WAUI

Web based Automation User Interface

Toni Vartiainen

Opinnäytetyö
Kesäkuu 2012
Tietotekniikka
Ohjelmistotekniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikka
Ohjelmistotekniikka

TONI VARTIAINEN:
WAUI
Web based Automation User Interface

Opinnäytetyö 42 sivua, josta liitteitä 0 sivua
Kesäkuu 2012

Tähän opinnäytetyöhön on kuvattu Demolassa alkanutta WAUI-projektia, jonka päämääränä oli suunnitella ja toteuttaa Metso Automationille Web-selainpohjainen automaatio-ohjaussovellus. Sovelluksella piti pystyä lukemaan ja kirjoittamaan ohjattavan prosessin laitteiden arvoja Metson järjestelmään mahdollisimman reaaliaikaisesti.

Sovellus perustuu asiakas-palvelin -arkkitehtuuriin. Asiakkaan käyttöliittymään on tarkoitus mallintaa graafisten käyttöliittymäkomponenttien avulla Metson ohjattava prosessi mahdollisimman tarkasti. Asiakkaan käyttöliittymäkomponentit on kirjoitettu HTML5:lla ja sovelluslogiikka jQuerylla. Palvelin toimii WAUI:ssa kommunikaatiopalvelimena asiakkaan ja Metson järjestelmän välissä. Palvelin hakee, tallentaa ja ohjaa jatkuvasti tietoa Metson järjestelmästä asiakkaalle sekä välittää asiakkaan käyttöliittymästä ohjattavan prosessin laitteille asetetut arvot Metson järjestelmään. Palvelimella on käytetty ohjelmointikielenä Pythonia.

WAUI-projektin seurauksena syntynyt sovellus on vasta prototyyppi. Valmiiseen tuotteeseen olisi toteutettava vielä merkittäviä lisäyksiä. Tällaisia lisäyksiä olisivat ainakin tietoliikenteen salaus asiakkaan ja palvelimen välillä sekä sovelluksen käyttöoikeuksiin liittyvät seikat. Sovelluksen jatkokehitysmahdollisuuksille jäi myös tilaa: käyttöliittymän voisi esittää kaksiulotteisen grafiikan sijaan kolmiulotteisella grafiikalla ja käyttöliittymäeditorin, joka jäi projektin päätyttyä vielä kehitysasteelle, voisi toteuttaa valmiiksi.

Asiasanat: Automaatio-ohjaus, HTML5, JavaScript

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Computer Science
Software Engineering

TONI VARTIAINEN:
WAUI
Web based Automation User Interface

Bachelor's thesis 42 pages, appendices 0 pages
June 2012

This thesis describes WAUI project which was implemented in Demola in co-operation with Metso Corporation. The goal of the project was to design and implement Web browser-based automation control application for Metso Automation. Application was supposed to be able to read and write the values of the process related devices into Metso´s system in real-time.

The application is based on client-server architecture. The goal of the user interface was to model Metso´s controlled process as accurately as possible using graphical user interface components. The user interface components of the client were written using HTML5 and the application logic using jQuery. In WAUI the server acts as communication server between the client and Metso´s system. The server retrieves, saves, and forwards information constantly from Metso´s system to client and forwards the values defined to controlled process devices from client's user interface to Metso´s system. The used programming language on the server side was Python.

The resulting WAUI-application is just a prototype. Significant additions should be implemented to complete the product. Such additions would include at least encryption of data communications between client and server and application access right related issues. There are still possibilities to develop the application further: the user interface could be presented by using three-dimensional graphics instead of two-dimensional graphics and user interface editor, which at the end of the project was still under development, could be completed.

Keywords: Automation control, HTML5, JavaScript

SISÄLLYS

LYHENTEET JA TERMIT

AJAX	Asynchronous JavaScript And XML, joukko Web-sovelluskehityksen tekniikoita
API	Application Programming Interface, rajapinta
CNP	Configurable and Name based Protocol, yhteyskäytäntö
CSS	Cascading Style Sheet, tyylikieli
DLL	Dynamic-Link Library, ajonaikainen kirjasto
DOM	Document Object Model, ohjelmointirajapinta
HTML	Hypertext Markup Language, merkintäkieli
HTTP	Hypertext Transfer Protocol, yhteyskäytäntö
HTTPS	Hypertext Transfer Protocol Secure, yhteyskäytäntö
IP	Internet Protocol, yhteyskäytäntö
JS	JavaScript, ohjelmointikieli
JSON	JavaScript Object Notation, tiedonsiirtomuoto
SSL	Secure Sockets Layer, tietoliikenteen salaamiseen käytettävä säännöstö
TCP	Transmission Control Protocol, yhteyskäytäntö
TLS	Transport Layer Security, tietoliikenteen salaamiseen käytettävä säännöstö
URI	Uniform Resource Identifier, merkkijono, joka yksilöi tiedon sijainnin
W3C	World Wide Web Consortium, työryhmä
WAUI	Web based Automation User Interface
WebGL	Web Graphics Library, JavaScript-rajapinta tai työryhmä
WHATWG	Web Hypertext Application Technology Working Group, työryhmä
ActiveX	Ohjelmointikieli
Adobe Illustrator	Adoben kehittämä vektorigrafiikkaan perustava piirto-ohjelma
Canvas	JavaScript-rajapinta, piirtoala
cTypes	Python-kirjasto, joka mahdollistaa C- ja yhteensopivilla ohjelmointikielillä kirjoitettujen funktioiden kutsumisen DLL-tiedostosta

ECMAScript	Ohjelmointikieli
Flash	Ohjelmointikieli
JavaScript	Ohjelmointikieli
JScript	Ohjelmointikieli
JQuery	JavaScript-kirjasto
JQueryUI	JavaScript-käyttöliittymäkirjasto
Metso CNP	Metson kirjasto CNP-yhteyskäytäntöä varten
Metso DNA	Metson automaatiojärjestelmä
Polling	Menetelmä, jossa asiakas ottaa palvelimeen yhteyttä tietyin väliajoin kyselläkseen palvelimen tilaa
Python	Ohjelmointikieli
Tornado	Pythonilla kirjoitettu Web-palvelin
Web Socket	JavaScript-rajapinta, joka mahdollistaa reaaliaikaisen kaksisuuntaisen kommunikaation asiakkaan ja palvelimen välillä
Web Storage	JavaScript-rajapinta, joka mahdollistaa evästetyylisen tiedon tallentamisen ja lukemisen Web-sovelluksissa

1 JOHDANTO

Tämä opinnäytetyö pohjautuu Demolassa helmikuun alussa aloitettuun WAUI-projektiin. Projektin tarkoitus oli suunnitella ja toteuttaa Metsolle Web-selainpohjainen automaatio-ohjaussovellus. Sovelluksella piti kyetä lukemaan ja kirjoittamaan arvoja Metson järjestelmään Web-selaimen käyttöliittymästä. Tähän opinnäytetyöhön on dokumentoitu Demolassa valmistuvaa sovellusta.

Tällä hetkellä Metsolla käytössä oleva sovellus on laaja ja sidottu Windows-ympäristöön. Nykyiseen sovellukseen verrattuna projektissa haluttiin painottaa joustavuutta ja uudistettua käyttöliittymää. Joustavuus tarkoittaa tämän projektin kohdalla erityisesti alustariippumattomuutta, keveyttä ja hyvää skaalautuvuutta.

Sovelluksen on tarkoitus toimia kaikilla alustoilla, joissa käyttäjällä on mahdollisuus modernin Web-selaimen käyttöön. Tällaisia alustoja ovat esimerkiksi puhelimet ja taulutietokoneet perinteisten tietokoneiden ohella. Skaalautuvuudella tarkoitetaan käytettävien resurssien suhdetta järjestelmän laajuuteen. Optimitilanteessa mahdollisimman pieni määrä resursseja voisi palvella mahdollisimman laajaa järjestelmää. WAUI pyrittiin ohjelmoimaan mahdollisimman hyvin skaalautuvaksi.

WAUI koostuu asiakkaasta ja palvelimesta. Asiakas tarkoittaa käyttäjän päätelaitteessa suoritettavaa sovellusta, joka käsittää WAUI:ssa Web-selaimeen avattavan käyttöliittymän ja siihen liittyvän sovelluslogiikan. Käyttöliittymään voidaan avata erilaisia näkymiä. Näkymien avulla on tarkoitus mallintaa mahdollisimman hyvin Metson ohjattavia prosesseja. Palvelin toimii kommunikaatiopalvelimena asiakkaan ja Metson järjestelmän välissä. Palvelimen tärkein tehtävä on hakea jatkuvasti tietoa Metson järjestelmästä. Palvelin ohjaa järjestelmästä saadut arvot asiakkaalle, jonka käyttöliittymässä arvot esitetään käyttäjälle graafisesti. Asiakkaan käyttöliittymästä voidaan arvojen lukemisen ohella kirjoittaa arvoja Metson järjestelmään käyttöliittymään toteutetun ohjauspaneelin avulla. Arvojen lukeminen ja arvojen kirjoittaminen käyttöliittymästä Metson järjestelmään on lähes reaaliaikaista.

Asiakkaan käyttöliittymä on ohjelmoitu lähes kokonaan HTML5:lla. Käyttöliittymän sovelluslogiikka on pääosin jQuerya. Palvelimella käytetty Web-palvelin on kirjoitettu Pythonilla. Myös kaikki Metson järjestelmään liittyvä koodi palvelimella on Pythonia.

Luvussa 2 käydään tarkemmin läpi sovelluksen käyttöliittymää. Luvussa kerrotaan käyttöliittymän ominaisuuksista sekä käyttöliittymäkomponenttien tiloista ja vuorovaikutuksesta. Luvussa 3 kuvataan sovelluksen arkkitehtuuri. Luvussa käydään läpi asiakkaan, palvelimen ja Metson järjestelmän välisen vuorovaikutuksen lisäksi käyttöliittymän dynaaminen luonti ja sen edut. Luvussa 4 esitellään käytetyt tekniikat ja olennaisia osia sovelluksen toteutuksesta. Luvussa 5 on yhteenveto projektista ja pohdintaa sovelluksen jatkokehitysmahdollisuuksista.

2 KÄYTTÖLIITTYMÄ

WAUI mahdollistaa Metson järjestelmän arvojen lukemisen ja kirjoittamisen Web-selaimeen avattavasta graafisesta käyttöliittymästä. WAUI:n käyttöliittymässä esitetään aina ohjattavan prosessin laitteiden mahdollisimman reaaliaikainen tila. Järjestelmän tilan esittämiseksi käyttöliittymään on toteutettu erilaisia mittareita, jotka esittävät Metson järjestelmästä palvelimen kautta saatavan tiedon graafisesti. Sen lisäksi, että käyttäjä voi mittareita tutkimalla todeta järjestelmän lähes reaaliaikaiset arvot, voi käyttäjä itse muuttaa järjestelmän arvoja käyttöliittymään toteutetun ohjauspaneelin avulla. Järjestelmän arvojen asettaminen ohjauspaneelista Metson järjestelmään tapahtuu arvojen esittämisen tapaan lähes reaaliajassa.

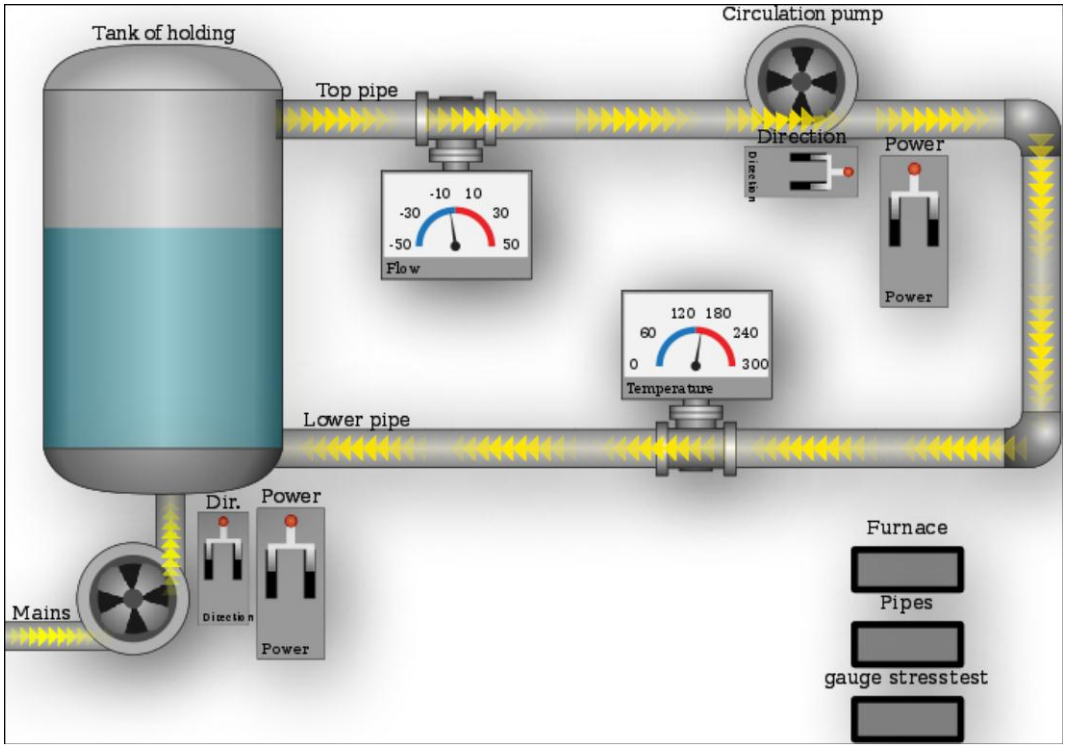
WAUI:n käyttöliittymä on tavallinen Web-sivu. Käyttöliittymä on kirjoitettu lähes kokonaan HTML5:lla. Käyttöliittymän voi avata millä tahansa modernilla Web-selaimella, joka tukee HTML5:n ominaisuuksia riittävällä tasolla. Käyttöliittymän ja Web-selaimien yhteensopivuutta käydään tarkemmin läpi tekniikat ja toteutus -luvussa.

Käyttöliittymän suunnittelussa lähdettiin liikkeelle siitä, että sovellusta tullaan käyttämään pöytätietokoneiden ohella mm. taulutietokoneilla ja puhelimilla. Puhelinten pienet näytöt sekä kosketusnäytöt asettavat rajoja käyttöliittymän toteutuksen suhteen. Tärkeiden käyttöliittymäkomponenttien kuten mittareiden ja ohjauspaneelin ohjaimien on oltava tarpeeksi suuria, jotta niiden käytettävyys olisi mahdollisimman hyvä pienemmältäkin näytöltä. Kosketusnäytöllisissä laitteissa taas Metson järjestelmään kirjoitettavia arvoja on huomattavasti helpompi ja nopeampi asettaa liukuohjaimien avulla kuin esimerkiksi suoraan tekstikenttään syötettävien arvojen avulla. Ohjauspaneeliin on jätetty kuitenkin myös suoran arvonsyötön mahdollisuus, koska sillä voidaan helposti asettaa ohjaimelle tarkka arvo.

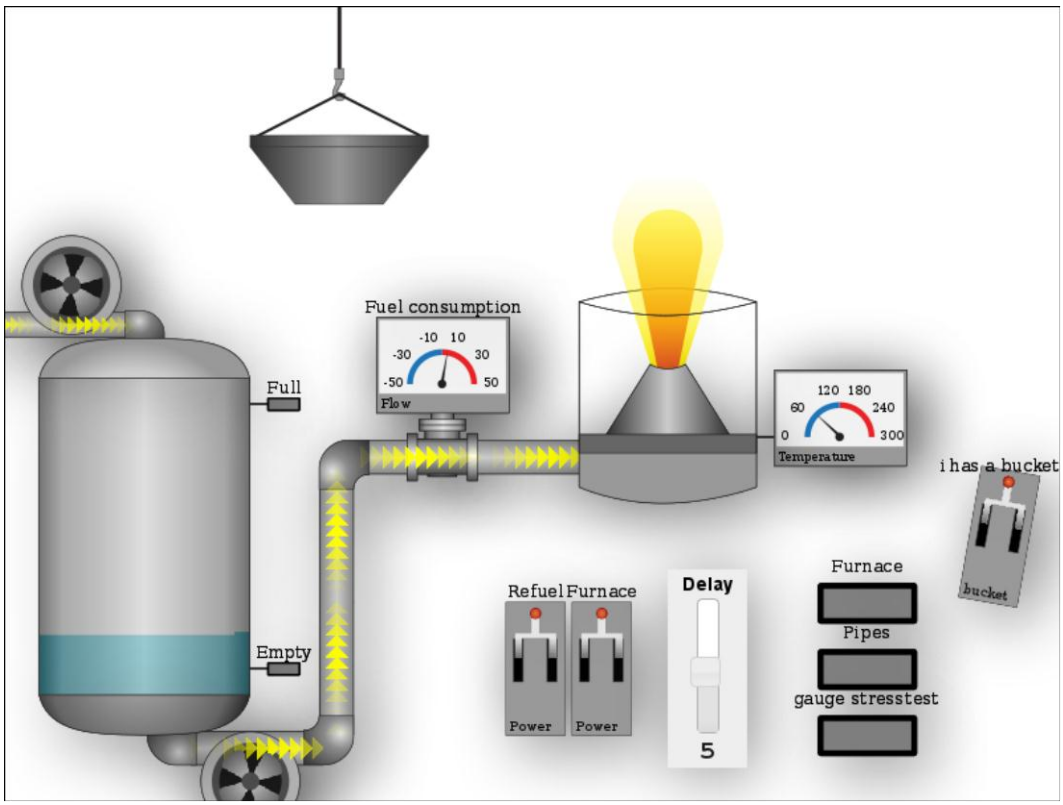
2.1 Näkymät

Käyttöliittymässä ohjattavat prosessit esitetään näkymien avulla. Yhdestä näkymästä voidaan nähdä ainoastaan osa koko prosessista. Näkymiä voidaan vaihtaa käyttöliittymästä nappia painamalla. Järjestelmän esittäminen näkymien avulla on huomattavasti

käytännöllisempää, kuin koko prosessin kuvaaminen käyttöliittymässä kokonaisuudessaan. Prosessin esittäminen näkymien avulla mahdollistaa sovelluksen käytön heikonkin suorituskyvyn laitteessa, kun käyttöliittymäkomponentteja piirretään näytölle vain vähän kerrallaan. Seuraavassa on muutama esimerkki erilaisista näkymistä (Kuva 1; Kuva 2).



Kuva 1. Esimerkki käyttöliittymän näkymästä.



Kuva 2. Esimerkki käyttöliittymän näkymästä.

Graafisten käyttöliittymäkomponenttien piirtäminen HTML5:n canvasilla on prosessori-intensiivistä sen toimintaperiaatteen takia. Lisäksi animoitujen käyttöliittymäkomponenttien suorituskykyistä toteutusta haittaa se, että canvas-elementti ei muista elementtiin piirrettyä kuvaa piirtokertojen välillä. Tämän takia animoidut käyttöliittymäkomponentit joudutaan piirtämään lyhyin väliajoin kokonaan uudelleen sen sijaan, että kuvalle annettaisiin vain uudet parametrit.

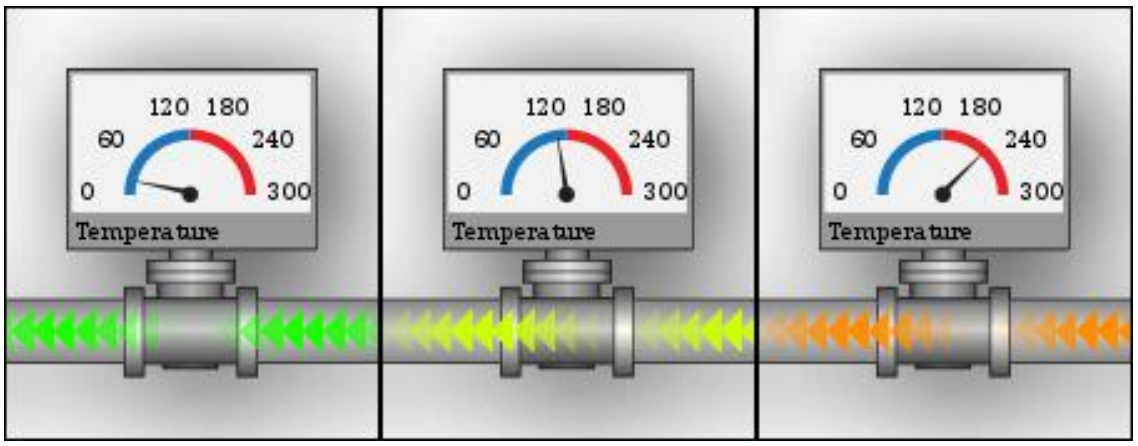
Näkymien animoitujen elementtien määrän ja monimutkaisuuden määräävät heikoimman suorituskyvyn laitteet. Tietokoneella laajemmankin prosessin kuvaaminen käyttöliittymässä onnistuisi ongelmitta, mutta esimerkiksi puhelimella sovellusta käytettäessä prosessin jakaminen näkymiin edesauttaa huomattavasti sovelluksen sulavaa käyttöä. Käyttöliittymän suoristuskykyä suhteessa animoitujen käyttöliittymäkomponenttien määrään on testattu WAUI:ssa erilaisten stressitestien avulla.

2.2 Komponentit

Kaikki näkymien käyttöliittymäkomponentit on ohjauspaneelia lukuun ottamatta ohjelmoitu HTML5:n canvas-elementtiin. Canvas-elementtiin voidaan piirtää kuvia JavaScriptin avulla. Ohjauspaneeli on muista käyttöliittymäkomponenteista poiketen kirjoitettu jQuerylla. JQueryn käyttöön on vaikuttanut muutama asia. Ensinnäkin jQueryn käyttöliittymäkomponenttikirjastosta löytyy valmis ja helppokäyttöinen toteutus liukuohjaimelle. Lisäksi käyttämällä jQuerya ohjauspaneelissa, vältytään piirtämästä ruudulle turhia käyttöliittymäkomponentteja HTML5:lla.

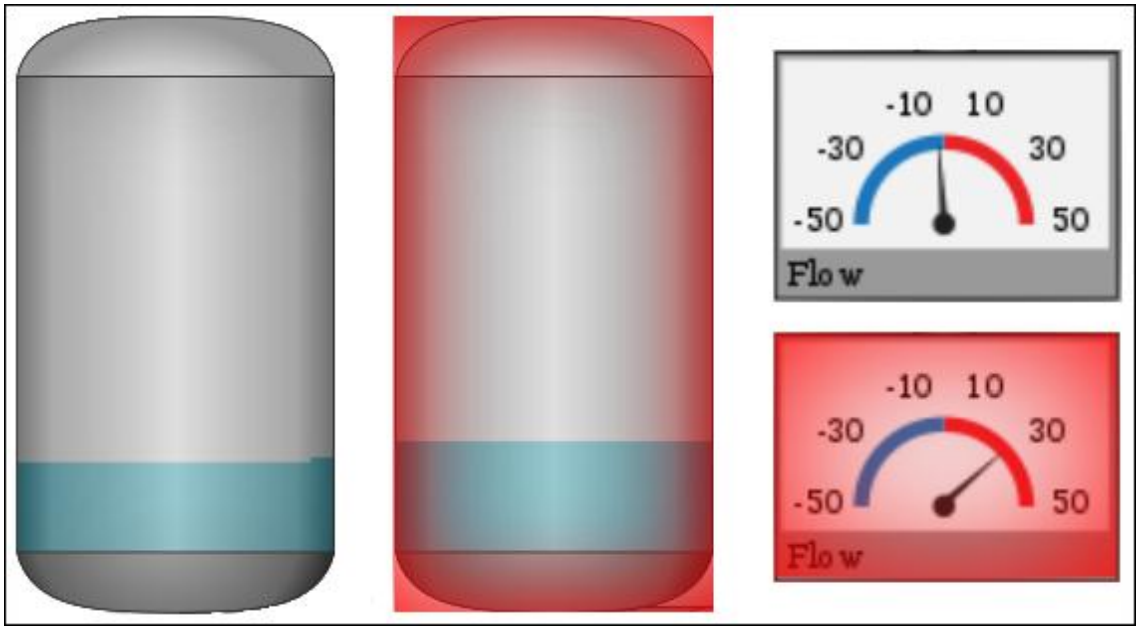
Käyttöliittymäkomponentit luodaan dynaamisesti nk. näyttötiedoston avulla. Käyttöliittymän komponenttien luominen näyttötiedoston avulla dynaamisesti mahdollistaa helpon käyttöliittymän luomisen koskematta ollenkaan sovelluslogiikkaan. Näyttötiedosto sisältää jokaisen käyttöliittymäkomponentin määrittelyn. Näyttötiedostossa määritellään muun muassa komponentin grafiikka, x-, y- ja z- koordinaatit sekä koko ja kierto. Jokaisen käyttöliittymäkomponentin grafiikka ja toiminnallisuus on toteutettuna niin sanottuun komponenttitiedostoon. Sovelluslogiikka suorittaa komponenttitiedostoja näyttötiedoston määrittelyiden mukaan. Tuloksena näytölle piirtyy määrittelyiden mukainen graafinen käyttöliittymäkomponentti. Näyttötiedostoja ja komponenttitiedostoja käydään arkkitehtuurin näkökulmasta läpi tarkemmin arkkitehtuuri-luvussa. Näyttötiedostojen, komponenttitiedostojen ja sovelluslogiikan välistä suhdetta käydään tarkemmin läpi tekniikka ja toteutus -luvussa.

Sovelluksen perustavanlaatuinen ominaisuus on, että komponentit voivat olla vuorovaikutuksessa keskenään. Käyttöliittymäkomponenttien vuorovaikutuksia voidaan asettaa näyttötiedostossa. Esimerkkinä laitteita yhdistävässä putki-käyttöliittymäkomponentissa kulkevat nuolet voidaan asettaa reagoimaan lämpötilamittarin arvoihin. Lämpötilan noustessa nuolet muuttavat väriään punaisemmiksi ja lämpötilan laskiessa nuolet muuttuvat vihreämmiksi (Kuva 3).



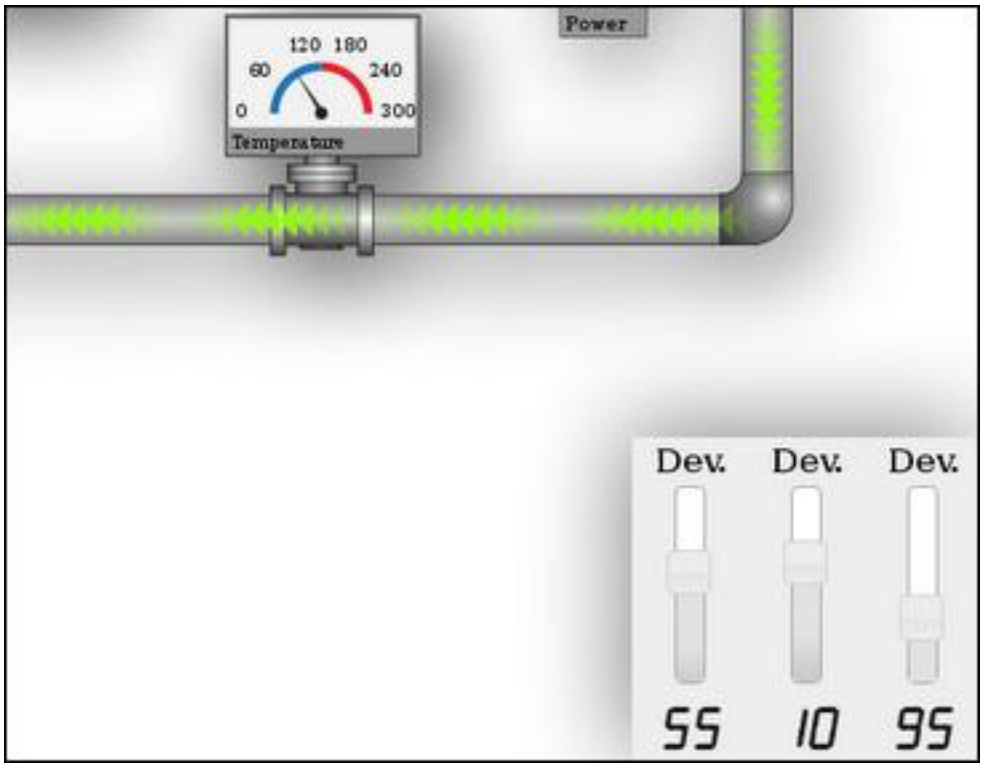
Kuva 3. Esimerkki käyttöliittymäkomponenttien välisistä suhteista.

Kaikkiin käyttöliittymän käyttöliittymäkomponentteihin liittyy minimi- ja maksimiarvot, jotka asetetaan myös näyttötiedostossa komponenttikohtaisesti. Ajatus on, että jos Metson järjestelmästä palvelimen kautta saatava arvo on pienempi kuin näyttötiedostossa komponentille määritelty minimiarvo tai vastaavasti suurempi kuin määritelty maksimiarvo, niin käyttöliittymäkomponentti asetetaan hälytystilaan. Hälytystilassa käyttöliittymäkomponentin ympärille piirretään punainen kehä, jonka tarkoitus on kiinnittää käyttäjän huomio virheellisesti toimivaan järjestelmään. Hälytystila liittyy lähinnä käyttöliittymän mittareihin, jotka välittävät tietoa ohjattavan prosessin laitteista käyttäjälle. Kuvassa 4 on muutama esimerkki hälytystilassa olevista käyttöliittymäkomponenteista.



Kuva 4. Esimerkkejä käyttöliittymäkomponenteista hälytystilassa

Ohjauspaneelin liukuohjaimien toteutuksessa on käytetty jQueryUI:a, joka on jQuerylla kirjoitettu käyttöliittymäkomponenttien kirjasto. Liukuohjaimiin on toteutettu sovelluslogiikka, joka lähettää palvelimen kautta Metson järjestelmään ohjaimessa asetetun arvon, kun käyttäjä päästää liu´usta irti. Liukua raahaamalla ylös ja alas liu´un arvo päivittyy jatkuvasti ohjaimen alla olevaan numerokenttään, jotta käyttäjän ei tarvitse arvailla liu´un arvoa. Liu´un yläpuolella näytetään ohjattavan laitteen nimi (Kuva 5).

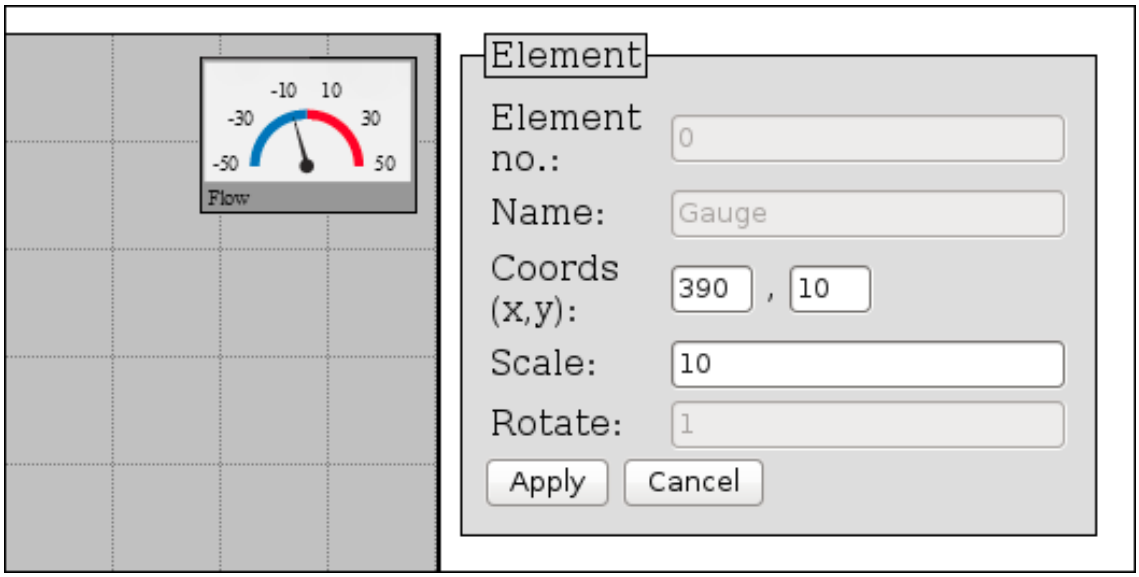


Kuva 5. Ohjauspaneeli.

2.3 Editori

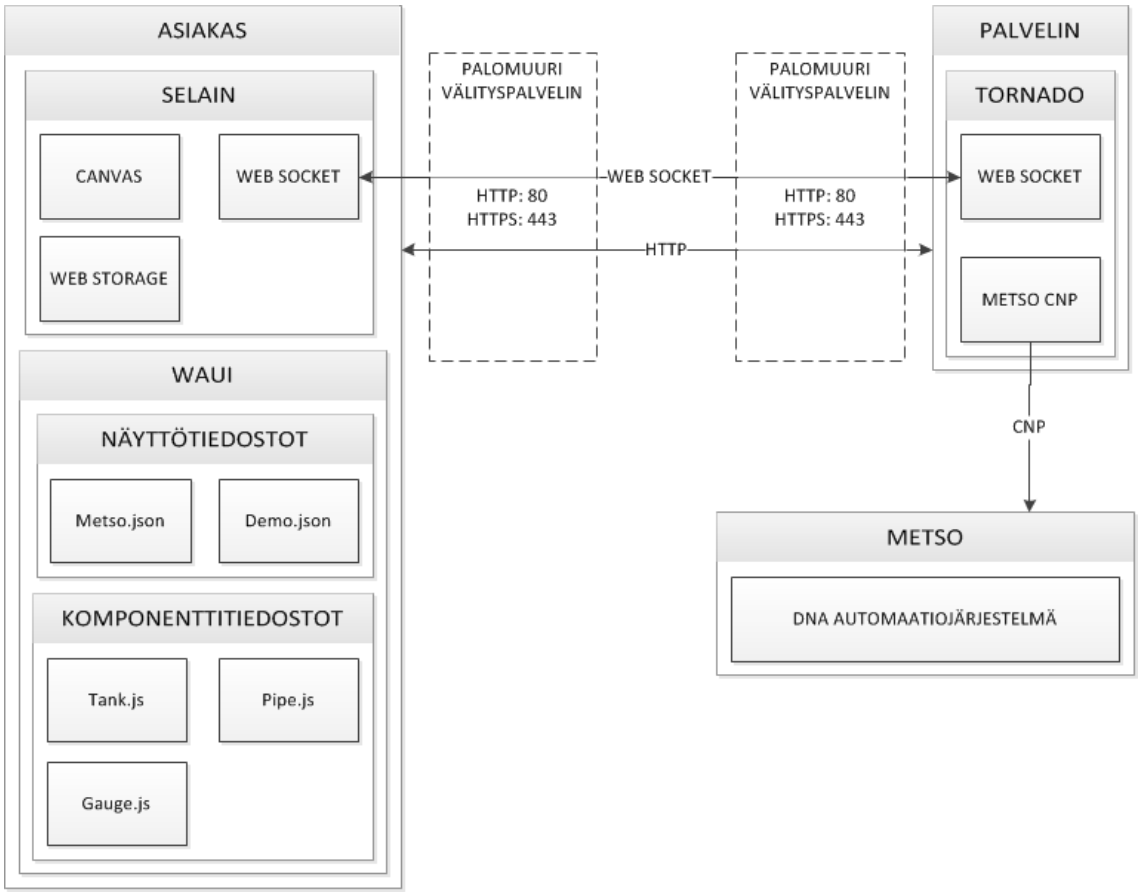
Käyttöliittymäkomponenttien dynaaminen luominen näyttötiedoston pohjalta mahdollistaa käyttöliittymän luomisen koskematta ollenkaan ohjelman sovelluslogiikkaan. Tämä mahdollistaa myös editorin helpohkon sovellustoteutuksen. Editori mahdollistaisi helpon tavan luoda käyttöliittymään minkälaisia näkymiä tahansa annettujen käyttöliittymäkomponenttien puitteissa. Editorissa voisi sijoitella käyttöliittymäkomponentit haluamilleen paikoille ja asettaa näille komponenteille ominaisuuksia kuten sijainti, koko, ja kierto.

Editori generoisi näkymän käyttöliittymäkomponenteista lopulta näyttötiedoston. Tämän näyttötiedoston avulla tuotettaisiin WAUI:n varsinaiseen käyttöliittymään uusi näkymä kaikkine toiminnallisuuksineen. Editori on kuitenkin keskeneräinen eikä tullut valmiiksi projektille varatun ajan puitteissa (Kuva 6).



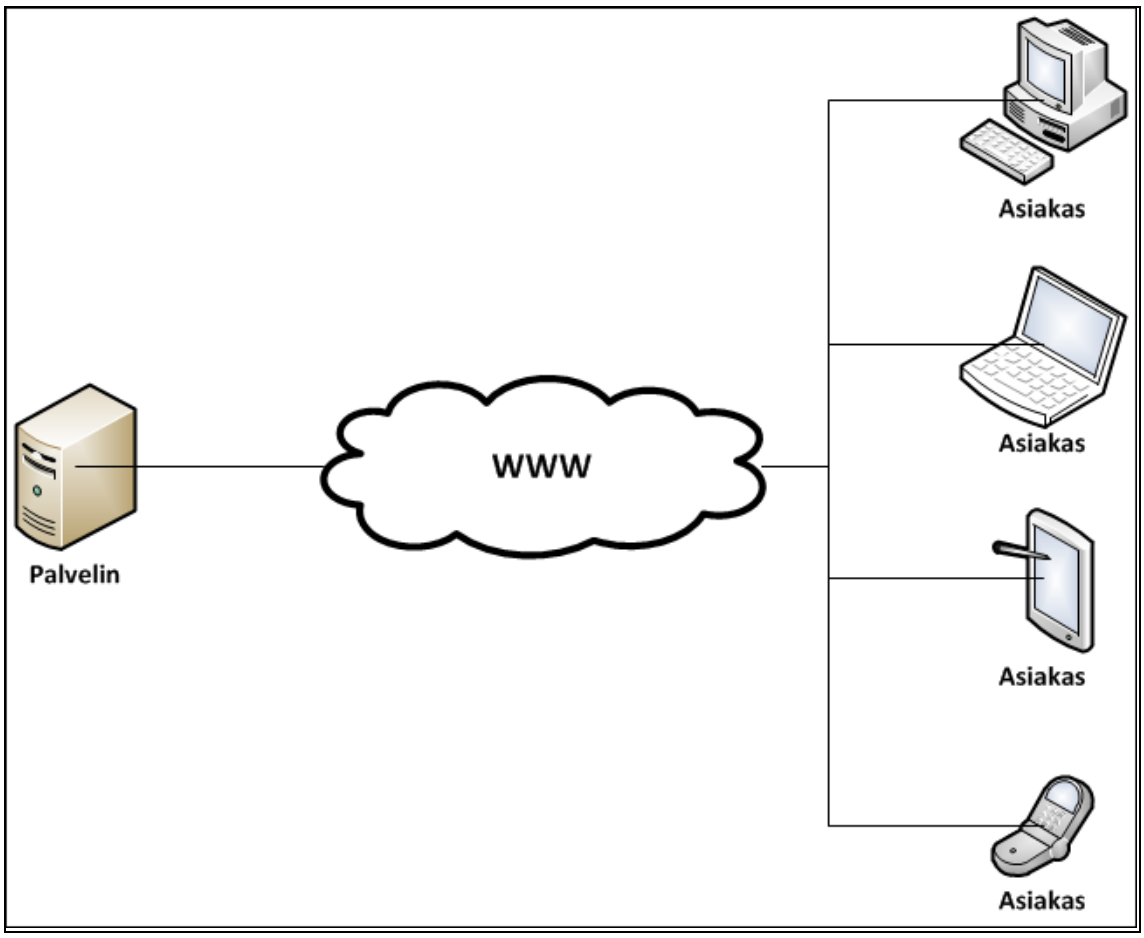
Kuva 6. Kehitysversio editorista.

3 ARKKITEHTUURI



Kuva 7. WAUI:n arkkitehtuuri.

WAUI perustuu asiakas-palvelin –malliin (Kuva 7; Kuva 8). Asiakkaaksi kutsutaan käyttäjän päätelaitteessa pyörivää sovellusta. Asiakas-palvelin -mallissa asiakas itsessään ei tiedä sovelluksen tilasta mitään olematta yhteydessä palvelimeen. WAUI:ssa palvelin toimii ns. kommunikaatiopalvelimena asiakkaan ja Metson järjestelmän välissä. Kommunikaatiopalvelin välittää asiakkaalle Metson järjestelmästä saatavan tiedon sekä asiakkaan Metson järjestelmään käyttöliittymästä asettamat arvot. Asiakas-palvelin -mallissa palvelin palvelee tavallisesti useitakin asiakkaita kerrallaan (Kuva 8).



Kuva 8. Asiakas-palvelin –malli.

WAUI perustuu siihen, että käyttöliittymästä on luettavissa järjestelmän mahdollisimman reaaliaikainen tila. HTTP perustuu kuitenkin pyyntö-vastaus -kommunikaatioon, jossa asiakas alustaa aina asiakkaan ja palvelimen välisen kommunikaation lähettämällä palvelimelle pyynnön haluttuun resurssiin. Palvelin vastaa asiakkaan pyyntöön palauttamalla tälle pyydetyn resurssin. Perinteisessä HTTP-protokollaan perustuvassa pyyntö-vastaus -kommunikaatiossa palvelin ei koskaan voi ottaa yhteyttä asiakkaaseen omaaloitteisesti. Tällaisen protokollan avulla on mahdoton toteuttaa reaaliaikaista kommunikaatiota, koska asiakas joutuu aina itse pyytämään palvelimelta muuttunutta resurssia. HTML5 kuitenkin mahdollistaa käytännössä reaaliaikaisen kommunikaation asiakkaan ja palvelimen välillä Web Socketin avulla. Asiakkaan ja palvelimen välille luodaan Web Socket -kanava ja tätä kanavaa pidetään auki niin kauan kuin toinen osapuoli sulkee yhteyden. Jatkuva yhteys asiakkaan ja palvelimen välillä mahdollistaa sen, että asiakkaalta palvelimelle lähtevien resurssien lisäksi palvelin voi työntää resursseja asiakkaalle reaaliajassa. Tämä ei ole ollut aikaisemmin mahdollista ilman kolmansien osapuolien sovelluksia.

Asiakkaan ja palvelimen välinen kommunikaatio alustetaan Web-sovelluksissa kuitenkin aina HTTP-protokollan mukaisesti. Pyynnöt ja vastaukset kulkevat HTTP-protokollassa perinteisesti portin 80 kautta. Jos asiakkaan ja palvelimen välillä käytetään HTTPS-protokollaa, joka on salattu HTTP-protokolla, kulkevat pyynnöt ja vastaukset perinteisesti portin 443 kautta. HTTPS-protokollan salaus perustuu SSL- tai TLS-protokollaan, joita käytetään suojaamaan tietoliikenne IP-verkon yli. HTTPS-protokolla perustuu varmenteisiin sekä yksityisiin ja julkisiin avaimiin, joilla tietoliikenne salataan ja salaus puretaan. Vaikka asiakkaan ja palvelimen välissä joku pääsisi käsiksi salattuna lähetettyyn tietoon, olisi tiedon alkuperästä käytännössä mahdoton päästä selville. Salattua yhteyttä käytetään yleensä arkaluontoisen tiedon välittämiseen asiakkaan ja palvelimen välillä. URI:n etuliite http:// tarkoittaa HTTP-protokollan ja https:// HTTPS-protokollan yhteyskäytäntöä.

WAUI:ssa asiakkaalta lähtevän ensimmäisen HTTP-pyynnön otsikkotiedoissa ehdotetaan kommunikointiin jatkossa käytettävän Web Socket -protokollaa. Protokollan vaihtaminen HTTP:stä Web Sockettiin on WAUI:ssa edellytys Metson järjestelmän arvojen päivittämiseen reaaliajassa asiakkaalta palvelimelle ja toisinpäin. Web Socket on HTTP-protokollan tavoin TCP-protokollaan perustuva protokolla. Web Socket -protokolla käyttää salaamattomaan kommunikointiin asiakkaan ja palvelimen välillä porttia 80 niin kuin HTTP:kin. Salattuun yhteyteen käytetään myös HTTP:n lailla porttia 443. Porttien 80 ja 443 käytön etuna palomuurillisissa ympäristöissä on se, että ne ovat standardinomaisia portteja HTTP/HTTPS-kommunikaatioon ja ovat sallittuina useissa palomuureissa. Web Socket -protokolla on laadittu niin, että myös välityspalvelimien tunnistaminen ja ohittaminen toimisi automaattisesti. URI:n etuliite ws:// tarkoittaa salaamattoman Web Socket -protokollan ja wss:// salatun Web Socket -protokollan yhteyskäytäntöä.

Metson nykyinen järjestelmä skaalautuu huonosti pienempään hinnallisesti. Metson järjestelmää valvotaan koko ajan erilaisten näyttöjen avulla. Isoissa sellutehtaissa valvomoita on useita ja pelkästään koko prosessin ohjaukseen on käytössä jopa 60 tietokonetta tehtaan alueella. Näyttöjä koko tehtaassa on useita satoja. Metson nykyisin käyttämän järjestelmän kustannusrakenne on kallis, jos tietokoneita ja näyttöjä on käytössä vain vähän.

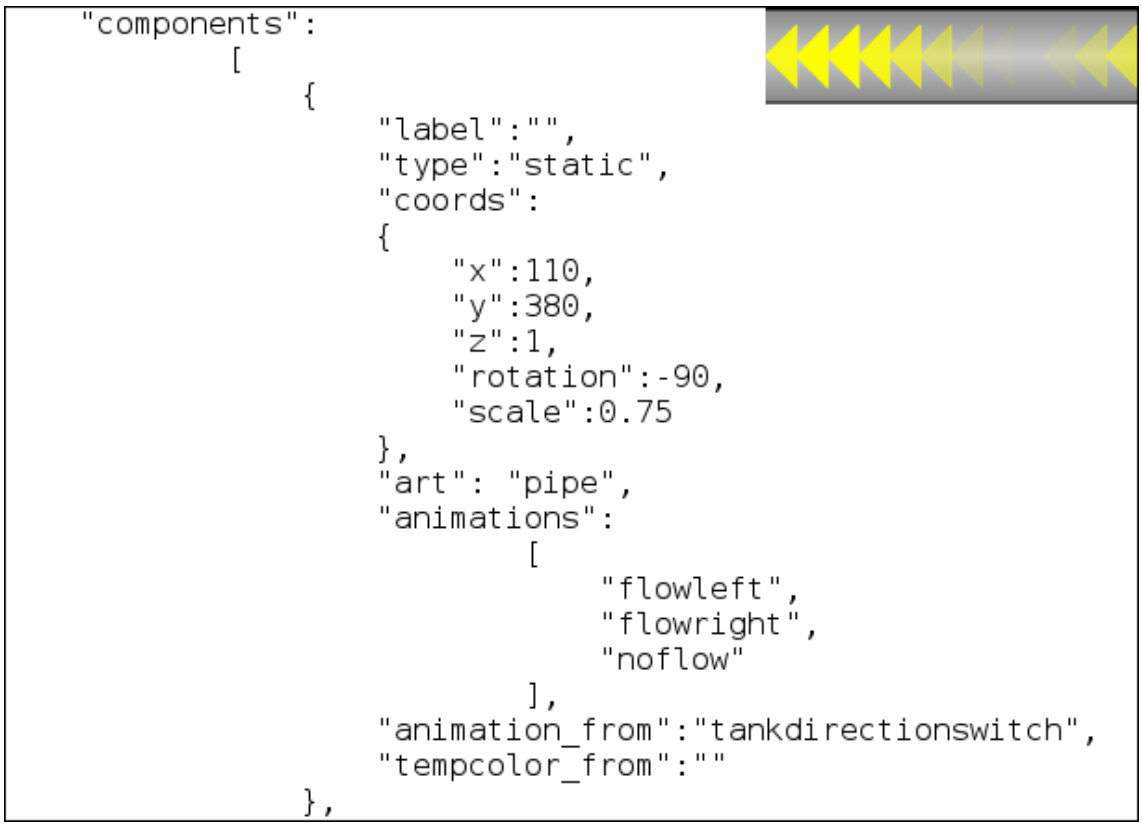
WAUI skaalautuu erittäin tehokkaasti eri laajuisiin järjestelmiin. WAUI:n käyttöliittymästä voidaan vaihtaa käyttöliittymässä näytettävää näkymää nappia painamalla. Nämä erilaiset näkymät kuvaavat prosessin eri osia ja vastaavat periaatteessa edellä kuvatussa järjestelmässä tietokoneisiin liitettäviä näyttöjä. WAUI:n käyttöliittymässä voi olla käytössä yhdestä näkymästä teoriassa loputtomaan määrään näkymiä. WAUI skaalautuu siis erittäin tehokkaasti niin pieniin kuin laajoihinkin järjestelmiin.

3.1 Asiakas

3.1.1 Näyttötiedosto

Kaikki käyttöliittymäkomponentit luodaan dynaamisesti näyttötiedoston avulla. Jokaiselle käyttöliittymäkomponentille on näyttötiedostossa oma lohko, joka määrittelee kaikki käyttöliittymäkomponenttiin liittyvät ominaisuudet. Näyttötiedosto on JSON-tiedosto, joka muunnetaan latauksen aikana JSON-olioksi, jota pidetään muistissa koko sovelluksen suorittamisen ajan. Tämä JSON-olio pitää sisällään sovelluksen käyttöliittymän tilaa.

Kuvassa 9 on esimerkki yhden käyttöliittymäkomponentin lohkosta. Kuvan 9 JSON-koodi määrittelee, kuinka osa laitteita yhdistävää putkea piirretään näytölle. Ohjelman sovelluslogiikka käsittelee kuvan 9 JSON-koodin ja sen seurauksena näytölle piirtyy kuvan 9 oikean yläkulman kaltainen putken osa.



```
"components":
        [
            {
                "label":"",
                "type":"static",
                "coords":
                {
                    "x":110,
                    "y":380,
                    "z":1,
                    "rotation":-90,
                    "scale":0.75
                },
                "art": "pipe",
                "animations":
                        [
                            "flowleft",
                            "flowright",
                            "noflow"
                        ],
                "animation_from":"tankdirectionswitch",
                "tempcolor_from":""
            },
```

Kuva 9. Yhden käyttöliittymäkomponentin lohko näyttötiedostossa.

Käyttöliittymäkomponenttien ominaisuudet määritellään näyttötiedostossa avain-arvo -pareilla. Kuvan 9 näyttötiedoston lohkossa määritellään ensimmäisenä putken etiketti. Etiketti määrittelee komponentin yhteyteen piirrettävän tekstin, joka saattaisi olla esimerkiksi komponentin nimi. Esimerkiksi nesteen virtauksen nopeutta ja lämpötilaa saattavat näyttää samanlaiset käyttöliittymäkomponentit. Tämän takia on oleellista määritellä label-avaimeen käyttöliittymäkomponentin näyttämä ominaisuus. Kuvan 9 tapauksessa etiketin arvo on tyhjä, koska putkelle ei haluta piirtää etikettiä.

Seuraavaksi kuvan 9 koodissa määritellään komponentin tyyppi. Putken tapauksessa tyyppi on static. Static ilmaisee komponentin olevan staattinen elementti eli se on pysyvä elementti käyttöliittymän näkymässä. Staattisten ja dynaamisten käyttöliittymäkomponenttien asetukset ovat hieman toisistaan poikkeavia.

Kolmas avain määrittelee komponentin sijainnin, asennon ja koon näytöllä (Kuva 9). X, y ja z määrittelevät komponentin sijainnin ko. akseleilla. Rotation määrää komponentin kierron ja scale määrittelee komponentin suhteellisen koon. Jokaisella komponentilla on alustava koko määriteltynä komponenttitiedostossa. Scale-avaimen arvo määrää, kuinka

paljon käyttöliittymäkomponenttia skaalataan suuremmaksi tai pienemmäksi ennen ruudulle piirtämistä.

Sovelluksen perustavanlaatuinen ominaisuus on, että komponentit ovat vuorovaikutuksessa toisiinsa. Putkessa kulkee tai ei kulje nuolia riippuen siitä, onko virtakytkin painettuna alas. Putkessa kulkevat nuolet ovat vihreitä tai punaisia riippuen esimerkiksi siitä, onko putkessa kulkevan nesteen lämpötila lähellä minimiä vai maksimia. Asetustiedoston animation_from ja tempcolor_from liittyvät käyttöliittymäkomponenttien vuorovaikutukseen.

Kuvan 9 putki-käyttöliittymäkomponentin animation_from-avaimen arvo on tankdirectionswitch. Tankdirectionswitch:lle on näyttötiedostossa samantyyppinen määrittely kuin putkelle. Tankdirectionswitch saa koodissa arvon 0 tai 1 sen mukaan, missä asennossa käyttöliittymäkomponentin kytkin on. Kytkimen ollessa arvossa 0, suoritetaan pipe.js-komponenttitiedoston flowleft-funktiota ja kytkimen ollessa arvossa 1, suoritetaan flowright-funktiota. Flowleft-funktiota suorittamalla putkessa saadaan kulkemaan nuolia vasempaan suuntaan ja flowright-funktiota suorittamalla nuolet kulkevat oikealle. Edelleen tankdirectionswitch-komponentin toiminta riippuu kuitenkin virtakytkimestä. Jos virtakytkin on ylhäällä, tankdirectionswitch saa arvon -1, jolloin suoritetaan pipe.js-komponenttitiedoston noflow-funktiota eli käytännössä käyttöliittymäkomponenttiin ei piirretä minkäänlaista animaatiota, koska noflow-funktion toteutus pipe.js-tiedostossa on tyhjä. Kuvan 9 näyttötiedoston lohkossa tempcolor_from on tyhjä, mutta sillä voidaan asettaa käyttöliittymäkomponentti hakemaan ja reagoimaan toisen käyttöliittymäkomponentin lämpötila-arvoon.

3.1.2 Komponenttitiedostot

Jokaisella sovelluksen käyttöliittymäkomponentilla on oma komponenttitiedosto. Komponenttitiedostot ovat js-tiedostoja, joissa on toteutettuna kaikki komponentin grafiikkaan ja animointiin liittyvä toiminnallisuus. Kuvan 9 näyttötiedoston lohkossa Art-avaimen arvo kertoo sovelluslogiikalle, mikä käyttöliittymäkomponentti näytölle piirretään. Kuvan 9 näyttötiedoston lohkossa näkymään määrätään piirrettäväksi putki-käyttöliittymäkomponentti. Komponenttitiedostoon liittyy myös näyttötiedoston animations-avain, joka määrää käyttöliittymäkomponentille suoritettavan funktion. Kuvan 9

näyttötiedoston lohkossa animations-avaimella on kolme erilaista arvoa: flowleft, flowright ja noflow. Nämä ovat kaikki funktioiden nimiä pipe.js-komponenttitiedostossa. Näihin funktioihin on toteutettuna käyttöliittymäkomponentien animointi. Komponenttitiedosto nimetään näyttötiedoston art-avaimen mukaiseksi. Näyttötiedoston art-avaimen arvo pipe viittaa siis komponenttitiedostoon pipe.js.

3.2 Palvelin

Palvelimella suoritetaan Tornadoa. Tornado on Pythonilla kirjoitettu Web-palvelin. Tornado on kaksisuuntainen ja ei-lukitseva. Kaksisuuntaisuus tarkoittaa, että kommunikaation asiakkaan ja palvelimen välillä voi aloittaa kumpi osapuoli tahansa. Tämän lisäksi kaksisuuntaisessa kommunikaatiossa tieto voi kulkea samaan aikaan asiakkaalta palvelimelle ja palvelimelta asiakkaalle. Kaksisuuntaisuus liittyy HTML5:n Web Socket -protokollaan. Tornado toteuttaa Web Socket -rajapinnan, mikä on oletus asiakkaan, jonka Web-selain toteuttaa myös Web Socket -rajapinnan, ja palvelimen väliselle Web Socket -kommunikaatiolle.

Ei–lukitseva palvelin on palvelin, joka voi palvella useita asiakkaita kerralla. Kaikki palvelimelle tulevat yhteydet asiakkailta saavat palvelimella oman säikeen sen sijaan, että kaikkia asiakkaita palveltaisiin ilman säikeistystä. Asiakas-palvelin -kommunikaation prosessointi ilman säikeistystä aiheuttaisi sen, että palvelimen kommunikoidessa yhden asiakkaan kanssa kaikki muut asiakkaat joutuisivat odottamaan edellisen asiakkaan ja palvelimen välisen kommunikaation loppumista. Säikeistys mahdollistaa usean asiakkaan ja palvelimen välillä käytävän kommunikaatioprosessin suorittamisen samanaikaisesti. Asiakkaan ja palvelimen välinen prosessi suoritetaan aina omassa säikeessä eikä säikeen tarvitse odottaa muiden säikeiden prosessien loppumista. Säikeissä suoritetut kommunikaatioprosessit eivät lähtökohtaisesti häiritse millään tavalla muissa säikeissä suoritettavia prosesseja.

Palvelin toimii WAUI:ssa kommunikaatiopalvelimena asiakkaan ja Metson järjestelmän välissä. Palvelin välittää Metson järjestelmän ohjattavien prosessien laitteiden arvoja asiakkaan ja palvelimen välillä. Tornadon koodiin on WAUI:ssa toteutettu asiakkaan ja palvelimen väliseen kommunikaatioon tarvittavan Web Socket -rajapinnan lisäksi toiminnallisuus, joka hakee ja tallentaa jatkuvasti tietoa ohjattavan prosessin laitteista. He-

ti, kun ohjattavan prosessin laitteiden arvot muuttuvat palvelimella, ne lähetetään asiakkaalle Web Socket -kanavaa pitkin.

4 TEKNIIKAT JA TOTEUTUS

Käyttöliittymän perustavanlaatuinen tarkoitus on alustariippumattomuus. HTML4 ei aivan ollut valmis tällaiseen sovellukseen ilman kolmansien osapuolten tekniikoiden, kuten Adoben Flashin tai Microsoftin ActiveX:n, käyttöä. Vaikka näiden tekniikoiden avulla oltaisiinkin päästy lähes vastaavanlaiseen toteutukseen, tekniikoiden käyttö olisi kuitenkin rajoittanut huomattavasti sovelluksen käyttöä erilaisilla alustoilla muun muassa yhteensopivuuden ja laitevaatimusten takia.

Käyttöliittymä toimii Web-selaimella ja hyödyntää moderneja tekniikoita kuten HTML5:a ja siihen läheisesti liittyvää JavaScript-rajapintaa. Seuraavassa esitellään WAUI:ssa käytettyjä tekniikoita tarkemmin. Käytettyjen tekniikoiden osalta käydään läpi ainoastaan se osa toiminnallisuutta, jota WAUI:ssa on käytetty. Osa läpikäytävistä tekniikoista on aivan uusia ja vanhat Web-selaimet eivät välttämättä tue niitä. Tarkempi selvitys näiden tekniikoiden yhteensopivuuksista Web-selainten kanssa löytyy Web-selainten yhteensopivuus-luvusta.

4.1 HTML

HTML on kuvauskieli, jolla kuvataan HTML-dokumentin semanttinen rakenne. Semanttisen rakenteen kuvaamiseen käytetään HTML-elementtejä, jotka merkitään koodiin niin sanotuilla tunnuksilla. Web-selain piirtää lopulta visuaalisen kokonaisuuden Web-selaimen käyttöliittymään HTML-dokumentin semanttisen rakenteen mukaan.

HTML-standardia ylläpitää ja kehittää WHATWG yhdessä W3C:n kanssa. WHATWG on kehittänyt HTML:a versiottomana vuodesta 2009 lähtien. WHATWG:n sivuilta löytyy jatkuvasti elävä HTML-spesifikaatio. W3C tulee kuitenkin tulevaisuudessa julkaisemaan HTML-standardin version 5.0 nimellä HTML5. HTML-standardin viimeisin virallinen versio on tällä hetkellä 4.01. W3C on vahvistanut, että HTML5-standardin odotetaan saavan suositus vuonna 2014. Käytännössä tuki HTML5:lle suosituimmissa selaimissa on kuitenkin jo lähes HTML5-spesifikaation mukainen.

WAUI:ssa on käytetty HTML5:a. HTML-elementtien osalta WAUI:ssa on käytetty canvas-elementtiä ja HTML-elementeissä käytettävää data-määrettä. Canvas-elementillä

sellaisenaan ei tee mitään Web-sovelluksissa. Canvas-elementtiin liittyy kuitenkin JavaScript-rajapinta, jonka avulla canvas-elementtiin voidaan piirtää. Canvas-elementin ja JavaScriptin yhteyttä selvitetään JavaScript-luvussa tarkemmin. Data-määre voidaan lisätä mille tahansa HTML-elementille. Data-määre on yleiskäyttöinen määre, jonne voidaan liittää sivuston tai sovelluksen toiminnan kannalta olennaista tietoa. Data-määreeseen voidaan esimerkiksi liittää musiikkikappaleiden pituuksia sivustolla, jossa käyttöliittymän sovelluslogiikka suodattaa tai uudelleenjärjestää kappaleita niiden keston mukaan. WAUI:ssa esimerkiksi ladataan oikea näyttötiedosto data-määreen avulla.

4.2 CSS

CSS on W3C:n ylläpitämä tyylikieli. CSS:lla kuvataan rakenteellisen dokumentin tyyli. Käytännössä tämä tarkoittaa dokumentin elementtien sijoittelun ja muotoilun määrittelemistä. CSS on kehitetty, jotta rakenteellisen dokumentin sisältö ja sen visuaalinen ilme voitaisiin erotella toisistaan. Tällainen erottelu tekee dokumentin luettavammaksi ja helpommin ylläpidettävämmäksi. CSS:ia käytetään usein HTML-dokumenttien yhteydessä: HTML:lla kuvataan dokumentin semanttinen rakenne ja CSS:lla määritellään dokumentin HTML-elementtien tyyli.

Tällä hetkellä CSS-standardin uusin versio on 2.1. CSS3 eli CSS-standardin versio 3.0 on vasta kehitteillä. Suuri osa CSS3-spesifikaation mukaisista ominaisuuksista on kuitenkin jo toteutettuna suosituimpien selainten uusimpiin versioihin.

WAUI:ssa CSS:ia on käytetty lähinnä elementtien sijoittamiseen oikeille paikoille. CSS:ia on kuitenkin käytetty myös jonkin verran elementtien ulkonäön määrittelemiseen. Esimerkiksi WAUI:ssa käytetyt elementtien varjostukset ovat CSS3-spesifikaation mukaiset.

4.3 JavaScript

JavaScript on olioperustainen, dynaaminen, heikosti tyypitetty, yleiskäyttöinen ohjelmointikieli. Olioparadigman lisäksi JavaScript tukee imperatiivista sekä funktionaalista ohjelmointiparadigmaa. JavaScript on kehittynyt tarpeesta luoda Web-sivuille dynaa-

mista sisältöä. JavaScript on formalisoitu ECMAScript-standardissa. ECMAScript-standardi on yhteinen useille johdannaisille kielille, kuten JavaScriptille ja JScriptille, joka on Microsoftin vastine Netscapen alun perin kehittämälle JavaScriptille. Valtaosassa nykyselaimia JavaScript-moottori on valmiiksi integroituna selaimeen ja näin selaimet tukevat JavaScriptin suorittamista sellaisenaan.

JavaScript on luokaton kieli eikä siis tue perinteistä olio-ohjelmointiin liittyvää luokkien periytymistä. JavaScriptissa funktiot toimivat olioina. Funktiosta voidaan luoda olioita new-avainsanalla samaan tapaan kuin luokista luodaan olioita perinteisissä olioparadigmaan pohjautuvissa kielissä. JavaScriptissa olioiden yhteinen rajapinta toteutetaan funktion, josta oliot luodaan, prototyyppiin. Kaikki prototyypistä johdetut oliot saavat automaattisesti käyttöönsä prototyyppiin kirjoitetun rajapinnan.

WAUI:n koodi on valtaosin JavaScriptiä. WAUI:ssa on käytetty nimenomaan JavaScriptiin perustuvaa jQuery-kehysohjelmistoa ja HTML5:n JavaScript-rajapintaa. Näillä tekniikoilla on toteutettu lähes koko asiakaspuoli käyttöliittymää ja sovelluslogiikkaa myöten. HTML5:n JavaScript-toiminnallisuutta ja jQuerya esitellään seuraavissa aliluvuissa tarkemmin.

4.3.1 Canvas

HTML5 määrittelee uuden canvas-nimisen HTML-elementin. HTML5:n canvas-elementti mahdollistaa dynaamisen scripti-perustaisen piirtämisen kyseiseen elementtiin. Canvas-elementtiä hallitaan JavaScript-rajapinnan avulla. Canvas-elementin käyttö vaatii tuen Web-selaimelta. Tällä hetkellä kaikki modernit Web-selaimet tukevat canvas-elementtiä, kuten Web-selainten yhteensopivuus-luvusta huomataan (Kuvio3; Kuvio 4).

Canvas on osa HTML5-spesifikaatiota, jota kehitetään W3C:n toimesta. HTML5-spesifikaatio määrittelee tällä hetkellä ainoastaan kaksiulotteisen grafiikan piirtämiseen tarkoitetun rajapinnan canvasille. Canvas-elementtiin on kuitenkin mahdollista piirtää myös kolmiulotteista grafiikkaa. Kolmiulotteista grafiikkaa voidaan piirtää käyttämällä WebGL-rajapintaa. WebGL:a ylläpitää oma WebGL-työryhmä. WebGL-rajapinta on

myös jo toteutettuna kaikissa suosituimpien selainten uusimmissa versioissa. WAUI:ssa on käytetty grafiikan piirtämiseen ainoastaan kaksiulotteista rajapintaa.

Canvas-elementtiin piirrettävät kuvat ovat bittikarttakuvia. Tämä tarkoittaa käytännössä sitä, että kerran Web-selaimessa piirrettyyn kuvaan ei voi millään tavalla viitata enää jälkeenpäin. Jos kuvaan halutaan tehdä muutoksia sen jälkeen, kun se on Web-selaimen käyttöliittymään piirretty, joudutaan koko kuva piirtämään uudelleen.

Canvas-tyylinen piirtäminen ei ole ollut mahdollista tähän asti. Canvas mahdollistaa hyvinkin monimutkaisen grafiikan piirtämisen. Hyvän kuvan canvasin mahdollisuuksista antaa se, että sillä on toteutettu WAUI:n lisäksi mm. Web-selain -pohjaisia tietokonepelejä.

Kaikki WAUI:ssa käytetty grafiikka on toteutettu canvas-elementtiä ja siihen läheisesti liittyvää JavaScript-API:a hyödyntäen lukuun ottamatta ohjauspaneelia, joka on toteutettu jQuerylla. Seuraavassa on esimerkki canvas-API:n käytöstä WAUI:ssa.

```
function draw(context) {

    context.save();

    const pipeWidth = 200;
    const pipeHeight = 50;

    var gradient = context.createLinearGradient(0, 0, 0, pipeHeight);

    gradient.addColorStop(0.0, "#888888");
    gradient.addColorStop(0.5, "#CCCCCC");
    gradient.addColorStop(1.0, "#888888");

    context.fillStyle = gradient;
    context.fillRect(0, 0, pipeWidth, pipeHeight);

    context.beginPath();

    context.moveTo(0,0);
    context.lineTo(pipeWidth,0);

    context.moveTo(0,pipeHeight);
    context.lineTo(pipeWidth,pipeHeight);

    context.strokeStyle = "rgb(255, 255, 255)";
```

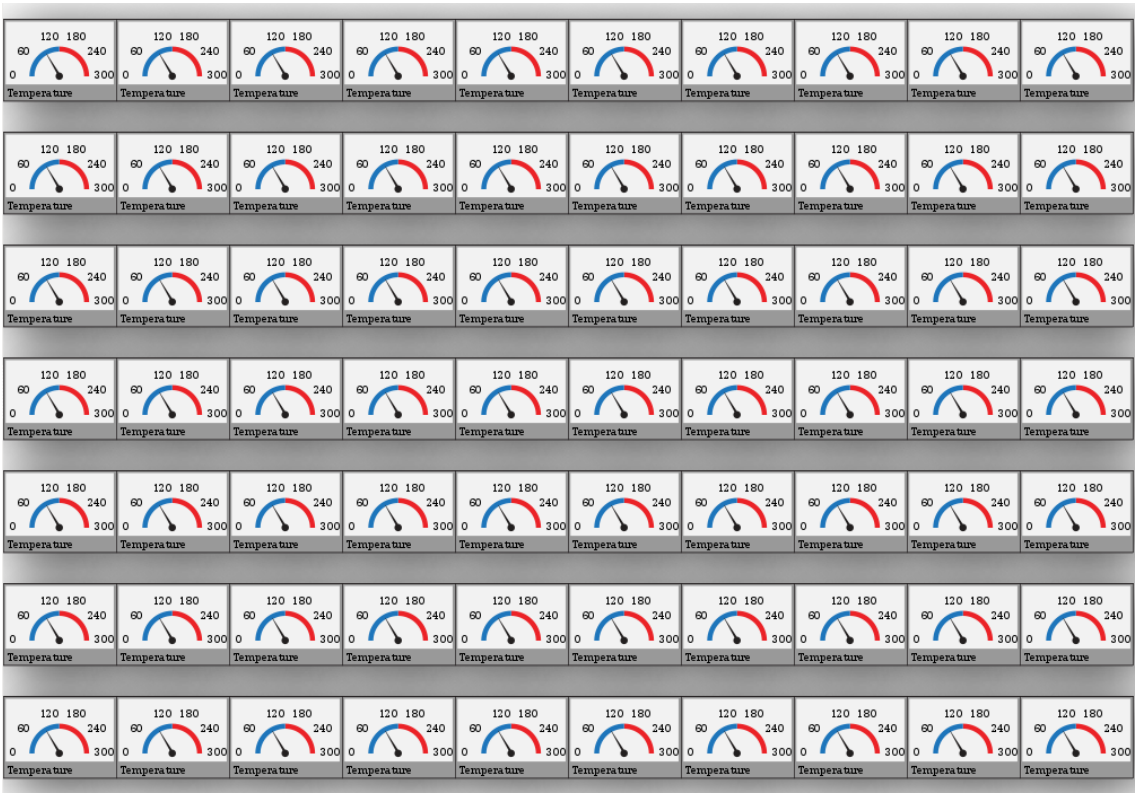
Kuva 10. Esimerkki canvas-API:n käytöstä putki-käyttöliittymäkomponentin komponenttitiedostossa.

Kuvassa 10 on osa erään käyttöliittymäkomponentin koodia. Koodista selviää hieman canvas-rajapinnan käyttöä. Jotta canvas-elementtiin voidaan piirtää, pitää canvas-elementiltä pyytää piirtoalue JavaScript-koodissa. Aiemmin kerrotiin canvas-elementin mahdollisuuksista kaksi- ja kolmiulotteiseen piirtämiseen. Nämä ovat kaksi erilaista canvas-elementin piirtoalueen tyyppiä.

Kuvan 10 esimerkissä käyttöliittymäkomponentin draw-funktio saa parametrinaan piirtoalueen, johon kuva on tarkoitus piirtää. Piirtoalue on itse asiassa CanvasRenderingContext2D-tyyppinen olio. Tältä oliolta pyydetään kuvan leveyden ja korkeuden määrittelemisen jälkeen CanvasGradient-tyyppistä oliota kutsumalla CanvasRenderingContext2D-olion createLinearGradient-metodia. CanvasGradient-olion rajapinnasta löytyy metodit liukuvärien käsittelyyn. Liukuväri liitetään piirtokontekstiin kutsumalla CanvasRenderingContext2D-olion fillStyle-metodia. Saman olion fillRect-metodi täyttää funktion alussa määritellyn kokoisen suorakaiteen olioon liitetyllä liukuvärillä. Koodin lopussa lisätään vielä piirtyvän suorakaiteen ympärille mustat viivat kutsumalla ensin moveTo-metodia, joka määrää koordinaatit, josta piirto aloitetaan ja vastaavasti lineTo, joka määrää koordinaatit, johon piirto lopetetaan. StrokeStyle-metodi määrää piirrettyjen viivojen tyylin, mitkä tässä tapauksessa määriteltiin vain väriltään mustiksi.

Käyttöliittymäkomponentit ovat pääosin piirretty Adobe Illustatorissa, josta ne on käännetty suoraan kuvasta JavaScriptiksi. Yhtä Adobe Illustatorissa piirrettyä kuvaa vastaa sovelluksen kansio- ja tiedostohierarkiassa yksi komponenttitiedosto. Sovelluslogiikka lukee näyttötiedoston asetuksia ja suorittaa sen mukaan erilaisia komponenttitiedostoja, joihin on toteutettu muun muassa se, kuinka käyttöliittymäkomponentti piirretään ruudulle.

Monimutkaisen kuvan piirtäminen canvas-elementtiin voi olla hyvinkin prosessori-intensiivistä. Varsinkin animoitujen elementtien piirtäminen on raskasta. Käyttöliittymäkomponenttien animaatiot toteutetaan käytännössä piirtämällä elementtiin toisistaan hieman poikkeavia kuvia peräkkäin. Riippuen animaation temmosta, kuvia voidaan joutua piirtämään hyvinkin lyhyin väliajoin. Animoitujen käyttöliittymäkomponenttien suhdetta suorituskykyyn eri laitteilla on testattu WAUI:n kehityksen ohessa seuraavanlaisilla stressitesteillä (Kuva 11).



Kuva 11. Stressitesti: 70 mittari-instanssia yhdessä näkymässä.

4.3.2 Web Storage

Web Storage on kuin eväste, jolle on löytynyt tuki jo varhaisimmistakin Web-selaimista. Web Storage oli alun perin osa HTML5-spesifikaatiota, mutta sille on nykyisin W3C:n ylläpitämä oma erillinen spesifikaatio. Web Storage on eräänlainen säilö, jonne voidaan tallentaa ja josta voidaan lukea tietoa. Web Storageen tallennetaan avain-arvo -pareja. Web Storageen voidaan tallentaa ainoastaan tekstimuotoista tietoa, mutta sinne voidaan tallentaa esimerkiksi JavaScript-olioita sarjallistamalla olio ensin JSON:ksi.

Evästeisiin liittyy suurehkoja rajoituksia ja niiden ominaisuudet eivät taivu kovin hyvin enää tämän päivän koko ajan kasvaviin ja monimutkaistuviin Web-sovelluksiin. Web Storage on kehitetty paikkaamaan evästeiden puutteita. Web Storage eroaa perinteisestä evästeestä seuraavalla tavalla: Web Storageen voidaan säilöä tietoa 5MB-10MB riippuen käytettävästä selaimesta. Evästeiden kapasiteetti on 4KB. Web Storageen mahtuu tietoa siis noin 1000-kertainen määrä verrattuna evästeeseen. Web Storage on tarkoitettu ainoastaan asiakaspuolen ohjelmointiin. Palvelimelta ei lähtökohtaisesti ole pääsyä Web Storageen eikä sen sisältämää tietoa lähetetä automaattisesti HTTP-pyynnön yhteydessä

palvelimelle toisin kuin evästeen tapauksessa. Web Storage koostuu kahdesta eri window-tason muuttujasta, joita ohjelmoija voi käyttää koodissaan käyttötarkoituksesta riippuen. Nämä ovat Local Storage ja Session Storage. Local Storage on verkkotunnuskohtainen säilö, jonne voidaan tallettaa pysyvää tietoa. Local Storagen tietoihin pääsee käsiksi ainoastaan samasta verkkotunnuksesta, josta tieto sinne tallennettiin. Session Storage on sivu- ja ikkunakohtainen säilö. Sinne voidaan tallettaa väliaikaista tietoa, joka pyyhitään muistista, kun käyttäjä sulkee selaimen ikkunan. Sivu- ja ikkunakohtainen säilö tarkoittaa sitä, että tietyn sivun tietyssä ikkunassa tallennettu tieto on saatavilla ainoastaan tässä ikkunassa eikä missään muualla. Tämän kaltainen erottelu säilöjen välillä poistaa tietyt evästeisiin liittyvät epäkohdat.

Web Storagea käytetään WAUI:ssa välimuistina raskaille käyttöliittymäkomponenteille. Riippuen tallennettavasta tiedosta, tieto tallennetaan joko Local Storageen tai Session Storageen JSON:na (Kuva 12). Käyttöliittymäkomponenttien raskas prosessointi jää pois, kun komponentit haetaan suoraan välimuistista, jossa ne ovat valmiiksi prosessoituina.

```
if (localstorageenabled) {
    try {
        sessionStorage.setObject("waui-controller" + jsonfile, controllerdata);
    } catch(e) {
        console.log("saving to sessionStorage failed:" + e);
    }
}
```

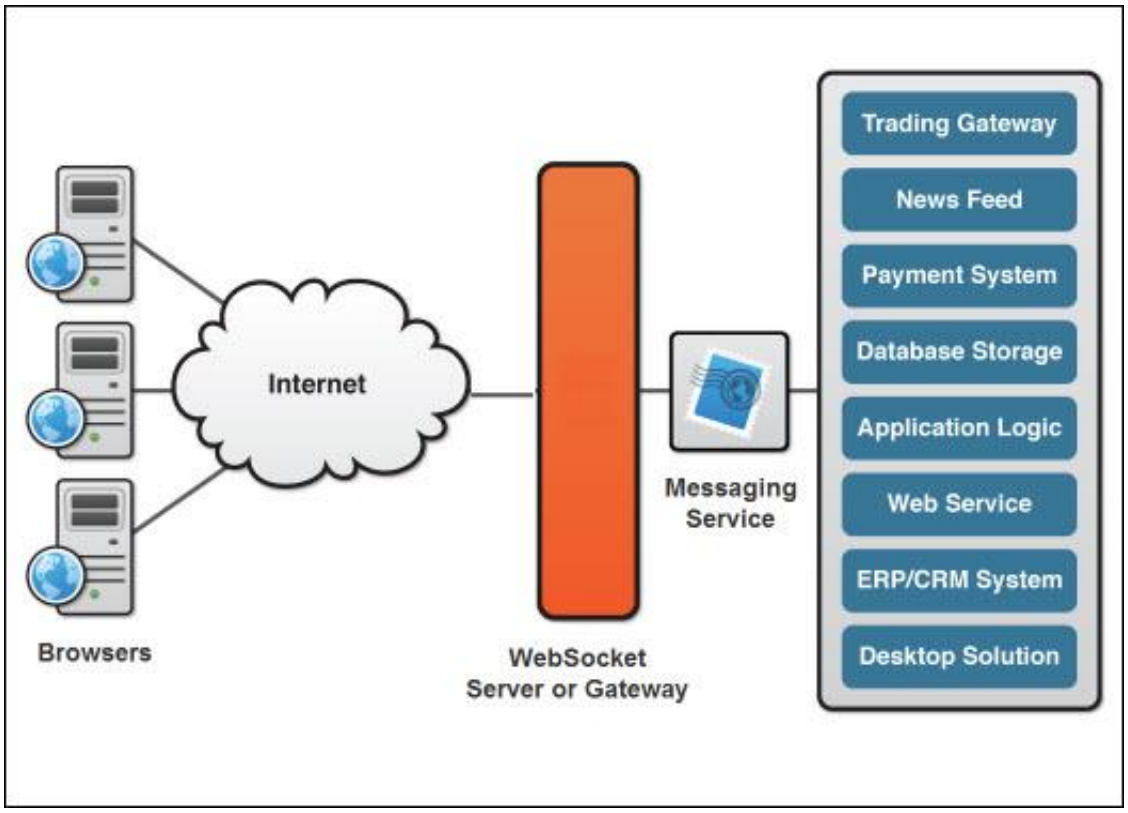
Kuva 12. SessionStorage-olion käyttöä sovelluslogiikassa.

SessionStorage-olion rajapintaan on lisätty WAUI:a varten setObject- ja getObject-metodit, joihin on toteutettu JSON-olioiden kirjoitus ja lukeminen sessionStoragesta (Kuva 12).

4.3.3 Web Socket

Web Socket on JavaScript-rajapinta, joka mahdollistaa käytännössä reaaliaikaisen, kaksisuuntaisen kommunikaatiokanavan asiakkaan ja palvelimen välillä. Kaksisuuntaisuudella tarkoitetaan, että tieto voi kulkea kanavaa pitkin kahteen suuntaan ja tiedon lähettäminen sekä vastaanottaminen voivat tapahtua samanaikaisesti. Web Socket oli alun

perin osa HTML5-spesifikaatioita, mutta spesifikaation ylläpitämisestä vastaa nykyään IETF.



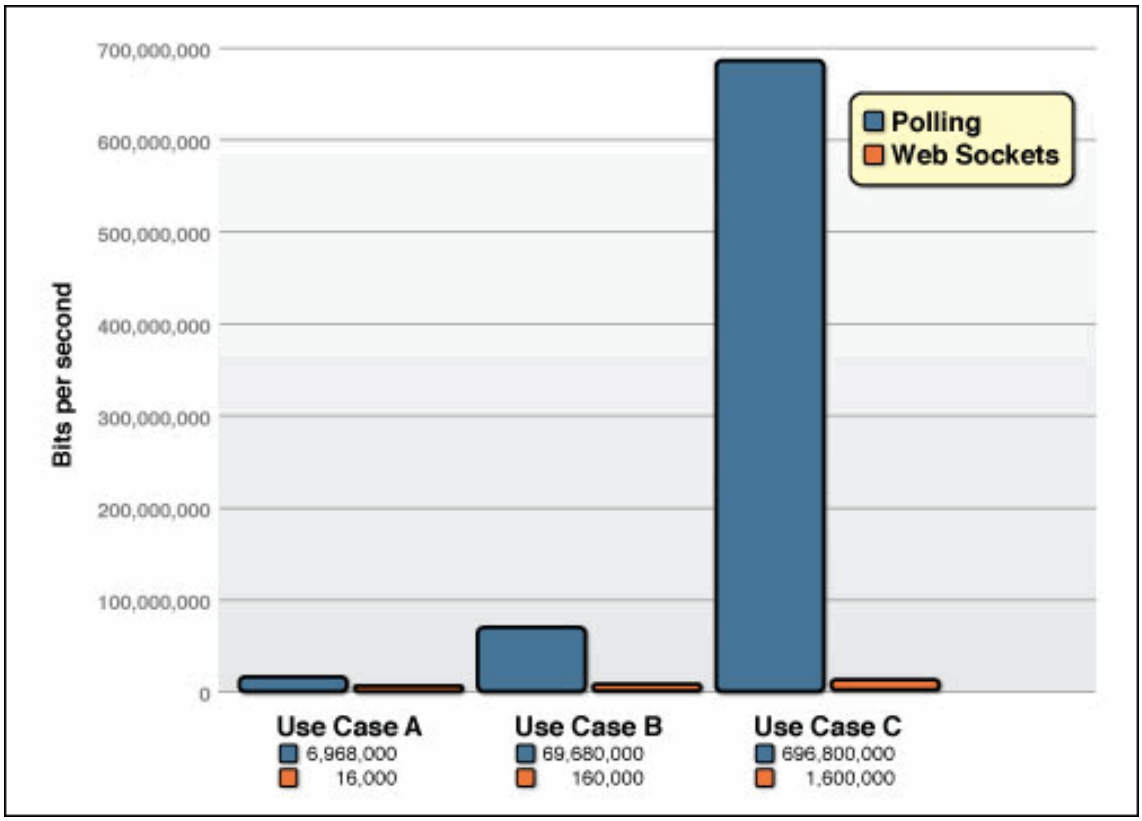
Kuva 13. Web Socket -arkkitehtuuri (http://www.websocket.org/aboutwebsocket.html 14.05.2012).

Perinteinen asiakas-palvelin -kommunikaatio HTTP-protokollassa perustuu pyyntö-vastaus -kommunikaatioon, jossa asiakas pyytää palvelimelta resurssia ja palvelin vastaa asiakkaalle palauttamalla pyydetyn resurssin. HTTP-protokollalle on ominaista, että asiakkaan ja palvelimen välisen kommunikaation aloittaa aina asiakas. Reaaliaikaisen resurssin toimittaminen asiakkaalle on käytännössä mahdotonta, koska asiakkaan pitää itse pyytää muuttunutta resurssia palvelimelta. Palvelin ei koskaan voi oma-aloitteisesti ottaa yhteyttä asiakkaaseen.

Perinteisissä Web-sovelluksissa resurssien reaaliaikaisuuteen on pyritty ottamalla asiakkaalta automaattisesti yhteyttä palvelimeen lyhyin väliajoin. Tällaista menetelmää kutsutaan termillä polling. Sen lisäksi, että tieto ei näin kuitenkaan ole reaaliaikaista, menetelmä saattaa rasittaa huomattavankin paljon Internet-yhteyttä. Tämä johtuu siitä, että niin asiakkaan palvelimelle lähettämässä pyynnössä kuin palvelimen asiakkaalle lähettämässä vastauksessa lähetetään mukana metatietoa, joka ei liity millään tavalla asiakkaan ja palvelimen välisen kommunikaation sisältöön.

Web Socket korjaa nämä perinteiseen pyyntö-vastaus -kommunikaatioon liittyvät epäkohdat tietynlaisissa sovelluksissa. Web Socket -kommunikaatio asiakkaan ja palvelimen välillä on käytännössä reaaliaikaista. Asiakas-palvelin -kommunikaation lisäksi myös palvelin-asiakas -kommunikaatio on mahdollista. Näiden lisäksi Web Socket säästää Internet-yhteyttä, koska asiakkaan ja palvelimen välillä lähetetään ainoastaan kommunikaation sisällön kannalta olennainen tieto (Kuvio 1).

Kuvio 1. Web Socket -polling -vertailu Internet-yhteyden kaistankulutuksen suhteen (http://www.websocket.org/quantum.html 14.05.2012)



Web Socket perustuu TCP-protokollaan ja protokollassa kommunikointi perustuu lähetettäviin ja vastaanotettaviin kehyksiin. Web Socket -protokollassa kehyksessä lähetetään ainoastaan varsinainen kommunikaatioon liittyvä sisältö sekä aloitus- ja lopetustavut, joiden avulla protokollassa merkitään viestin alkamista ja loppumista.

Kuviosta 1 selviää hyvin, kuinka oleellisesti Web Socket -kommunikaatio vähentää tietoliikenteen määrää asiakkaan ja palvelimen välillä verrattuna perinteiseen polling-ratkaisuun. Kuvaajasta selviää myös hyvin se, kuinka paljon polling-menetelmässä lähetetään turhaa, kommunikaation sisältöön liittymätöntä, tietoa asiakkaan ja palvelimen

välillä. Web Socket -palkki kuviossa 1 kuvaa käytännössä tietoliikenteen määrää, joka vähintään tarvitaan, jotta kommunikaatioon liittyvä varsinainen tieto voidaan saada välitettyä asiakkaalta palvelimelle tai toisinpäin.

```
REQUEST

GET /socket HTTP/1.1
Host: localhost:8888
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:12.0) Gecko/20100101 Firefox/12.0 Iceweasel/12.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive, Upgrade
Sec-WebSocket-Version: 13
Origin: http://localhost
Sec-WebSocket-Key: w+Uyiq/vu5I9tXHBRSEAlg==
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket

RESPONSE

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: g+yDauR5bRw70J93EGw43++u7tc=
```

Kuva 14. Web Socket -kanavan alustaminen HTTP-protokollalla.

Web Socket -kanava alustetaan asiakkaan ja palvelimen välille perinteisellä HTTP-protokollan mukaisella pyyntö-vastaus -menetelmällä. Web Socket on oma protokolla, joten HTTP-pyynnössä pyydetään protokollanvaihdosta Web Sockettiin (Kuva 14). Protokollan vaihdoksen jälkeen kommunikaatioon asiakkaan ja palvelimen välillä perustuu Web Socket -kanavaan eikä jatkossa kommunikaatioon tarvita enää HTTP-protokollaa.

```
[Constructor(DOMString url, optional (DOMString or DOMString[]) protocols)]
interface WebSocket : EventTarget {
  readonly attribute DOMString url;

  // ready state
  const unsigned short CONNECTING = 0;
  const unsigned short OPEN = 1;
  const unsigned short CLOSING = 2;
  const unsigned short CLOSED = 3;
  readonly attribute unsigned short readyState;
  readonly attribute unsigned long bufferedAmount;

  // networking
  [TreatNonCallableAsNull] attribute Function? onopen;
  [TreatNonCallableAsNull] attribute Function? onerror;
  [TreatNonCallableAsNull] attribute Function? onclose;
  readonly attribute DOMString extensions;
  readonly attribute DOMString protocol;
  void close([Clamp] optional unsigned short code, optional DOMString reason);

  // messaging
  [TreatNonCallableAsNull] attribute Function? onmessage;
           attribute DOMString binaryType;
  void send(DOMString data);
  void send(ArrayBufferView data);
  void send(Blob data);
};
```

Kuva 15. Web Socket -rajapinta (http://dev.w3.org/html5/websockets/ 14.05.2012).

Web Socket -rajapinta on yksinkertainen, mutta riittävä. Rajapinta koostuu lähinnä tapahtumankäsittelijöistä: onopen, onerror, onclose ja onmessage. Tapahtumankäsittelijät reagoivat erilaisiin tapahtumiin Web Socket -kanavassa. Onopen-funktio suoritetaan, kun Web Socket -kanava asiakkaan ja palvelimen välillä saadaan alustetuksi. Onerror-funktio suoritetaan kanavassa sattuneen virheen seurauksena. Onclose suoritetaan, kun kanava asiakkaan ja palvelimen välillä katkeaa. Onmessage-funktioon toteutetaan kanavan toiselta osapuolelta saatavaan viestin käsittely. Send-funktiota käytetään viestin lähettämiseen kanavan vastakkaiselle osapuolelle (Kuva 15).

Web Socketin käyttämiseksi vaaditaan modernin Web-selaimen lisäksi palvelimen tuki Web Socketille. Palvelimella täytyy pyöriä prosessi, joka osaa kommunikoida Web Socketin kanssa. Toisin sanoen palvelimella pyörivän sovelluksen pitää toteuttaa Web Socket -rajapinta. WAUI:ssa palvelimella suoritetaan Tornadoa, joka on Pythonilla kirjoitettu Web-palvelin. Tornado toteuttaa Web Socket -rajapinnan.

4.3.4 JQuery

JQuery on sovelluskehys, joka on kirjoitettu JavaScriptilla. JQueryn tarkoitus on abstrahoida käyttäjän näkökulmasta JavaScriptin käyttöä. JQueryn kirjastoihin on kirjoitettu suurempia toiminnallisia kokonaisuuksia, joita ohjelmoija voi käyttää esimerkiksi yhdellä funktiokutsulla sen sijaan, että kirjoittaisi vastaavan toiminnallisuuden itse alusta alken. JQueryn painopiste on DOM:ssa, tapahtumankäsittelyssä, animoinnissa sekä Ajax:ssa. WAUI:ssa lähes kaikki JavaScript on kirjoitettu jQuerylla.

JQueryUI on jQueryn päälle toteutettu käyttöliittymäkirjasto. JQueryUI:sta löytyy valmiita käyttöliittymäkomponentteja. WAUI:ssa jQueryUI:a on käytetty niukasti. Sillä on toteutettu ainoastaan ohjauspaneelin ohjaimet, jotka mahdollistavat ohjattavan prosessin laitteiden arvojen säätelyn.

4.4 Python

Python on yleiskäyttöinen, korkean tason ohjelmointikieli. Pythonin suunnittelufilosofia perustuu mahdollisimman luettavaan koodiin. Python tukee useita eri ohjelmointiparadigmoja. Tällaisia ovat esimerkiksi olioperustainen, imperatiivinen ja funktionaalinen ohjelmointiparadigma. Pythonissa, samoin kuin WAUI:ssa muutenkin paljon käytetyssä JavaScriptissa, muuttujat ovat dynaamisesti tyypitettyjä. Dynaamisesti tyypitetyille muuttujille ei tarvitse antaa staattista tyyppiä alustuksen yhteydessä vaan niiden tyyppi muuntuu aina muuttujaan tallennetun tiedon mukaan. Python toimii useilla eri alustoilla, kuten Windows, Linux/Unix ja Mac OS X.

WAUI:ssa Pythonia on käytetty palvelimella. Palvelimella suoritetaan Tornado-nimistä Pythonilla kirjoitettua Web-palvelinta. Tornado toteuttaa Web Socket -rajapinnan. Web Socket -rajapinnan toteuttaminen asiakas- ja palvelinpäässä on edellytys Web Socketin käyttämiselle.

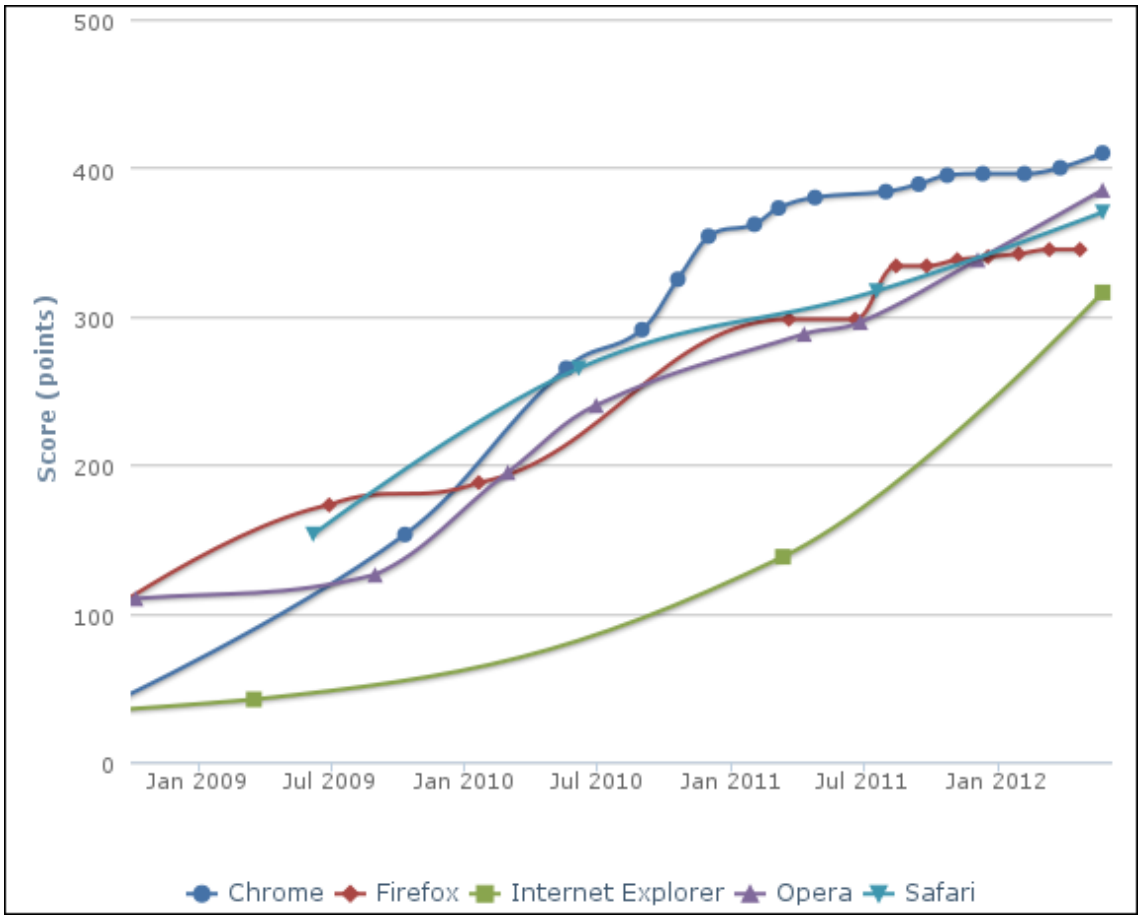
Tornadon koodiin on asiakkaan ja palvelimen väliseen Web Socket -kommunikaatioon tarvittavan Web Socket -rajapinnan lisäksi toteutettu toiminnallisuus, joka päivittää jatkuvasti ohjattavan prosessin laitteiden arvoja palvelimelle Metso DNA automaationjärjestelmästä. Metso DNA automaatiojärjestelmän kanssa kommunikoidaan CNP-

protokollan avulla. Tornadon koodissa CNP-protokollaa käytetään Metso CNP-kirjaston avulla. Kirjasto toteuttaa luku- ja kirjoitusrajapinnan Metso DNA -automaatiojärjestelmään. Kirjastoa käytetään Python-koodista ctypesin avulla. Ctypes on kirjasto, joka mahdollistaa DLL-kirjastojen funktioiden suorittamisen Pythonista käsin.

4.5 Web-selainten yhteensopivuus

WAUI:n ohjelmointiin on käytetty moderneja Web-tekniikoita. Käyttäjän Web-selaimen pitää olla tarpeeksi uusi ja sen pitää ennen kaikkea tukea sovelluksessa käytettyjä tekniikoita, jotta sovellus toimii tarkoituksenmukaisesti. Suurin yhteensopivuusongelma WAUI:n ja Web-selainten suhteen on HTML5. Modernit Web-selaimet tukevat kuitenkin jo pääsääntöisesti hyvin HTML5:a. Kuvio 2 antaa yleiskäsityksen siitä, miten Web-selainten tuki HTML5:lle on lisääntynyt ajan mittaan. Kuviossa 2 500 pistettä vastaa täydellistä tukea kaikille HTML5:een liittyville ominaisuuksille.

Kuvio 2. Web-selaimien HTML5-yhteensopivuus aikajanalla (http://html5test.com/results/desktop.html 20.05.2012).



HTML5 on kuitenkin vain yläkäsite monille eri tekniikoille. WAUI:ssa käytettiin vain pientä osaa HTML5:n ominaisuuksista. Käytettyjä ominaisuuksia olivat Canvas, Web Socket ja Web Storage. Canvas mahdollistaa scripti-perustaisen piirtämisen canvas-elementtiin, Web Socket mahdollistaa kaksisuuntaisen kommunikaation asiakkaan ja palvelimen välillä ja Web Storage mahdollistaa tiedon säilömisen Web-sovelluksissa. Seuraavassa on hieman tarkempi erittely selaimien HTML5-tuesta. Kuviosta 3 selviää puhelimissa ja taulutietokoneissa käytettyjen Web-selainten tuki WAUI:ssa käytetyille HTML5-ominaisuuksille. Kuviosta 4 selviää työpöytätietokoneissa käytettävien selainten tuki.

Kuvio 3. Puhelimissa ja taulutietokoneissa käytetyiden Web-selainten HTML5-yhteensopivuus WAUI:ssa käytetyille tekniikoille.

Feature	Safari on iOS	Android Browser		Google Chrome	Amazon Silk	BlackBerry Browser	
Platform	iPhone, iPad	Phones (2.x & 4.0)	Tablets (3.0+)	Android 4.0	Kindle Fire	Phones	Tablet
Minimum version tested	3.2	1.5	3.0	16 Beta	1.0	5.0	1.0
Latest version tested	5.1	4.0	4.0	16 Beta	1.0	7.0	2.0
Web storage	true	2.0+	true	true	true	6.0+	true
Web Sockets	4.2+	□	□	true	□	6.1+	true
Feature	Safari on iOS	Android Browser		Google Chrome	Amazon Silk	BlackBerry Browser	
Version tested	iPhone, iPad	Phones (1.0-2.3)	Tablets (3.0+)	Android 4.0	Kindle Fire	Phones	Tablet
Canvas API	true	true	true	true	true	true	true

Feature	Nokia Browser		Internet Explorer	Opera Mobile	Opera mini	Firefox	webOS Browser	
Platform	MeeGo - N9	Symbian	Windows Phone 7.5	Android & Symbian	Java, iOS & Android	Android	HP Phones	HP TouchPad
Minimum version tested	1.2	^3	9	11	5	6	1.4	3.0
Latest version tested	1.2	Belle	9	12	7	12 beta	2.0	3.0.5
Web storage	true		true	true		true	true	true
Web Sockets	□		□	true		7+	□	3.0.5+
Feature	Nokia Browser		Internet Explorer	Opera		Firefox	webOS Browser	
Version tested	Meego - Nokia N9	Symbian	Windows Phone	Mobile	Mini	Android	Phones 1.x-2.x	TouchPad 3.0
Canvas API	true	Anna+	true	true	true	true	true	true

Kuvio 4. Työpöytätietokoneissa käytettyjen Web-selainten HTML5-yhteensopivuus WAUI:ssa käytetyille tekniikoille.

Canvas	Internet Explorer	Firefox	Chrome	Safari	Opera
		3.6			
	6.0	9.0			
	7.0	10.0	17.0		
	8.0	11.0	18.0	5.0	
Current	9.0	12.0	19.0	5.1	11.6
Near future	10.0	13.0	20.0	5.2	12.0
Farther future		14.0	21.0		
Web Socket	**Internet Explorer**	**Firefox**	**Chrome**	**Safari**	**Opera**
		3.6			
	6.0	9.0*			
	7.0	10.0*	17.0		
	8.0	11.0	18.0	5.0	
Current	9.0	12.0	19.0	5.1	11.6
Near future	10.0	13.0	20.0	5.2	12.0
Farther future		14.0	21.0		
Web Storage	**Internet Explorer**	**Firefox**	**Chrome**	**Safari**	**Opera**
		3.6			
	6.0	9.0			
	7.0	10.0	17.0		
	8.0	11.0	18.0	5.0	
Current	9.0	12.0	19.0	5.1	11.6
Near future	10.0	13.0	20.0	5.2	12.0
Farther future		14.0	21.0		

5 YHTEENVETO

Sovelluksesta tuli juuri sellainen kuin Demolassa alkaneesta WAUI-projektista kaavailtiinkin. WAUI:n käyttöliittymästä voidaan nyt lukea ja kirjoittaa takaisin Metson järjestelmän arvoja. WAUI:n toteutus on erittäin joustava: WAUI on alustariippumaton ja skaalautuu erinomaisesti erilaajuisiin järjestelmiin. WAUI:n käyttöliittymästä löytyy Metson toivomia animointeja ja kaiken kaikkiaan käyttöliittymästä saatiin aikaiseksi näyttävä ja käytännöllinen kokonaisuus.

Sovelluksen käyttö vaatii modernin Web-selaimen, koska sovelluksessa on käytetty moderneja tekniikoita, kuten HTML5:a. Vanhemmat Web-selaimet eivät ole yhteensopivia sovelluksen kanssa kuten on todettu tekniikat ja toteutus -luvussa.

Vaikka projektissa päästiinkin sovittuun päämäärään, jäi sovelluksen jatkokehittämiselle silti mahdollisuus. Käyttöliittymä-luvussa kuvattiin lyhyesti kehitysversiota käyttöliittymäeditorista. Editoria ei kuitenkaan ehditty ohjelmoimaan projektin puitteissa valmiiksi, koska WAUI:n toteutus itsessään vei niin kauan. Editorin jatkaminen loppuun ei kuitenkaan olisi pitkä projekti johtuen käyttöliittymän näkymien dynaamisesta luonnista. Toinen jatkokehitysmahdollisuus liittyy WAUI:n käyttöliittymän kolmiulotteisuuteen. Tällä hetkellä käyttöliittymässä käytetty grafiikka on kaksiulotteista, mutta kuten tekniikka ja toteutus -luvussa mainittiin, HTML5:n canvas-elementtiin on mahdollista ohjelmoida myös kolmiulotteista grafiikkaa WebGL-rajapinnan avulla.

WAUI on vasta prototyyppi valmiista sovelluksesta. Valmiiseen tuotteeseen olisi lisättävä vielä ainakin tietoliikenteen salaus sekä tietoturvamekanismit liittyen sovelluksen käyttöoikeuksiin. Näihin asioihin ei ehditty keskittymään projektin aikana.

LÄHTEET

Caniuse. 2012. When can I use... Support tables for HTML5, CSS3, etc. Luettu: 14.05.2012. http://caniuse.com/#feat=websockets, http://caniuse.com/#feat=namevalue-storage, http://caniuse.com/#feat=canvas

Ecma International. 2011. Final Draft Standard ECMA-262 edition 5.1, March 2011 (rev. 6). Luettu: 20.05.2012. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

Html5test. 2012. The HTML5 test - How well does your browser support HTML5?. Luettu: 20.05.2012. http://html5test.com/results/desktop.html

IETF. 2011. draft-ietf-hybi-thewebsocketprotocol-17 – The WebSocket protocol. Luettu: 14.05.2012. http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-17

JQuery. 2012. jQuery: The Write Less, Do More, JavaScript Library. Luettu: 10.05.2012. http://jquery.com/

JQueryUI. 2012. JQueryUI – Home. Luettu: 10.05.2012. http://jqueryui.com/home

Khronos. 2012. WebGL Specification. Luettu: 14.05.2012. https://www.khronos.org/registry/webgl/specs/1.0/

Mobilehtml5. 2012. Mobile HTML5 - compatibility tables for iPhone, Android, BlackBerry, Symbian, iPad and other mobile devices. Luettu: 20.05.2012. http://mobilehtml5.org/

Python. 2012. About. Luettu: 14.05.2012. http://www.python.org/about/

Tornadoweb. 2012. tornado.websocket — Bidirectional communication to the browser – Tornado 2.3 documentation. Luettu: 14.05.2012 http://www.tornadoweb.org/documentation/websocket.html

W3C. 2011. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. Luettu 14.05.2012. http://www.w3.org/TR/CSS2/

W3C. 2012. Embedding custom non-visible data with the data-* attributes. Luettu: 14.05.2012. http://dev.w3.org/html5/spec/global-attributes.html#embedding-custom-non-visible-data-with-the-data-attributes

W3C. 2012. The WebSocket API. Luettu: 14.05.2012. http://dev.w3.org/html5/websockets/

W3C. 2012. Web Storage -. Luettu: 14.05.2012. http://dev.w3.org/html5/webstorage/

WebSocket.org. WebSocket.org | About WebSocket. Luettu: 20.05.2012. http://www.websocket.org/aboutwebsocket.html

WHATWG. 2012. HTML standard. Luettu: 14.05.2012. http://www.whatwg.org/specs/web-apps/current-work/multipage/

WHATWG. 2009. Switching to an unversioned development model. Luettu: 14.05.2012. http://lists.whatwg.org/pipermail/whatwg-whatwg.org/2009-December/024477.html

WHATWG. 2012. The canvas element – HTML standard. Luettu: 14.05.2012. http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html