavien selainmoottoreiden lukumäärä on kuitenkin huomattavasti pienempi, koska useat eri Internet-selaimet käyttävät samoja selainmoottoreita haettavan sisällön näyttämiseen. Taulukkoon 2 on koottu yleisimpiä Internet-selaimia ja niiden käyttämiä selain- ja JavaScript-moottoreita.

Taulukko 2. Internet-selaimet ja niiden käyttämät selain- ja JavaScript-moottorit [19; 20; 21; 22; 23; 24; 25].

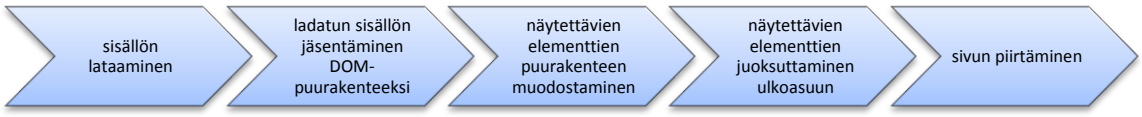
Selain	**Selainmoottori**	**JavaScript-moottori**
Chrome	WebKit	V8
Safari	WebKit	Nitro (JavaScriptCore)
Firefox	Gecko	SpiderMonkey
Opera	Presto	Carakan
Internet Explorer	Trident	Chakra

Mobiilipuolella WebKit-selainmoottori on saavuttanut selkeän markkinajohtajuuden, sillä älypuhelinkäyttöliittymämarkkinoita hallitsevien Androidin ja iOS:n oletusselaimet käyttävät WebKit-selainmoottoria. Oletusselainten lisäksi eri mobiilialustoille voi ladata myös muita Internet-selaimia. Esimerkiksi Prestoa käyttävä Opera ja Geckoa käyttävä Firefox ovat ladattavissa muiden muassa Androidille ja Maemolle.

3.2 Sivun hakemisen ja näyttämisen eri vaiheet

Selainmoottorin tehtävä on hakea sille annetun Unified Resource Identifierin (URI) takaa löytyvät sisällöt ja esittää ne halutussa muodossa käyttäjälle. Yleisin käyttötapaus on hakea URI:n osoittaman HTML-tiedoston ja sen viittaamien ulkoisten resurssien sisällöt. Ulkoisia resursseja ovat muun muassa kuvat, tyylitiedostot ja skriptitiedostot.

HTML5:n määrittely antaa referenssikuvauksen HTML-sivun sisältöjen lataamisen ja näyttämisen välivaiheista [10]. Suosituimpien selainmoottorien toiminta vastaa määrittelyiden mukaista prosessia, jonka rakenne näkyy kuvassa 3.

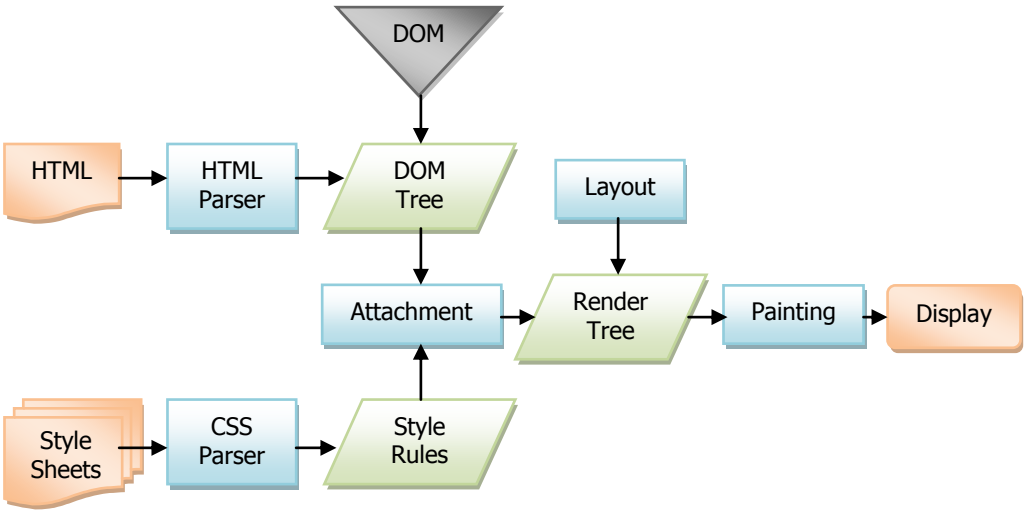


Kuvio 3. HTML-sivun esittämisen prosessi [26].

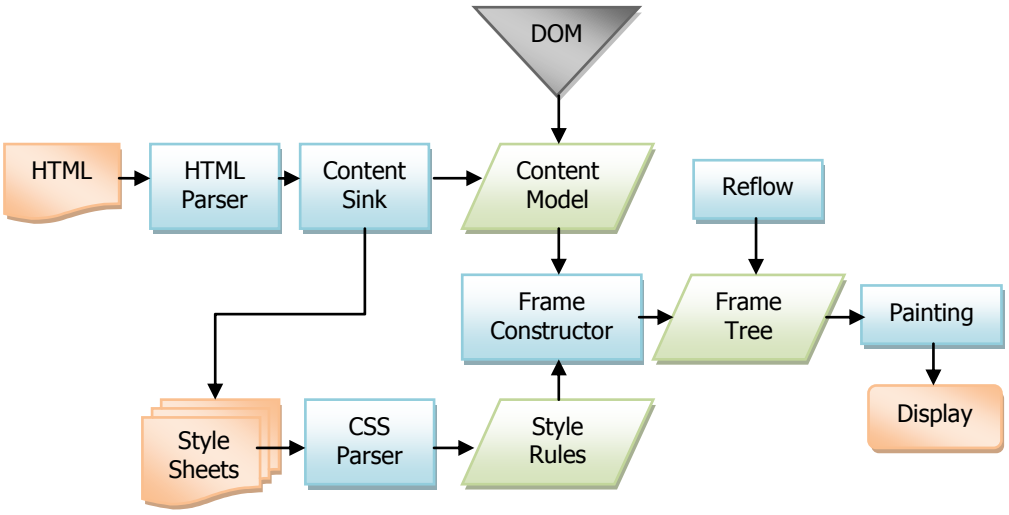
Sivun näyttäminen alkaa sivun sisältöjen lataamisella. Sisältöjen sijainti on määritelty URI:lla, ja se voi osoittaa esimerkiksi julkisessa Internetissä sijaitsevaan tai paikallisessa tiedostojärjestelmässä olevaan resurssiin. Alustavasti ladatut tiedot tulevat selainmoottorille tavuina, jotka sisältävät tekstidatan koodattuna jollakin merkistökoodauksella. Eri merkistökoodauksia on useita, ja selainmoottorin pitää tunnistaa sivun käyttämä merkistökoodaus. [10; 26.]

Merkistökoodauksen purkamisen jälkeen selainmoottori käy ladattujen tavujen sisältämän tekstidatan läpi ja muodostaa sen sisältämistä elementeistä Document Object Model (DOM) -puurakenteen ja erillisen puurakenteen, joka sisältää sivun määrittelemät tyylitiedot. Näitä kahta puumallia käyttäen selainmoottori voi luoda visuaalisen mallin sivun sisällöistä. Tämä visuaalinen malli sisältää kaikki sivulla näkyvät elementit ja niiden sijainnit. Visuaalinen malli voidaan tulostaa käyttäjän näytölle hyödyntäen käyttöjärjestelmän tarjoamia piirtorajapintoja. [26.]

Kuvissa 4 ja 5 on esitelty WebKitin ja Geckon toimintamallit sivujen lataamisesta sivun näyttämiseen. Vaikka selainmoottorit käyttävätkin hieman eri termistöä, ovat niiden toimintamallit hyvin lähellä toisiaan.



Kuvio 4. WebKit-selainmoottorin toimintamalli [26].



Kuvio 5. Gecko-selainmoottorin toimintamalli [26].

Dokumentin rakennepuun muodostaminen

HTML-sivujen jäsentäminen hierarkkiseksi rakennepuuksi ei ole yhtä yksinkertainen prosessi kuin esimerkiksi XML-tiedostojen vastaava operaatio. HTML:n hyvin anteeksiantava syntaksi ja vanhojen huonojen käytäntöjen tukeminen tekevät sen jäsentämisestä vaikeaa [27]. Näiden lisäksi jäsennettävä teksti voi sisältää skriptejä, jotka saattavat lisätä sivulle uusia elementtejä kesken jäsennyksen [10].

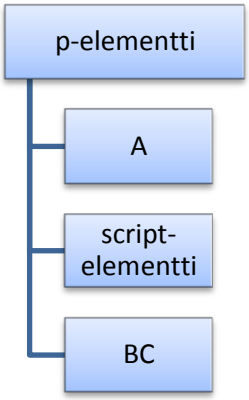
Vaikka HTML:lle on määritelty syntaksi, jota HTML-dokumenttien tulisi noudattaa, ei syntaksivirhe estä selainta näyttämästä dokumenttia käyttäjälle. Eräs suurimmista syistä tähän toimintamalliin ovat vanhat selaimet, jotka tulkitsivat ja näyttivät epävalideja HTML-dokumentteja muun muassa senaikaisten suppeiden HTML:n standardien tai selaimen omien ohjelmistovirheiden vuoksi. Tämän lisäksi eri selaimet saattoivat lisätä omia standardista poikkeavia ominaisuuksia. Toinen syy on HTML:n käyttötarkoitus ja käyttäjä- sekä kehittäjäystävällisyys. Koska kyseessä on informaation rakenteen kuvaamiseen käytettävä kieli, koetaan tärkeämmäksi informaation näyttäminen edes jossain muodossa syntaksivirheistä huolimatta kuin kokonaan näyttämättä jättäminen. [27.] Virheellinen syntaksi on niin yleistä HTML-dokumenteissa, että HTML5-standardi määrittelee, miten selainten tulisi tulkita erilaisia yleisimpiä syntaksivirheitä [10].

HTML tukee dokumentin sisällä ajettavia skriptikieliä. Tämä ominaisuus mahdollistaa tilanteen, jossa dokumentin käsittelyn aikana skripti voi lisätä, muokata tai poistaa elementtejä käsittelyn alla olevaan dokumenttiin.

Selaimen jäsennettyä esimerkkikoodin 1 tekstin DOM-puurakenteeseen muodostuu neljä elementtiä kuvan 6 mukaisesti.

```
<p>
    A
    <script>
        var text = document.createTextNode('B');
        document.body.appendChild(text);
    </script>
    C
</p>
```

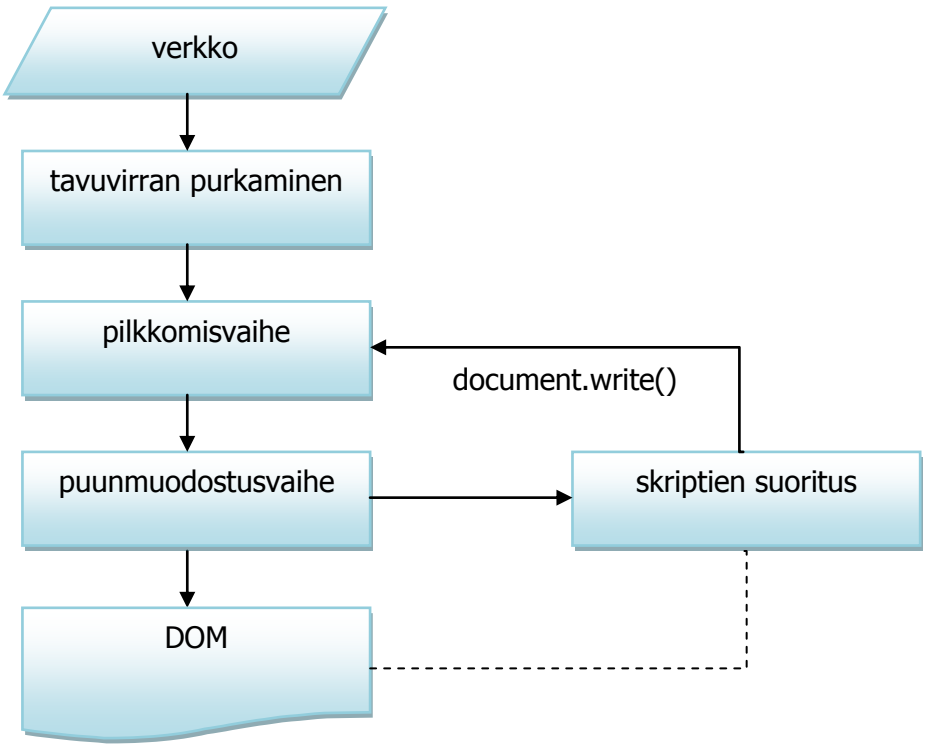
Esimerkkikoodi 1. HTML-koodiin upotettu script-elementti, joka lisää dokumenttiin uuden elementin.



Kuvio 6. Esimerkistä muodostuva DOM-puurakenne.

Käyttäjälle puolestaan näytetään ainoastaan kappale, jonka sisältönä on teksti ABC.

Sivun DOM-puurakenne muodostuu sivulla olevista elementeistä. Sen muodostaminen koostuu kahdesta vaiheesta: pilkkomis- ja puunmuodostusvaiheesta. Pilkkomisvaiheessa selainmoottori käy syötetekstiä merkki kerrallaan läpi ja antaa valmiita elementtien osia puunmuodostusvaiheelle. Valmiita elementtien osia ovat esimerkiksi aloitustagi ja lopetustagi. [10.] Kuvassa 7 on kuvattu HTML-dokumentin jäsentämisen vaiheet.



Kuvio 7. HTML-dokumentin jäsentämisen vaiheet [10].

Pilkkomisvaiheessa käytetään avuksi siihen tarkoitettua tilakonetta, jonka tila muuttuu riippuen merkeistä, joita se kohtaa käydessään läpi syötetekstiä [10]. Esimerkiksi tekstiä `<strong>Esimerkki</strong>` jäsentäessään tilakone saa taulukon 3 mukaisesti arvoja.

Taulukko 3. Pilkkomisvaiheen tilakoneen saamat arvot.

Merkki	**Uusi tila**	**Toiminto**
Alku	Data state	
<	Tag open state	
s		Luodaan uusi elementti
trong	Tag name state	Lisätään merkki luodun elementin nimeen
>	Data state	Välitetään -tagi eteenpäin
Esimerkki		Välitetään kukin kirjain kerrallaan eteenpäin
<	Tag open state	
/	End tag open state	
s		Luodaan uusi elementti
trong	Tag name state	Lisätään merkki luodun elementin nimeen
>	Data state	Välitetään -tagi eteenpäin

Pilkkomisvaihe lähettää valmiit elementtien osat puunmuodostusvaiheeseen. Myös puunmuodostusvaiheessa käytetään avuksi siihen tarkoitettua tilakonetta. Vastaanotetuista pilkotuista elementtien osista muodostetaan kokonaisia elementtejä, ja ne lisätään DOM-puurakenteeseen tilakoneen senhetkisen tilan määrittelemien sääntöjen mukaisesti. HTML-dokumentti muodostuu muutamasta korkean tason elementistä, jotka jakavat dokumentin eri osiin. Nämä elementit on kuvattu taulukossa 4. Puunmuodostusvaiheen tilakoneen tila määräytyy sen mukaan, mikä näistä dokumentin osista on käsiteltävänä. [10.]

Taulukko 4. HTML:n korkean tason elementit.

Elementti	Kuvaus
<html>	– Kertoo selaimelle dokumentin olevan HTML:ää – Dokumentin juurielementti, joka sisältää dokumentin muut elementit
<head>	Sisältää dokumentista tietoja, jotka eivät välttämättä näy käyttäjälle
<body>	Dokumentin sisältö, jonka selain näyttää käyttäjälle

Puunmuodostusvaiheen tilakone saa arvoja perustuen taulukossa 4 oleviin elementteihin: ”before HTML”, ”before head”, ”in head”, ”after head”, ”in body”, ”after body”, ”after after body” [26].

Dokumentin tyylien jäsentäminen

HTML-dokumenttien sisältöjen tyylit määritellään tavallisesti CSS:n avulla. Joillakin HTML-elementeillä on tyyleihin liittyviä attribuutteja, mutta niitä tulkattaessa selaimet muuttavat ne vastaaviksi CSS-tyylimäärittelyiksi [26]. HTML:n määrittelyiden [10] mukaan selaimen ei kuitenkaan ole pakko tukea CSS-tyylimäärittelyitä.

CSS:n kielioppi on kontekstiton kielioppi, eikä sen syntaksi ole yhtä monimutkainen tai anteeksiantava kuin HTML:n. Näin ollen sen jäsentäminen on helpompaa ja siihen voidaan käyttää yleisiä jäsentämismenetelmiä. CSS:n määrittelyissä on mukana CSS:n kielioppi Flex-formaatissa, joka muodostuu useista säännöllisillä lausekkeilla määritellyistä säännöistä. Muun muassa WebKitissä hyödynnetään tekniikkaa, joka osaa generoida tämän kielioppimäärittelyn perusteella jäsentimen automaattisesti. [26.]

Tyylien liittäminen HTML-elementteihin

DOM-puurakenteen lisäksi selainmoottori muodostaa toisen puurakenteen sivulla näkyvistä elementeistä. Elementtejä, jotka on määritelty näkymättömiksi CSS:n avulla, tai elementtejä, jotka ovat luonnostaan näkymättömiä, ei lisätä tähän puuhun. Elementit

on järjestetty siihen järjestykseen, jossa ne näytetään käyttäjälle. Jokainen rakennepuussa oleva elementti kuvaa suorakaiteen muotoista aluetta, jonka mitat on usein asetettu elementille annettujen CSS-määrittelyiden perusteella. Jokaisella elementillä on leveys, korkeus ja sijainti. [10; 26.]

Elementtien leveys, korkeus ja sijainti pitää laskea niille määriteltyjen tyylien perusteella. Yksittäisen elementin lopullinen tyyli voi olla useammassa eri paikassa määriteltyjen tyylien yhdistelmä. Jokaisella elementillä on selaimen määrittelemä oletustyyli, jonka lisäksi sivun tekijä on voinut määritellä elementille tyylejä CSS-tyylitiedostoilla tai upottaa tyylimäärityksiä itse HTML-elementtiin. Tämän lisäksi esimerkiksi Firefox tukee käyttäjän itse määrittelemiä tyylitiedostoja, jotka sivuston käyttäjä voi halutessaan ottaa käyttöön. Taulukossa 4 on listattuna CSS:n määrittelyiden mukainen tärkeysjärjestys eri lähteistä tuleville tyylimäärityksille. [26.]

Taulukko 5. CSS-tyylimääritysten lähteiden tärkeysjärjestys [26].

Tärkeys pienimmästä suurimpaan	**Tyylimäärityksen lähde**
1	Selaimen oletustyyli
2	Käyttäjän normaalin tärkeyden määritys
3	Sivun tekijän normaalin tärkeyden määritys
4	Sivun tekijän tärkeäksi merkitsemä määritys
5	Käyttäjän tärkeäksi merkitsemä määritys

Eri tyylimääritysten tärkeys tulee vastaan niissä tilanteissa, joissa elementille on määritelty samalle tyyliattribuutille eri arvo useammassa kuin yhdessä lähteessä. Mikäli samassa lähteessä on määritelty sama tyyliattribuutti useampaan kertaan, valitaan oikea tyylimääritys CSS:n määrittelyiden mukaisesti siihen tarkoitetulla pisteitysjärjestelmällä. Järjestelmässä annetaan pisteitä tyylimäärityksille niiden sääntöjen vastaavuudesta elementtiin nähden. [28.]

CSS:n avulla voidaan määritellä elementeille tyylejä niiden tunnisteen, luokan tai tyypin perusteella. Gecko ja WebKit muodostavat hajautustaulujen tyyppiset hakurakenteet jokaiselle eri tyylimääritysluokalle. Näytettävien elementtien puurakennetta muodostaessaan selainmoottorit hakevat elementtien tyylimääritykset näistä hajautustauluista.

Haun jälkeen tyylimäärityksistä karsitaan pois tyylit, jotka eivät sääntöjensä perusteella sovi elementille. Esimerkiksi tyylimäärityksen säännössä on voitu määrittää elementille eri hierarkia, vaikka muuten sääntö vastaisikin elementtiä. Lopuksi säännöt järjestetään oikeaan järjestykseen CSS:n pisteitysjärjestelmän mukaisesti. [26.]

CSS:n määrittelyiden [28] määrittelemän pisteitysjärjestelmän avulla laskettu vastaavuus muodostuu nelinumeroisesta luvusta. Jokainen luku lasketaan erikseen, minkä jälkeen luvut yhdistetään yhdeksi luvuksi. Yksittäiset luvut määräytyvät seuraavasti:

1. Luku on 1, jos tyyli on määriteltynä itse HTML-elementtiin style-attribuutin avulla. Muissa tapauksissa luku on 0.
2. Luku on säännössä käytettyjen ID-attribuuttien lukumäärä.
3. Luku on säännössä käytettyjen muiden attribuuttien lukumäärä.
4. Luku on säännössä käytettyjen elementtien lukumäärä.

Sivun ulkoasun muodostaminen

Kun kaikki näytettävät elementit on järjestetty oikeaan järjestykseen ja niille on liitetty tyylimäärittelyt, voidaan elementtien sijainnit laskea ja muodostaa sivun ulkoasu. HTML:ssä käytetään juoksutuspohjaista ulkoasun muodostamista. Juoksutus tapahtuu vasemmalta oikealle ja ylhäältä alas. Jokaisen elementin sijainti lasketaan sen isäntäelementin perusteella. Juurielementtinä toimii alue, johon sivun sisällöt tulostetaan. [26.]

Elementtien sijaintien laskeminen on rekursiivinen operaatio. Selainmoottori pyytää näytettävien elementtien puurakenteen juurielementtiä laskemaan omat mittansa ja sijaintinsa. Tietääkseen omat mittansa elementin pitää kuitenkin tietää lapsielementtiensä mitat. Elementin sijainnin laskeminen noudattaa usein seuraavaa kaavaa:

1. Isäntäelementti laskee oman leveytensä.
2. Isäntäelementti käy läpi sen mahdolliset lapsielementit ja
    a. asettaa lapsielementtien sijainnit (x- ja y-koordinaatit)
    b. pyytää lapsielementtejä laskemaan omat leveytensä.

3. Isäntäelementti laskee oman korkeutensa lapsielementtien mittasuhteiden perusteella.

Elementtien juoksuttaminen sivulle on raskas operaatio. Tämän vuoksi selainmoottoreihin on toteutettu optimointeja operaation nopeuttamiseksi. Ensimmäisen juoksutuksen jälkeen selainmoottorit pyrkivät laskemaan ainoastaan niiden elementtien sijainnit, joiden sijainti tai mitat ovat mahdollisesti muuttuneet. Esimerkiksi kun sivulle lisätään dynaamisesti sisältöä tai muokataan vain tiettyjen elementtien tyylimäärittelyitä, ei kaikille sivun elementeille tarvitse laskea uusia sijainteja. Tällaisissa tilanteissa selainmoottorit merkitsevät muuttuneet elementit ja niiden lapsielementit niin sanotusti likaisiksi, mikä tarkoittaa, että niiden sijainnit pitää laskea uudelleen. Gecko ja WebKit käyvät tietyin väliajoin näytettävien elementtien puuta läpi etsien mahdollisia likaisia elementtejä, ja niitä kohdatessaan selainmoottorit laskevat niille uudet sijainnit ja mitat. Joskus tämän operaation yhteydessä likaisten elementtien sijaintien ja mittojen päivitys aiheuttaa ketjureaktion, jossa myös päivitetyn elementin viereiset elementit tarvitsevat uudet sijainnit. [26.]

Sivun piirtäminen näytölle

Kun dokumentin elementeille on laskettu sijainnit ja mitat, voidaan sivu tulostaa näytölle. Selainmoottori käy näytettävien elementtien puun läpi ja tulostaa elementit näytölle käyttäen aiemmin laskettuja geometriatietoja [26]. CSS [28] määrittelee järjestyksen, jossa yksittäisen elementin osat tulostetaan:

1. taustaväri
2. taustakuva
3. reunus
4. lapsielementit
5. elementin ääriviivat.

3.3 Skriptikielet ja dynaaminen HTML

Useimmista selainmoottoreista löytyy tuki skriptikielille. Skriptikielten avulla voidaan muokata HTML-sivun sisältöä, tyylejä ja rakennetta dynaamisesti. Näitä operaatioita

kuvaa järjestelmästä ja ohjelmointikielestä riippumaton DOM-rajapinta. DOM-rajapinta määrittelee dokumenttien loogisen rakenteen ja muun muassa metodien nimet, joilla dokumentista voidaan valita tiettyjä elementtejä ja kysyä, mitä attribuutteja elementeillä on. Kumpikaan, HTML tai DOM, ei ota kantaa, millä ohjelmointikielellä tuki sivun muokkaamiseen on toteutettu. [29.]

Vaikka uusimmat määrittelyt pyrkivät olemaan mahdollisimman järjestelmä- ja kieliriippumattomia, on ECMAScriptiin pohjautuva JavaScript ylivoimaisesti suosituin Internetissä käytetty skriptikieli. Käytännössä jokaisesta nykyaikaisesta selaimesta löytyy tuki sille, ja se on ainut yleisessä käytössä oleva kieli. Arkkitehtuurisesti selainten tuki JavaScriptille on toteutettu erillisinä JavaScript-moottoreina, jotka on liitetty selainmoottoreihin [18, s. 6]. Vaikka eri selaimet käyttävät samoja selainmoottoreita, niissä voi olla eri JavaScript-moottorit. Esimerkiksi WebKitiä käyttävät Applen Safari ja Googlen Chrome käyttävät eri JavaScript-moottoreita. Taulukossa 2 (s. 10) on listattu eri selainmoottoreiden käyttämät JavaScript-moottorit.

Tyypillinen JavaScript-moottori muodostuu kolmesta eri komponentista: jäsentimestä, tulkista ja erillisestä moduulista, joka tarjoaa skriptien suorituksen aikana JavaScriptin käyttämät standardiluokat. Jäsennin pilkkoo suoritettavan JavaScript-koodin syntaksirakenteeksi, jonka tulkki suorittaa. [30.]

JavaScript-moottoreissa on usein mekanismi, jonka avulla ohjelmat, joihin JavaScript-moottorit on sisällytetty, voivat sitoa ja välittää omia olioita JavaScript-moottorin käytettäväksi. Selainmoottorit käyttävät tätä mekanismia jakamaan DOM-rakennepuun JavaScript-moottoreille. [30.]

JavaScript-koodia voidaan kutsua ja suorittaa selaimessa eri tavoilla [10]:

- sisällyttämällä koodia script-elementteihin
- hyödyntämällä JavaScript URL -notaatiota (JURL) esimerkiksi linkkien ja kuvien URL:ssa
- määrittelemällä tapahtumakuuntelijoita esimerkiksi käyttäjän suorittamille toiminnoille.

4 Selainpohjaiset ratkaisumallit

Selainpohjaisilla ratkaisumalleilla tarkoitetaan malleja, joilla voidaan laajentaa selainpohjaisten sovellusten mahdollisuutta käyttää käyttöjärjestelmän palvelurajapintoja. Käytännössä tämä tehdään toteuttamalla ohjelmallinen silta selaimen skriptiympäristön ja palvelurajapintojen välille.

4.1 Netscape Plugin API -selainlisäosat

Netscape Plugin API (NPAPI) on alun perin Netscapen kehittämä rajapinta, jonka avulla voidaan luoda lisäosia Internet-selaimiin. Nimestään huolimatta muutkin kuin Netscapen selaimet tukevat NPAPI-selainlisäosia. Suosituimmista selaimista ainoastaan Microsoftin Internet Explorer ja Applen mobiilikäyttöjärjestelmä iOS:n Safari eivät tue NPAPI:a. NPAPI on selainriippumaton teknologia, joten NPAPI:a tukevat selaimet osaavat käyttää samoja lisäosia. Eri käyttöjärjestelmien välillä on kuitenkin eroja NPAPI:n toteutuksessa liittyen siihen, miten selain alustaa ja käynnistää lisäosan. Tämän lisäksi jos lisäosa haluaa piirtää näytölle jotakin, sen toteutuksen pitää käyttää käyttöjärjestelmän tarjoamia piirtopalveluita, jotka voivat poiketa toisistaan paljonkin. [31.]

NPAPI-rajapinta on toteutettu C-kielellä, ja käytännössä suurin osa NPAPI-selainlisäosista on toteutettu C- ja C++-ohjelmointikielillä. NPAPI-selainlisäosat ovat konekieleksi käännettyjä jaettuja kirjastoja, jotka linkittyvät dynaamisesti selaimen ohjelmakoodiin (Windows-ympäristöissä Dynamic Linked Library (DLL) ja Unix-pohjaisissa ympäristöissä Shared Object (SO)) [32]. Koska NPAPI-selainlisäosia suoritetaan osana selainta, niillä on samat mahdollisuudet käyttää käyttöjärjestelmän palvelurajapintoja kuin itse selaimella.

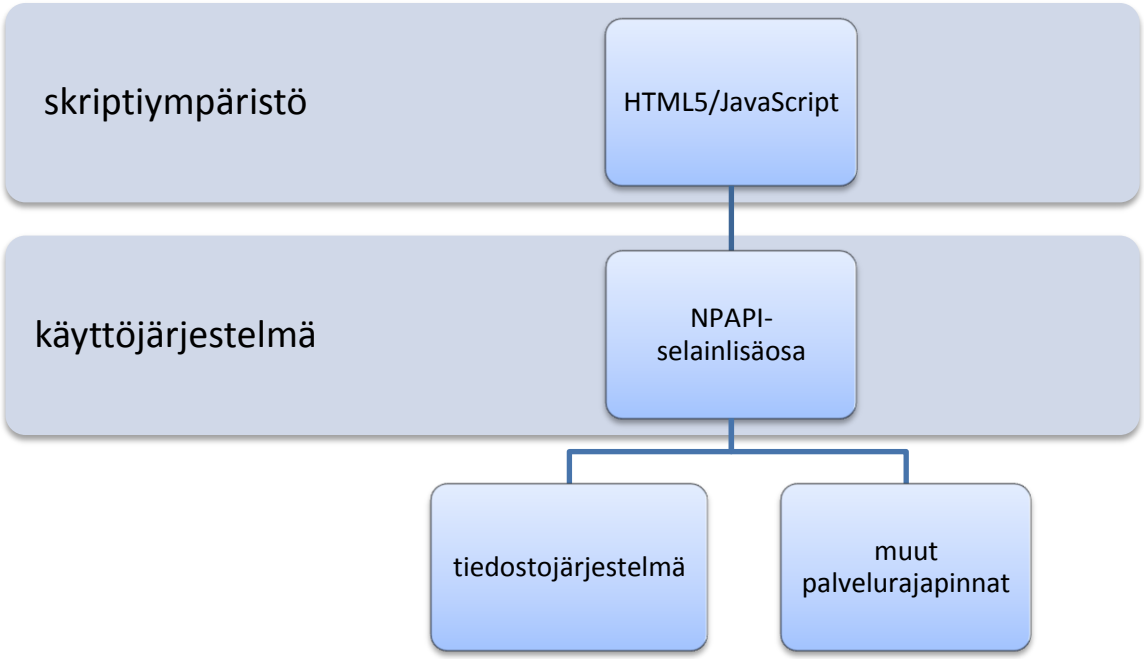
NPAPI-selainlisäosat kertovat selaimille Multipurpose Internet Mail Extensions -tyypin (MIME) avulla, minkälaista sisältöä ne käsittelevät. Mikäli selain kohtaa MIME-tyypin, jota se ei itse osaa käsitellä, se tarkistaa, tukeeko mikään sen mahdollisista lisäosista kyseistä tyyppiä. MIME-tyypit muodostuvat päätyypistä ja alityypistä ja ovat muodossa päätyyppi/alityyppi. Lisäosien käyttämä päätyyppi on application, mikä käytännössä tarkoittaa, että sisältö on binäärimuodossa. Alityypin määrittää lisäosan kehittäjä. MIME-tyypeistä pitää kirjaa Internet Engineering Task Force (IETF), johon kaikki uudet

MIME-tyypit tulisi rekisteröidä. Ennen kuin uusi MIME-tyyppi on rekisteröity, tulisi sen alityypissä käyttää ”x-”-etuliitettä. [31.]

NPAPI-selainlisäosia voidaan käyttää kahdella eri tavalla. Yleisempi tapa on HTML-dokumenttiin sisällytetty elementti, joka on osa sivua ja käyttäytyy kuin mikä tahansa muukin HTML-elementti. Elementin sisällön piirtäminen on kuitenkin selainlisäosan vastuulla. Tämän lisäksi selainlisäosat voivat toimia HTML-dokumentin ulkopuolella koko selainikkunan käyttävinä ohjelmina. Esimerkiksi Adobe Flash -selainlisäosan käyttämät SWF-tiedostot on usein upotettu HTML-dokumenttiin, kun taas Adobe Reader -selainlisäosa näyttää PDF-tiedostoja täysin irrallaan HTML-dokumentista. [31.]

HTML-dokumenttiin upotetut selainlisäosia käyttävät elementit lisätään selaimen toimesta muiden elementtien tavoin sivun DOM-rakenteeseen. Näin ollen skriptikielillä on DOM-rajapinnan avulla mahdollisuus käsitellä dokumentista löytyviä lisäosia käyttäviä elementtejä. Jotta tietoa pystyttäisiin välittämään skriptikielten ja selainlisäosien välillä, on NPAPI-rajapintaan tehty jälkikäteen npruntime-laajennus. Npruntimen avulla NPAPI-selainlisäosilla on mahdollisuus paljastaa niin sanottu skriptattava olio skriptikielille, jotka voivat kutsua skriptattavan olion metodeja. [33.]

Koska NPAPI-selainlisäosilla on mahdollisuus käyttää käyttöjärjestelmän palvelurajapintoja ja npruntimen avulla tietoa pystytään välittämään selaimessa suoritettavien skriptien ja selainlisäosien välillä, muodostuu kuvan 8 kaavion mukainen silta palvelurajapintojen ja skriptikielten välille.



Kuvio 8. NPAPI-selainlisäosaan pohjautuvan ratkaisumallin rakenne.

NPAPI-rajapinta muodostuu kahdesta eri ryhmästä funktioita, jotka on esitelty taulukossa 6. Näiden lisäksi NPAPI-rajapinta määrittelee useita selainlisäosille tarkoitettuja tietorakenteita. Näiden tietorakenteiden nimet alkavat NP-etuliitteellä.

Taulukko 6. NPAPI-rajapinnan funktioryhmät [32].

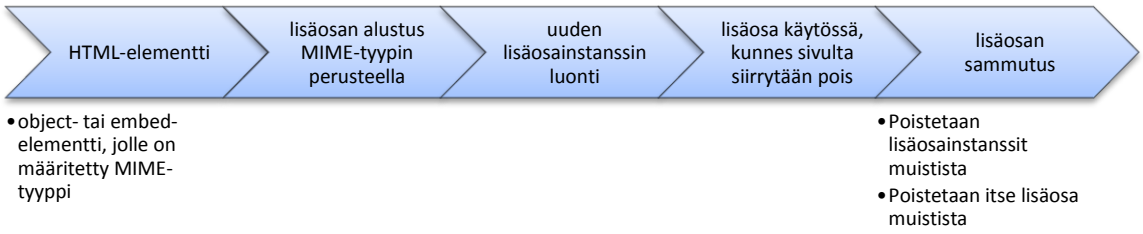
Funktioryhmä	**Kuvaus**
Netscape Plug-in: Plug-in Defined (NPP)	Lisäosan toteuttamat funktiot, joita selain kutsuu. Näiden funktioiden nimissä on NPP_-etuliite. Tähän ryhmään kuuluvat kuitenkin myös NP_-etuliitteiset niin sanotut liityntäfunktiot.
Netscape Plug-in: Navigator Defined (NPN)	Selaimen toteuttamat funktiot, joita lisäosa kutsuu. Näiden funktioiden nimissä on NPN_-etuliite.

NPAPI-rajapinta määrittelee muutaman liityntäfunktion, jotka lisäosan tulee toteuttaa ja joita selain kutsuu. Vaikka NPAPI onkin selainriippumaton standardi, on nämä liityntäfunktiot määritelty eri tavoin eri käyttöjärjestelmissä. [31.]

Npruntime-laajennus lisää NPAPI-rajapintaan määrittelyt funktioista, jotka skriptattavan olion tulee toteuttaa. Toteuttamalla nämä funktiot lisäosa voi paljastaa haluamansa metodit ja kentät selaimen skriptiympäristölle. Npruntime tuo myös lisäosille mahdollisuuden kutsua ja suorittaa selaimen skriptiympäristön skriptejä. [33.]

NPAPI-selainlisäosan elinkaari

NPAPI-selainlisäosia tukeva Internet-selain käy käynnistyessään läpi kaikki saatavilla olevat lisäosat ja rekisteröi ne niiden määrittelemää MIME-tyyppiä vasten. Saatavilla olevien lisäosien tunnistaminen vaihtelee käyttöliittymästä riippuen. [31.] Kuvassa 9 on esitetty NPAPI-selainlisäosainstanssin elinkaari.



Kuvio 9. NPAPI-selainlisäosan elinkaari.

NPAPI-selainlisäosa upotetaan HTML-dokumenttiin joko object- tai embed-elementillä. Object-elementin käyttöä suositellaan, koska se tukee vaihtoehtoisen sisällön näyttämistä, mikäli tarvittavaa selainlisäosaa ei ole käytettävissä. Esimerkkikoodi 2 esittää object-elementin käyttöä.

```
<object id="esimerkki" type="application/x-esim-plugin">
     Ei tarvittavaa lisäosaa
</object>
```

Esimerkkikoodi 2. Object-elementti.

Elementille voidaan määrittää id-attribuutilla tunniste, jolla sitä voidaan hakea DOM-rakenteesta. Type-attribuutti ilmoittaa selaimelle käytettävän lisäosan MIME-tyypin, jonka perusteella selain osaa ottaa käyttöön oikean lisäosan [31]. Elementin sisältöteksti näytetään, jos selaimeen ei ole asennettu MIME-tyyppiä vastaavaa lisäosaa tai jos selain ei muusta syystä voi käyttää lisäosaa [10]. Type-attribuutti ei ole kuitenkaan pakollinen. Jos selainlisäosa käyttää jotakin ulkoista resurssitiedostoa, se voidaan määritellä data-attribuutilla. Tässä tilanteessa selain saa käytettävän MIME-tyypin palvelimelta ladatessaan resurssitiedostoa [32]. Samalla periaatteella toimivat myös HTML-dokumentin ulkopuolella toimivat lisäosat. Kun käyttäjä pyytää selainta noutamaan jonkin lisäosan käyttämän resurssitiedoston, käynnistää selain tarvittavan lisäosan palvelimelta saadun MIME-tyypin perusteella.

Selain käynnistää ja alustaa lisäosan siihen tarkoitetulla NPAPI-rajapinnan määrittelemällä liityntäfunktiolla. Tämän alustusfunktion mukana selain antaa kaksi rajapintaosoitinta. Rajapintaosoitin on osoitin, joka osoittaa toiseen osoittimeen. Käytännössä nämä kaksi rajapintaosoitinta osoittavat kahteen eri tietorakenteeseen, jotka sisältävät osoittimia taulukossa 6 kuvattuihin funktioihin. Tässä vaiheessa lisäosan tulee ottaa talteen toinen osoittimista, joka viittaa selaimen toteuttamiin funktioihin, jotta se voi kutsua niitä tarvittaessa. Tämän lisäksi lisäosan pitää täyttää toisen osoittimen viittaama tietorakenne, joka sisältää NPP-ryhmän funktiot. Lisäosa täyttää tietorakenteeseen sen toteuttamat NPAPI-rajapinnan mukaiset funktiot, minkä jälkeen selaimella on niiden muistiosoitteet ja se voi kutsua niitä. [34.]

Kun lisäosa on alustettu ja ladattu käyttömuistiin, voi selain luoda siitä yksittäisiä instansseja. Yhdellä sivulla voi olla useampia samaa lisäosaa käyttäviä elementtejä. Kuitenkin ainoastaan ensimmäisen kohdalla suoritetaan lisäosan alustus. Jälkimmäisten kohdalla voidaan suoraan luoda uusi instanssi lisäosasta. Uuden instanssin luomiseen käytetään siihen tarkoitettua NPP_New-funktiota, johon saatiin selaimelta viite alustusvaiheessa [31].

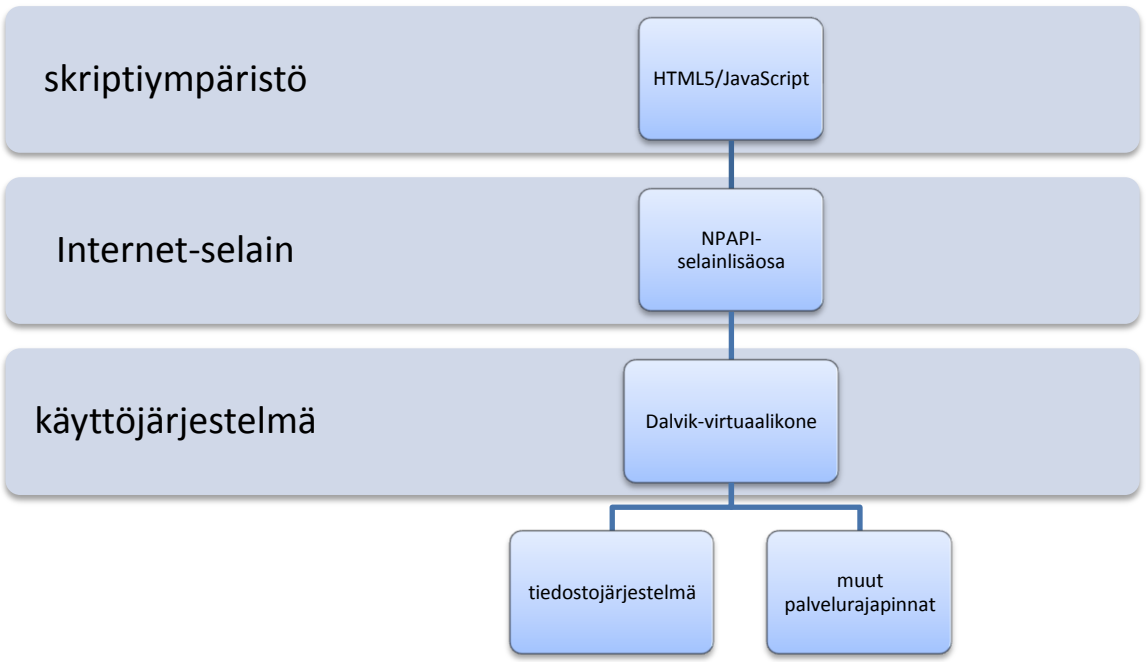
Instanssin luonnin jälkeen lisäosa on valmiina käytettäväksi. Se pystyy halutessaan piirtämään grafiikkaa sille määritellyn elementin alueelle, tai se voi pyytää selainta suorittamaan erilaisia toimintoja [31]. Mikäli instanssi on paljastanut skriptattavan olion, voivat selaimen skriptikielet kutsua sen metodeja [33].

Kun käyttäjä poistuu sivulta, jolla lisäosia käytettiin, tuhotaan jokainen lisäosan instanssi. Tuhottuaan kaikki instanssit selain kutsuu lisäosan liityntäfunktiota, jossa voidaan vapauttaa muistivaraukset, jotka olivat kaikkien instanssien käytössä. [31.]

NPAPI ja Android-käyttöjärjestelmän palvelurajapinnat

Suurin osa Android-käyttöjärjestelmälle tehdyistä sovelluksista on toteutettu Java-ohjelmointikielellä. Valmis ohjelma käännetään tavukoodiksi, jota Androidin käyttämä Dalvik-virtuaalikone osaa tulkita. Nykyään Androidille on mahdollista toteuttaa myös niin sanottuja natiiveja sovelluksia, jotka käännetään konekieleksi ja suoritetaan Dalvik-virtuaalikoneen ulkopuolella. Natiiveiksi sovelluksiksi kutsutaan sovelluksia, jotka käännetään jonkin tietyn prosessoriarkkitehtuurin konekieleksi. Tätä kirjoitettaessa kaikki Androidin tarjoamista palvelurajapinnoista eivät kuitenkaan ole natiiviympäristön käytettävissä.

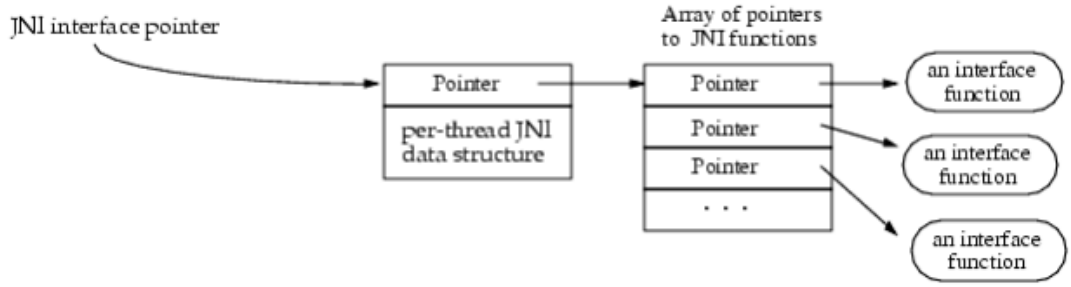
NPAPI-selainlisäosia suoritetaan natiiviympäristössä. Näin ollen käyttääkseen kaikkia Androidin palvelurajapintoja ne tarvitsevat yhden ylimääräisen kerroksen natiiviympäristön ja Dalvik-virtuaalikoneen välille. Tämä on otettu huomioon Androidin NPAPI-rajapinnan toteutuksessa. Kun selain luo uuden instanssin selainlisäosasta, se antaa argumenttina referenssin Dalvik-virtuaalikoneen suoritusympäristöstä. Tämän referenssin avulla NPAPI-selainlisäosat voivat halutessaan käynnistää Java-ohjelmia Dalvik-virtuaalikoneeseen ja kutsua niiden metodeja. Ohjelmien käynnistämiseen Dalvik-virtuaalikoneessa käytetään Java Native Interface -ohjelmistokehystä (JNI). Kuvan 10 kaavio määrittelee, kuinka Android-ympäristössä selainten skriptikielillä voidaan käyttää käyttöjärjestelmän palvelurajapintoja NPAPI-selainlisäosan avulla. [35.]



Kuvio 10. Kaavio NPAPI-ratkaisun rakenteesta Androidilla.

JNI on natiivin ympäristön ohjelmistokehys, ja se mahdollistaa yhteistoiminnan Java-yhteensopivassa virtuaalikoneessa suoritettavien Java-ohjelmien ja muilla ohjelmointikielillä toteutettujen ohjelmien ja kirjastojen välillä. Usein JNI:a käytetään, kun virtuaalikoneen rajoitteet estävät halutun ohjelmiston toiminnallisuuden toteutuksen. Androidilla tämä on kuitenkin toisin päin, koska sen tarjoamat palvelurajapinnat ovat kattavammat Dalvik-virtuaalikoneessa suoritettaville ohjelmille. [36.]

Natiivi koodi käyttää virtuaalikoneen ominaisuuksia JNI-funktioiden avulla. JNI-funktioita kutsutaan rajapintaosoittimien läpi. Kuvassa 11 on esitelty rajapintaosoittimen rakenne. JNI-osoitin osoittaa taulukkoon, jossa on osoittimia. Taulukon osoittimet puolestaan osoittavat rajapintafunktioihin. [36.]



Kuvio 11. JNI-rajapintaosoitin [36].

Javan System-luokassa on loadLibrary-metodi, jolla voidaan ladata natiiveja kirjastoja. Natiiveista kirjastoista löytyvät metodit esitellään Java-luokissa native-avainsanalla. Tämän jälkeen niitä voidaan kutsua kuin mitä tahansa muutakin Java-metodia, mutta sen suoritus tapahtuu natiivissa ympäristössä. Java-koodissa natiivina esitelty metodi ja itse natiiviympäristön toteutus kyseiselle metodille linkitetään niiden nimien perusteella. Natiiviympäristön metodin nimi muodostuu seuraavien komponenttien yhdistelmästä:

- Java_-etuliite
- Java-luokan nimi mukaan lukien paketti, johon se kuuluu
- alaviiva
- metodin nimi. [36.]

Esimerkiksi esimerkkikoodin 3 Java-luokan metodi sampleMethod on natiiviympäristössä muodossa Java_pkg_Cls_sampleMethod.

```
package pkg;

class Cls {
    native void sampleMethod();
    void anotherMethod();
}
```

Esimerkkikoodi 3. Java-luokka, joka esittelee natiiviympäristön metodin.

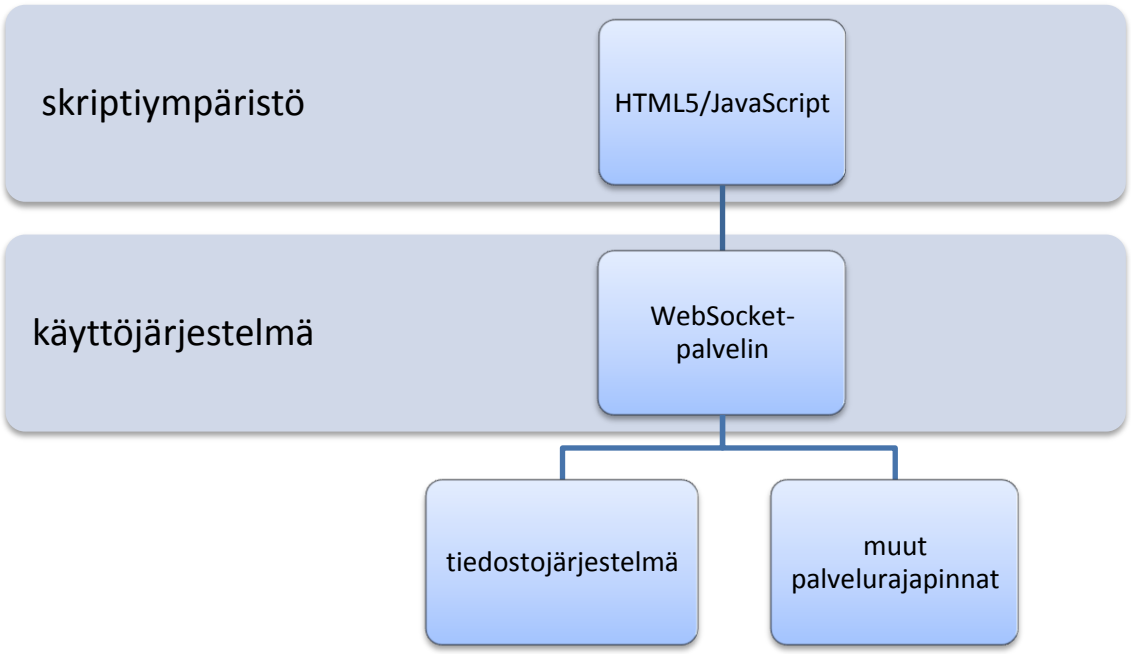
JNI tarjoaa monia eri metodeja Java-metodien kutsumiseen natiivipuolelta. JNI tunnistaa Java-metodit niiden symbolisten nimien ja allekirjoitusten perusteella. Java-metodien kutsu natiivipuolelta on aina kaksivaiheinen operaatio. Ensiksi pitää hakea Java-metodin jmethodID-tietotyyppinen tunniste. Tämän jälkeen metodia voidaan kutsua haetun tunnisteen avulla useita kertoja, ilman että metodi pitäisi etsiä uudelleen. [36.]

4.2 WebSocket-palvelin

WebSocket on HTML5:n määrittelemä teknologia, jonka avulla voidaan muodostaa kaksisuuntaisia yhteyksiä palvelimen ja asiakasohjelman välillä. Yleisemmin Web-

Socket-teknologiaa käsiteltiin luvussa 2.4. Koska WebSocket-yhteys on kaksisuuntainen ja reaaliaikainen, voidaan sen avulla välittää nopeasti ja tehokkaasti tietoa sitä tukevien palvelinten ja Internet-selaimien välillä.

Hyvien tiedonsiirto-ominaisuuksien ansiosta WebSocket-teknologialla voidaan tarjota palvelurajapintoja selaimen käytettäviksi. Käytännössä tämä saadaan aikaiseksi toteuttamalla WebSocket-palvelin käyttöjärjestelmälle, jonka palvelurajapintoja halutaan antaa verkkosovellusten saataville. Koska WebSocket-palvelin on tavallinen sovellus, joka toimii samassa ympäristössä kuin muutkin sovellukset, sillä on mahdollisuus käyttää käyttöjärjestelmän tarjoamia palvelurajapintoja ja jakaa niitä verkkosovellukselle. WebSocket-palvelinta suoritetaan taustalla, ja Internet-selaimessa toimivat verkkosovellukset voivat muodostaa yhteyden siihen. Yhteyden muodostamisen jälkeen palvelin ja verkkosovellus voivat vaihtaa tietoja reaaliajassa kuvan 12 kaavion mukaisesti.



Kuvio 12. WebSocket-ratkaisumallin rakenne.

WebSocket-teknologiaan kuuluu sen oma WebSocket-protokolla, joka määrittelee säännöt teknologiaa käyttävien palvelinten ja asiakasohjelmien välille. Protokolla muodostuu kahdesta osasta. Ensimmäinen osa on niin kutsuttu kättely, jonka tarkoitus on muodostaa yhteys turvallisesti palvelimen ja asiakasohjelman välille. WebSocket-

protokollan kättely on toteutettu laajentamalla HTTP-protokollaa, jotta samaa porttia voidaan käyttää kuuntelemaan niin HTTP- kuin WebSocket-pyyntöjä. Tämä voi johtaa tilanteisiin, joissa tavallinen HTTP-kutsu voi sekoittua WebSocket-kutsun kanssa. WebSocket-protokollan kättelyssä on mekanismi, jonka avulla palvelin ja asiakasohjelma voivat varmistua kyseessä olevan WebSocket-yhteys. Handshake on käytännössä tavallinen HTTP Upgrade -kutsu, johon on lisätty muutama otsaketieto. [16.]

Esimerkki asiakasohjelman lähettämästä kättelyotsakkeesta työn tekovaiheelta [37]:

```
GET /demo HTTP/1.1
Host: example.com
Connection: Upgrade
Sec-WebSocket-Protocol: sample
Upgrade: WebSocket
Sec-WebSocket-Key1: 18x 6]8vM;54 *(5:  {   U1]8  z [  8
Sec-WebSocket-Key2: 1_ tx7X d  <  nw  334J702) 7]o}` 0
Origin: http://example.com

Tm[K T2u
```

HTTP-protokollan tavoin otsakkeen kentät voivat olla missä järjestyksessä tahansa. Asiakasohjelman lähettämä pyyntö sisältää niin sanotun haasteen, johon palvelimen tulee vastata oikein todistaakseen vastaanottaneensa WebSocket-pyynnön. Haaste muodostuu kolmesta eri tiedosta, jotka löytyvät otsakkeesta: Sec-WebSocket-Key1- ja Sec-WebSocket-Key2-kentistä ja otsakkeen viimeisistä kahdeksasta tavusta. Kaikki kolme tietoa ovat näennäisesti satunnaisia merkkijonoja, joiden perusteella palvelimen tulee muodostaa vastaus asiakasohjelman haasteeseen. Palvelin muodostaa vastauksen seuraavalla prosessilla:

1. Palvelimen tulee poimia molemmista Sec-WebSocket-Key-kentistä numerot ja muodostaa niistä luvut. Esimerkin otsakkeesta muodostuisi luvut 1868545188 ja 1733470270.
2. Palvelimen tulee laskea molempien Sec-WebSocket-Key-kenttien sisältämät välilyönnit. Esimerkissä välilyöntejä on 12 ja 10.
3. Palvelimen tulee jakaa kohdassa 1 saadut numerot kohdassa 2 saaduilla numeroilla. Lopputulokseksi esimerkissä tulevat numerot 155712099 ja 173347027.

4. Palvelimen pitää yhdistää kohdassa 3 saadut luvut ja jatkaa saatua yhdistelmää asiakasohjelman lähettämän otsakkeen viimeisillä 8 tavulla muodostamalla niistä 16-tavuisen merkkijonon.
5. Kohdassa 4 muodostetun merkkijonon perusteella muodostetaan MD5-summa, joka on vastaus asiakasohjelman lähettämään haasteeseen. [37.]

Kokonaisuudessaan palvelimen vastaus asiakasohjelman lähettämään on seuraava:

```
HTTP/1.1 101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Origin: http://example.com
Sec-WebSocket-Location: ws://example.com/demo
Sec-WebSocket-Protocol: sample

fQJ,fN/4F4!~K~MH
```

Palvelimen lähettämän otsakkeen viimeiset 16 tavua ovat vastaus asiakasohjelman lähettämään haasteeseen. Mikäli vastaus ei täsmää haasteen kanssa, asiakasohjelman tulee katkaista yhteys. [37.]

Työtä kirjoitettaessa WebSocket-protokollan kättelyyn tehtiin muutoksia, eikä yllä kuvattu prosessi haasteesta ole enää voimassa. Uuden protokollan haasteessa asiakasohjelma lähettää ainoastaan yhden Sec-WebSocket-Key-kentän kättelyotsakkeen mukana. Kentän arvo on base64-koodattu satunnainen merkkijono. Palvelimen pitää purkaa base64-koodaus ja yhdistää purettuun merkkijonoon protokollan ennalta määrittelemä Globally Unique Identifier (GUID) "258EAFA5-E914-47DA-95CA-C5AB0DC85B11". Yhdistetystä merkkijonosta otetaan SHA-1-tiiviste, joka palautetaan asiakasohjelmalle base64-koodattuna. [16.]

Esimerkki uudemman standardin mukaisesta asiakasohjelman lähettämästä kättelyotsakkeesta [16]:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Kun yhteys on saatu muodostettua, voivat palvelin ja asiakasohjelma alkaa välittää tietoa keskenään. Työn tekovaiheessa WebSocket-protokolla tuki ainoastaan UTF-8-tekstimuotoisten viestien välitystä. Jokainen viesti alkaa 0x00-tavulla ja loppuu 0xFF-tavuun. Alku- ja lopputavujen välinen osa muodostaa viestin sisällön. Yhteyden sulkemiseksi puhtaasti käytetään tyhjää viestiä, joka muodostuu ainoastaan 0x00- ja 0xFF-tavusta. [37.]

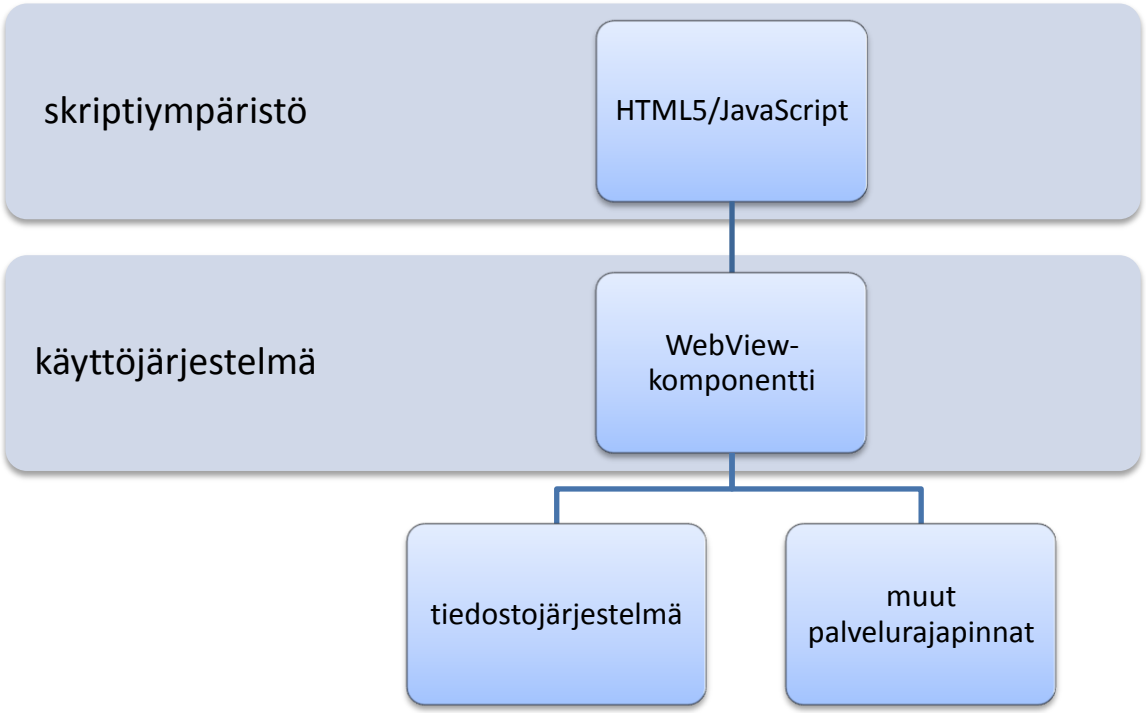
WebSocket API -rajapinta määrittelee, miten selainpohjaiset sovellukset voivat yhdistää ja kommunikoida WebSocket-palvelimen kanssa. Yhteys luodaan luomalla uusi `WebSocket`-olio, jolle annetaan argumenttina halutun WebSocket-palvelimen osoite. Mikäli yhteys saatiin muodostettua onnistuneesti, kutsutaan olion `onopen`-metodia. Tämän jälkeen palvelimelle voidaan lähettää viestejä `WebSocket`-olion `send`-metodilla. Palvelimelta tulevat viestit laukaisevat `WebSocket`-olion `onmessage`-tapahtuman, jolle annetaan argumenttina saapunut viesti. [38.]

4.3 WebView-komponentti

WebView on ohjelmistokomponentti, jonka avulla voidaan hakea ja näyttää verkkosivuja helposti tavallisen ohjelman näyttöalueella. Käytännössä se on selainmoottori upotettuna ohjelmistokomponenttiin. Se ei tarjoa ohjelman käyttäjälle tavanomaisen Internet-selaimen ominaisuuksia, kuten mahdollisuutta syöttää sille osoite, joka sen tulisi hakea. WebView-komponenttiin ladatun verkkosivun linkkien avulla käyttäjä voi kuitenkin navigoida eteenpäin. [39.]

WebView-komponentti kuuluu lähes jokaisen suositun käyttöjärjestelmän kehitystyökalujen tarjoamiin valmiisiin ohjelmistokomponentteihin. Se kuitenkin voi olla nimetty eri nimillä eri ympäristöissä. Esimerkiksi Androidilla se on WebView, iOS:llä UIWebView ja Maemolla QWebView.

WebView-komponenteilla on usein ominaisuus, jonka avulla on mahdollista altistaa natiiviympäristön olio WebView-komponentissa suoritettavan skriptiympäristön käyttöön. Tämä olio ja sille toteutetut metodit toimivat rajapintana natiivi- ja selainympäristöjen välillä. Kuvassa 13 on esitetty WebView-ratkaisumallin rakenne.



Kuvio 13. WebView-ratkaisumallin rakenne.

Tämä ratkaisumalli poikkeaa kuitenkin toisista ratkaisumalleista, koska selainpohjaisia sovelluksia ei suoriteta tavallisessa Internet-selaimessa vaan erillisessä ohjelmassa.

Tätä ratkaisumallia hyödyntäviä ohjelmistokehyksiä on jo olemassa muutamia, esimerkiksi PhoneGap, Apparat.io ja Sencha v2. Ne kaikki osaavat tuottaa selainpohjaisista sovelluksista natiivisovelluksia WebView-komponettien avulla. [4, s. 23.]

5 Selainpohjaisten sovellusten palvelurajapintojen lisääminen

5.1 Lähestymistapojen suunnittelu

Insinöörityön kohdeympäristöt ovat Maemo ja Android, jotka ovat molemmat Linux-pohjaisia käyttöjärjestelmiä. Maemolle saatavissa olevassa Firefox-selaimessa käytetään Gecko-selainmoottoria, ja Android Browserin selainmoottori pohjautuu WebKit-selainmoottoriin. Valitsemalla eri selainmoottorit pyrittiin selvittämään lähestymistapojen toimivuutta erilaisissa ympäristöissä.

Vaadittavaksi toiminnallisuudeksi valittiin yksi käyttöjärjestelmien peruspalvelurajapinnoista eli mahdollisuus selata tiedostojärjestelmää. Kaikki palvelurajapinnat sijaitsevat

arkkitehtuurisesti samassa kerroksessa. Näin ollen mikäli Internetissä sijaitsevan selainpohjaisen sovelluksen avulla voidaan selata mobiililaitteen tiedostojärjestelmää, voidaan olettaa muidenkin palvelurajapintojen olevan käytettävissä.

Muita mahdollisia palvelurajapintoja, jotka eivät vielä ole selainpohjaisten sovellusten käytettävissä, on koottuna taulukossa 7.

Taulukko 7. Palvelurajapintoja, joita ei voida käyttää selainpohjaisilla sovelluksilla.

Palvelurajapinta	**Kuvaus**
Kalenteri	Mahdollistaa laitteen kalenterin käsittelyn.
Kamera	Antaa mahdollisuuden laitteen kameran käyttämisen kuvien ottamiseen.
Osoitekirja	Mahdollistaa laitteen osoitekirjan sisältämien yhteystietojen käsittelyn.
Viestintä	Mahdollistaa viestien lähetyksen.
Anturit	Antaa mahdollisuuden kuunnella laitteen eri antureita (esimerkiksi kiihtyvyysanturit ja kompassi).
Puhelin	Mahdollistaa puheluiden muodostamisen.

5.2 Netscape Plugin API -selainlisäosan toteutus

Projektissa toteutettiin NPAPI-selainlisäosa ja sitä hyödyntävä JavaScript-ohjelma, joiden avulla Internetissä sijaitsevalla verkkosivulla voidaan selata sitä käyttävän mobiililaitteen tiedostojärjestelmää. Toteutukseen lisättiin myös toiminto, jonka avulla verkkosivulla voidaan näyttää mobiililaitteen tiedostojärjestelmässä sijaitsevia kuvia. Lisäosa toteutettiin Androidille ja Maemolle, koska niissä on eri selainmoottorit. Tällä haluttiin varmistaa NPAPI-lähestymistavan yleistettävyys.

NPAPI-rajapinta määrittelee muutaman niin kutsutun liityntäfunktion, jotka lisäosan tulee toteuttaa ja joita selain kutsuu. Taulukossa 8 on esitetty NPAPI-selainlisäosien liityntäfunktiot ja niiden käyttötarkoitukset.

Taulukko 8. NPAPI:n liityntäfunktiot [31].

Liityntäfunktio	**Kuvaus**
`NP_Initialize`	Kutsutaan, kun selain kohtaa sivua ladatessaan ensimmäistä kertaa kyseisen lisäosatyypin. Käytetään varaamaan muistia, jonka kaikki lisäosan instanssit jakavat.
`NP_GetValue`	Selain kutsuu tätä funktiota kysyäkseen lisäosan nimen ja kuvauksen.
`NP_GetMIMEDescription`	Kutsutaan selaimen käynnistyessä selvittämään ja rekisteröimään lisäosan tukemat MIME-tyypit.
`NP_Shutdown`	Kun jokainen instanssi lisäosasta on tuhottu, selain kutsuu tätä funktiota. Käytetään vapauttamaan muisti, joka varattiin lisäosan suorituksen aikana.

Koska Android ja Maemo ovat molemmat Linux-pohjaisia, niiden NPAPI-määrittelyt ovat hyvin lähellä toisiaan. Ainoastaan yksi liityntäfunktio poikkeaa Androidilla toisista. `NP_Initialize` saa Androidilla yhden parametrin enemmän. Ylimääräinen parametri on osoitin Dalvik-virtuaalikoneen ajoympäristöön. Esimerkkikoodeissa 4 ja 5 ovat eri käyttöjärjestelmien `NP_Initialize`-funktion allekirjoitukset.

```
NPError NP_Initialize(NPNetscapeFuncs *aNPNFuncs,
    NPPluginFuncs *aNPPFuncs)
```

Esimerkkikoodi 4. Standardi Linux-pohjaisten käyttöjärjestelmien NP_Initialize.

```
NPError NP_Initialize(NPNetscapeFuncs *aNPNFuncs,
    NPPluginFuncs *aNPPFuncs, void *java_env)
```

Esimerkkikoodi 5. Androidin käyttämä NP_Initialize.

`NP_Initialize`-liityntäfunktion argumentteina saamat osoittimet on esitelty luvussa 4.1. `NPPluginFuncs`-tietorakenne sisältää NPAPI-rajapinnan funktiot, jotka lisäosan pitää toteuttaa. Tässä projektissa käytettiin kuitenkin vain muutamaa näistä funktioista. Merkittävimmät näistä on esitelty taulukossa 9.

Taulukko 9. Merkittävimmät NPP-funktiot [31].

Funktio	**Kuvaus**
`NPP_New`	Selain kutsuu tätä luodakseen uuden instanssin lisäosasta.
`NPP_Destroy`	Käytetään lisäosainstanssin tuhoamiseen ja sen muistivarausten vapauttamiseen.
`NPP_GetValue`	Selain kysyy tällä funktiolla tietoja lisäosalta. Tässä projektissa funktiota käytetään ainoastaan skriptattavan olion paljastamiseen.

Jokaiselle lisäosainstanssille luodaan uniikki tunniste, joka annetaan `NPP_New`-funktion kutsun argumenttina. Tunnisteen avulla samalla sivulla olevat saman lisäosan instanssit voidaan erottaa toisistaan. Tunniste on `NPP`-tietue, joka muodostuu kahdesta kentästä:

- `pdata`, kenttä lisäosainstanssin tiedoille
- `ndata`, kenttä selaimen lisäosainstanssikohtaisille tiedoille.

`NPP_New`-funktion toteutuksessa pitää testata, tukeeko selain npruntime-laajennusta NPAPI-rajapintaan. Tuki skriptattaville oliolle tuli NPAPI:n versiossa 14. Selaimen käyttämää NPAPI-versiota voidaan kysyä selaimelta `NPN_Version`-funktiolla. Mikäli selain tukee skriptattavia olioita, voidaan olio luoda ja asettaa se `NPP`-tietueen `pdata`-kenttään, jolloin luotu skriptattava olio linkittyy oikeaan lisäosainstanssiin.

Esimerkkikoodissa 6 luodaan skriptattava olio käyttäen selaimen `NPN_CreateObject`-funktiota. Parametriksi annettu `npClassRef`-osoitin osoittaa `NPClass`-tietorakenteeseen, joka koostuu npruntimen-mukaisista funktio-osoittimista.

```
NPError NPP_New(NPMIMEType    pluginType,
                NPP instance, uint16 mode,
                int16 argc,   char *argn[],
                char *argv[], NPSavedData *saved) {
   if (browser->version >= 14) {
       instance->pdata = browser->createobject(instance,
           &npClassRef);
   }
}
```

Esimerkkikoodi 6. Skriptattavan olion luonti NPP_New-funktiossa.

Taulukkoon 10 on koottu `NPClass`-tietueen funktiot, joita käytetään tässä projektissa.

Taulukko 10. Projektissa käytetyt NPClass-tietueen funktiot [31].

Funktio	**Kuvaus**
`Allocate`	Käytetään uuden skriptattavan olion alustamiseen.
`Deallocate`	Käytetään skriptattavan olion tuhoamiseen.
`Invoke`	Kutsutaan skriptattavan olion metodia.

Käytännössä kun selaimen `NPN_CreateObject`-funktiota kutsutaan, selain kutsuu `NPClass`-tietueen osoittamaa `Allocate`-funktiota, joka puolestaan palauttaa selaimelle osoittimen `NPObject`-tietueeseen. `NPObject`-tietuetta käytetään rajapintana, jonka avulla selaimet ja lisäosat välittävät olioita keskenään.

Tässä projektissa tehtiin oma `PluginObject`-tietue, jonka ensimmäinen elementti on `NPObject`-tietue ja toinen itse skriptattava olio. C-kielessä tietueen ja sen ensimmäisen muuttujan osoittimet osoittavat samaan muistipaikkaan [40, s. 103]. Näin ollen kun selain antaa lisäosalle osoittimen `NPObject`-tietueeseen, voidaan sen perustella hakea muistista oikea `PluginObject`-tietue, joka sisältää halutun skriptattavan olion.

Kun skriptattava olio on alustettu, sitä voidaan käyttää selaimen skriptiympäristöstä. Esimerkkikoodi 7 näyttää, miten JavaScriptillä saadaan referenssi lisäosainstanssista, joka on upotettu HTML-dokumenttiin object-elementtinä.

```
<object type="application/x-testbrowserplugin" height="1"
width="1" id="testPlugin"></object>

<script>
    var obj = document.getElementById("testPlugin");
</script>
```

Esimerkkikoodi 7. Skriptattavan olion referenssin haku JavaScriptillä hyödyntäen DOM-rajapintaa.

Kun skriptiympäristöllä on referenssi lisäosaan, voidaan lisäosan metodeja kutsua kuten minkä tahansa muun olion metodeja esimerkkikoodin 8 tavoin.

```
obj.getDirectory(”/”);
```

Esimerkkikoodi 8. Skriptattavan olion metodin kutsuminen JavaScriptillä.

Skriptiympäristöstä tehdyt metodikutsut saavat selaimen kutsumaan lisäosan `Invoke`-funktiota. `Invoke`-funktiolle välitetään kutsuttava olio, kutsuttavan metodin nimi, alkuperäisen kutsun argumentit ja osoitin `NPVariant`-tietueeseen, johon täytetään metodin palautusarvo. `Invoke`-funktion rakenne on kuvattu esimerkikoodissa 9.

```
static bool Invoke(NPObject* obj, NPIdentifier methodName,
                   const NPVariant *args, uint32_t argCount,
                   NPVariant *result) {
    char *name = browser->utf8fromidentifier(methodName);
    bool rval = false;
    if (name && !strcmp((const char *)name, "getDirectory")) {
        // suoritetaan getDirectory-metodi
    } else if (...) {
       // muita metodeja
    }
    browser->memfree(name);
    return rval;
}
```

Esimerkkikoodi 9. Kutsuttavan metodin selvittäminen Invoke-funktiossa.

Metodin nimi annetaan `NPIdentifier`-tietotyypin muodossa, joka voidaan muuntaa merkkijonoksi käyttämällä selaimen `NPN_UTF8FromIdentifier`-funktiota. Metodin nimen perusteella voidaan suorittaa haluttu tehtävä. Tehtävän suorittamisesta vastaa skriptattava olio.

Skriptattavan olion mallintamiseen toteutettiin `BackgroundObject`-luokka. Luokan rakenne ja otsaketiedosto ovat samanlaiset Android- ja Maemo-lisäosilla. Luokan toteutukset kuitenkin poikkeavat toisistaan niiltä osin, joissa käytetään käyttöjärjestelmän palvelurajapintoja. Android-toteutuksessa kutsut palvelurajapinnoille välitetään JNI:n kautta Java-luokalle, joka käsittelee palvelurajapintakutsut ja palauttaa niiden vastaukset takaisin skriptattavalle oliolle. Tähän tarkoitukseen toteutettiin `ServiceApiProxy`-luokka. Maemolla voidaan käyttää suoraan Qt:n tarjoamia palvelurajapintoja.

BackgroundObject-luokalle toteutettiin metodi, joka palauttaa halutun hakemiston sisällöt taulukkona selaimen skriptiympäristölle. Metodille annetaan argumenttina osoitin NPVariant-tietueeseen, johon hakemiston sisällöt täytetään. Palautettava arvo on taulukko, jota skriptiympäristön pitää osata käsitellä. Näin ollen palautettava taulukko pitää luoda selaimen avulla. Esimerkkikoodissa 10 luodaan selaimen avulla tyhjä taulukko.

```
NPVariant cbVar;
NPIdentifier arrayIdenfifier = browser->
    getstringidentifier("Array");
browser->invoke(instance, windowObject, arrayIdentifier, NULL, 0,
    &cbVar);
NPObject* arrayObject = NPVARIANT_TO_OBJECT(cbVar);
```

Esimerkkikoodi 10. Tyhjän taulukon luonti selaimen avulla.

Selainta voidaan pyytää NPN_Invoke-funktion avulla suorittamaan skriptiympäristön window-olion metodeita. Taulukon luomiseksi kutsutaan Array-metodia, joka palauttaa tyhjän taulukon. Kun tyhjä taulukko on luotu, siihen voidaan lisätä alkioina halutun hakemiston sisällöt. Tämä toteutettiin muuntamalla hakemiston sisällöt NPVariant-tietueiksi. Tämän jälkeen voidaan selaimen NPN_Invoke-funktiolla kutsua aiemmin luodun taulukko-olion push-metodia lisäämään siihen NPVariant-taulukon sisällöt esimerkkikoodin 11 tavoin.

```
NPIdentifier pushIdentifier = browser->
    getstringidentifier("push");
browser->invoke(instance, arrayObject, pushIdentifier, args,
    sizeof(args)/sizeof(args[0]), &cbVar);
```

Esimerkkikoodi 11. Alkioiden lisäys taulukkoon käyttäen push-metodia.

Kun hakemiston sisällöt on lisätty taulukkoon, annetaan taulukon sisällöt skriptiympäristölle.

Selainlisäosaan toteutettiin ainoastaan mahdollisuus kysyä yksittäisen hakemiston sisältöjä. Varsinaista tiedostojärjestelmän selailua varten toteutettiin JavaScriptillä selainpohjainen sovellus, joka kuuntelee käyttäjän komentoja ja näyttää käyttäjän määrittämän hakemiston sisällöt verkkosivulla hyödyntäen toteutettua selainlisäosaa.

Sovellus tulostaa käyttäjälle listan valitun hakemiston sisältämistä hakemistoista ja tiedostoista ja muodostaa niistä linkit, joilla voi siirtyä syvemmälle tiedostojärjestelmään. Mikäli listattava tiedosto on kuva, sovellus antaa käyttäjälle mahdollisuuden avata sen.

Teknisesti sovellus on yksinkertainen, ja se muodostuu ainoastaan yhdestä JavaScript-luokasta. Luokan metodi, jolla näytetään käyttäjälle hakemiston sisällöt, päättelee hakemiston tiedostonimien päätteiden perusteella, onko tiedosto kuva. Kuvatiedostot erotellaan muista tiedostoista säännöllisellä lausekkeella, joka tarkistaa, onko tiedoston nimen pääte jokin yleinen kuvatiedoston pääte esimerkkikoodin 12 tavoin.

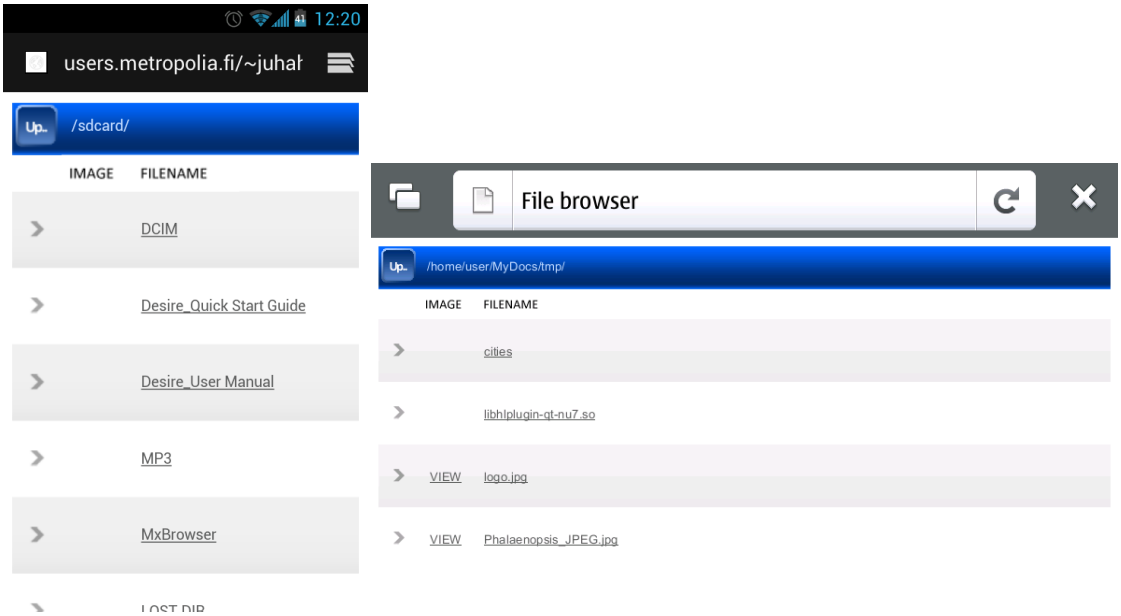
```
if (this.files[i].match(/(\.jpg|\.jpeg|\.gif|\.png)$/)) {
  // kyseessä on kuva, annetaan käyttäjälle linkki,
  // jonka avulla voidaan hakea kuvan kuvapistetiedot lisäosan
  // avulla
} else {
  // näytetään tavallinen hakemisto/tiedosto
}
```

Esimerkkikoodi 12. Kuvatiedostojen erottelu muista tiedostoista hyödyntäen säännöllistä lausekketta.

Alun perin kuvatiedostot suunniteltiin näytettäväksi tavallisen img-elementin avulla. Toteutuksen aikana kuitenkin havaittiin, ettei Internetissä sijaitsevalla HTML-sivulla voida tietoturvasyistä näyttää kuvia, jotka sijaitsevat paikallisessa tiedostojärjestelmässä. Tämän kiertämiseksi selainlisäosaan toteutettiin metodi, jonka avulla voidaan hakea kuvien kuvapistetietoja ja palauttaa ne skriptiympäristölle.

Kuvatiedostojen hakeminen on selainlisäosan kannalta hyvin samankaltainen toimenpide kuin tiedostolistauksen hakeminen. Hakemistorakennetaulukon sijaan lisäosa palauttaa halutun kuvan kuvapistetiedot taulukkona. Kun kuvan kuvapistetiedot on saatu haettua, ne voidaan piirtää sivulla olevaan canvas-elementtiin.

Kuvassa 14 on kuvakaappauksia toteutetusta sovelluksesta Android Browserilla ja Maemon Firefoxilla.

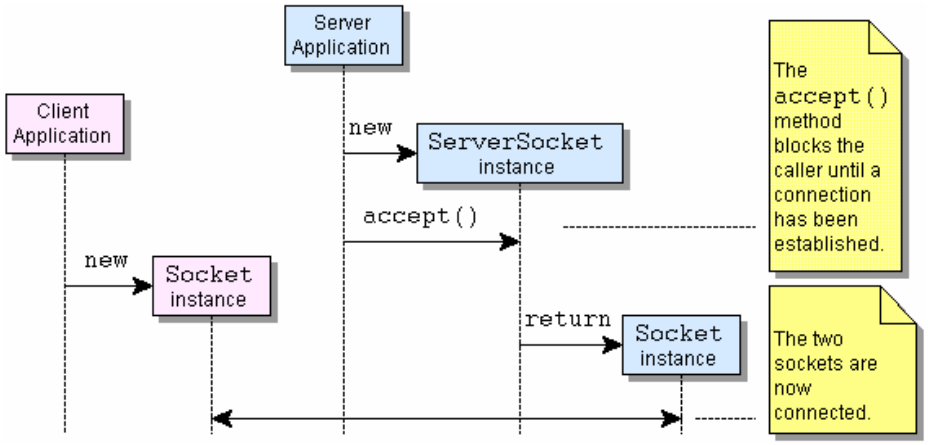


Kuvio 14. Kuvakaappauksia toteutetusta tiedostolistaussovelluksesta. Android Browser kuvassa vasemmalla ja Maemon Firefox Mobile oikealla.

5.3 WebSocket-palvelimen toteutus

WebSocket-palvelin toteutettiin Android-käyttöjärjestelmälle. Kehityksen helpottamiseksi palvelin toteutettiin aluksi tavallisena Java-ohjelmana, ja sen toimivuus testattiin Windows-tietokoneella. Testauksen jälkeen palvelin muutettiin Androidille sopivaksi. Käytännössä muuntaminen tarkoitti yksinkertaisen Android-sovelluksen tekoa, jonka ainut tehtävä on käynnistää WebSocket-palvelin taustalla suoritettavana Android-palveluna.

Koska WebSocket-protokolla toimii TCP-protokollan päällä, voitiin WebSocket-palvelin toteuttaa Javan `ServerSocket`- ja `Socket`-luokkien tarjoamien pistokkeiden avulla. Kuvassa 15 on kuvattu yhteyden muodostamista käyttäen `ServerSocket`- ja `Socket`-luokkia.



Kuvio 15. Sekvenssikaavio Javan ServerSocket- ja Socket-luokkien toiminnasta [41].

Android-palveluiden tulisi suorittaa raskaat operaatiot omissa säikeissään [42]. `ServerSocket`-luokan yhteyksiä kuunteleva metodi on niin sanotusti blokkaava, eli metodia suoritetaan niin kauan, kunnes palvelimelle tulee yhteyspyyntö. Pyynnön jälkeen se jää odottamaan seuraavaa yhteyspyyntöä, eikä näin ollen muun sovelluksen suoritus etene. Tästä syystä sen suoritus piti siirtää omaan säikeeseensä. Tätä varten toteutettiin `WSServerThread`-luokka, jonka tehtävänä on alustaa `ServerSocket`-luokan instanssi kuuntelemaan haluttua porttia ja odottaa tulevia yhteyspyyntöjä omassa säikeessään, kunnes palvelin sammutetaan. [41.]

Kun `ServerSocket`-luokka havaitsee tulevan yhteyspyynnön, se palauttaa varsinaisen pistokkeen, jota mallintaa Javan `Socket`-luokka. `Socket`-luokan avulla voidaan kuunnella ja vastaanottaa asiakasohjelman lähettämiä viestejä [41]. Myös tämä operaatio on blokkaava, joten jokainen pistoke kääritään omaan `ListenerThread`-luokan instanssiin, jonka tehtävänä on kuunnella pistokkeelta saapuvia viestejä omassa säikeessään. Tämä mahdollistaa useiden samanaikaisten pistokkeiden kuuntelun. `ListenerThread`-luokkaan toteutettiin tuki WebSocket-protokollan kättelylle, ja jokaiselle uudelle pistokkeelle suoritetaan luvussa 4.2 kuvattu prosessi. Mikäli kättely onnistutaan suorittamaan, jää `ListenerThread` kuuntelemaan pistokkeesta saapuvia tavuja. Saapuvia tavuja tulkkaamaan toteutettiin `WebSocketReader`-luokka, jonka tehtävänä on erotella ja palauttaa kokonaiset viestit `ListenerThread`ille saapuvien tavujen joukosta WebSocket-protokollan mukaisesti.

`WebSocketReader`-luokan palauttamia selkokielisiä viestejä tulkitsemaan toteutettiin `ServiceApiProtocol`-luokka, jonka tehtävänä on selvittää asiakasohjelman haluama palvelurajapinta saapuneen viestin perusteella ja toimia sen mukaisesti.

Asiakasohjelmaksi toteutettiin yksinkertainen selainpohjainen sovellus, joka muodostaa yhteyden WebSocket-palvelimeen käyttäen WebSocket API -rajapintaa. Kunnollista testisovellusta ei pystytty toteuttamaan Androidille, koska työn tekovaiheessa Androidin selain ei tukenut WebSocket-teknologiaa. WebSocket-palvelimen toimivuus pystyttiin kuitenkin testaamaan onnistuneesti kolmannen osapuolen valmistamalla Android-sovelluksella, joka osaa muodostaa yhteyden palvelimeen WebSocket-protokollalla [43].

5.4 WebView-komponentin testaus

WebView-komponentti ratkaisumallista ei toteutettu varsinaisia demosovelluksia, koska ne eivät sopineet alkuperäiseen vaatimukseen, jossa tavallisessa Internet-selaimessa suoritettavat selainpohjaiset sovellukset voisivat käyttää käyttöjärjestelmän palvelurajapintoja. Työssä kuitenkin päädyttiin testaamaan WebView-komponenttiin perustuvaa ratkaisumallia, jotta sitä voitaisiin verrata muihin ratkaisumalleihin. Komponentin toimintaa testattiin Androidilla ja Maemolla.

Molemmat ympäristöt mahdollistavat natiiviolion sitomisen WebView-komponentissa olevan skriptikielen globaaliksi muuttujaksi. Tämä antaa skriptikielelle mahdollisuuden kutsua natiiviolion metodeja. Natiivioliolla tarkoitetaan instanssia natiiviympäristössä suoritettavasta luokasta. Molemmissa sitominen tehdään käyttämällä siihen tarkoitettua WebView-komponentin metodia. Esimerkkikoodeissa 13 ja 14 näkyy, miten sidonta tapahtuu.

```
webView.addJavascriptInterface(new DemoJavaScriptInterface(),
    "demo");
```

Esimerkkikoodi 13. Natiiviolion sitominen WebView-komponenttiin Androidilla.

```
webView->page()->mainFrame()->
    addToJavaScriptWindowObject("demo", this);
```

Esimerkkikoodi 14. Natiiviolion sitominen WebView-komponenttiin Maemolla.

Sitomisen jälkeen WebView-komponentin skriptiympäristöllä on globaali demo-niminen olio, jonka läpi on mahdollista kutsua natiiviolion metodeja.

5.5 Tulokset

NPAPI-selainlisäosilla saatiin onnistuneesti toteutettua vaatimusten mukainen selainpohjainen sovellus, joka pystyy käyttämään mobiililaitteen palvelurajapintoja. Lisäosa toteutettiin Androidille ja Maemolle, joiden selaimet käyttävät eri selainmoottoreita. Tästä huolimatta toteutukset olivat itse palvelurajapintojen käsittelyä lukuun ottamatta lähes identtiset. NPAPI-selainlisäosan toteuttaminen oli kuitenkin varsin haastavaa käytettyjen teknologioiden ja niukkojen dokumentaatioiden vuoksi. Tätä kirjoitettaessa Androidin selain suostuu käynnistämään ainoastaan Adoben Flash-lisäosan, ellei laitetta käytetä pääkäyttäjäoikeuksilla. Lähes kaikista myynnissä olevista Android-laitteista on oletusasetuksilla poistettu pääkäyttäjäoikeudet käytöstä ja niiden käyttöönotto voi olla työlästä.

WebSocket-palvelimeen perustuva ratkaisumalli onnistuttiin todistamaan toimivaksi teoriassa. Käytännössä testausta vaikeutti työn tekohetkellä vallinnut heikko tuki WebSocket-teknologialle mobiiliselaimissa. Työn tekemisen jälkeen on kuitenkin ilmestynyt useampia mobiiliselaimia, jotka tukevat WebSocket-teknologiaa. Esimerkiksi Applen iOS:n Safari, Opera Mobile, Firefox ja Googlen Chromen mobiiliversio tukevat teknologiaa ainakin osittain. WebSocket-protokolla on kuitenkin edelleen kehitysvaiheessa, ja siihen voi tulla jatkossakin isoja muutoksia, jotka mahdollisesti rikkovat nykyiset siihen perustuvat ohjelmat. Selainpohjaisen sovelluksen toteutuksen kannalta WebSocket-ratkaisumalli on hieman työläämpi kuin muiden insinöörityössä tutkittujen ratkaisumallien, koska kaikki pyynnöt ja niiden sisältämät palvelinrajapintakyselyt ovat asynkronisia, eli käytännössä kun palvelimelle tehdään pyyntö, ei siihen saada vastausta samassa metodissa, vaan vastausta pitää odottaa niin kauan, kunnes se saapuu palvelimelta. Vastauksen saapuminen saa Internet-selaimen laukaisemaan tapahtuman eri puolella koodia, mikä voi johtaa vaikeaselkoisempaan koodiin.

WebView-komponenttiin perustuva ratkaisumalli ei vastannut alkuperäisiä vaatimuksia, joiden mukaan mobiiliselaimessa suoritettavan selainpohjaisen sovelluksen tulisi pystyä käyttämään mobiililaitteen palvelurajapintoja. Internet-selaimen sijaan WebView-komponenttia hyödyntävät ohjelmat ovat omia itsenäisiä ohjelmiaan. Tästä muodostuu tilanne, jossa jokaiselle selainpohjaiselle sovellukselle pitäisi luoda oma WebView-komponenttia käyttävä ohjelma. WebView-komponentin avulla onnistuttiin kuitenkin käyttämään palvelurajapintoja selainpohjaisesta sovelluksesta. Tämän toteuttaminen oli todella yksinkertaista ja nopeaa, koska WebView-komponentissa on tähän tarkoitukseen suunniteltuja ominaisuuksia.

6 Yhteenveto

Älypuhelimet yleistyvät, niiden käyttö lisääntyy jatkuvasti ja palveluntarjoajat tuovat palveluitaan entistä enemmän mobiilikäyttäjien saataville. Saadakseen suurimman mahdollisen kohdeyleisön, palveluntarjoajien tulee tarjota palveluitaan usealle eri mobiilialustalle, jolloin ongelmaksi muodostuvat usean eri ohjelman kehitys- ja ylläpitokustannukset. Selainpohjaisten sovellusten toteuttaminen poistaa useamman erillisen ohjelman tarpeen, koska ne toimivat laitteilla, joissa on Internet-selain. Toisaalta HTML5-standardin tuomista uudistuksista huolimatta selainpohjaisilla sovelluksilla ei ole mahdollista käyttää kaikkia käyttöjärjestelmien tarjoamia palvelurajapintoja.

Selainpohjaisten sovellusten yleistyessä niiden suoritusympäristö, Internet-selain, nousee entistä merkittävämpään rooliin. Insinöörityössä perehdyttiin tarkemmin yleisimpien Internet-selainten toimintaan ja HTML5-standardin suosituksiin verkkosivujen lataamisesta ja näyttämisestä.

Puuttuvien palvelurajapintojen käytön mahdollistamiseksi insinöörityössä suunniteltiin ja toteutettiin kolme erilaista ratkaisumallia, joilla muodostetaan ohjelmallinen silta Internet-selaimen skriptiympäristön ja natiiviympäristön välille. NPAPI-selainlisäosaan perustuvasta ratkaisumallista toteutettiin onnistuneesti selainpohjainen sovellus sitä suorittavan laitteen tiedostojärjestelmän selaamiseen. Lisäosan toteuttaminen oli haastavaa vaikeiden teknologioiden ja niukkojen dokumentaatioiden takia. Vaikka NPAPI on selainriippumaton teknologia, ei sille löydy tukea kaikista markkinoilla olevista mobiilikäyttöjärjestelmistä.

WebSocket-teknologiaa hyödyntävä ratkaisumalli onnistuttiin todistamaan toimivaksi vain teoriassa, koska tuki teknologialle on vielä rajoittunutta mobiiliselaimissa. WebSocket-protokolla on edelleen kehityksen alla, ja tämän insinöörityön aikana siihen tuli muutoksia, jotka rikkoivat sen aiempaan versioon perustuneet toteutukset. WebSocket-teknologia on kuitenkin osa HTML5:tä, joten tulevaisuudessa sille voi odottaa kattavaa tukea myös mobiiliselaimissa.

Kolmas toteutettu ratkaisumalli perustuu käyttöjärjestelmien tarjoamaan WebView-ohjelmistokomponenttii. Koska WebView-ratkaisumallissa verkkopohjaisia sovelluksia suoritetaan Internet-selaimen sijasta omissa sovelluksissaan, se ei vastannut alkuperäi-

siä vaatimuksia. WebView-komponenttia käyttämällä onnistuttiin kuitenkin toteuttamaan ohjelmallinen silta selainpohjaisten sovellusten ja palvelurajapintojen välille. Sillan toteuttaminen oli nopeata ja helppoa, koska komponentissa on siihen tarkoitettu metodi, jolla voidaan sitoa natiiviympäristön olio skriptiympäristön käytettäväksi.

Kaikista ratkaisumalleista saatiin toteutettua toimivat ohjelmat, jotka ratkaisevat alkuperäisen ongelman. NPAPI-selainlisäosaan ja WebSocket-teknologiaan perustuvilla ratkaisumalleilla ei kuitenkaan tavoiteta koko asiakaskuntaa, koska kaikki markkinoiden laitteista eivät tue niitä täysin. WebView-ohjelmistokomponenttiin perustuva ratkaisumalli puolestaan ei täyttänyt alkuperäistä vaatimusta, koska se vaatii erillisen ohjelman tavallisen Internet-selaimen sijaan.

Lähteet

1 Connors, Adam & Sullivan, Bryan. 2010. Mobile Web Application Best Practices. Verkkodokumentti. <http://www.w3.org/TR/mwabp/>. 14.12.2010. Luettu 16.5.2012.

2 Stark, Jonathan & Jepson, Brian. 2012. Building Android Apps with HTML, CSS, and JavaScript. Sebastobol, CA: O'Reilly.

3 Global mobile statistics 2012. 2012. Verkkodokumentti. dotMobi. <http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats>. Luettu 16.5.2012.

4 Jones, Seth, Voskoglou, Christina, Vakulenko, Michael, Measom, Vanessa, Constantinou, Andreas & Kapetanakis, Matos. 2012. Cross-platform developer tools 2012. London: VisionMobile.

5 Laakso, Henri. 2012. Android sirpaloituu ja saa kehittäjät siirtymään html5:een. Verkkodokumentti. <http://www.mikropc.net/kaikki_uutiset/android+sirpaloituu+ja+saa+kehittajat+siirtymaan+html5een/a792004>. 20.3.2012. Luettu 16.5.2012.

6 Publishing an App in the App Store. 2012. Verkkodokumentti. Apple Inc. <http://developer.apple.com/library/ios/#documentation/General/Conceptual/ApplicationDevelopmentOverview/DeliverYourAppontheAppStore/DeliverYourAppontheAppStore.html>. 9.1.2012. Luettu 16.5.2012.

7 List of mobile phones and tablets having Webkit based browser. 2011. Verkkodokumentti. MobiTechie. <http://www.mobitechie.com/browser/list-of-webkit-based-mobile-phones-and-tablets/>. Luettu 16.5.2012.

8 Mobiililaajakaista. 2011. Verkkodokumentti. Mobiililaajakaista. <http://www.mobiililaajakaista.com/>. 29.6.2011. Luettu 16.5.2012.

9 Nokia Siemens Networks White Paper: Latency. 2009. Nokia Siemens Networks.

10 Hickson, Ian. 2012. HTML5. Verkkodokumentti. <http://www.w3.org/TR/html5>. 29.3.2012. Luettu 16.5.2012.

11 CSS Specifications. 2012. Verkkodokumentti. W3C. <http://www.w3.org/Style/CSS/current-work>. 5.5.2012. Luettu 16.5.2012.

12 Casario, Marco, Elst, Peter, Brown, Charles, Wormser, Nathalie & Hanquez, Cyril. 2011. HTML5 Solutions: Essential techniques for HTML5 developers. New York: Springer Science+Business Media LLC.

13 WebGL - OpenGL ES 2.0 for the Web. Verkkodokumentti. Khronos Group. <http://www.khronos.org/webgl/>. Luettu 16.5.2012.

14 CSS3 Transitions. Verkkodokumentti. W3Schools. <http://www.w3schools.com/css3/css3_transitions.asp>. Luettu 16.5.2012.

15 Hogan, Brian P. 2010. HTML5 and CSS3 Develop with Tomorrows Standards Today. USA: The Pragmatic Bookshelf.

16 Fette, I. & Melnikov A. 2011. The WebSocket Protocol. Verkkodokumentti. <http://tools.ietf.org/html/rfc6455>. Luettu 16.5.2012.

17 Tibbett, Richard. 2011. Contacts API. Verkkodokumentti. <http://www.w3.org/TR/contacts-api/>. 16.6.2011. Luettu 16.5.2012.

18 Grosskurth, Alan & Godfrey, Michael W. 2006. A reference architecture for web browsers. Waterloo: David R. Cheriton School of Computer Science.

19 Google Chrome: FAQ for web developers. 2011. Verkkodokumentti. Google. <http://www.google.com/chrome/intl/en/webmasters-faq.html>. Luettu 16.5.2012.

20 Safari Features. 2012. Verkkodokumentti. Apple Inc. <http://www.apple.com/safari/features.html>. Luettu 16.5.2012.

21 JavaScriptCore. Verkkodokumentti. WebKit. <http://trac.webkit.org/wiki/JavaScriptCore>. Luettu 16.5.2012.

22 Gecko. 2012. Verkkodokumentti. Mozilla. <https://developer.mozilla.org/en/Gecko>. 18.3.2012. Luettu 16.5.2012.

23 SpiderMonkey. 2012. Verkkodokumentti. Mozilla. <https://developer.mozilla.org/en/SpiderMonkey>. 11.5.2012. Luettu 16.5.2012.

24 Opera version history. 2012. Verkkodokumentti. Opera Software ASA. <http://www.opera.com/docs/history/>. 26.4.2012. Luettu 16.5.2012.

25 Internet Explorer 9 Guide for Developers. 2011. Verkkodokumentti. Microsoft. <http://msdn.microsoft.com/en-us/ie/hh410104.aspx>. 14.3.2011. Luettu 16.5.2012.

26 Garsiel, Tali & Irish, Paul. 2011. How browsers work: behind the scenes of modern web browsers. Verkkodokumentti. <http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/>. 5.8.2011. Luettu 16.5.2012.

27 History of the Web Standards Project. Verkkodokumentti. Web Standards Project. <http://www.webstandards.org/about/history/>. Luettu 16.5.2012.

28 Bos, Bert, Celik, Tantek, Hickson, Ian & Wium Lie, Håkon. 2011. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. Verkkodokumentti. <http://www.w3.org/TR/CSS2/>. 7.6.2011. Luettu 16.5.2012.

29 Le Hors, Arnaud, Nicol, Gavin, Wood, Lauren, Champion, Mike & Byrne, Steve. 2000. Document Object Model Core. Verkkodokumentti. <http://www.w3.org/TR/DOM-Level-2-Core/core.html>. Luettu 16.5.2012.

30 Hidayat, Ariya. 2010. JavaScript Engines: How to Compile Them. Verkkodokumentti. <http://www.sencha.com/blog/javascript-engines-how-to-compile-them/>. 12.10.2010. Luettu 16.5.2012.

31 Gecko Plugin API Reference. 2011. Verkkodokumentti. Mozilla. <https://developer.mozilla.org/en/Gecko_Plugin_API_Reference>. 10.8.2011. Luettu 16.5.2012.

32 Oliphant, Zan. Creating Netscape Navigator Plug-Ins. Verkkodokumentti. <http://www.webbasedprogramming.com/Web-Programming-Unleashed/ch32.htm>. Luettu 16.5.2012.

33 Scripting plugins. 2011. Verkkodokumentti. Mozilla. <https://developer.mozilla.org/en/Gecko_Plugin_API_Reference/Scripting_plugins>. 26.8.2011. Luettu 16.5.2012.

34 Building a firefox plugin - part one. 2009. Verkkodokumentti. ColonelPanic. <http://colonelpanic.net/2009/03/building-a-firefox-plugin-part-one/>. 1.3.2009. Luettu 16.5.2012.

35 SampleBrowserPlugin. Verkkodokumentti. Android Open Source Project. <https://github.com/android/platform_development/tree/master/samples/BrowserPlugin>. Luettu 16.5.2012.

36 Java SE Documentation. Verkkodokumentti. Oracle. <http://docs.oracle.com/javase/6/docs/technotes/guides/jni/spec/design.html>. Luettu 16.5.2012.

37 Hickson, I. 2010. The WebSocket protocol. Verkkodokumentti. <http://tools.ietf.org/html/draft-hixie-thewebsocketprotocol-76>. 6.5.2010. Luettu 16.5.2012.

38 Hickson, Ian. 2011. The WebSocket API. Verkkodokumentti. <http://www.w3.org/TR/websockets/>. 8.12.2011. Luettu 16.5.2012.

39 Web Apps Overview. 2012. Verkkodokumentti. Android Developers. <http://developer.android.com/guide/webapps/index.html>. 9.5.2012. Luettu 16.5.2012.

40 ISO/IEC 9899:1999. Programming languages - C. 2000. New York: American National Standards Institute.

41 Java Networking Overview. Verkkodokumentti. Oracle. <http://docs.oracle.com/javase/6/docs/technotes/guides/net/overview/overview.html>. Luettu 16.5.2012.

42 Android API: Service. Verkkodokumentti. Android Developers. <http://developer.android.com/reference/android/app/Service.html>. Luettu 16.5.2012.

43 What is jWebSocket? Verkkodokumentti. jWebSocket. <http://jwebsocket.org/>. Luettu 16.5.201