Vesa-Pekka Vihantomaa

Selaimen toiminta ja tiedonsiirto

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Opinnäytetyö

5.12.2013

Metropolia

Tekijä Otsikko Sivumäärä Aika	Vesa-Pekka Vihantomaa Selaimen toiminta ja tiedonsiirto 44 sivua 5.12.2013
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaajat	yliopettaja Erja Nikunen lehtori Simo Silander

Insinöörityössä luodaan katsaus web-teknologioihin. Tavoitteena on käsitellä selaimen toimintaa ja tiedonsiirtoa. Erityisesti keskitytään selaimen ja palvelimen väliseen tiedonsiirtoon.
Aiheenkäsittely perustuu lähdemateriaaleihin tutustumiseen. XMLHttpRequest, WebSocket ja server-sent event tuovat omanlaisensa ratkaisunsa tietojen välittämiseen selaimelle. Näiden tekniikoiden käyttäminen vaati myös palvelinpuolen soveltuvuutta. WebRTC:n myötä myös reaaliaikainen kommunikaatio on tullut selaimiin. Selainsivusta toiseen selainsivuun tapahtuva tiedonsiirto on saanut oman rajapintansa. Tämä tuo selkeän tavan selainten sivujen väliseen tiedonsiirtoon.

Avainsanat	HTTP, JavaScript, MessageEvent, HTML, soketti, komponentti

Author Title Number of Pages Date	Vesa-Pekka Vihantomaa Web Browser and Data Transfer between Browser and Server 44 5 December 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Erja Nikunen, Principal Lecturer Simo Silander, Senior Lecturer

This thesis provides an overview of web technologies. The aim is to understand how the web browser behaves and also to understand some aspects of data transfer. The study concentrates especially on data transfer between web browser and server.
The contents of the study are mostly based on source materials from the internet. XMLHttpRequest (XHR), WebSocket and Server-Sent Events (SSE) brings their own kind of solution to data transfer to the web browser. Using these technologies requires compatibility from the server side. WebRTC is for real-time communication between two web browsers. The web browser has new API for data transfer between web pages, this brings an unambiguous way to communicate between Web pages.

Keywords	HTTP, JavaScript, MessageEvent, HTML, socket, component

Sisällys

LYHENTEITÄ

CSS	*Cascading Style Sheet*. Tyylisivukieli, jonka avulla määritellään verkkosivun ulkoasu.
CORS	*Cross origin resource sharing*. Toimintatapa, joka sallii verkkosivulla olevan JavaScript-koodin tekemään XMLHttpRequest-pyynnön eri verkkoalueelle.
DOM	*Document Object Model*. Alustariippumaton ja ohjelmointikielestä riippumaton tapa kuvata ja käsitellä HTML-, XHTML- ja XML-dokumentien objekteja.
HTML	*HyperText Markup Language*. Sivukuvaus- ja ohjauskieli, jolla luodaan hypertekstiasiakirjoja. WWW-sivun kuvaus.
HTTP	*Hypertext Transfer Protocol*. Sovellustason protokolla, jolla asiakkaat ja palvelimet viestivät keskenään hajautetuissa järjestelmissä. Käytetään Internetissä muun muassa WWW-sivujen siirtoon.
JavaScript	Netscapen luoma olio-ohjelmointikieli.
JSON	*JavaScript Object Notation*. JavaScript-kieleen pohjautuva kevytrakenteinen ja kieliriippumaton tiedonkuvaus.
JSONP	*JSON with padding*. Tapa lähettää pyyntö eri verkkoalueeseen. Vastauksena saadaan JSON-muotoista tietoa
MIME	*Multipurpose Internet Mail Extensions*. Määrittelee tavan, miten Internetissä kulkevien viestien sisältöosa määritellään ja kuvataan. Käytössä muun muassa sähköpostiviesteissä.
SOP	*Same Origin Policy*. Saman alkuperän käytäntö. Selaimen valvoma käytäntö, jossa selaimeen ladattu sivu ei käytä muualta ladattuja resursseja.
SSE	*Server-Sent Events*. Tekniikka, jolla siirretään tietoa palvelimelta verkkoselaimelle HTTP-protokollan avulla.
URL	*Uniform Resource Locator*. Osoitetieto, josta resurssi löydetään. WWW-sivun osoite.
XHR	*XMLHttpRequest*. Ohjelmistorajapinta, jolla saa luotua HTTP-protokollan mukaisen pyynnön verkkoselaimelta palvelimelle. Tiedonsiirtoon tarkoitettu olio.

1 Johdanto

Web koostuu verkkodokumenteista, jotka voivat sisältää linkkejä toisiin verkkodokumentteihin. Verkkodokumentit saadaan verkkopalvelimilta, jotka haetaan Internet-verkon kautta käyttäjille nähtäviksi esimerkiksi verkkoselaimella. Suurin osa verkkodokumenteista on kuvailtu HTML-kielellä. HTML-dokumentti on tekstipohjainen kuvaus verkkodokumentista. Tällä hetkellä uusin käytettävissä oleva HTML:n versio on viisi, joka voidaan ilmaista merkinnällä HTML5. Tosin tätä merkintää käytetään kuvaamaan uusia selaimen ominaisuuksia kuten esimerkiksi WebGL:ää.

Verkkosovellus voidaan määrittää tietokoneohjelmaksi, mikä sijaitsee verkkopalvelimella, ymmärtää HTTP-protokollaa ja luo verkkosivuja tai muuta sisältöä dynaamisesti verkkoselaimen pyynnöstä [1]. Selainpuolella verkkosovellus voi koostua HTML-sivusta, CSS-tyylimäärittelyistä ja JavaScript-koodista. CSS-tyylimäärittely määrittelee HTML-sivun ulkoasun ja elementtien sijainnin. JavaScript-koodi luo HTML-sivulle toiminnallisuuden. Näillä kolmella osalla voidaan tehdä käyttöjärjestelmiin widgettejä ja sovelluksia. Samoin mobiililaitteisiin voidaan luoda sovelluksia tai widgettejä. Web-sovelluksessa widget on itsenäinen pieni ohjelma, mikä voidaan liittää verkkosivulle. Widget voi olla myös käyttöliittymäkomponentin osakokonaisuus.

Verkkodokumentit voivat olla staattisia tai dynaamisia dokumentteja. Staattiset dokumentit ovat verkkopalvelimella valmiina HTML-tiedostona palvelimella, kun taas dynaamiset dokumentit luodaan verkkosovelluksella yleensä silloin, kun käyttäjä niitä pyytää verkkopalvelimelta. [1.]

Staattisen dokumentin ja dynaamisen dokumentin määrittäminen ei ole välttämättä aivan suoraviivaista mutta voidaan sanoa, että staattinen dokumentti on sivu, jota ei ole generoitu ohjelmallisesti ja palvelin noutaa kyseisen sivun suoraan käyttäjälle. Toisaalta staattisiakin dokumentteja on voitu ohjelmallisesti generoida ja dynaaminenkin sivu on voinut olla muuttumattomana pidemmän aikaa, kun dynaaminen sivu on haettu käyttäjälle jonkinlaisesta välimuistista. Tällöin dynaaminen sivu ei ole luotu silloin, kun käyttäjä on sitä pyytänyt verkkopalvelimelta. Näin tehdään, jotta verkkosovellusta voisi käyttää mahdollisimman moni ihminen samanaikaisesti. Ajatuksena on siis palvella mahdollisimman nopeasti montaa käyttäjää, jotta useampi käyttäjä pysyisi tyytyväisenä. Kun käyttäjä on tyytyväinen, hän palaa uudelleen verkkosivustolle. Tämä

toivon mukaan antaa mahdollisuuden esimerkiksi yritykselle parempaan myyntitulokseen.

Verkkoselaimen ja verkkopalvelimen yhdistää HTTP-protokolla. Tämän protokollan avulla verkkoselain ja verkkopalvelin kommunikoivat keskenään. On myös muitakin protokollia ja arkkitehtuureja.

Nykyään on myös paremmin mahdollista ladata sivusto tai ohjelma omalle laitealustalle esimerkiksi omaan tietokoneeseen ja käyttää tai selata sivustoa ilman Internet-yhteyttä. Kun Internet-yhteys on käytössä, voidaan mahdolliset muutokset lähettää tai vastaanottaa palvelimen verkkosovellukselta. Selainta on myös mahdollista käyttää thin clientina eli niin sanottuna tyhmänä päätteenä. Tällöin tietokoneohjelma toimii palvelimella, ja tietokoneohjelman käyttöliittymä siirretään tässä tapauksessa selaimen selainikkunaan. HTML-, CSS- ja JavaScript-kieltä voidaan käyttää myös erilaisissa muissa laitteissa kuten esimerkiksi televisioissa tai pelikonsoleissa. Näissä laitteissa on todennäköisesti jonkinlainen selainkomponentti, mikä mahdollistaa myös kaksi- tai kolmiulotteisten kuvien piirtämisen, verkkopuhelinyhteyden ja muiden uudempien selaimiin liittyvien tekniikoiden käytön. Selainta voidaan näin ollen pitää monipuolisena ohjelmointialustana. Palvelinpuolella eri tekniikoiden valinta voi olla täysin vapaata. Selain on hieman rajoitetumpi, ellei sitten itse koosta tai ohjelmoi haluamiansa ominaisuuksia samalla tavalla kuin palvelinpuolella.

Verkkosovelluksissa käyttäjien määrä voi vaihdella. Käyttäjiä voi olla yhtenä päivänä muutamia satoja ja toisena päivänä satoja tuhansia tai jopa miljoonia. Ympäristö, missä ohjelma, on vaihteleva. On erilaisia palvelinympäristöjä, käyttöympäristöjä, tietokantoja ja selaimia, jossa ohjelman pitäisi toimia.

Tämän opinnäytetyön tavoitteena on tutkia verkkoselaimen toimintaa ja verkkoselaimeen liittyvää ohjelmointia. Tarkastelun kohteena on myös verkkoselaimen ja palvelimen tietojenvaihto. Seuraavassa luvussa käsitellään tarkemmin HTTP-protokollaa, jonka jälkeen tarkastellaan verkkoselaimen toimintaa ja siihen liittyvää ohjelmointia. Tämän jälkeen käsitellään tarkemmin verkkoselaimen toimintaa HTML-elementtien avulla. Viimeiseksi käsitellään uudempia tapoja miten verkkoselain kommunikoi palvelimen ja eri verkkosivujen kesken.

2 HTTP-protokolla

HTTP-protokolla on TCP/IP:n sovellustason protokolla. Protokolla on tarkoitettu muun muassa verkkodokumenttien siirtoon. HTTP-protokollaa käytetään yleensä verkkoselaimen ja verkkopalvelimen väliseen kommunikointiin. TCP/IP on verkkoarkkitehtuurimalli, jonka perusteella voidaan rakentaa tietoverkkoja. Verkkoarkkitehtuuri malli sisältää neljä kerrosta, joista yksi niistä tai ylimpänä niistä on sovellustasokerros. TCP/IP:n nimi muodostuu kahdesta tietoverkoissa käytettävistä nimistä TCP ja IP. HTTP-protokolla käyttää TCP-protokollaa asiakkaan ja palvelimen yhdistämiseen. TCP-protokolla on tilallinen protokolla, kun taas HTTP-protokolla on tilaton protokolla. Protokollaa voidaan käyttää muuhunkin kuin verkkodokumenttien siirtoon, kuten esimerkiksi nimipalvelimissa. Nimipalvelin muuttaa selväkielisen verkko-osoitteen numeeriseksi IP-verkko-osoitteeksi. [2.]

HTTP-protokollan spesifikaatioista käytössä olevat versiot ovat HTTP 1.0 ja HTTP 1.1. Protokollan spesifikaatioita pidetään monitulkintaisena. HTTP 1.1 -versio mahdollistaa muun muassa pipelining-yhteyden. Pipelining-yhteydessä verkkoselain voi lähettää useamman pyynnön kerralla odottamatta palvelimen vastausta. Tällöin esimerkiksi HTML-dokumentin kuvia voidaan pyytää palvelimelta ilman palvelimen vastausta. Pipelining-yhteys saattaa olla parempi verrattuna yhteyteen ilman pipelining-yhteyttä, jos yhteydessä on viivettä. Vain idempotentit HTTP-protokollan metodeilla pipelining toimii. Idempotenttisesta ominaisuudesta kerrotaan kappaleessa 2.6. Idempotentteja metodeja ovat tai näitä metodeja pitäisi olla ainakin GET ja HEAD. GET- ja HEAD-metodien aikaansaamat toiminnot voivat olla ohjelmantekijän vastuulla. HTTP-protokollan mukaan nämä metodit pitäisi ohjelmoida idempotenteiksi. Versiossa 1.1 yhteys on aina päällä (engl. persistent) yhteyden muodostamisen jälkeen, ellei asiasta toisin päätetä. Tästä huolimatta verkkopalvelimet saattavat tosin katkaista yhteyden automaattisesti tietyn ajan jälkeen verkkopalvelimet voivat pyrkiä pitämään yhteysajan mahdollisimman lyhyenä. Myös selain saattaa katkaista yhteyden, koska pipelining-yhteys saattaa aiheuttaa käyttäjän huomaavan viiveen selaimen toiminnassa. Yleisin versio HTTP-protokollasta on sen uusin versio 1.1.

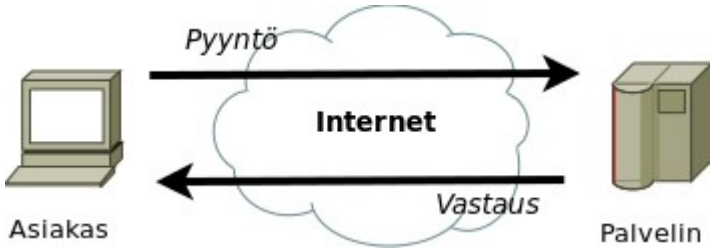
Protokollasta ollaan tekemässä parhaillaan uutta versiota 2.0. Jos version uusin spesifikaatio tulee voimaan, on se HTTP-protokollan ensimmäinen muutos 1990-luvun lopun jälkeen. Uudella spesifikaatiolla yritetään muun muassa selventää protokollan spesifi-

kaatiota, parantaa autentikointia eli käyttäjän tunnistamista, yksinkertaistaa protokollan toteutusta ja nopeuttaa yhteyksiä. Uuden HTTP-protokollan pohjana on Google-yhtiön kehittämä SPDY-protokolla. Yhtiö on kehittänyt SPDY-protokollan yhteyksien nopeuttamiseen. Standardin pohjaksi valittiin SPDY-protokolla, koska protokollasta oli valmiina toimiva toteutus.

Seuraavassa luvussa käsitellään verkkoarkkitehtuureja ja tilan käsitettä.

2.1 Asiakas-palvelin-malli

HTTP-protokolla on tilaton protokolla, koska se ei seuraa selaimen ja palvelimen välisiä yhteyksiä [1]. Protokollatasolla kahden eri pyynnön välillä ei tallenneta tietoa. Tällöin saman käyttäjän toimia ei pystytä seuraamaan verkkosovelluksen puolella ilman mitään ylimääräisiä toimia.



Kuva 1: Asiakas/palvelin-malli.

Protokolla noudattaa perinteistä asiakas-palvelin-arkkitehtuuria. Asiakas-palvelin-mallissa katsotaan olevan asiakasprosessi ja palvelinprosessi, jotka kommunikoivat keskenään. Kuvassa 1 asiakas lähettää pyynnön ja jää odottamaan palvelimen vastausta. Asiakkaaksi katsotaan se, kumpi yhteyden on aloittanut. Vuoroittain asiakkaana voi toimia joko asiakas tai palvelin riippuen siitä kumpi yhteyden on aloittanut. Yhteyden välissä voi olla useita muita palvelimia tai verkkolaitteita. Asiakas-palvelin-malli on hajautetun ohjelman malli. Asian voi myös ilmaista niin, että palvelin on odottamassa milloin asiakas ottaa yhteyttä. Tällaisen mallin avulla palvelin pystyy palvelemaan montaa asiakasta yhtäaikaisesti. Asiakas-palvelin-mallissa pitää sopia yhteyskäytännöstä eli protokollasta. Tällainen protokolla voi olla HTTP-protokolla. Useimmat Internet-sovellukset toimivat tämän mallin mukaisesti.

HTTP-protokolla noudattaa siis asiakas ja palvelin -arkkitehtuuria. On myös muita arkkitehtuureja kuten P2P-verkko eli vertaisverkko (engl. peer-to-peer). P2P-verkko on hajautettu ja keskittämätön verkkoarkkitehtuuri. Siinä ei käytetä yhtä keskitettyä palvelinta, jota kaikki asiakkaat käyttävät, niin kuin asiakas ja palvelin -arkkitehtuurissa.

Tapoja, joilla tuodaan tila selaimen ja palvelimen väliseen yhteyteen on ainakin kolme kappaletta.

- eväste
- URL-osoite
- FORM-lomake.

Evästeitä (engl. cookies) käytetään verkkoselaimissa. Verkkoselaimet tallentavat evästeen asiakaskoneelle ja lähettävät evästeen pyynnön mukana palvelimelle. Evästeisiin voi tallentaa rajallisen määrän tietoa. Pyynnössä tai vastauksessa evästeet esitetään nimi-arvo-pareina viestin otsikkokenttiin. Evästeitä voivat käsitellä selainohjelma ja palvelinohjelma. Viestien otsikkokentissä evästeet ovat näkyvillä, joten on parempi salata evästeet, jos ne sisältävät arkaluontoista tietoa. URL-osoitteeseen voidaan laittaa tilatieto parametrin arvona tai arvoina. HTML-dokumentin FORM-lomakkeeseen voidaan laittaa tilatieto piilotettuun kenttään. Olipa tapa mikä tahansa, tilatietoon ei pidä tallentaa mitään arkaluonteista ja kyseinen tilatieto pitää salata. Tilatiedon avulla voidaan tunnistaa pyynnöt saman asiakkaan lähettämäksi ja luoda istunto (engl. session). Istunto koostuu käyttäjän toimista. Tämän avulla voidaan tallentaa tietoja seuraavia pyyntöjä varten. Istunto on jossain tilassa, ja istunnon tila haetaan esimerkiksi tietokannasta. Yleensä istunnon tilatieto on jokin avainarvo, jonka perusteella tilatiedot haetaan verkkosovelluksessa esimerkiksi tietokannasta.

Evästeeseen, piilotettuun kenttään ja URL:n laitettava tilatiedon koko on rajallinen ja aiheuttaa turvallisuusongelmia. Istunnon tila on turvallisinta tallentaa palvelinohjelmassa ja sinne pystyy myös tallentamaan suuremman määrän erilaisia verkko-ohjelman tarvitsemia tietoja. Ennen istuntotunnisteen lähettämistä on parasta tunnistaa käyttäjä ensin luotettavasti varsinkin, jos kyseessä on verkkosovellus, jossa käyttäjän pitää kirjautua sisään verkkopalveluun. Parhaimmista tavoista autentikoida käyttäjä on HTML-sivun lomake tai käyttämällä hyvää tunnistautumispalvelua.

Turvallisuusongelma syntyy siitä, että ilman salausta tilatieto on käyttäjän muokattavissa. Tilatiedot näkyvät HTTP-pyynnön tai -vastauksen otsaketiedoissa. Tämän vuoksi käytetään esimerkiksi tietokantaa tilatietojen tallentamisen apuna paitsi ehkä yksinkertaisimmissa verkkosovelluksissa. Yksi suositeltavista tavoista on käyttää selaimen evästeitä tilatiedon apuna. Tilatiedon avulla verkkosovelluksessa haetaan esimerkiksi käyttäjän ystävät tai ostoskori. Joka kerta, kun käyttäjä tekee pyynnön palvelimelle, tilatiedot haetaan palvelimen tallennetusta paikasta verkkosovelluksen käyttöön. Kun pyyntö on käsitelty, tilatiedot hylätään. Verkkosovelluksessa voidaan käyttää jonkinlaista tilatietojen hakujen tallennusta tai tilatietojen väliaikaista tallennusta, jotta tilatietojen haku ja käyttöönotto olisi nopeampaa. Yksi mahdollinen ongelma evästeissä on, jos käyttäjä estää evästeiden käytön tai ulkopuolinen käyttäjä saa evästeen omaan käyttöönsä.

Seuraavassa luvussa kirjoitetaan resurssien hakutavoista.

2.2 URI

Taulukossa 1 on joitakin IANA:n rekisteröimiä tapoja hakea resurssi. Nämä on jaettu pysyviin (permanent URI scheme), väliaikaisiin (provisional URI scheme) ja historiallisiin (historical URI scheme). IANA eli The Internet Assigned Numbers Authority on ICANN:n eli Internet Corporation for Assigned Names and Numbers osasto, jonka tehtävänä on muun muassa rekisteröidä Internetissä käytössä olevia protokollia [4]. ICANN on voittoa tavoittelematon järjestö, jonka yhtenä tehtävänä on jakaa Internetissä käytettäviä verkko-osoitteita. Suoritetun epätieteellisen laskennan jälkeen voidaan todeta, että IANA:n rekisteröimiä skeemoja löytyy noin 180 kappaletta. On myös epävirallisia mutta yleisesti käytössä olevia URI-skeemoja, kuten esimerkiksi javascript ja jdbc [5]. Skeemalla javascript voidaan suorittaa JavaScript-koodia ja skeemalla jdbc voidaan luoda yhteys tietokantaan JDBC-tekniikalla. JDBC on Java-pohjainen teknologia tietokannan yhdistämiseen ja käsittelyyn. URI-skeemojen käyttö vaati ohjelman, joka tukee kyseistä skeeman käyttöä. Verkkoselaimet tunnistavat osan skeemoista. Verkkoselainten eri versiot ja eri verkkoselaimet tunnistavat vaihtelevan määrän URI-skeemoja.

URI koostuu merkeistä, jotka muodostavat nimen, sijainnin tai jonkin muun tunnisteen resurssille. Nimet jaetaan kahteen luokkaan, jotka ovat URL ja URN. URI voi sisältää

URL:in, URN:in tai molemmat. URL on URI, jonka perusteella resurssi tunnistetaan ja haetaan esimerkiksi verkko-osoitteen perusteella. URN on historiallinen nimi URI:lle, jonka skeema on urn. Se määrittää nimiavaruuden. Esimerkiksi "urn:isbn:1-23-456789-0" on URN-nimiavaruudessa ja isbn on URN-nimiavaruudentunnus. URN-nimiavaruuden tunnuksia rekisteröi IANA. Yksi rekisteröidyistä tunnuksista on siis isbn. URN on tarkoitettu osoittamaan resurssi yksilöllisesti ja pysyvästi, jopa silloin kun resurssia ei ole saatavilla tai sitä ei ole enää olemassa. [6.]

Taulukko 1: IANA:n rekisteröimiä URI skeemoja [7].

URI scheme	**Description**	**Registries**
http	Hypertext Transfer Protocol	permanent URI scheme
ftp	File Transfer protocol	permanent URI scheme
mailto	Electronic mail address	permanent URI scheme
bitcoin	bitcoin	provisional URI scheme
data	data	permanent URI scheme
sms	Short Message Service	provisional URI scheme
news	Usenet news	permanent URI scheme
wais	Wide Area Information Servers	historical URI scheme

Seuraavassa luvussa kirjoitetaan asiakkaan ja palvelimen välisistä viesteistä.

2.3 Viestit

HTTP-viestit ovat asiakkaan tai palvelimen tekemiä. Viestit ovat pyyntöjä tai vastauksia. Yleinen rakenne HTTP-viesteille on seuraavanlainen

```
Request-Line | Status-Line
*(message-header CRLF)
CRLF
[ message-body ]
```

Viesti sisältää joko request-line:n eli pyyntörivin tai status-line:n eli tilatietorivin. Tämän jälkeen tulee viestin message-headerit eli otsikkotiedot. Jokainen otsikkotietorivi päättyy CRLF-merkkiin eli rivinvaihtoon. Otsikkotietojen jälkeen tulee rivinvaihtomerkki, jonka jälkeen alkaa mahdollisesti message-body eli viestin sisältö. [3.]

2.4 Pyyntörivi tai tilatietorivi

Kun käyttäjä haluaa mennä verkkoselaimella verkkosivulle, verkkoselain lähettää esimerkiksi seuraavanlaisen pyynnön.

```
GET /index.html HTTP/1.1
Host: www.service.fi
```

Ensimmäinen rivi on pyyntörivi. Verkkoselain ilmoittaa pyynnössään, että käyttää metodia GET, resurssin polku on www.service.fi/index.html ja käyttää protokollan versiota HTTP/1.1. Toisella rivillä olevalla Hostilla ilmoitetaan, mihin koneeseen otetaan yhteys. Tämä otsatieto on pakollinen HTTP/1.1-protokollassa. Verkkopalvelimella voi olla monta nimeä käytössään, ja yksi palvelin voi palvella montaa verkkosivustoa. Tällä tavalla tieto haluttuun osoitteeseen säilyy. [1.]

Vastaukseksi palvelin voi lähettää selaimelle rivin.

```
HTTP/1.1 200 OK
```

Ensimmäinen vastausrivi on status line eli tilatietorivi. Tässä vastauksessa palvelin ilmoittaa, että käytettävä protokollan versio on HTTP/1.1, tilatiedon koodi on 200 ja tilatiedon teksti on OK. Tilatiedon koodi 200 tarkoittaa, että vastaus onnistui. Tilatiedon teksti on lyhyt kuvaus tilatiedon koodista, joka on ymmärrettävämpi muoto ihmisille. Tilatiedon tekstit ovat spesifikaatiossa suosituksia, joten nämä voisivat olla jotain muitakin. Tilatietorivin tilakoodin ensimmäisen numeron perusteella vastaukset luokitellaan taulukon 2 mukaisesti. Esimerkiksi numerolla 4 alkava tilakoodi tarkoittaa selaimesta johtuvaa virhettä ja numerolla 5 alkava tilakoodi palvelimesta johtuvaa virhettä. Numerolla 2 alkava tilakoodi tarkoittaa siis onnistumista, niin kuin tapahtui aikaisemmin esimerkissä palvelimen vastauksessa.

Taulukko 2: Tilakoodien (engl. status code) jaottelu [3].

Tilakoodi	**Luokittelu**
1xx	Informational
2xx	Success
3xx	Redirect
4xx	Client error
5xx	Server error

Seuraavassa luvussa käsitellään enemmän, mitä muuta pyyntöviesti tai vastausviesti voivat sisältää.

2.5 Otsikkotiedot

Otsikkotiedot alkavat pyyntörivin tai tilatietorivin jälkeen. Näitä rivejä voi olla useampikin, mutta verkkoselain tai verkkopalvelin voi rajoittaa näiden määrää tai kokoa.

Sisällönkoodaus voi olla tietona otsikkotiedossa. Sisältö pitää vastaanottaa samalla tavalla koodattuna asiakas- ja palvelinpuolella. Kun esimerkiksi saatu sisältö tallennetaan palvelimella, sisältö pitää tallentaa samalla tavalla koodattuna kuin se oli vastaanottaessa. Ellei sitten erikseen muuta sisällön koodausta ohjelmallisesti. Oikealla tavalla koodatun sisällön tallentaminen, lähettäminen ja vastaanottaminen on tärkeätä. On syytä varmistaa, että sisältö on koodattu oikein. Sisällön väärä koodaus aiheuttaa sisältöön virheitä tai saa aikaan tietoturvaongelmia verkkosovelluksissa. On esimerkiksi syytä varmistua, että tietokantaan tallennetaan tietoa samalla tavalla koodattuna kuin tietoa vastaanottaessa oli verkkosovelluksessa. Samoin tietoa lähettäessä tieto pitää olla samalla tavalla koodattuna vastaanottoon asti, kuin se on tietokantaan tallennettu. Otsikkotiedon otsikolla Content-Type ilmoitetaan viestin MIME-tyyppi. MIME eli Multipurpose Internet Mail Extensionsin avulla ilmoitetaan, mikä on sisällön tyyppi ja miten se on koodattu.

Otsikotiedon otsikot voivat olla minkä nimisiä tahansa, kunhan ne ovat HTTP-protokollamääritelmän mukaisia.

Seuraavassa luvussa käsitellään HTTP-protokollan metodeja.

2.6 Metodit

HTTP/1.1 -protokolla määrittelee yhteisiä metodeja pyynnöille. Metodi laitetaan pyyntöriville heti ensimmäisenä. Tämän jälkeen tulevat haettavan resurssin URI ja protokollan versio. Metodeja ovat muun muassa GET, POST, HEAD ja DELETE. Vakiintuneena tapana pidetään sitä, että GET- ja HEAD-metodia käytetään vain tiedon hakemiseen. Näin ollen näitä metodeja voidaan pitää turvallisena. Tärkeintä on, että muutoksia ei ole saatu aikaan käyttäjän pyynnöstä, vaikka joitain muutoksia tapahtuisi palvelimessa. GET-metodi on kaikkein yleisin metodi, jota Internetissä käytetään. HTTP/1.1 protokol-

lassa määritetään myös idempotenttinenominaisuus metodille. Tällä tarkoitetaan metodia, jonka vaikutus on sama useammalla pyynnöllä, kuin yhdellä pyynnöllä. Pyyntöjen pitää olla samoja. Ainakin metodeilla GET, HEAD, PUT ja DELETE on tämä ominaisuus. Myös sellaiset pyyntösarjat, joilla ei ole sivuvaikutuksia sanotaan idempotenteiksi. Uusia metodeja voi tehdä itsekin, kunhan vain asiakas ja palvelin ne tunnistavat. Vaikka protokolla määrittelee ominaisuuksia metodeille, kuten turvallinen tai idempotenttinen, niin on ohjelmantekijän omalla vastuulla, ovatko metodien ominaisuudet tällaisia. [3.]

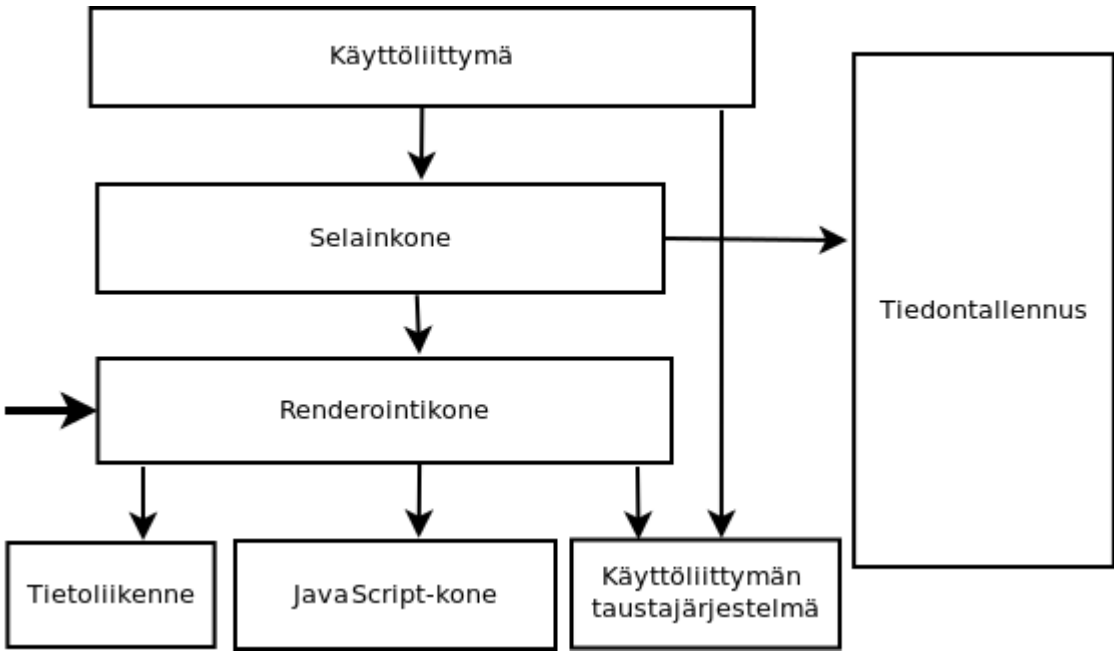
3 Selain

Käytettyjä WWW-selaimia ovat muun muassa Apple Safari, Mozilla Firefox, Google Chrome ja Microsoft Internet Explorer. Selaimia on tehty eri käyttöjärjestelmille pöytäkoneisiin ja mobiililaitteisiin. Lisäksi käytössä on uudempia ja vanhempia versioita samasta selaimesta. Tämä saattaa hankaloittaa verkkosivujen tai verkkosovellusten tekoa, koska vanhemmissa selainten versioissa ei ole käytössä uudempia ominaisuuksia. Laitteet, joissa selaimia käytetään, ovat erilaisia. Näissä laitteissa kiinnitetään enemmän huomiota näytön suuruuteen ja näytön resoluutioon verkkosivuja tai verkko-ohjelmia tehdessä. Laitteen näytön koko voidaan ottaa huomioon CSS-tyylitiedostoissa, joissa niin sanottujen media queryn avulla voidaan valita, mitä verkkoselaimessa näytetään. Tällaista suunnittelua kutsutaan responsiiviseksi suunnitteluksi (engl. responsive design). Javascript-kieltä voidaan myös käyttää tähän tarkoitukseen.

Jos selaimessa ei ole jotain haluttua ominaisuutta, yritetään haluttu ominaisuus saada aikaan täydentämällä (engl. polyfill). Tällöin yritetään luoda haluttu ominaisuus jollakin tavalla. Haluttu toiminto voidaan jättää myös toteuttamatta. Ominaisuuksien puutteet johtuvat selainten eroista. Aina ei ole kyse ominaisuuksien puutteista vaan toimintojen erilaisesta toiminnasta selaimissa. On myös mahdollista lisätä ominaisuuksia selaimeen, joita ei ole yhdessäkään selaimessa toteutettu. Toisenlainen lähestymistapa on ominaisuuksien kasvattaminen (engl. progressive enhancement), jossa toimintoja otetaan käyttöön selaimen niitä tukiessa. Selaimet ovat erilaisia, mutta niissä on myös joitain yhteisiä osia. Seuraavassa luvussa kirjoitetaan näistä lisää.

3.1 Selaimen komponentit

Selaimen käyttöliittymälle ei ole mitään yleistä mallia, minkälainen sen pitäisi olla, mutta vuosien kuluessa selaimet ovat alkaneet muistuttaa toisiaan. Kuvassa 2 kuvataan selaimen komponentteja. Tätä voidaan pitää jonkinlaisena referenssiarkkitehtuurina. [8.]



Kuva 2: Verkkoselaimen komponentit [8].

Eri verkkoselaimissa komponenttien nimet voivat olla erilaisia, ja rakennekin voi olla hieman erilainen. Pääosiltaan kuitenkin verkkoselaimen toiminta on samankaltainen eri verkkoselaimilla. Selain voi jakaantua seuraaviin osiin käyttöliittymään, selainkoneeseen, renderointikoneeseen, tietoliikenne, JavaScript-koneeseen, käyttöliittymän taustajärjestelmään ja tiedontallennukseen. Käyttöliittymä tarkoittaa selaimen käyttöliittymää lukuun ottamatta selaimen pyytämää ja näyttämää verkkosivua. Selainkone (engl. browser engine) ohjaa toimintoja käyttöliittymän ja renderointikoneen välillä. Renderointikone (engl. rendering engine) on vastuussa selaimen pyytämän verkkosivun näyttämisestä käyttäjälle. Selaimet saattavat käyttää jokaisella välilehdellä omaa prosessiaan renderointikoneelle. [8.]

Selaimet käyttävät erilaisia renderointikoneita, joiden pohjalta ne ponnistavat. Firefox perustuu Gecko Engineen, Internet Explorer perustuu Trident Engineen ja Chrome

perustuu Webkit Engineen. Opera-selain perustuu Blink Engineen, mutta on siirtymässä käyttämään WebKit Enginea. Näitä koneita on muitakin, mutta nämä ovat yleisimmät renderointikoneet, mitä muut verkkoselaimet tai muut ohjelmat käyttävät.

Tietoliikenneosa on vastuussa tiedonsiirrosta kuten esimerkiksi HTTP-protokollan mukaisista pyynnöistä. JavaScript-konetta käytetään JavaScript-koodin tulkintaan (engl. parse) ja ajamiseen. Verkkoselaimet käyttävät erilaisia JavaScript-koneita. Selaimen JavaScript-konetta voidaan käyttää myös muualla JavaScript-koodin ajamiseen kuin pelkästään verkkoselaimissa. Käyttöliittymän taustajärjestelmä (engl. UI Backend) on yleinen rajapinta käyttöliittymän osiin, jotka ovat alustasta riippumattomia. Se käyttää käyttöjärjestelmän käyttöliittymän toimintoja avukseen. Tiedontallennusosa on tarkoitettu selaimen tarvitsemiin tiedontallennuksiin. Selain käyttää tätä evästeiden tallentamiseen. Selain tukee myös muitakin tiedontallennustapoja, kuten esimerkiksi IndexedDB:tä ja WebSQL:ää. [8.]

Seuraavassa luvussa käsitellään verkkoselaimen toimintaa.

3.2 Selaimen toiminta

Selaimet noudattavat saman alkuperän käytäntöä (engl. same origin policy, SOP). Kun selain lataa verkkosivuja eri verkko-osoitteista, verkkosivut ovat eristettyjä toisistaan. Saman alkuperän käytäntö sallii sivujen vuorovaikutuksen vain, jos yhteyskäytäntö, verkkotunnus (engl. domain) ja portin numerot ovat samat. Selain valvoo, että JavaScript-kielellä ei voi muokata jonkin toisen sivun DOM-puuta tai HTML-sivu ei käytä muualta ladattuja resursseja. Saman alkuperän käytännön jyrkällä toteuttamisella ei voisi ladata esimerkiksi HTML-sivun kuvia muualta kuin haetun sivun osoitteesta.

Cross site scripting on tietoturva-aukko tai hyökkäys, jolla voidaan esimerkiksi ohittaa verkkoselaimen saman alkuperän käytäntö. Kun saman alkuperän käytäntö on ohitettu, hyökkääjä voi "varastaa" uhrin selaimesta evästeen. Cross site scripting -hyökkäyksestä voidaan käyttää lyhennettä XSS. Saman alkuperän käytäntö on alunperin luotu estämään Cross site scripting -haavoittavuus. Verkkosivu voi muuttaa alkuperäänsä tietyin rajoituksin. Selaimessa voidaan JavaScript-kielellä asettaa sivun document.domain -arvon verkkotunnus lyhyemmäksi. Kun esimerkiksi skripti on verkkosivulla http://www.metropolia.fi/p/index.html ja muuttaa verkkotunnukseksi metropolia.fi. Tämän jälkeen verkkosivu läpäisee saman alkuperän käytännön tarkistuksen sivun

http://metropolia.fi/p/index.html:n kanssa. Samoin perustein laurea.fi-sivun skripti ei voi muuttaa document.domain-arvoaan arvoksi metropolia.fi. Saman alkuperän käytäntö koskee myös XMLHttpRequest-objektia. Taulukossa 3 on tuloksia, kun vertaillaan saman alkuperän käytännöllä verkko-osoitetta http://store.company.com/dir/page.html taulukossa oleviin verkko-osoitteisiin. [9.]

Taulukko 3: Saman alkuperän vertailu. Otettu pienin muutoksin suoraan lähteestä [9].

URL	**Tulos**	**Syy**
http://store.company.com/dir2/other.html	Kyllä	
http://store.company.com/dir/inner/another.html	Kyllä	
https://store.company.com/secure.html	Ei	Eri protokolla
http://store.company.com:81/dir/etc.html	Ei	Eri portti
http://news.company.com/dir/other.html	Ei	Eri kone (engl. host)

Verkkosivut voidaan tehdä HTML-kielellä, CSS-kielellä ja JavaScript-kielellä. HTML kuvaa verkkosivun rakenteen. CSS eli Cascading Style Sheet on sivunkuvauskieli, jolla kuvataan verkkosivun ulkoasu. JavaScript-kieli tuo verkkosivulle toiminnallisuutta. HTML-sivu koostuu HTML-sivun elementeistä. Elementien attribuutteihin voidaan sijoittaa arvoja, tai attribuutteihin ei voi sijoittaa mitään arvoa. Elementtien attribuutit voivat antaa elementeille muun muassa yksilöllisen tunnisteen, luokkatunnisteen ja lisäominaisuuden. Verkkoselain tai verkkoselaimen ohjelmaskripti voi käyttää näitä elementtien attribuuttien tietoja toiminnassaan. HTML-sivu elementteineen voisi olla tämän näköinen.

```
<!DOCTYPE html>
<html manifest>
          <body>
               <p orange="" apple=""> Hello </p>
          </body>
</html>
```

HTML-elementissä on attribuuttina manifest. Tämä tarkoittaa sitä, että verkkoselain voi käyttäjän niin halutessa ladata verkkosivun sisällön verkkoselaimen muistiin. Tällaista verkkosivua voi käyttää ilman verkkoyhteyttä. Manifest-nimiseen tiedostoon voidaan lisätä ne tiedostot, jotka verkkoselain hakee muistiinsa. Attribuutti voi olla nimeltään mitä tahansa, kunhan se on oikean muotoinen. Verkkoselain tunnistaa elementin attri-

buutin ja käyttää sitä toiminnassaan. Jos elementin attribuuttia ei tunnisteta, verkkoselain ei käytä sitä "toiminnassaan". Tässä tapauksessa elementin attribuutti löytyy DOM-puusta kyseisen elementin solmusta. Ylläolevassa esimerkissä laitoin P-elementtiin attribuutit orange ja apple. Verkkoselain ei tällä hetkellä tunnista kyseisiä attribuutteja, mutta attribuutit löytyvät DOM-puusta. Ylläolevan esimerkin attribuutit eivät myöskään ole HTML-spesifikaatiossa. Attribuutit saadaan esille esimerkiksi seuraavalla tavalla.

```
var pelem = document.getElementsByTagName('p')[0];
var orange_apple = pelem.attributes;
```

Nyt orange_apple-muuttujasta löytyvät P-elementin attribuuttilista. Attribuuttilistalla ovat jollakin tavalla järjestettynä arvot orange ja apple.

Seuraavassa luvussa käsitellään selaimessa käytettävästä ohjelmointikielestä.

3.3 EcmaScript alias JavaScript

Yleisin selaimissa käytettävä ohjelmointikieli on EcmaScript, joka tunnetaan nimeltä JavaScript. Aikoinaan Netscape-yhtiö määritteli ja toteutti JavaScript-ympäristön selaimeen. Brendan Eich suunnitteli kielen kymmenessä päivässä ollessaan töissä Netscape yhtiössä. Ajan kuluessa JavaScriptin käyttö on levinnyt eri verkkoselaimiin ja moniin muihin paikkoihin. Eri selaimissa voi olla toteutuseroja JavaScript-tulkin toiminnallisuudessa. Microsoft Internet Explorer käyttää JScript-nimistä toteutusta, joka on käytännössä sama kuin JavaScript. Internet Explorer -selaimessa voi käyttää myös Microsoftin omaa VBScript-kieltä. Yleisesti ottaen selaimissa voidaan käyttää muitakin skriptikieliä joko erilaisten selainten liitännäisten niin sanottujen pluginien avulla tai käyttämällä JavaScript-kielellä toteutettua tulkkia. Tällöin JavaScript-kielinen ohjelmaa tulkitsee selaimessa esimerkiksi HTML-dokumentissa olevaa toisenkielistä ohjelmaa. Selaimissa yleisesti käytetty Adobe Flash -plugin, jolla toistetaan Adobe Flash -sisältöä, ohjelmoidaan ActionScript-kielellä. Tämä kieli pohjautuu EcmaScript-standardiin, joten ActionScript on kuin JavaScript. Selaimissa JavaScript-kieli on pienin askelin siirtymässä kohti nykyisestä EcmaScript 3 -standardista EcmaScript 5 -standardiin. Yleisesti käytetty EcmaScript 3 -standardin versio on 1.5. Uudempiakin versioita on kehitelty. JavaScript-tulkin versio on riippuvainen isäntäympäristössä. Versio saattaa vaihdella eri ohjelmissa, kuten verkkoselaimissa.

JavaScript-kieli on tulkattava moniparadigmainen ohjelmointikieli. Moniparadigmainen tarkoittaa, että JavaScript tukee useampaa ohjelmointityyliä. JavaScript on oliosuuntautunut, dynaaminen ja prototyyppipohjainen ohjelmointikieli.

Oliosuuntautunut- tai olio-ohjelmointi tarkoittaa ohjelmointityyliä, jossa käsitteitä mallinnetaan objektien avulla. Objekteilla on attribuutteja ja metodeja. Metodit käsittelevät attribuutteja. Objektien voidaan ajateltavan vastaavan reaalimaailman esineitä, kuten esimerkiksi omena. Omenalla on ominaisuuksia eli attribuutteja esimerkiksi väri tai koko. Dynaaminen tarkoittaa sitä, että JavaScript-kielessä tiedon tyyppi ei ole pysyvä. Se voi vaihtua ohjelman suorituksen aikana. Uudemmassa JavaScript-versiossa voidaan jokin muuttujan arvoa nimetä pysyväksi esimerkiksi **const**-määreellä. Tällöin myös muuttujan tyyppi pysyy samana.

Prototyyppipohjainen ohjelmointi on oliosuuntautunutta ohjelmointia, jossa objektit periytyvät toisista objekteista ja objektit sisältävät linkin toiseen objektiin. Kun esimerkiksi JavaScript-objektista *o* etsitään metodia, käydään ensin läpi objektin *o* metodit. Jos mitään ei löydy, siirrytään objektin *o.prototype* attribuutin osoittamaan objektiin ja jatketaan etsintää sieltä. Ketjua käydään läpi, kunnes jotain löytyy. Jos mitään ei löydy, annetaan virheilmoitus. Sitä voidaan kutsua myös luokista vapaaksi ohjelmoinniksi, koska siinä ei käytetä periytymisessä luokkia niin kuin esimerkiksi C++- tai Java-ohjelmointikielessä. JavaScript-kielessä on varattuna sanana **class**, mutta sitä ei tällä hetkellä käytetä missään. Se on varmuuden vuoksi varattu avainsana. Uudemmassa versiossa tätä käytetään syntaktisena sokerina (engl. syntactic sugar) objektien määrittelyssä. Olio-ohjelmoinnissa on kaksi koulukuntaa, jotka ovat luokka ja prototyyppi. JavaScript kuuluu jälkimmäiseen ryhmään, eli JavaScript on prototyyppikieli.

JavaScript-kielellä voi myös tehdä funktionaalista ohjelmointia. Funktiot ovat JavaScript-kielessä niin sanottuja ensimmäisen luokan kansalaisia. Tämä tarkoittaa sitä, että funktioita kohdellaan samalla tavalla kuin muita tietotyyppejä. Funktiot voivat olla esimerkiksi kutsuttavan funktion parametreissa. JavaScript tarvitsee isäntäympäristön, jonka palveluksia JavaScript-ohjelma käyttää. Tällainen isäntäympäristö voi olla esimerkiksi verkkoselain. Muitakin ohjelmistoalustoja on, kuten Node.js.

JavaScript ei tarjoa tällä hetkellä suoraan tapaa, jolla voisi ladata koodia turvallisesti ilman nimikonflikteja. Tämä hankaloittaa koodin jakamista itsenäisiin yksiköihin eli moduuleihin. Myös erilaisten riippuvuuksien käsittely voi olla hankalaa. Tähän voi

käyttää erilaisia kirjastoja, sovelluskehyksiä tai työkaluja. Työkaluohjelmia käytetään riippuvuuksien hallintaan. CommonJS on spesifikaatio palvelinpuolen JavaScript-moduulien rakentamiseen ja asynkroniseen lataamiseen. AMD eli Asynchronous Module Definition on määritelmä, jolla voidaan jakaa JavaScript-koodia ladattaviin moduuleihin ja hallita moduulien riippuvuuksia. Tämä on tarkoitettu selainpuolen ja palvelinpuolen käyttöön. RequireJS-kirjastolla voi käyttää JavaScript-kieltä AMD-määritelmän mukaisesti. Moduuli sanaa tässä ei pidä sekoittaa JavaScript:issa yleisesti tiedettyyn moduulimalliin (engl. module pattern), jossa moduuli on funktio. Tämä funktio määrittelee yksityiset muuttujat ja metodit. Tällainen moduulimalli voidaan luoda esimerkiksi seuraavalla tavalla. [10.]

```
var my_module = function ( ) {
           var private_value = 0;
           var private_function = function () { };
           return {
                 public_function: function ( ) {
                       console.log(private_value);
                 }
} ( );
```

Edellisessä esimerkissä käytetään sulkeumia, joiden avulla public_function voi käyttää muuttujaa private_value. Funktiossa public_function on pääsy ulommassa funktiossa määriteltyihin muuttujiin tai funktioihin.

On olemassa myös ohjelmointikieliä, joita generoimalla saadaan aikaan JavaScript-koodia. Yksi tällainen kieli on CoffeeScript. CoffeeScript-kieli on JavaScript-kieltä toisenlaisella syntaksilla. Ohjelmakoodi käännetään JavaScript-kielen koodiksi, joka voidaan liittää verkkosivulle. CoffeeScript-koodia voi ajaa myös suoraan verkkosivulta, jos verkkosivulle on asennettu JavaScript-kielinen ohjelma ymmärtämään CoffeeScript-koodia. Tätä ei kuitenkaan kannata tehdä, koska CoffeeScript-kielinen ohjelmakoodin tulkinta on riippuvainen ohjelmakoodin sisennyksistä. Tällainen ohjelmointikielen ominaisuus ei ole sopiva silloin, kun ohjelmakoodia siirrellään ympäri Internet-verkkoa. CoffeeScript-koodi itsessään on tiivimpää kuin JavaScript. Toisin sanoen CoffeeScript on hieman ilmaisuvoimaisempi kuin JavaScript. Lisäksi CoffeeScript-kielellä pyritään välttämään JavaScript-kielen karikot. Muitakin ohjelmointikieliä on, joissa alkuperäinen ohjelmointikieli käännetään JavaScript-kieleksi. On myös mahdollista kääntää C++-koo-

dia JavaScript-koodiksi. Tämä mahdollistaa esimerkiksi C++-kielellä tehdyn pelin kääntämisen ja ajamisen selaimessa käyttäen WebGL-grafiikkaa.

JavaScript-kieltä käytetään myös joissakin NoSQL-tietokannoissa hakukielenä SQL-kielen sijasta. NoSQL-nimeä käytetään yleisterminä sellaisista tietokannoista, jotka eivät ole relaatiotietokantoja.

JavaScript ei voi toimia, jos käyttäjä on kieltänyt selaimessa JavaScript-tulkin käytön. Selaimen lisäosat voivat estää tarkoituksellisesti JavaScript-tulkin käytön. On myös mahdollista, että HTML-sivun JavaScript-koodi ei ole latautunut. Tämä voi johtua esimerkiksi huonosta tietoliikenneyhteydestä tai JavaScript-koodia yritetään hakea väärästä verkko-osoitteesta, koska SCRIPT-elementtiin on laitettu väärä URL-osoite. JavaScript-koodin pitää olla myös virheetöntä. Muutoin JavaScript-tulkki ei osaa tulkata koodia. On myös mahdollista, että JavaScript-koodi on ristiriidassa JavaScript-kirjastojen kanssa. JavaScript-koodi voi olla myös ristiriidassa itsetehtyjen tai toisten tekemien JavaScript-koodien kanssa. Tämä johtuu globaalista nimialueesta, jonka takia kannattaa välttää turhaa globaalin nimialueen käyttöä. [11.]

JavaScript-koodia voidaan laittaa HTML-elementtien attribuutteihin, joihin laitetaan URL-arvo. Tämä voidaan laittaa kaikkiin niihin elementteihin, joissa selain tukee tätä toiminnallisuutta. Se on mahdollista, koska javascript on epävirallinen URI-skeema. Kirjainmerkki (engl. bookmark) tai linkki, jossa on JavaScript-koodia, kutsutaan bookmarkletiksi. Tarkoituksena on luoda verkkosivulle "yhden hiiren klikkauksen toiminnallisuuksia". Bookmarklet voidaan laittaa linkkiin eli HTML:n A-elementtiin. Tätä voi kokeilla kirjoittamalla selaimen osoitekenttään sopivat arvot. Linkkiin bookmarklet laitetaan seuraavalla tavalla. [12.]

```
<a href="javascript:console.log("YeeYee!");">
YeeYee</a>
```

Tämä ohjelman pätkä tulostaa tekstin "YeeYee!" verkkoselaimen konsoli-ikkunaan, kunhan vain verkkoselain tunnistaa JavaScript-koodin metodikutsun console.log. Parempi tapa voi olla laittaa JavaScript-koodi välittömästi suoritettavan anonyymiin funktioon (engl. immediate anonymous function). [12.]

```
<a href="javascript:(function ( )
      {
            console.log("YeeYee!");
      }( ));">
YeeYee</a>
```

Edellä olevaa koodia on muokattu niin, että se näyttäisi selkeämmän näköiseltä. Tavallisesti bookmarklet-koodi on kirjoitettu yhdelle riville. Selain tunnistaa protokollan ja tulkitsee koodin suoritettavaksi JavaScript-koodiksi. Ajettava skripti voi tutkia ja muokata senhetkistä verkkosivua. Suorituksen tuloksena saatava String-tyyppistä tulosta käytetään seuraavan verkkosivun luomiseen. Jos palautettava tyyppi on undefined, niin verkkosivun latausta ei tapahdu. Tämä mahdollistaa verkkosivulla tapahtuvan muutoksen ilman uudelleen latausta. Jos funktio ei palauta mitään, se palauttaa arvon undefined, paitsi jos kyseessä on konstruktorifunktio. Konstruktorifunktiolla luodaan objekteja, ja se voisi olla vaikka tämän näköinen.

```
var Student = function (name) {
            this.name = name;
};
```

Tällä tavalla voidaan luoda toinen objekti objektista Student, kun sitä kutsuttaisiin **new**-operaattorilla.

```
var lisa = new Student("Lisa Writer");
```

Ongelmia syntyy, jos Student-funktiota ei kutsuta **new**-operaattorilla. Silloin funktio muokkaisi tai lisäisi globaaleja arvoja. Yksi tapa välttää tämä on se, että ei käytä tällaista tapaa. Toinen tapa saattaisi olla se, että tunnistetaan, onko funktiota kutsuttu **new**-operaattorilla. Tämä tapahtuu seuraavasti.

```
function Student(name) {
            if (!(this instanceof Student)) {
                  return new Student(name);
            }

            this.name = name;
}
```

Esimerkki on otettu lähteestä [13, s. 46] hieman muokattuna. Tällä tavalla myös prototyyppiketju säilyy. Kolmas tapa on käyttää 'use strict'-merkkijonoa ilmaisemaan JavaScript-tulkille, että se käyttää strict mode -tilaa. Tällä tavalla ohjelmoija saa käyttöönsä "paremman version" JavaScript-kielestä, kunhan vain JavaScript-tulkki tukee

tätä tilaa. Konstruktorifunktiota kutsutaan siis **new**-operaattorilla, ja se palauttaa luotavan objektin. Vaikka konstruktorifunktion loppuun ei olisi laitettu **return**-lausetta, niin tämä konstruktorifunktio palauttaisi objektin **new**-operaattorin yhteydessä. Konstruktorifunktiot pitäisi nimetä isolla alkukirjaimella alkavalla nimellä. Tämä on vain nimeämiskäytäntö. JavaScript-kielessä kaikki ovat objekteja paitsi perustietotyypit null ja undefined. Funktiotkin ovat objekteja. Tämä tarkoittaa myös sitä, että objektia luotaessa ei tarvitse käyttää konstruktorifunktiota, koska lähes kaikki ovat objekteja. Konstruktorifunktio on vain yksi tapa luoda objekteja. Välittömästi suoritettavan anonyymin funktion käyttö on tässä tapauksessa suositeltavaa, koska sen suorittaminen palauttaa arvon undefined ja välittömästi suoritettava anonyymi funktio ei sijoita arvoja julkisesti näkyville muille "JavaScript-ohjelmille". Näkyvyysalue ohjelmakoodin suorituksessa olisi funktio. Välittömästi suoritettava funktio voi olla tämän näköinen.

```
(function ( ) {
          console.log("After evaluation I'm executed.");
} ( ));
```

Anonyymiksi funktion tekee se, että sillä ei ole nimeä. Näin on tämän koodiesimerkin tapauksessa. Funktion määrittely alkaa sanalla function, jonka jälkeen tulee funktion nimi. Tässä tapauksessa funktiolla ei ole nimeä. Funktio suoritetaan, kun sitä kutsutaan. JavaScriptissä funktiota voi kutsua muun muassa funktion nimellä tai funktion määrittelyn yhteydessä, kun funktion määrittelyn lopussa on sulkumerkit eli ( )-merkintä. Välittömästi suoritettavan funktiota voidaan kutsu myös antamalla funktiolle parametreja esimerkiksi seuraavalla tavalla.

```
var result = (function (val1, val2) {
          console.log(val1);
          console.log(val2);
} ('value1', 'value1'));
```

Tämä koodiesimerkki ei tee mitään järkevää. Esimerkissä tulostetaan isäntäympäristön konsoli-ikkunaan funktiolle annetut kaksi argumenttia. Muuttujan result arvoksi tulisi undefined, koska funktio "ei palauta mitään". Koska funktio ei palauta mitään arvoa, se palauttaa arvon undefined. Välitön suoritettava anonyymi funktio ei sijoita JavaScriptin käyttämälle globaalille nimialueelle arvoja ilman ohjelmoijan omia toimenpiteitä.

JavaScript-kielessä muutujien näkyvyysalue on funktio. Tämä on yksi niistä asioista, mikä mahdollistaa sulkeumien (engl. closure) käytön. Tällöin funktion sisäisellä funk-

tiolla on pääsy ulompana olevan funktion arvoihin. Muualla määritellyt muuttujat näkyvät globaalisti. Sulkeumia voisi käyttää esimerkiksi tällä tavalla.

```
function outer( ) {
           var value = 0;
           var inner = function ( ) {
                 value = value + 1;
           }
           return inner;
}
```

Funktio outer palauttaa funktion inner, jolla on pääsy muuttujaan value. Muuttuja value on määritelty funktiossa outer. Funktiota outer voisi käyttää tällä tavalla.

```
var counter = outer();
counter();
```

Funktio outer palauttaa funktion inner muuttujaan counter. Muuttujasta counter tulee funktio. Kun funktiota counter kutsutaan, kasvaa funktiossa outer oleva value-muuttuja yhdellä.

Uudemmassa JavaScript-versiossa on mahdollista sitoa muuttujan näkyvyysalue esimerkiksi **for**-silmukkaan **let**-määreellä tällä tavalla.

```
for (let i = 0; i < 10; i++) {
           console.log(i);
}
```

Tällä määritelmällä pystyy määrittelemään muuttujia ja rajaamaan näkyvyysaluetta lohkoon, käskyyn tai lausekkeeseen. Ohjelmakoodin lohko merkitään samalla tavoin kuin monissa muissakin ohjelmointikielissä {-merkillä ja }-merkillä. Ylläolevassa esimerkissä for-silmukan ohjelmalohkossa on käskyrivi "console.log(i);". Kun määritellään muuttujaa, näkyvyysalue on ympärillä oleva funktio tai globaalinäkyvyysalue. Kun muuttujaa esitellään käytetään avainsanaa **var**. Muuttujan määrittäminen ja muuttujaan sijoittaminen voivat tapahtua eri kohdissa ohjelmaa. Muuttujat siirretään ylös näkyvyysalueen alkuun. Tätä kutsutaan englanninkielisellä sanalla hoisting. Tämän takia pitää olla tarkkana, kun määrittelee muuttujia avainsanalla **var**. Sama tapahtuu myös funktioille. For-silmukan esittelyosaan ei saisi laittaa muuttuja esittelyjä. Muuttujat pitää esitellä esimerkiksi funktion alussa. Välittömästi suoritettavan anonyymin funktion ja tavallisen nimettömän funktion ero on se, että välittömästi suoritettava anonyymi

funktio suoritetaan ilman erillistä kutsua. Funktio voi olla myös nimetty ja välittömästi suoritettava. Tällainen funktio voisi olla seuraavanlainen.

```
(function box() {
          console.log("Fritz the Cat.");
}());
```

Välittömästi suoritettavan anonyymin funktion tai välittömästi suoritettavan nimetyn funktion nimi tai viite katoaa tällä tavalla. Eli tähän funktioon on vaikeata viitata, koska sen viitettä ei löydy globaalista nimiavaruudessa. Yleensä JavaScript-koodia ei kannata liittää suoraan HTML-elementteihin, koska se hankaloittaa koodin ylläpitoa ja testaamista. Ehkä bookmarklet on erikoistapaus.

Yksi tapa tehdä verkkosovellusta on ohjelmalogiikan siirtäminen selaimen puolelle. JavaScript-kielellä tai selaimen ymmärtämällä skriptikielellä tehdään verkkosovellusta, joka toimii enimmäkseen verkkoselaimessa. Palvelinpuolen ohjelmassa tehdään mahdollisimman vähän. JavaScript-kieltä voidaan käyttää myös palvelinpuolella. Toisaalta selaimeen voidaan myös ladata Java-sovelmia (engl. Java Applet), mutta tämä ei ole yleinen tapa. Danske Bankin verkkopankissa käytetään Java-sovelmaa. Selaimeen voi ladata myös muita selaimen liitännäisen (engl. plugin) tukemia ohjelmia.

Toinen tapa tehdä verkkosovellusta on, että ohjelman toimintalogiikka ei siirretä selaimeen. Palvelimessa generoidaan valmiita sivuja selaimelle. Sovellukset ovat erilaisia tai toteutukset ovat erilaisia. Toteutukset voivat olla edellä mainittujen tapojen yhdistelmiä. Lisäksi on olemassa koontisovelluksia (engl. mashup), jossa yhdistetään tietolähteitä muista palveluista ja yritetään luoda uusi verkkosovelluskokonaisuus. Tämä mahdollistaa sovelluksen luonnin ilman omaa palvelinpuolen ohjelmaa ja selaimessa on kaikki sovelluksen ohjelmakoodit. Tällä hetkellä eri tietolähteistä tietojen siirtäminen ei ole helppoa, joten verkkosovelluksessa voidaan tarvita palvelinpuolen ohjelmaa, joka muokkaa tietoa ymmärrettävämpään muotoon selainpuolen ohjelman käyttöön. Se minkälainen koontisovelluksen toteutus on, riippuu kyseisestä verkkosovelluksesta. Palvelinpuolen ohjelman ja selainpuolen ohjelman käyttäminen on tapauskohtaista. Koontisovelluksessa kuitenkin selainpuolen ohjelma koostaa sivulle toiminnallisuuksia kolmannenosapuolen tietolähteistä tai pienistä verkko-ohjelmista. Ulkopuolisen toiminnallisuuden tai tietolähteen liittäminen voi olla myös tietoturvallinen ongelma, johon pi-

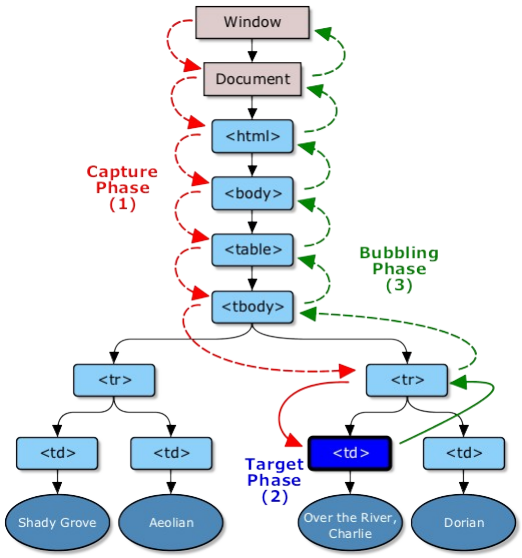
tää kiinnittää huomiota sovellusta tehdessä. Ulkopuolisen JavaScript-koodin liittäminen voi olla erittäin vaarallista.

JavaScript-kieltä käytetään verkkosivujen käyttöliittymissä. Tästä kirjoitetaan seuraavassa luvussa.

3.4 Käyttöliittymä

Käyttöliittymän ohjelmointi on tapahtumapohjaista. Suuremman osan ohjelmansuoritusajasta odotetaan käyttäjän toimenpiteitä. Käyttäjän aiheuttamat tapahtumat, kuten hiiren napin painaminen, voivat tulla milloin tahansa. Silloin, kun jotain tapahtuu, ohjelman pitää reagoida tarpeeksi nopeasti käyttäjän aiheuttamaan tapahtumaan. Verkkoselaimen verkkosivun käyttöliittymän ohjelmointi on tällaista. Tapahtumasilmukassa käydään tapahtumia läpi ja kutsutaan tapahtumia seuraavia kuuntelijoita. Kun käyttäjä painaa hiiren napilla linkkiä, syntyy tapahtuma. Tämä tapahtuman etsintä (engl. event capture) lähtee DOM-puun juuresta ja päättyy tapahtuman lähteeseen. Tässä vaiheessa tapahtuma voidaan ottaa käsittelyyn ennen kuin se saapuu tapahtuman aiheuttajaan. Tämän jälkeen lähdetään kulkemaan takaisin kohti DOM-puun juurta niin pitkälle, kunnes tapahtuma käsitellään. On myös mahdollista, että tapahtumaa ei käsitellä lainkaan. Paluuvaihetta kutsutaan englanninkielisellä termillä event bubbling.

Kuvassa 3 kuvataan tapahtuman käsittelyä. Lisäksi kyseisessä kuvassa on vielä yksi vaihe etsintä- ja kuplintavaiheen eli paluuvaiheen välissä. Tämän vaiheen nimi on kohdevaihe (engl. target phase), joka tapahtuu silloin, kun saavutaan tapahtuman aiheuttaneeseen elementtiin ja aloitetaan kulkemaan kohti DOM-puun juurta. Kohdevaiheessa saadaan tapahtuman aiheuttanut kohde (engl. target). Kaikissa selaimissa ei ole toteutettu etsintä vaihetta tai siihen ei ohjelmallisesti pääse käsiksi. Paluuvaihe yleensä on. Selaimen sovelluksessa tähän vaiheeseen yleensä liitetään tapahtumankäsittelijä, jos selaimen ohjelmassa on tarpeellista käsitellä DOM-puun solmun tapahtumia. Ohjelmoija voi itse valita kumpaan vaiheeseen tapahtuman käsittelijän liittää. Molempiinkin vaiheisiin voi liittää tapahtuman käsittelijän, jos ohjelmoija niin haluaa. Tapahtumankäsittelijöiden suoritusjärjestys HTML-sivun sisäkkäisissä elementeissä on riippuvainen siitä, kumpaan vaiheeseen tapahtumankäsittelijät on liitetty.



Kuva 3: Tapahtuman käsittely DOM-puussa. Kuva otettu suoraan lähteestä [26].

Jos tapahtumankäsittelijän viite katoaa, syntyy niin sanottu elävä kuollut (engl. zombie) tapahtuman käsittelijä. Tällaisia tapahtumankäsittelijöitä on syytä varoa, koska ne voivat haitata ohjelman toimintaa. Elävät kuolleet tapahtumankäsittelijät voivat hidastaa selaimen ohjelmaa kuluttamalla muistia tai suoritustehoa. Kun esimerkiksi HTML-elementti poistetaan ja siitä ei ole poistettu tapahtumankäsittelijää niin, silloin syntyy elävä kuollut tapahtumankäsittelijä, jos viite tapahtumankäsittelijään ei ole tallessa.

Räätälöity HTML-elementtien kokonaisuus HTML-dokumentissa on web-komponentti. Verkkosivulla voidaan luoda komponentti, joka koostuu HTML-sivusta, CSS-tyylisivusta ja JavaScript-koodista. HTML-sivu luo komponentin rakenteen, CSS-tyylisivu komponentin ulkonäön ja komponentin HTML-elementtien sijainnin. JavaScript-koodi luo komponentin toiminnallisuuden. Kun näitä kolmea osaa pidetään erillään tiedostoissa ja muutenkin erillisinä, saadaan aikaan itsenäinen komponentti. Tällainen komponentti liitetään verkkosivuun kopioimalla komponentin HTML-koodi sopivaan paikkaan verkkosivulla, lisäämällä komponentin tyylisivu verkkosivulle LINK-elementin avulla ja lisäämällä komponentin JavaScript-koodi verkkosivulle SCRIPT-elementin avulla. Tästä toimintatavasta voi seurata ongelmia. Kopioi-liimaa-tekniikalla komponentin liittäminen voi olla virhealtista. Komponentin CSS-tyylitiedosto ja liitettävään verkkosivuun liittyvä

oma CSS-tyylimäärittelyt voivat aiheuttaa epätoivottuja verkkosivun tai komponentin ulkoasun muutoksia. Komponentin toiminnallisuudesta huolehtiva JavaScript-koodi tai verkkosivun oma JavaScript-koodi voivat haitata toisiaan. Verkkosivuun liitetty komponentti voi olla toimimaton tämän takia.

Selaimen käyttöliittymäkehyksillä on omia ratkaisuja widgeteille. Tällaisia käyttöliittymäkehyksien widgettejä ovat muun muassa Dojo widgets, YUI widgets ja JQuery plugins.

Web Component on mahdollinen komponenttistandardi Web-sivuille. Web Component koostuu tällä hetkellä seuraavista osista:

- Templates
- Decorators
- Custom Elements
- Shadow DOM
- Imports.

Templates on sivupohjamalli, joka sisältää kasan HTML-elementtejä. Nämä elementit voidaan aktivoida myöhemmin. Sivupohjamallit toimivat osakokonaisuutena HTML-sivulla muodostaen näkyvän elementtikokonaisuuden aktivoinnin jälkeen. Yleensä dynaamiset verkkosovellukset käyttävät sivupohjamalleja muodostaessaan esimerkiksi HTML-sivujen osia. Myös selaimissa toimivissa ohjelmissa käytetään sivupohjamalleja samoihin tarkoituksiin. Templates on määritelmä sivupohjien käyttöön verkkoselaimissa. Kun käytetään TEMPLATE-elementtiä, niin sen sisällä olevat elementit eivät aiheuta aktivointia. TEMPLATE-elementin sisällä voi olla JavaScript-koodia, joka jäsennetään (engl. parse) vasta TEMPLATE-elementin aktivoinnin jälkeen. [14.]

Decorators on määritelmäluonnos, joka ei ole tällä hetkellä tuettuna missään selaimessa. Tämä nimi tulee Decorator-suunnittelumallista, jossa valmiiseen objektiin saadaan lisättyä uutta toiminnallisuutta. Decoratos parantaa tai vaihtaa elementin ulkonäköä. [14.]

Custom Elements eli itseluodut elementit, joita voidaan käyttää HTML-sivulla. Itseluotujen elementtien alkutoimenpiteiden jälkeen, itseluotu elementti voisi olla seuraavanlainen. [14.]

```
<my-experiment></my-experiment>
```

Tätä my-experiment -elementtiä voisi käyttää muun HTML-koodin seassa verkkodokumentissa.

Shadow DOM eli varjo DOM piilottaa DOMiin alipuun, jossa on HTML-elementtejä. Shadow DOMiin voidaan lisätä HTML-elementtejä, jotka ovat eivät näy suoraan DOM-puussa. HTML-elementin solmuihin liitettävän Shadow DOMin mukana tulee uudenlainen solmu. Tällaisen solmun nimi on shadow root. Elementti, johon on liitetty shadow root, kutsutaan shadow hostiksi. Shadow root -elementtiin voi lisätä muita HTML-elementtejä. Vain shadow rootin sisältö piirretään selaimessa. [14.]

Imports eli sisällyttäminen määrittää, miten kaikki edelliset osat paketoidaan ja tuodaan HTML-sivulle "selaimen käyttöön" [14].

Web Component on keskeneräinen spesifikaatio, joka saattaa olla tulossa käyttöön. Osa spesifikaation määrittelemistä osioista on käytössä uudemmissa selaimissa. Web componentin avulla kehittäjä voisi luoda omia HTML-elementtejä toisin sanoen laajentaa HTML-sanastoa ohjelman käyttöön. Ohjelmoija ei olisi riippuvainen esimerkiksi selainvalmistajien toimista. Tällainen tapa saattaa auttaa kehittäjää, joka on tekemässä käyttöliittymäkehystä. Shadow DOM on piilotettu näkyvistä selaimissa, vaikka uudet HTML-elementit käyttävät sitä, kuten esimerkiksi video-elementti uuden version HTML-verkkodokumentissa. Kun tulevaisuus on sanotusti epävarmaa, voidaan joitakin edellä mainittuja tekniikoita käyttää selaimilla. Tämä tapahtuu selainten ominaisuuksia täydentämällä.

4 Tiedonvälitystapoja HTML-elementeillä

Kun verkkoselain kutsuu palvelinta ja pyytää verkkosivun, palvelin lähettää vastauksena verkkosivun. Tämän jälkeen verkkoselain kutsuu verkkosivun osia palvelimelta yksittäin tai rinnakkain useampaa osaa kerrallaan. Osa kutsuista ovat kaikesta huolimatta niin sanotusti estäviä eli blokkaavia. Tällöin verkkoselain kutsuu yksittäisiä verkkosivunosia.

Verkkosivulta voidaan kutsua palvelinta monella tavalla. Esimerkiksi IMG-elementin src-attribuuttilla, A-elementillä, IFRAME-elementillä, FORM-elementillä, SCRIPT-elementin src-attribuuttilla, XMLHttpRequest-objektilla ja WebSocketilla.

Muita tapoja ovat link-elementit, jotka muun muassa määrittävät CSS-tyylitietojen paikan tai sivun ikonin selaimeen (engl. favicon). Tyylitietoja voidaan myös suoraan määritellä HTML-sivulle STYLE-elementillä tai suoraan kirjoittamalla elementtiin. Jälkimmäistä tapaa ei pidetä hyvänä käytäntönä. Monia muitakin elementtejä on, joissa selain tekee pyynnön palvelimelle. Seuraavassa luvussa kirjoitetaan kuvaelementistä.

4.1 IMG-elementti

HTML-sivulla voi olla alla olevan koodin kaltainen IMG-elementti.

```
<img src="http://www.myservice.org/web/myimage.png">
```

Tämä elementti lataa kuvan verkkosivulle. Kun verkkoselain kohtaa tämän elementin, niin verkkoselain lähettää pyynnön palvelimelle ja pyytää kyseistä kuvaa. Selaimen lähettämä pyyntö on GET-pyyntö. Tarvittavat tiedot selain löytää IMG-elementin src-attribuutista. Palvelin käsittelee pyynnön ja lähettää onnistuneen käsittelyn jälkeen vastauksena kuvan takaisin.

IMG-elementin src-atribuutti voi osoittaa paikkaan, jossa on palvelimessa ajettava ohjelma. Pyynnön yhteydessä selain suorittaa tämän skriptin. Joissakin toteutuksissa palautetaan yhden pikselin kokoinen kuva. Hyvänä toimintatapana on ilmoittaa, että sisältö on tyhjä. Tämä onnistuu laittamalla vastauksen Content-header-otsikkokentän arvoksi 0.

Tätä käytetään esimerkiksi käyttäjien seurantaan. HTML-sähköpostissa voi olla tällaisia kuvaelementtejä. Kun käyttäjä avaa sähköpostiviestin, niin samalla HTML-sähköpostissa olevat kuvat latautuvat.

HTML-elementin src-attribuutin skeema data mahdollistaa tiedon suoran lisäämisen elementtiin seuraavalla tavalla [15].

```
<img src="">
```

Yllä olevan esimerkin kuvatietoa on lyhennetty, jotta kuvatieto ei veisi liikaa tilaa. Varsinainen kuvatieto alkaa pilkun jälkeen, joka on sanan base64 jälkeen. Tällä tavalla kuvaelementti ladataan heti palvelimen lähettämän ensimmäisen vastauksen aikana. Elementtiin voi liittää tietoa myös ilman MIME-tyypin määrittämistä esimerkiksi HTML-koodia tai JavaScript-koodia. Tämä skeema ei toimi jokaisessa HTML-elementissä. Lisäksi

selaimet määrittävät liitettävän tiedolle maksimikoon eivätkä vanhemmat selaimet tunnista tätä skeemaa.

Seuraavassa luvussa käsitellään linkkielementtiä.

4.2 A-elementti

Linkit HTML-sivulle laitetaan A-elementillä seuraavalla tavalla.

```
<a href="http://www.serv.org/mypage.html">Linkki</a>
```

Kun käyttäjä painaa verkkosivulla linkkiä Linkki, selain hakee GET-pyynnöllä linkin määrittämän sivun. Linkissä voi olla myös parametreja ja ankkuri seuraavalla tavalla.

```
<a href="http://www.s.fi/p.html?p1=a&p2=2#myanchor">Link</a>
```

Parametrit ovat osoitteen loppuosassa. Ne alkavat ?-merkillä, ja parametrit erotellaan toisistaan &-merkillä. Nämä parametrit laitetaan lähetettävän pyynnön otsaketietoihin osana haettavaa osoitetta, koska kyseessä on GET-pyyntö. Ankkuriosaa ei välitetä eteenpäin otsaketiedossa. Se on kohta, mihin siirrytään sivulla. Tämä ankkuri-merkintä pitää olla verkkosivulla esimerkiksi seuraavalla tavalla, jos halutaan käyttäjän pystyvän siirtymään kyseiseen kohtaan.

```
<a name="myanchor">myanchor</a>
```

Niin sanotuissa yhden sivun sovelluksissa (engl. SPA, single page application) ankkuria saatetaan käyttää "sivunavigoinnin" apuna. Tämän merkinnän avulla tallennetaan sivuhistoriaa. Merkintää on vain aloitettu käyttämään yhden sivun sovelluksissa. Tällä tavalla sivun linkki voidaan laittaa kirjamerkiksi ja palata linkin osoittamaan kohtaan ohjelmassa. Parempi tapa voisi olla käyttää History API -ohjelmistorajapintaa.
Hakukone ei löydä sivua, jos palvelin ei vastaa annettuun linkkiin. Tämä saattaa olla pienoinen ongelma. Tätä varten on voitu rakentaa erilaisia järjestelmiä. Tähän saatetaan käyttää ikkunattomia selaimia (engl. headless browser), jotka antavat toimintansa tuloksena haettavan sivun esimerkiksi kuvana tai HTML-koodina.
Seuraavassa luvussa kirjoitetaan IFRAME-elementistä.

4.3 IFRAME-elementti

IFRAME-elementillä voidaan sisällyttää HTML-dokumenttin sisään toinen HTML-dokumentti. IFRAME-elementti merkitään seuraavalla tavalla.

```
<iframe src="http://www.serv.org/index.html">
</iframe>
```

Elementti saa aikaan GET-pyynnön src-attribuutin ilmoittamaan URL-osoitteeseen. Elementti saa aikaan itsenäisen kehyksen, joka sisällyttää toisen verkkosivun alkuperäiselle verkkosivulle. IFRAME-elementissä oleva sivu on oma itsenäinen kokonaisuutensa. Se verkkosivu, johon sisällytettiin IFRAME-elementti, on nimeltään parent. IFRAME-elementistä on yhteys isäntäikkunaan window.parent-viitteen kautta. Saman alkuperän käytäntö vaikuttaa IFRAME-elementissä olevaan skriptiin ja isäntäikkunan skriptiin.

Seuraavassa luvussa käsitellään lyhyesti lomake-elementtiä.

4.4 FORM-elementti

FORM-elementti luo HTML-lomakkeen verkkosivulle. Lomake näyttää seuraavanlaiselta.

```
<form method="post" action="myform.pl">
          Name: <input type="text" name="myname">
          <input type="submit" value="Send me">
</form>
```

FORM-lomakkeella voidaan lähettää tietoa GET- tai POST-metodilla palvelinohjelmalle. GET-metodia käytettäessä lomakkeesta saadut tiedot lisätään URL-osoitteeseen. HTTP-protokollan mukaan GET-metodin pitäisi olla idempotentti. POST-metodia käytettäessä lomakkeesta saadut tiedot lisätään pyynnön viestikenttään. Tiedon koodaus on riippuvainen Content-type-otsaketiedosta. Tämän määrittää FORM-elementin enctype-attribuutti.

Seuraavassa luvussa käsitellään elementtiä, jonka avulla liitetään verkkosivuun toiminnallisuus.

4.5 SCRIPT-elementti

HTML-koodin SCRIPT-elementtillä liitetään JavaScript-koodi verkkosivulle. SCRIPT-elementtiin voidaan liittää myös jokin muu selaimen ymmärtämä ohjelmakoodi tai ohjelmakoodi voi olla SCRIPT-elementin sisällä. Verkkoselain hakee JavaScript-koodin SCRIPT-elementin src-attribuutin osoittamasta paikasta GET-pyynnöllä. Latauksen jälkeen selain tulkitsee ja suorittaa JavaScript-koodin. JavaScript-koodit suoritetaan sii-

nä järjestyksessä, kun ne verkkosivulla tulevat verkkoselaimelle tulkittavaksi. SCRIPT-elementin src-attribuuttiin ei vaikuta verkkoselaimen saman alkuperän käytäntö. Tämä tarkoittaa sitä, että verkkoselain voi hakea JavaScript-koodia mistä tahansa verkko-osoitteesta. Tällaisella JavaScript-koodilla on samanlaiset oikeudet kuin verkkosivun muilla JavaScript-koodeilla. Tämä on erittäin vaarallista verkkosivun turvallisuuden kannalta ja myös erittäin kätevää. Tällaista tapaa ei ole syytä käyttää. On parempi ladata JavaScript-koodit samalta verkkoalueelta, kuin mistä verkkosivu ladattiin.

JSONP eli JSON with Padding on tapa lähettää pyyntö eri verkkoalueeseen. Vastauksena saadaan JSON:n mukaista tietoa. JSONP:ssä laitetaan SCRIPT-elementin src-attribuutin verkko-osoitteeksi, mikä tahansa verkko-osoite. Vastauksena saadaan JSON-muotoista tietoa. Annettuun verkko-osoitteeseen laitetaan kyselyparametriksi callback tai jsonp. Tämän kyselyparametrin arvoksi laitetaan verkkoselaimessa sijaitseva JavaScript-funktio. Tämä funktio käsittelee vastauksena saadun JSON-tiedon. JSON eli JavaScript Object Notation on kevytrakenteinen ja kieliriippumaton tiedonkuvaus. JSON-merkintää on ihmisten helppo kirjoittaa ja lukea, koska se on tekstimuotoista tietoa. Tietokoneohjelmien on helppo käsitellä ja generoida JSON-tietoa. Muita tapoja kuvata tietoa tekstimuotoisesti on XML. JSON on XML-kuvauskieleen verrattuna kevytrakenteisempi. XML eli extensible markup language on merkintäkieli tai standardi dokumenttien ja rakenteisen tiedon kuvaukseen. JSON perustuu JavaScript-kielistandardin osajoukkoon. Syntaksiltaan JSON-tiedonkuvaus on samanlainen kuin JavaScript-kielen objektien luontitapa. Tämä tapa on vain eräs tapa luoda JavaScript-kielessä objekteja. Tämä tiedonvaihtoon tarkoitettu tiedonkuvaus perustuu kahteen kokonaisuuteen. Nämä ovat nimi-arvo-parien kokoelmat ja järjestetyt eri arvojen listat. JSON näyttää seuraavanlaiselta. [16.]

```
{
          "product": "apple",
          "quantity": 25,
          "categories": [1, 2, 3]
}
```

JSON-tietoa käytetään verkkopalvelimen ja verkkoselaimen väliseen tiedonsiirtoon. SCRIPT-elementti voi olla seuraavanlainen uudemman HTML-standardin mukaan.

```
<script src="http://www.m.fi/serv?callback=handleResponse">
</script>
```

Kun SCRIPT-elementti aktivoituu, tehdään pyyntö verkko-osoitteeseen www.m.fi/serv. Tästä kutsusta tulevan JSON-tiedon saatuaan verkkoselain kutsuu verkkoselaimessa olevaa JavaScript-funktiota handleResponse. Yleensä JSONP:ssä SCRIPT-elementti luodaan dynaamisesti. Tämä tarkoittaa, että HTML-sivulle luodaan uusi SCRIPT-elementti. Kun verkkoselain on käsitellyt tämän SCRIPT-elementtin, tapahtuu GET-pyyntö SCRIPT-elementin src-attribuutin määrittelemään verkko-osoitteeseen. Silloin kun näin tehdään jatkuvasti, saadaan luotua pyyntö ja vastaus ketju verkkoselaimen ja verkkopalvelimen välille.

JSONP-kutsuissa ei ole virheidenkäsittelyä. Jos JavaScript-koodin lisääminen onnistuu, niin kyseinen koodi suoritetaan. Virhetilanteessa mitään ei tapahdu. Pyyntöä ei voi keskeyttää tai aloittaa uudelleen. Kuten aikaisemmin todettiin, JSONP voi olla vaarallinen tekniikka, jos vastausviesti saadaan epäluotettavasta lähteestä. JSONP-palvelu palauttaa JSON-tiedon käärittynä funktion kutsuun ja tätä funktiota kutsutaan verkkoselaimessa. JSONP altistaa verkkosovelluksen monenlaisille hyökkäyksille. [17.]

JavaScript-ohjelma voi käyttää sivupohjia selainpuolen ohjelmassa. Näihin sivupohjiin generoidaan haluttuja arvoja samalla tavalla kuin palvelinpuolen sivujen generoinneissa. Palvelinpuolen sivujen generoinnit voivat tapahtua myös JavaScript-kieltä käyttämällä. Yksi tapa sivupohjien käytölle on piilottaa elementtikokonaisuus CSS-tyylimäärittelyn avulla. Piilotettavan elementin sisällä on sivupohjassa käytettäviä osia. Tällainen piilotettava elementti voisi olla esimerkiksi DIV-elementti. Tämä tapa ei välttämättä ole hyvä. Toinen tapa sivupohjien käytölle on JavaScript-kirjastojen tai sovelluskehyksien käyttämä SCRIPT-elementti. SCRIPT-elementin sisälle voidaan laittaa HTML-elementtejä. Selaimen tulkinta sisällöstä riippuu SCRIPT-elementin type-attribuutin arvosta. Itse asiassa SCRIPT-elementin sisälle voidaan laittaa, mitä tahansa sisältöä. SCRIPT-elementin käyttäminen on tietynlainen ohjelmointikikka (engl. hack), parempi tapa voisi olla käyttää esimerkiksi TEMPLATE-elementtiä. Monissa JavaScript-kirjastoissa käytetään sivupohjina SCRIPT-elementtiä. Niissä SCRIPT-elementin type-attribuutin arvona käytetään erilaisia arvoja. Kun SCRIPT-elementtiä käytetään sivupohjana, on tärkeätä käyttää type-attribuutin arvona muuta kuin tavallisesti käytettävää text/javascript-arvoa. Kaikkia muitakin type-attribuutin arvoja pitää välttää, joilla selain tulkitsee

SCRIPT-elementin sisällön JavaScript-kieleksi. Attribuutin type arvona voidaan käyttää esimerkiksi "text/html", jolloin sisältöä ei tulkita JavaScript-koodiksi.

JavaScript-koodin voi laittaa suoraan HTML-koodin sekaan esimerkiksi SCRIPT-elementin sisälle. Parempi tapa on laittaa JavaScript-koodi erilliseen tiedostoon ja hakea se palvelimelta. Suositeltavin tapa laittaa SCRIPT-elementti HTML-dokumenttiin, on laittaa se juuri ennen BODY-elementin loppuelementtiä. HTML-elementti SCRIPT estää muiden verkkosivulla olevien osien lataamisen. Muut osat ladataan vasta sen jälkeen, kunnes JavaScript-koodi on ladattu, tulkattu ja suoritettu. Tämän takia on parempi sijoittaa SCRIPT-elementti sivun loppuun, jotta esimerkiksi verkkosivulla olevien kuvien lataaminen ei estyisi. Käyttäjä kokee tällä tavalla sivun latautuvan nopeammin.

5 Muita tiedonvälitystapoja

Seuraavissa luvuissa on kirjoitettu selaimesta löytyvästä tiedonsiirto-objektista, soketista ja reaaliaikaisesta kommunikaatiossa. Lisäksi seuraavissa luvuissa on kirjoitettu palvelimen lähettämistä viesteistä ja dokumenttien välisestä tiedonsiirrosta. Aloittakaamme tutustuminen tiedonsiirto-objektiin, josta kirjoitetaan seuraavassa luvussa.

5.1 Tiedonsiirto-objekti

XMLHttpRequest-objekti on turvallisempi ja luotettavampi tapa käsitellä HTTP-protokollan pyyntöjä ja vastauksia kuin SCRIPT-elementti. Alunperin XMLHttpRequest-objektin kehitteli Microsoft. Tämän jälkeen se on levinnyt muidenkin selainvalmistajien selaimiin. Tällä hetkellä se on W3C:n standardoima. XMLHttpRequest-objekti löytyy ECMAScriptin HTTP API:sta. API on ohjelmointirajapinta, joka on tarkoitettu ohjelmoijien käyttöön. Ohjelmointirajapinta piilottaa rajapinnan takana olevan ohjelmakoodin. Ohjelmakoodia käytetään ohjelmointirajapinnan määrittelemällä tavalla. XMLHttpRequest-objekti mahdollistaa tiedonsiirron ilman verkkosivun päivitystä. Verkkoselaimet voivat kuitenkin rajoittaa XMLHttpRequest-objektien yhteyksien määrää. Saman alkuperän käytäntö koskee myös XMLHttpRequest-objektia. CORS mahdollistaa pyynnön lähettämisen muihin verkko-osoitteisiin. Nimi on yhteensopivuussyiden takia valittu XMLHttpRequestiksi, vaikka jokainen osa nimessä on harhaanjohtava. Objekti tukee tekstipohjaista tiedonsiirtoa mukaan lukien XML. Pyynnöt voivat olla HTTP/HTTPS-protokollan mukaisia ja lisäksi pyynnöt ovat HTTP-protokollan metodeja [2].

XMLHttpRequest-objekti tukee myös binääripohjaista tiedonsiirtoa ja joissakin selaintoteutuksissa voidaan käyttää muitakin protokollia kuin HTTP/HTTPS. HTTP-pyynnöt ovat joko asynkronisia tai synkronisia. Erään tunnetun JavaScript-gurun mukaan XMLHttpRequest-nimitys on esimerkki huonosta nimeämisestä. Hieman parempi nimitys voisi olla esimerkiksi DataTransferRequest. Erilaisissa kirjoituksissa XMLHttpRequest-nimi on lyhennetty kolmikirjaimiseksi nimeksi XHR. Mutta tässä kirjoituksessa ei käytetä tuota nimeä. [19.]

Kun halutaan lähettää lomaketietoja XMLHttpRequest-objektilla, pitää lomakkeesta lähetettävät tiedot mahdollisesti liittää URL-osoitteeseen JavaScript-koodin avulla. Eli lomakkeesta lähetettävistä tiedoista on muodostettava nimi-arvo-pareja pyynnön URL-osoitteeseen tai pyynnön viestiosaan. Tämä riippuu käytettävästä HTTP-protokollan metodista, joita ovat GET tai POST. Muitakin HTTP-protokollan metodeja voidaan käyttää tai itse luotuja HTTP-protokollan mukaisia metodeja. On myös mahdollista, että lähetettävän pyyntöön on lisättävä sopivia otsikkokenttiä ja sovelluskohtaisia otsikkokenttiä. Yksi yksinkertaisempi tapa käsitellä lomaketietoja on käyttää FormData-objektia. Tämän avulla voidaan luoda avain-luku-pareja XMLHttpRequest-objektin käyttöön. [19.]

Tämän jälkeen käsittelyvuorossa on soketti, josta kirjoitetaan seuraavassa luvussa.

5.2 Soketit

Kaikkia niitä tekniikoita, joilla saadaan palvelin lähettämään tietoa silloin, kun palvelimella on jotain lähetettävää verkkoselaimelle, kutsutaan englanninkielisillä termeillä Comet tai Push. Niitä tekniikoita, joilla verkkoselain voi vastaanottaa tietoa ilman sivunpäivitystä kutsutaan termillä Ajax. Ajax-teknologiaan liittyy XMLHttpRequest-objektin käyttö. WebSocket on selaimelle tarkoitettu protokolla. Tämä mahdollistaa selaimen ja palvelimen välisen kaksisuuntaisen tiedonsiirron. Protokollan on standardoinut IETF vuonna 2011. Standardi on nimeltään RFC 6455. WebSocket API:n on W3C:n standardoima [20.]

WebSocket käyttää HTTP-protokollaa tiedonsiirtokerroksena. Se on itsenäinen TCP-pohjainen protokolla. Ainut yhteys HTTP-protokollaan WebSocket-protokollalla on se, että kättely tulkitaan palvelimessa päivityspyyntönä. Päivityspyynnössä pyynnön otsaketiedossa on otsake Upgrade. Tämän protokollan URI-skeemoja ovat suojaamat-

tomassa yhteydessä käytetty ws ja suojattussa yhteydessä käytetty wss. Skeeman ws:n oletusporttina on 80 ja skeeman wss:n käyttämä portti on 443. Kättelyn jälkeen "ei käytetä enää" HTTP-protokollaa. Tiedonsiirto tapahtuu suoraan valmiiksi avattuun TCP-sokettiin kirjoittamalla WebSocket-protokollan määrittämiä kehyksiä (engl. frame). TCP-soketti aukeaa silloin kun HTTP-yhteys muodostetaan ja HTTP-yhteys käyttää TCP-sokettia tiedonsiirrossaan. WebSocket on matalan tason protokolla. [21.]

WebSocket-spesifikaatio määrittelee API:n, jolla muodostetaan soketti verkkoselaimen ja palvelimen välille. Verkkoselaimen saman alkuperän käytäntö ei rajoita WebSocket-yhteyttä eli yhteys voidaan muodostaa mihin tahansa verkko-osoitteeseen. Palvelimessa yhteydenottoa rajoitetaan Origin-otsaketiedolla. Tätä tietoa käyttämällä palvelin valitsee sallitut yhteydenotot. Verkkoselaimet saattavat rajoittaa WebSocket-yhteyksien määrä. Kaikki verkkoselaimet eivät tue WebSocket-yhteyttä, mutta on mahdollista yrittää täydentää verkkoselaimen ominaisuuksia sopivalla JavaScript-kirjastolla. Yksi tällainen on socket.io-kirjasto. Tätä kirjastoa on mahdollista käyttää verkkoselaimen ohjelmassa tai palvelinohjelmassa. Kun WebSocket-ominaisuutta ei löydy, yrittää socket.io-kirjasto käyttää parasta mahdollista seuraava vaihtoehtoista yhteyttä. Tämä jatkuu kokeillen erilaisia tapoja yhteyden muodostamiseen. Yhteyden muodostaminen voi kokeilun tuloksena onnistua tai epäonnistua. Mahdolliset WebSocket-yhteyden välissä olevat välityspalvelimet (engl. proxy server) eivät välttämättä ymmärrä selaimen lähettämää päivityspyyntöä. Tämän vuoksi välityspalvelin voi katkaista yhteyden. [20.]

WebSocketien käyttö luo uudenlaisen käyttötavan palvelinpuolen sovelluksille. Perinteiset palvelinpuolen ohjelmat on suunniteltu HTTP-protokollan pyyntöjen ja vastauksien ympärille. Nämä eivät aina toimi usean WebSocket-yhteyden kanssa. Usean yhteyden ylläpito vaatii sellaista arkkitehtuuria, joka on korkeasti rinnakkainen pienellä tietokoneresurssien kulutuksella. Tällaiset arkkitehtuurit ovat yleensä toteutettu säikeillä tai esteettömällä (engl. non-blocking) siirrännällä (engl. I/O). [20.]

Seuraavassa luvussa kirjoitetaan reaaliaikaisesta kommunikaatiosta.

5.3 Reaaliaikainen kommunikaatio

WebRTC mahdollistaa verkkoselainten välisen reaaliaikaisen kommunikaation JavaScript-ohjelmistorajapinnan kautta ilman verkkoselaimen lisäosia. Tarkoituksena on

mahdollistaa esimerkiksi äänipuhelut, videopuhelut ja P2P-vertaisverkko verkkoselaimen verkkosovelluksissa. WebRTC toteuttaa ainakin kolme ohjelmistorajapintaa. Nämä ovat MediaStream, RTCPeerConnection ja RTCDataChannel. Kaikki selaimet eivät tue näitä ohjelmistorajapintoja. MediaStream-rajapinta on tarkoitettu kuvaamaan mediavirtaa (engl. stream), kuten esimerkiksi ääntä tai kuvaa videokamerasta. RTCPeerConnection on WebRTC-komponentti, joka käsittelee tietovirtaa verkkoyhteydessä olevien selainten välillä. Käytännössä yhteyden saamiseksi tarvitaan palvelinta, jotta kaksi käyttäjää löytäisivät toisensa ja voisivat kommunikoida keskenään. Palvelimessa täytyy mahdollisesti olla toiminnot käyttäjien löytäminen ja kommunikointi, signalointi, NAT/palomuuri traversal ja välityspalvelin. NAT (engl. network address translation) traversal on yleinen termin nimi, jolla pyritään pitämään yhteys päällä NAT:n läpi. Tämä tapa piilottaa verkkoon yhdistyneitä laitteita osoitteita muuttamalla tai osoitteita ja porttia muuttamalla. NAT:n kautta toimiva laite saa verkko-osoitteeksi NAT:n verkko-osoitteen. Tällä tavoin ulkopuolinen laite ei tiedä NAT:n takana olevan laitteen osoitetta. Sisäinen laite on tällä tavalla piilotettu. RTCDataChannel mahdollistaa minkä tahansa tyyppisen tiedon siirtämisen P2P-tiedonsiirrolla. Tiedonsiirto tapahtuu suoraan kahden selaimen välillä. Tämä voi olla nopeampaa kuin WebSocketin käyttäminen. [22.]

Reaaliaikainen kommunikointi voi aiheuttaa turvallisuusongelmia. Salaamaton tieto esimerkiksi salaamaton kuvapuhelu voidaan kaapata selainten välisestä liikenteestä tai selaimen ja palvelimen välisestä liikenteestä. Sovellus voi tallentaa ja jakaa käyttäjän tietämättä videota tai ääntä. Haittaohjelma tai virus voi asentua selaimen lisäosan tai ohjelman mukana. WebRTC pyrkii estämään näitä ongelmia. WebRTC-toteutuksissa käytetään "turvallisia" protokollia, salaaminen on pakollista kaikille WebRTC-komponenteille ja WebRTC ei ole selaimen lisäosa. Lisäksi kameran ja mikrofonin käyttäminen vaativat käyttäjän hyväksymisen sekä kameran ja mikrofonin päälläolo on selkeästi näytetty käyttöliittymässä. [22.]

Seuraavassa luvussa käsitellään palvelimen lähettämiä viestejä selaimeen.

5.4 Palvelimen lähettämät viestit

Server sent events-spesifikaatio määrittelee miten palvelin voi lähettää tietoa verkkosivulle [23]. Tähän voidaan käyttää HTTP-protokollaa [23]. Server sent eventin idea pe-

rustuu siihen, että verkkosovellus tilaa palvelimen lähettämiä viestejä [24]. Tämä mahdollistaa sen, että vastaanotettu viesti voidaan käsitellä tapahtumana ja tietona verkkosivulla. Kun tapahtuma tapahtuu, tiedotus siitä lähetetään selaimen verkkosovellukselle [24]. Server sent events-spesifikaatioon liittyy EventSource-rajapinta. Uudemman HTML-version myötä W3C on standardoinut Server sent events EventSource APIn [23]. Tätä rajapintaa käyttämällä luodaan EventSource-objekti ja tapahtuman käsittelijöitä. Tämä tapahtuu seuraavalla tavalla. [23.].

```
var source = new EventSource('updates.php');
source.onmessage = function (event) {
          console.log(event.data);
};
```

Esimerkkikoodia on otettu muokattuna lähteestä [23].

Kun palvelin työntää päivityksen, onmessage-tapahtuma käynnistyy. Uuden viestin tieto löytyy event.data-muuttujasta. Palvelinpuolella updates.php-skripti lähettää viestejä. Viestien MIME-tyyppi pitää olla text/event-stream. Viestit on merkistökoodattu aina UTF-8 muotoon, tätä merkistökoodausta ei voi muuttaa. Taulukossa 4 on mahdolliset viestin kenttien arvot. Kaikki muut arvot hylätään. Jos viestin rivi alkaa :-merkillä, kyseessä on kommenttirivi. Palvelinpuolen ohjelma voi käyttää kommenttiriviä yhteyden ylläpitoon. Lähettämällä viestejä palvelinpuolen ohjelma varmistuu siitä, että yhteyttä ei katkaista. Palvelinpuolen ohjelma voisi lähettää seuraavia viestejä. [23.]

```
data: Ensimmäinen.

data: Toinen,
data: viesti.
```

Taulukko 4: Server-sent event-spesifikaation määrittämät viestien kentät [23].

Kenttä	**Selitys**
event	tapahtuman tyyppi
data	viestin tietokenttä
id	tapahtuman id-tunnus
retry	uudelleen yhdistämisaika

Viestit päättyvät kahteen rivinvaihtoon eli kahteen \n\n-merkkiin. Kun viestirivejä on useampia, erotellaan viestirivit yhdellä rivinvaihdolla. Viestin viimeiseen riviin tulee kak-

si rivinvaihtomerkkiä. Palvelin voi erotella tapahtumat käyttämällä erilaisia tapahtuman tyyppejä. Perustapahtuma on message. Viestien tapahtumien erottelu tapahtuu seuraavalla tavalla. [23.]

```
: This is a comment line

event: add
data: {"fruit": "apple", "reason": "keep the doctor away."}

event: remove
data: 109831
```

Näillä viesteillä on kaksi tapahtumaa add ja remove. Viestien sisältö on tekstimuotoista, joten siinä voi olla mukana esimerkiksi JSON-muotoista tietoa. Lisäksi viestissä on mukana kommentti. Viestiin voi liittää yksilöllisen id-tunnisteen aloittamalla rivi "id:"-merkinnällä. Tämä tapahtuu seuraavalla tavalla. [23.]

```
id: 1234
data: Kanada
```

Tällä tavoin selain voi seurata tapahtumia, ja yhteyden katketessa voidaan lähettää uuden HTTP-protokollan mukaisen pyynnön yhteydessä erityinen otsaketieto Last-Event-ID. Tämä auttaa selainta päättämään, mikä tapahtuma laukaistaan. Viestin tapahtuma sisältää event.lastEventId-muuttujan. Selaimen uudelleen yhdistämisaikaa voi säätää lähettämällä viestin, jonka rivi alkaa "retry:"-merkinnällä. Tämän jälkeen tulee aika-arvo millisekunteina. [24.]

Viestien pyynnöt voidaan uudelleen ohjata samalla tavalla kuin HTTP-protokollan pyynnöt. Verkkoselain ottaa yhteyden uudelleen, jos viestiyhteys on katkennut. Palvelinohjelma voi pysäyttää selaimen yhteyden kytkentäpyynnöt lähettämällä HTTP-protokollan vastausviestin ilman viestin sisältöosuutta vastauskoodilla 204. Tämä tarkoittaa sitä, että pyyntö on käsitelty mutta vastausviestissä ei ole vastausviestin sisältöosuutta. Selainpuolen JavaScript-kielinen koodi näyttäisi seuraavanlaiselta. [23.]

```
var source = new EventSource('updates.php');
source.addEventListener('add', addHandler, false);
source.addEventListener('remove', removeHandler, false);
```

Esimerkkikoodia on otettu muokattuna lähteestä [23].

Kun yhteys aukeaa, voidaan tapahtumankuuntelija liittää tapahtumaan open. Virheiden varalta voidaan EventSourcen liittää virheiden käsittelijä käyttämällä addEventListener-funktiota tai seuraavalla tavalla.

```
source.onerror = function(e) {
          console.log("EventSource failed.");
};
```

Normaalisti selain muodostaa yhteyden uudelleen, jos yhteys menee poikki. Tämän toiminnallisuuden saa lopetettua joko selain- tai palvelinpuolelta. Verkkoselain puolelta yhteys voidaan katkaista sulkemalla yhteysvirta (engl. stream) käskyllä source.close(). Palvelinpuolelta yhteys voidaan katkaista asettamalla viestin MIME-tyyppiksi muu kuin text/event-stream. Toinen tapa on palauttaa HTTP-tilatietorivi muulla arvolla kuin "200 OK". [24.]

Viestinnän turvallisuuden kannalta on erittäin tärkeätä tarkistaa, että viestin käsittelijässä tarkistetaan onko viestinlähettäjän verkko-osoite se mistä viestien pitäisi tulla. Tämä tapahtuu seuraavalla tavalla. [24.]

```
source.addEventListener('message', function(e) {
  if (e.origin != 'http://metropolia.fi') {
    alert('Origin was not http://metropolia.fi');
    return;
  }
}, false);
```

Esimerkkikoodi on otettu muokattuna lähteestä [24]. On myös erittäin tärkeätä tarkistaa, että vastaanotettu tieto on halutun kaltaista [24].

Käyttämällä tätä ohjelmistorajapintaa mieluimmin kuin XMLHttpRequest-objektia tai IFRAME-elementtiä annetaan verkkoselaimelle paremmat mahdollisuudet hallita verkkoyhteyden resursseja. Tämä saattaa myös parantaa laitteiden akkukestoa. [23.]

WebSockect:n verrattuna Server Sent Events on yksisuuntainen tiedonsiirtokanava. Server Sent Events käyttää HTTP-protokollaa, kun taas WebSocket käyttää omaa protokollaansa, joka tarvitsee oman toteutuksensa palvelimelle. WebSocketissa ei ole automaattista yhteyden uudelleen muodostusta, tapahtumien tunnuksia (event ID) tai mahdollisuutta lähettää mitä tahansa tapahtumia. [24.]

Seuraavassa luvussa käsitellään verkkosivujen välistä kommunikaatiota.

5.5 Dokumenttien kommunikaatio

Verkkoselaimet kieltävät eri verkko-osoitteista olevien verkkosivujen skriptien vaikuttamisen toisiinsa saman alkuperän käytännöllä. Web Messaging on tarkoitettu verkkosivujen skriptien väliseen kommunikointiin. Tämä mahdollistaa eri alkuperästä olevien sivujen skriptien viestinnän hieman turvallisemmin. Web Messaging tai cross-document messaging on verkkoselaimen ohjelmistorajapinta. [25.]

Cross-document messaging, Web socket, server-sent event ja channel messaging viestit käyttävät samaa MessageEvent-tapahtumaa. [23.]

Metodilla window.postMessage voidaan oikein käytettynä turvallisesti lähettää viestejä toisen verkko-osoitteen omaavalle verkkosivulle. Kun tätä metodia kutsutaan, aiheuttaa se kohde ikkunassa MessageEvent-tyyppisen tapahtuman lähettämisen. Metodia kutsunut ohjelmanosa pitää ensin suorittaa loppuun, ennen kuin tapahtuma lähetetään. Jotta postMessage-metodia voi kutsua, pitää metodin objektilla olla viite toiseen ikkunaan. Tällaisen viitteen saa esimerkiksi IFRAME-elementistä hakemalla contentWindow-ominaisuuden arvon, window.open-metodin palauttamalla objektilla tai taulukon kaltaisesta window.framesista indeksin avulla [27]. Tämä tapahtuu esimerkiksi IFRAME-elementillä seuraavalla tavalla. [25.]

```
var o = document.getElementsByTagName('iframe')[0];
o.contentWindow.postMessage('Böö!', 'http://example.com/');
```

Esimerkkikoodia on otettu muokattuna lähteestä [25].

IFRAME-elementti pitää olla samalla sivulla, jossa esimerkin koodi suoritetaan. Lisäksi IFRAME-elementin sisällä pitää olla seuraavanlainen JavaScript-koodi, jotta se voisi vastaanottaa viestin. [25.]

```
function receiver(e) {
          if (e.origin === 'http://example.net') {
               if (e.data === 'Böö!') {
                    e.source.postMessage('A!', e.origin);
               } else {
                    alert(e.data);
               }
          }
}
window.addEventListener('message', receiver, false);
```

Esimerkkikoodia on otettu muokattuna lähteestä [25].

Viestinkäsittelijä tarkistaa, että lähettäjällä on sopiva osoite. Tämän jälkeen katsotaan, onko viestin arvo "Böö!", jos on niin lähetetään viesti vastaanottajalle. Lopussa liitetään tämä tapahtumankäsittelijä seuraamaan MessageEvent-tyyppistä tapahtumaa. [25.]

Turvallisuuden kannalta on parempi, että ei laita tapahtumankuuntelijoita viestitapahtumille, jos ei odota viestejä toisista verkko-osoitteista. Tämä on yksinkertaisin tapa välttää turvallisuusongelmia. Jos kuitenkin pitää vastaanottaa viestejä toisilta verkko-osoitteilta, on aina tarkistettava lähettäjän identiteetti tarkistamalla viestinlähettäjän verkko-osoite ja mahdollisesti viestin lähettäjän objektin viite. Samoin viestin sisältö on syytä tarkistaa, että se on halutun kaltainen. Jos viestin sisältöä ei tarkisteta, se mahdollistaa cross-site scripting -hyökkäyksen. Aina kun viesti lähetetään toisiin ikkunoihin, on syytä laittaa tarkka verkko-osoite, jotta kukaan ulkopuolinen ei pääse viestiin käsiksi. [27.]

Channel messaging luo kaksisuuntaisen putken kommunikointiin. Putken molemmissa päissä on portti. Kun viesti lähetetään portista sisään, tulee se toisesta portista ulos. Samalla tavalla kuin cross-document messagingissa viestit aiheuttavat MessageEvent-tyyppisen tapahtuman. Yhteys luodaan seuraavalla tavalla. [23.]

```
var channel = new MessageChannel();
```

Esimerkki on otettu lähteestä [23]. Yhtä porteista pidetään paikallisena porttina ja toinen porteista lähetetään vastaanottajalle esimerkiksi käyttämällä postMessage-metodia seuraavalla tavalla. [23.]

```
ob.postMessage('hello', 'http://example.com',
                [channel.port2]);
```

Esimerkkikoodi on otettu muokattuna lähteestä [23]. Kun viestiä lähetetään, käytetään portin metodia postMessage näin [23].

```
channel.port1.postMessage('hello');
```

Esimerkkikoodi on otettu lähteestä [23]. Viestinvastaanottamiseen käytetään tapahtumankuuntelijaa, joka seuraa viestintapahtumia. Tämä tapahtuu seuraavalla tavalla. [23.]

```
channel.port1.onmessage = handleMessage;
function handleMessage(event) {
  // message is in event.data
}
```

Esimerkkikoodi on otettu lähteestä [23].

Näin saadaan luotua suorayhteys kahden eri objektin välillä verkkosivulla seuraavalla tavalla [23].

```
<script>
          var channel = new MessageChannel();
          object1.method(channel.port1);
          object2.method(channel.port2);
</script>
```

Esimerkkikoodia on otettu muokattuna lähteestä [23].

Objekteissa object1 voisi olla esimerkiksi seuraavanlainen viestin tapahtumakäsittelijä [23].

```
var object1 = {
          method: function (port) {
                port.onmessage = function (event) {
                      console.log(event.data);
                };
          }
};
```

Esimerkkikoodi on otettu muokattuna lähteestä [23]. Objektissa object2 pitäisi olla myös porttiin liitetty tapahtuman käsittelijä, jotta se voisi vastaanottaa viestejä. Itse tapahtuman käsittelijän toteutus voisi olla hyvinkin erilainen. Esimerkiksi objektissa object2 voisi lähettää viestin seuraavalla tavalla, jos viestiportti objekti olisi nyt objektissa port. [23.]

```
port.postMessage('hello from object2');
```

Esimerkkikoodi on otettu muokattuna lähteestä [23]. Toiseenkin suuntaan samankaltainen viestin lähettäminen onnistuisi objektista object1 objektiin object2.

6 Pohdintaa

Web-ohjelmointi on monipuolinen aihealue. Selainsovelluksen ohjelmoinnissa tai palvelinsovelluksen ohjelmoinnissa on eroavaisuuksia, vaikka yhteisiäkin piirteitä löytyy. Selainsovelluksen ja palvelinsovelluksen tiedonsiirto pitää olla turvallista ja nopeaa.

Selainsovellus voi toimia myös ilman omaa palvelinsovellusta. Koontisovellukset ovat tästä hyvä esimerkki. Ohjelmoinnin lisäksi erilaiset laitteistoarkkitehtuurit ovat verkkosovellukselle tärkeitä, mutta huomaamattomia mahdollisiin ongelmiin asti.

Vaikka on erilaisia web-teknologioiden standardeja ja spesifikaatioita, selaimissa on eroavaisuuksia. Eroavaisuuksien vuoksi käyttöliittymän ohjelmointi voi olla haasteellista. Siksi on hyvä tietää, miten selain toimii. Selaimissa on yhtenäisiä piirteitä ja tapahtumankäsittely on samankaltaista. Verkkosivulle laitettavien sivupohjien osalta on syytä käyttää SCRIPT-elementtiä tai TEMPLATE-elementtiä. Jälkimmäisen elementin osalta, selaimen ominaisuuksien täydentämisen avulla, tämä ominaisuus voidaan ottaa käyttöön joissakin selaimissa. Monimutkaisemman käyttöliittymän tapauksessa saattaa käyttöliittymäsovelluskehysten käyttäminen olla suositeltavaa. Selaimessa JavaScript-koodin suoritus ei yleensä vie paljon aikaa. Enemmän aikaa kuluu selainsivun näyttämiseen. Paremman version JavaScript-kielestä saa, jos JavaScript-tulkki on strict mode -tilassa.

Selainsovellus voi lähettää tietoa palvelinsovellukselle monella tapaa. Uudempien tiedonsiirtotapojen myötä on syntynyt käsite "reaaliaikainen" web. Reaaliaikaisessa webissä palvelinsovellus lähettää tietoa selainsovellukselle heti, kun on jotain lähetettävää. Samoin useat käyttäjät voivat keskustella tai työskennellä yhtäaikaisesti. Tämän saavuttamiseksi on käytetty esimerkiksi JSONP:tä ja IFRAME-elementtiä sekä Flash Socket -sokettia. XMLHttpRequest, WebSocket ja server-sent event tuovat omanlaisensa ratkaisun tietojen välittämiseen selaimelle. Näiden tekniikoiden käyttäminen vaati myös palvelinpuolen soveltuvuutta. WebRTC:n myötä myös reaaliaikainen kommunikaatio on tullut selaimiin. Selainsivusta toiseen selainsivuun tapahtuva tiedonsiirto on saanut oman rajapintansa. Tämä tuo selkeän tavan selainten sivujen väliseen tiedonsiirtoon.

Onko nykyinen vallitseva asiakas-palvelin-malli toimiva verkkosovelluksen kannalta? Missä tämä malli toimii? Missä tämä malli ei toimi? Näihin kysymyksiin ei vastattu tässä opinnäytetyössä. Turvallinen tiedonsiirtotapa selaimen ja palvelimen välillä voisi olla sopiva tapa laajentaa tätä opinnäytetyötä. Kaiken tämän lisäksi tutkimusta voisi jatkaa tutkimalla palvelimen verkkosovelluksen toimintaa esteettömänä siirrännällä tai säikeistettynä silloin, kun selainohjelman viesteihin pitää vastata palvelinsovelluksen muun toiminnan ohessa. Uudempien Web-teknologioiden liittäminen valittuun verkko-

sovelluskehykseen tai palvelinympäristöön voisi olla mielenkiintoinen tutkimuskohde. Toisaalta voisi myös selvittää erilaisten ohjelmointitapojen ja suunnittelumallien soveltuvuutta selainsovelluksen käyttöliittymän ohjelmoinnissa.

Tässä opinnäytetyössä käsiteltiin erilaisia Web-teknologioita enimmäkseen Internetin tietolähteitä käyttäen. Alussa tutkittiin HTTP-protokollan toimintaa. Tämän jälkeen käsiteltiin hieman selaimen toimintaa ja selaimeen liittyvää ohjelmointia. Lopuksi tutkittiin erilaisia tiedonsiirron mahdollisuuksia selaimen ja palvelimen välillä sekä selaimessa olevien eri verkkosivujen kesken.

Lähteet

1 CS253 - Lesson 1: The Basics. 2012. Verkkodokumentti. Udacity <www.udacity.com/wiki/cs253/unit_1>. Luettu 7.11.2013.

2 Britt, David T., Davis, Chuck, Forrester, Jason, Liu, Wei, Matthews, Carolyn, Parziale, Lydia & Rosselot, Nicolas. 2006. TCP/IP Tutorial and Technical Overview. Verkkodokumentti. <www.redbooks.ibm.com/redbooks/pdfs/gg243376.pdf>. Luettu 7.11.2013.

3 Berners-Lee, T., Fielding, R., Frystyk, H., Gettys, J, Irvine, UC, Leach, P., Masinter, L. & Mogul, J. 1999. Hypertext Transfer Protocol -- HTTP/1.1. Verkkodokumentti. <www.ietf.org/rfc/rfc2616.txt>. Luettu 19.11.2013.

4 Introducing IANA. Verkkodokumentti. IANA. <www.iana.org/about>. Luettu 20.11.2013.

5 URI scheme. Verkkodokumentti. Wikipedia. <https://en.wikipedia.org/wiki/URI_scheme>. Luettu 19.11.2013.

6 Berners-Lee, T., Fielding, R. & Masinter, L. 2005. Uniform Resource Identifier (URI): Generic Syntax. Verkkodokumentti. <tools.ietf.org/html/rfc3986>. Luettu 18.11.2013.

7 Uniform Resource Identifier (URI) Schemes. Verkkodokumentti. IANA. <www.iana.org/assignments/uri-schemes/uri-schemes.xhtml>. 26.11.2013. Luettu 27.11.2013.

8 Garsiel, Tali. How Browsers Work. Verkkodokumentti. <taligarsiel.com/Projects/howbrowserswork1.htm>. Luettu 20.11.2013.

9 Ruderman, Jesse. Same-origin policy. Verkkodokumentti. <developer.mozilla.org/en-US/docs/Web/JavaScript/Same_origin_policy_for_JavaScript>. 14.11.2013. Luettu 27.11.2013.

10 Osmani, Addy. Writing Modular JavaScript With AMD, CommonJS & ES Harmony. Verkkodokumentti. <addyosmani.com/writing-modular-js/>. Luettu 24.11.2013.

11 How to build custom form widgets. Verkkodokumentti.

<developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms/How_to_build_custom_form_widgets>. 22.11.2013. Luettu 25.11.2013.

12 Bookmarklet. Verkkodokumentti. Wikipedia. <en.wikipedia.org/wiki/Bookmarklet>. 14.11.2014. Luettu 24.11.2013.

13 Stefanov, Stoyan. 2010. JavaScript Patterns. Sebastopol: O'Reilly.

14 Introduction to Web Components. 2013. Verkkodokumentti. <w3c.github.io/webcomponents/explainer/>. Luettu 27.11.2013.

15 Masinter, L. 1998. The "data" URL scheme. Verkkodokumentti. <tools.ietf.org/html/rfc2397>. Luettu 19.11.2013.

16 Introducing JSON. Verkkodokumentti. <www.json.org>. Luettu 28.11.2013.

17 Ergül, Salih, Özses, Seda, 24.2.2009. Cross-domain communications with JSONP, Part 1: Combine JSONP and jQuery to quickly build powerful mashups. Verkkodokumentti. <www.ibm.com/developerworks/library/wa-aj-jsonp1/>. Luettu 28.11.2013.

18 XMLHttpRequest. 6.12.2012. Verkkodokumentti. <www.w3.org/TR/XMLHttpRequest>. Luettu 24.11.2013.

19 Using XMLHttpRequest. Verkkodokumentti. <developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest>. 21.11.2013. Luettu 24.11.2013.

20 Kitamura, Eiji, Ubl, Malte. 20.10.2010. Verkkodokumentti. Introducing WebSockets: Bringing Sockets to the Web. <www.html5rocks.com/en/tutorials/websockets/basics/>. Luettu 30.11.2013.

21 Fette, I., Melnikov, A. The WebSocket Protocol. 2011. Verkkodokumentti. <http://tools.ietf.org/html/rfc6455>. Luettu 30.11.2013.

22 Dutton, Sam. Getting Started with WebRTC. 23.7.2012. Verkkodokumentti. <http://www.html5rocks.com/en/tutorials/webrtc/basics/>. Luettu 26.11.2013.

23 HTML Living Standard. Verkkodokumentti. <http://www.whatwg.org/specs/web-apps/current-work/>. 25.11.2013. Luettu 28.11.2013.

24 Bidelman, Eric. 30.9.2010. Stream Updates with Server-Sent Events. Verkkodokumentti. <www.html5rocks.com/en/tutorials/eventsource/basics/>. Luettu 28.11.2013.

25 Web Messaging. Verkkodokumentti. <en.wikipedia.org/wiki/Web_Messaging>. 9.10.2013. Luettu 29.11.2013.

26 Le Hégaret, Philippe, Höhrmann, Björn, Kacmarcik, Gary, Leithead, Travis, Pixley, Tom, Rossi, Jacob & Schepers, Doug. 2013. Document Object Model (DOM) Level 3 Events Specification. Verkkodokumentti. W3C. <www.w3.org/TR/DOM-Level-3-Events/>. Luettu 26.11.2013.

27 Window.postMessage. Verkkodokumentti. <developer.mozilla.org/en-US/docs/Web/API/Window.postMessage>. 25.11.2013. Luettu 29.11.2013.