Tampereen ammattikorkeakoulu

Videosoittimen toteuttaminen HbbTV-alustalle

Miikka Ylisiurunen

OPINNÄYTETYÖ
Marraskuu 2025

Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

YLISIURUNEN, MIIKKA:
Videosoittimen toteuttaminen HbbTV-alustalle

Opinnäytetyö 48 sivua, joista liitteitä 0 sivua
Marraskuu 2025

Opinnäytetyössä analysoitiin HbbTV-alustalle yhteensopivan videosoittimen toteutusta, sen teknisiä erityispiirteitä sekä kehityksessä kohdattavia haasteita. Työn tarkoituksena oli muodostaa konkreettisiin esimerkkeihin perustuva opas, joka tarjoaa tietoa HbbTV-kehittäjille. Työ tehtiin yhteistyössä digitaalisen television ratkaisuihin erikoistuneen Sofia Digital Oy:n kanssa.

Työssä esitellään modulaarinen ohjelmistoarkkitehtuuri laitehajanaisuuden hallintaan. Arkkitehtuuri erottaa yhteisen logiikan erillisistä moduuleista, jotka vastaavat standardiversioiden vaatimista soitinteknologioista, kuten vanhemmasta OIPF-rajapinnasta ja moderneista HTML5- sekä MSE-pohjaisista toteutuksista. Tämä malli helpottaa alustan erityispiirteiden ja versioiden käsittelyä.

Opinnäytetyötä varten tehtiin videosoitin, jonka toiminnallisuus todennettiin useilla eri HbbTV-laitteilla. Raportissa kuvataan soittimen tärkeimmät osa-alueet ja niiden tekninen toteutus. Lisäksi työssä käsitellään ratkaisuja yleisiin yhteensopivuusongelmiin, kuten tekstitysten luotettava näyttäminen HbbTV-versiosta riippumatta.

Työssä esitelty arkkitehtuuri ja toteutustavat tarjoavat hyvän lähtökohdan HbbTV-videosoitinten jatkokehitykselle. Tulosten avulla voidaan rakentaa uudelleenkäytettävä soitinkomponentti, joka nopeuttaa tulevien HbbTV-projektien kehitystä.

Asiasanat: hbbtv, oipf, videosoitin, javascript

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Engineering

YLISIURUNEN, MIIKKA:
Creating a Video Player for the HbbTV Platform

Bachelor's thesis 48 pages, appendices 0 pages
November 2025

The objective of this thesis was to analyse the implementation of a video player for the HbbTV platform. The purpose of the study was to create a practical guide based on concrete examples for HbbTV developers to use as a reference. The thesis was done in collaboration with Sofia Digital Oy, a company specialising in digital television solutions.

A modular architecture was designed to help with the challenges of device fragmentation. The architecture separates common player logic from specialised modules that handle the technologies used by different HbbTV versions, such as the older OIPF interface and modern HTML5 and MSE implementations.

A video player was developed based on this architecture and its functionality was verified on a wide range of HbbTV devices. The report describes the core features of the player and their implementation. The study also presents solutions to common compatibility issues, such as a device-independent method for rendering subtitles.

The findings indicate that the architecture is an efficient solution for developing and maintaining HbbTV video players. The results of this thesis provide a solid foundation for creating a reusable player component for future HbbTV projects.

Key words: hbbtv, oipf, video player, javascript

SISÄLLYS

LYHENTEET JA TERMIT

Adaptiivinen suoratoisto	Tekniikka, joka vaihtaa videon laatua verkkoyhteyden mukaan
AIT	Application Information Table, signaali, joka kertoo televisiolle HbbTV-sovelluksesta
CSS	Cascading Style Sheets, kieli HbbTV-sovellusten ja verkkosivujen ulkoasun määrittelyyn
DRM	Digital Rights Management, tekniikka, joka estää sisällön luvatonta käyttöä
Fokus	Tila, joka näyttää, mikä sovelluksen elementti on valittuna
HbbTV	Hybrid Broadcast Broadband TV, teknologia, joka yhdistää internet-palvelut televisiolähetyksiin
HDR	High Dynamic Range, tekniikka, joka parantaa kuvan laatua ja värejä
HEVC	High Efficiency Video Coding, pakkausmenetelmä, joka pienentää videotiedoston kokoa ilman laadun heikkenemistä
HLS	HTTP Live Streaming, Applen kehittämä protokolla adaptiiviselle suoratoistolle
HTML	HyperText Markup Language, kieli HbbTV-sovellusten ja verkkosivujen rakenteen luomiseen
JavaScript	Ohjelmointikieli, jolla toteutetaan HbbTV-sovellusten ja verkkosivujen toiminnallisuus
MPD	Media Presentation Description, MPEG-DASH-suoratoiston manifestitiedosto, joka sisältää videon tiedot
MPEG-DASH	Dynamic Adaptive Streaming over HTTP, kansainvälinen standardi adaptiiviselle suoratoistolle
MSE	Media Source Extensions, selainten rajapinta, joka mahdollistaa adaptiivisen suoratoiston selaimissa
OIPF	Open IPTV Forum, organisaatio, joka kehitti määrityksiä televisiopalveluille

Polyfill	Koodi, joka lisää uuden ominaisuuden vanhaan selaimeen, joka ei tue sitä
Transpilointi	Prosessi, joka muuntaa uudemman koodin vanhemmaksi eli yhteensopivammaksi versioksi

1 JOHDANTO

Opinnäytetyön tarkoituksena on analysoida ja esitellä HbbTV (Hybrid Broadcast Broadband TV) -videosoittimen tärkeimpien toimintojen toteutusta sekä siihen liittyviä erityispiirteitä. Työssä käsitellään HbbTV-alustan vaatimuksia ja yleisimpiä kehityksessä kohdattavia haasteita sekä syvennytään videosoittimen toteutukseen eri HbbTV-versioiden vaatimusten mukaisesti. Tavoitteena on tuottaa konkreettisiin esimerkkeihin perustuva opas, josta on hyötyä sekä uusille että kokeneemmille HbbTV-kehittäjille.

HbbTV-teknologiaa hyödynnetään tapana tuoda internet-pohjaisia lisäpalveluita televisioihin. Teknologian laaja levinneisyys käy ilmi HbbTV Associationin maaliskuussa 2025 julkaisemasta tiedotteesta, jonka mukaan jo yli 100 miljoonaa eurooppalaista kotitaloutta omistaa HbbTV-yhteensopivan television (HbbTV Association 2025). Sovellusten kehitys alustalle eroaa huomattavasti normaalista verkkokehityksestä, sillä kehityksessä on huomioitava laitehajanaisuus, televisioiden heikko suorituskyky sekä kaukosäätimen asettamat haasteet. Toimivan ja luotettavan videosoittimen toteuttaminen vaatii syvällistä ymmärrystä alustan toiminnasta ja rajoitteista. Opinnäytetyö tehdään yhteistyössä digitaalisen television ratkaisuihin erikoistuneen Sofia Digital Oy:n kanssa.

2 HBBTV-TEKNOLOGIA JA KEHITYSYMPÄRISTÖ

2.1 HbbTV-standardi ja versiot

HbbTV yhdistää internet-yhteyden kautta ladattavat lisäpalvelut televisiolähetyksiin. Sen tavoitteena on parantaa katselukokemusta lisäämällä interaktiivisia palveluita osaksi televisiolähetystä. Tämä näkyy usein televisiolähetyksen aikana ilmestyvänä kehotteena, joka ilmoittaa käyttäjälle, että lisäpalvelu tai sovellus on saatavilla. Käyttäjä voi käynnistää palvelun kehotteessa esitetyllä painikkeella, kuten punaisella väripainikkeella. (HbbTV Association n.d.h.)

Teknologian toiminta perustuu televisiolähetyksen mukana lähetettävään tietoon, joka ilmoittaa televisiolle saatavilla olevasta sovelluksesta tai palvelusta. Televisio voi vastaanottaa tämän osana lähetystä tai käyttää internet-yhteyttään sen lataamiseen. Sovellus, eli verkkosivu, käynnistetään television sisäänrakennetussa selainympäristössä. (HbbTV Association n.d.i.) Sovellus voi sisältää esimerkiksi ohjelmaoppaan, uutisia, äänestyksiä tai jopa pelejä. Sovellus näyttää olevan osa television omaa käyttöliittymää, mutta se on teknisesti verkkosivu, joka on rakennettu HTML:n, CSS:n ja JavaScriptin avulla. Standardin ylläpidosta ja kehityksestä vastaa HbbTV Association. (HbbTV Association n.d.k.) HbbTV-standardista on julkaistu vuosien varrella useita eri versioita, jotka ovat lisänneet alustalle uusia ominaisuuksia. Versioiden väliset erot ovat merkittäviä etenkin videosoittimen kehityksessä.

2.1.1 HbbTV 1.0 ja 1.5

Ensimmäinen versio, HbbTV 1.0 (ETSI TS 102 796 V1.1.1), julkaistiin vuonna 2010. Sen toiminta perustui vanhempiin web-standardeihin ja OIPF-rajapintoihin (Open IPTV Forum). Videoiden toistaminen tapahtui `<object>`-elementin ja OIPF-määriteltyjen rajapintojen avulla, mutta sen tuki rajoittui progressiiviseen lataukseen ja yksinkertaisiin suoratoistoprotokolliin. (ETSI TS 102 796 V1.1.1 2010, 14, 36–38, 43–45.)

Seuraava versio, vuonna 2012 julkaistu HbbTV 1.5 (TS 102 796 V1.2.1), oli merkittävä parannus videotoiston kannalta. Se perustui edelleen samaan OIPF-arkkitehtuuriin kuin HbbTV 1.0, mutta se lisäsi tuen MPEG-DASH-suoratoistoprotokollalle (Dynamic Adaptive Streaming over HTTP), eli adaptiiviselle suoratoistolle. Standardi asetti tarkkoja vaatimuksia DASH-videoiden teknisille tiedoille sekä videotoiston toiminnalle eri tilanteissa. (ETSI TS 102 796 V1.2.1 2012, 16, 44–45, 82–86.)

2.1.2 HbbTV 2.x

HbbTV 2.0 (TS 102 796 V1.3.1) lisäsi tuen modernille HTML5-standardille, jonka myötä tuli useita parannuksia videotoistoon liittyen:

- HTML5:n `<video>`-elementille lisättiin tuki, jota voi käyttää videoiden toistamiseen aiemman `<object>`-toteutuksen lisäksi.
- HEVC-videokoodekin (High Efficiency Video Coding) tuki, joka tarjosi paremman kuvanlaadun pienemmällä verkkokuormituksella. (ETSI 2015; ETSI TS 102 796 V1.3.1 2015, 13, 24–26, 58–59.)

Alkuperäinen 2.0-versio päivitettiin kuitenkin nopeasti 2.0.1-versioon (TS 102 796 V1.4.1) jo ennen laajempaa käyttöönottoa. Päivitys sisälsi Iso-Britannian markkinan vaatimia muutoksia sekä korjauksia havaittuihin ongelmiin. (HbbTV Association n.d.b; ETSI TS 102 796 V1.4.1 2016, 13.)

Uudemmat versiot ovat jatkaneet ominaisuuksien lisäystä ja havaittujen ongelmien korjausta. Versiossa 2.0.2 lisättiin tuki HDR-tekniikalle (High Dynamic Range) (HbbTV Association n.d.c; ETSI TS 102 796 V1.5.1 2018, 15). Versiossa 2.0.3 lisättiin tuki MSE-rajapinnalle (Media Source Extensions), jota käytetään adaptiiviseen suoratoistoon selaimissa (HbbTV Association n.d.d; ETSI TS 102 796 V1.6.1 2021, 106, 111–113).

2.1.3 Standardin haasteet kehityksessä

Television ilmoittama HbbTV-versionumero ei aina takaa kaikkien standardin ominaisuuksien täydellistä toimintaa. Laitevalmistajien välisissä toteutuksissa on usein eroja ja ominaisuudet voivat toimia puutteellisesti tai eri tavalla laitteesta riippuen. Vanhempiin standardiversioihin on lisäksi julkaistu päivityksiä ja korjauksia vuosien varrella, mikä luo epäselvyyttä laitteiden todellisista ominaisuuksista, varsinkin kun laitteita päivitetään uudempiin versioihin vain todella harvoin. (HbbTV Association n.d.h.)

HbbTV-kehityksen yksi suurimmista haasteista on laitehajanaisuus. Laitehajanaisuudella tarkoitetaan sitä, että HbbTV-yhteensopivia laitteita on saatavilla lukuisilta eri valmistajilta ja ne vaihtelevat ominaisuuksiltaan merkittävästi. Laitteissa esiintyy vaihtelua esimerkiksi valmistajien, vuosimallien, käyttöjärjestelmien ja HbbTV-versioiden välillä. Tämän takia pelkästään standardin mukainen koodi ei välttämättä toimi kaikilla laitteilla. Haaste korostaa kattavan testaamisen tärkeyttä, jonka vuoksi sovelluksen toiminnallisuus on todennettava useilla eri valmistajien ja sukupolvien laitteilla. Tämän lisäksi sovelluksiin on usein lisättävä vararatkaisuja, joita käytetään, jos jokin ominaisuus ei toimi laitteessa odotetusti.

2.2 HbbTV-sovelluksen toiminta

HbbTV-sovellukset perustuvat web-teknologioihin, mikä tarkoittaa, että ne ovat verkkosivuja, jotka on suunniteltu ja optimoitu toimimaan television selainympäristössä. Sovellukset toteutetaan kolmella teknologialla, jotka ovat HTML rakenteen määrittelyyn, CSS ulkoasun hallintaan ja JavaScript toiminnallisuuden toteuttamiseen (ETSI TS 102 796 V1.1.1 2010, 11). Näitä teknologioita käytetään myös tavallisten verkkosivujen kehityksessä, mikä tarkoittaa, että samoilla taidoilla voidaan tehdä sovelluksia myös televisioille.

Sovelluksen toiminta ja elinkaari eroavat kuitenkin merkittävästi tavallisista verkkosivuista. Prosessi alkaa televisiolähetyksestä, jonka mukana lähetetään AIT (Application Information Table) -signaali. Tämä signaali sisältää tiedon saatavilla

olevasta HbbTV-sovelluksesta sekä sen verkko-osoitteen. Kun HbbTV-yhteensopiva televisio tunnistaa tämän signaalin, se voi näyttää käyttäjälle siinä määritellyn sovelluksen. (HbbTV Association n.d.g.) Usein sovellukset näyttävät automaattisesti vain pienen kehotteen televisiolähetyksen päällä, joka kertoo käyttäjälle, että sovellus on saatavilla ja ohjeistaa avaamaan sen. Kun käyttäjä avaa sovelluksen, eli painaa kehotteessa määritettyä painiketta, sovelluksen koko käyttöliittymä tulee näkyviin. Ladattu sovellus käynnistetään ja esitetään tyypillisesti televisiolähetyksen ohella, peittäen sen osittain tai kokonaan. Sovellus pysyy käynnissä, kunnes käyttäjä sulkee sen, tyypillisesti painamalla kaukosäätimen EXIT-painiketta tai siirtymällä pois kanavalta. (ETSI TS 102 796 V1.1.1 2010, 20–23.)

2.3 Kehitystyökalut ja -teknologiat

HbbTV-sovellusten kehitys eroaa verkkokehityksestä siinä, että selaimen kehitystyökaluihin ei yleensä ole pääsyä. Vaikka joissakin uusissa Samsungin televisioissa on tuki selaimen kehitystyökaluille, tämä on harvinaista (Samsung Developer n.d.). Tätä ongelmaa voi helpottaa sillä, että sovelluksen päällä näytetään tietoa sen toiminnasta, kuten tärkeiden muuttujien arvoja tai virheviestejä. Tämä ei kuitenkaan korvaa selaimen kehitystyökaluja, sillä niiden avulla saa tarkemman käsityksen sovelluksen toiminnasta.

Kehityksen alkuvaiheessa sovelluksen rakennetta, käyttöliittymää ja yksinkertaista logiikkaa voi kehittää tietokoneen selaimessa. Tietokoneella testattaessa ja kehitettäessä tulee kuitenkin ottaa huomioon erot todelliseen ympäristöön, jossa sovellusta käytetään. Televisioiden selainympäristöt ovat huomattavasti rajoittuneempia, sillä ne perustuvat usein vanhoihin versioihin ja on tyypillistä, että uuden television selain on useita vuosia vanha (HbbTV Association n.d.a).

Seuraava vaihe on testaus emulaattorilla. Emulaattorit ovat selainlaajennuksia, jotka jäljittelevät television selainympäristöä ja rajapintoja. Ne ovat hyödyllisiä sovelluksen yksinkertaisessa testaamisessa, mutta ne eivät koskaan vastaa täysin aitoa laiteympäristöä. Ne eivät pysty emuloimaan laitteiden toteutusten ja selainten välisiä eroja tai suorituskykyyn liittyviä rajoituksia. Sovelluksen toiminnan ja

yhteensopivuuden voi varmistaa ainoastaan testaamalla oikeilla HbbTV-yhteensopivilla laitteilla. On erittäin tärkeää tehdä lopullinen testaus mahdollisimman laajalla ja monipuolisella laitevalikoimalla, sillä eri valmistajien ja vuosimallien laitteiden HbbTV-toteutuksissa saattaa olla merkittäviä eroja. (HbbTV Association n.d.e.) Tämän lisäksi emulaattorit eivät pysty toistamaan DASH-videoita lainkaan OIPF- tai HTML5-rajapinnalla, mikä tekee videosoittimen testaamisesta todella haastavaa.

Suurin teknologioihin liittyvä haaste on laitteiden vaihteleva ja vanhentunut JavaScript-tuki. Moderni JavaScript-kehitys hyödyntää ominaisuuksia, joita monet televisiot eivät tue ollenkaan. (HbbTV Association n.d.a.) Tätä haastetta hallitaan transpiloimalla koodia. Transpilointiin tarkoitetut työkalut, kuten Babel, muuntavat modernilla syntaksilla kirjoitetun JavaScript-koodin vanhemmaksi eli yhteensopivammaksi versioksi (Babel n.d.).

Transpiloinnin lisäksi käytetään usein polyfillejä. Polyfill on koodinpätkä, joka lisää vanhempaan selaimeen toiminnallisuuden, jota kyseinen versio ei vielä tue (MDN Web Docs n.d.a). Jos selainympäristöstä puuttuu jokin uusi JavaScript-ominaisuus, polyfill saattaa lisätä sille toteutuksen. Polyfillit eivät kuitenkaan pysty lisäämään tukea aivan kaikille ominaisuuksille, joten kehityksessä on suositeltavaa käyttää vanhempia ominaisuuksia toiminnallisuuden varmistamiseksi. Koodin transpilointi ja polyfillit ovat tärkeitä työkaluja, jotka helpottavat sovellusten kehittämistä selainten vanhempiin versioihin. Niiden avulla voidaan kehityksessä hyödyntää moderneja JavaScript-ominaisuuksia, vaikka päätelaitteet eivät niitä normaalisti tukisi.

Verkkokehityksen ja HbbTV-kehityksen apuna käytetään lähes aina työkaluja, jotka automatisoivat useita vaiheita kehityksen ja lopputuloksen parantamiseksi. Yksi näistä on Vite, joka tekee edellä mainitun transpiloinnin ja yhdistää useat JavaScript- ja CSS-tiedostot muutamaksi optimoiduksi tiedostoksi, mikä parantaa sovelluksen latausaikoja (Vite n.d.).

3 VIDEOTOISTO HBBTV-SOVELLUKSISSA

3.1 Videon suoratoistoprotokollat

Ennen adaptiivisen suoratoiston yleistymistä sovellukset latasivat videot yksittäisinä isoina tiedostoina. Tämä tarkoittaa sitä, että soitin lataa yhtä suurta videotiedostoa ja toistaa sitä samalla. Tätä kutsutaan progressiiviseksi lataamiseksi ja sen suurin heikkous on sen kyvyttömyys mukautua käyttäjän internet-yhteyden nopeuden vaihteluihin, mikä johtaa usein huonoon ja pätkivään katselukokemukseen. Jos käyttäjän internet-yhteys hidastuu, videon toisto pysähtyy ja jää odottamaan, että soitin ehtii ladata, eli puskuroida, videota tarpeeksi. Videoiden progressiivinen lataaminen on edelleen mahdollista, mutta nykyaikainen videotoisto perustuu lähes aina adaptiiviseen suoratoistoon. (Francis 2023.)

Adaptiivisen suoratoiston avulla videon laatu mukautuu automaattisesti käyttäjän internet-yhteyden nopeuteen, mikä parantaa katselukokemusta sekä hitailla että nopeilla verkkoyhteyksillä. Jos internet-yhteyden nopeus hidastuu katselun aikana, soitin vaihtaa videon heikomman laatuiseksi versioksi. Vastaavasti nopeuden kasvaessa soitin parantaa videon laatua. Tämän mahdollistaa se, että videot jaetaan useisiin pieniin, muutaman sekunnin pituisiin osiin eli segmentteihin. Samasta videosta tehdään useita eri laatuisia versioita, joilla on eri bittinopeus ja resoluutio. Versioiden ja segmenttien tiedot kuvataan erillisessä manifestitiedostossa. Kun videon toisto aloitetaan, soitin lataa ensin manifestitiedoston ja alkaa sen jälkeen ladata videota segmentti kerrallaan. Soitin tarkkailee verkkoyhteyden nopeutta videotoiston aikana ja vaihtaa tarvittaessa automaattisesti eri laatutasojen välillä. Yleisimmät adaptiivisen suoratoiston protokollat ovat MPEG-DASH ja HLS (HTTP Live Streaming). (Francis 2023.) Adaptiivinen suoratoisto on tärkeää varsinkin HbbTV-sovelluksissa, sillä verkkoyhteyksien nopeudet voivat vaihdella paljon.

MPEG-DASH on kansainvälinen standardi adaptiiviselle suoratoistolle. DASH-protokollassa manifestitiedosto on XML-muotoinen MPD-tiedosto (Media Pre-

sentation Description), joka sisältää saatavilla olevat video- ja ääniraidat, eri laatutasot, koodekit ja segmenttien sijainnit (Mueller 2022). HbbTV-ympäristö on tukenut DASH-protokollaa versiosta 1.5 lähtien.

HLS on Applen vuonna 2009 kehittämä suoratoistoprotokolla. Se käyttää tekstimuotoisia soittolistatiedostoja, jotka kuvaavat saatavilla olevat versiot, kuten eri resoluutiot sekä tiedot videosegmenttien kestosta ja sijainnista. (Mux n.d.) HLS-tuki löytyy myös monista HbbTV-laitteista sen laajan yleistymisen vuoksi. Tukeen ei kuitenkaan voi luottaa, sillä HbbTV-standardi ei virallisesti tue sitä. On vaarallista olettaa, että kaikki HbbTV-laitteet tukisivat HLS:ää oikein. (HbbTV Association n.d.a.)

3.2 Videotoiston toteutustavat

Videoiden toistaminen on HbbTV-sovellusten yleinen toiminnallisuus ja sen tekninen toteutus on muuttunut merkittävästi versioiden välillä. Eri versiot tukevat erilaisia rajapintoja videoiden toistamiseen. Toteutustapa on valittava sen perusteella, minkä vuosimallin laitteiden yhteensopivuutta sovellukselta vaaditaan. Toteutuksessa on mahdollista käyttää joko vanhempien versioiden rajapintoja tai uudempien versioiden tukemia toteutuksia.

3.2.1 Toteutus OIPF-rajapinnalla

HbbTV-versioissa 1.0 ja 1.5 videotoisto perustui OIPF-rajapintaan ja videosoitin lisättiin sovellukseen `<object>`-elementtinä (HbbTV Association n.d.j). HbbTV 1.0 -versiossa elementti tuki pelkästään videoiden progressiivista latausta, mutta HbbTV 1.5 -versiosta alkaen se tuki myös videoiden adaptiivista suoratoistoa.

Videon toistoa pystyi ohjaamaan JavaScriptillä OIPF-standardin määrittelemien rajapintojen avulla. Rajapinta tarjosi yleisimmät toiminnot videon toistoon ja niiden toiminta eri tilanteissa oli tarkkaan määritelty. (HbbTV Association n.d.j.) Haasteena oli, että rajapintojen toteutus ja toiminta saattoi vaihdella laitteiden välillä, mikä teki yhdenmukaisen toiminnan toteuttamisesta haastavaa.

3.2.2 Toteutus HTML5-rajapinnalla

HbbTV 2.0 lisäsi tuen HTML5:n `<video>`-elementille (ETSI TS 102 796 V1.3.1 2015, 13, 24–26). Tämän version myötä videon toistossa pystyi käyttämään selaimista tuttua elementtiä, jonka rajapinta on myös monipuolisempi. Se tarjoaa laajan valikoiman ominaisuuksia ja metodeja, jotka antavat tarkan hallinnan soittimen tilaan. (MDN Web Docs n.d.b.)

HbbTV 2.0 ja uudemmat laitteet tukevat myös MPEG-DASH-videoiden toistamista `<video>`-elementin avulla (ETSI TS 102 796 V1.3.1 2015, 85–86). Testauksen aikana kuitenkin havaittiin, että tämä toimii virheellisesti joissain laitteissa. Ongelmia voi olla esimerkiksi ääniraidan vaihtamisessa tai tietynlaisten MPD-manifestien käsittelyssä.

3.2.3 Toteutus MSE-rajapinnalla

Jos HTML5-toteutuksessa ilmenee ongelmia, MSE-versio saattaa auttaa. MSE-tuki lisättiin osaksi standardia HbbTV 2.0.3 -versiossa, mutta se on saattanut toimia joissain laitteissa jo aiemmin (HbbTV Association n.d.d). MSE-rajapinnan monimutkaisuutta hallitaan käyttämällä avoimeen lähdekoodiin perustuvia soitinkirjastoja, kuten dash.js tai Shaka Player. Nämä kirjastot hyödyntävät MSE-rajapintaa ja ne vastaavat monimutkaisesta logiikasta, kuten sopivan laatutason valinnasta ja verkkoyhteyden tilan tarkkailusta (Dash Industry Forum n.d.; Shaka Project n.d.).

MSE-versio on myös todella hyödyllinen jo kehityksen aikana. Sen avulla voi toistaa DASH-videoita tietokoneen selaimella, toisin kuin OIPF- ja HTML5-versioilla. Tämä nopeuttaa kehitystä huomattavasti, sillä soittimen toiminnallisuuden testaamiseen ei tarvita televisiota.

3.3 Kaukosäädinohjaus

HbbTV-sovellusten suunnittelussa täytyy huomioida alustan käytettävyyden erot tavalliseen sovelluskehitykseen verrattuna. Sovellusta ohjataan kaukosäätimen rajallisella näppäinvalikoimalla, eikä hiiren tai kosketusnäytön avulla. Rajallisten näppäinten vuoksi sovelluksen kaikki toiminnot on suunniteltava käytettäviksi muutamalla eri painikkeella. Sovelluksen täytyy käsitellä käyttäjän tekemät painallukset ja yhdistää ne loogisiin toimintoihin sovelluksen sisällä. Navigointi tapahtuu pääasiassa kaukosäätimen nuolinäppäimillä ja OK-painikkeella. Usein sovelluksissa hyödynnetään myös kaukosäätimen väripainikkeita, jotka toimivat oikopolkuina sovelluksen eri osioihin tai toimintoihin. Monissa kaukosäätimissä olevat medianäppäimet, kuten play-, pause- ja kelauspainikkeet, toimivat usein vaihtoehtoisena tapana ohjata videosoitinta nuolinäppäinten lisäksi.

Sovellusten suunnittelun tärkeimpiä osa-alueita ovat selkeä navigointi ja fokuksen hallinta. Fokuksella tarkoitetaan sitä, mikä sovelluksen elementti, esimerkiksi painike, on valittuna. Käyttäjän painaessa kaukosäätimen painiketta sovellus toteuttaa valitun elementin toiminnon. Sovelluksessa on oltava aina yksi ja vain yksi kohde fokusoituna ja sen on erotuttava visuaalisesti muista elementeistä esimerkiksi eri värisen reunuksen tai taustavärin avulla. (Tizen n.d.) Fokuksen ja navigoinnin logiikka täytyy aina tehdä sovellukseen itse ja HbbTV-sovelluksissa tämä tehdään lähes aina JavaScriptillä. Kun käyttäjä painaa nuolinäppäintä, sovelluksen koodin on pääteltävä, mihin elementtiin fokus siirretään. Tämän siirtymän on oltava käyttäjälle looginen ja ennustettava, sillä huonosti suunniteltu ja toteutettu navigointi voi johtaa tilanteeseen, jossa sovelluksen käytöstä tulee epäselvää ja pahimmassa tapauksessa käyttäjä ei enää pysty ohjaamaan sovellusta. Tässä opinnäytetyössä esiteltävässä soittimessa ei käsitellä ollenkaan toiminnallisuuden toteuttamista elementtien välillä liikkumiseen.

HbbTV 1.0 -versiossa määritellään kaksi tapaa sovelluksen navigoinnille, JavaScript-navigointi sekä CSS3-ominaisuuksiin perustuva navigointi (ETSI TS 102 796 V1.1.1 2010, 51–52). Näistä ensisijainen ja yleisimmin käytetty on JavaScript-navigointi, jossa sovellus kaappaa näppäintapahtumat ja toteuttaa navigointiin liittyvän logiikan kokonaan itse. Toinen tapa on CSS3-ominaisuuksien hyödyntäminen, mutta sen toiminnallisuus on huomattavasti rajallisempaa.

HbbTV 2.0.2 -versiosta alkaen on määritelty, että CSS3-navigointi on vanhentunut ominaisuus ja sen käyttöä ei enää suositella uusissa sovelluksissa (ETSI TS 102 796 V1.5.1 2018, 119, 315).

3.4 Videosoittimen käyttöliittymä

Videosoittimen käyttöliittymän ja ohjauksen toteutukseen on yleistynyt useita erilaisia tapoja. Työssä esiteltyjä toteutuksia soittimen toiminnoille voi soveltaa ja käyttää minkä tahansa tavan kanssa. Yleisiä toteutustapoja ovat esimerkiksi seuraavat:

- Käyttöliittymä: Soittimen käyttöliittymä pidetään piilotettuna videon aikana. Se tulee esiin, kun käyttäjä painaa esimerkiksi OK- tai nuolinäppäintä. Käyttöliittymä piilotetaan jälleen muutaman sekunnin kuluttua, jos video on käynnissä, jotta se ei häiritse katselua. Käyttöliittymä sisältää usein videon edistymistä kuvaavan aikajanan sekä kuvakkeita asetuksille.
- Kelaaminen: Yksinkertaisin ja yleisin tapa toteuttaa kelaaminen on, että nuolinäppäimet kelaavat videota tietyn sekuntimäärän. Toisessa yleisessä toteutustavassa käyttöliittymässä on kelauspainikkeita, joiden avulla videota kelataan.
- Asetusvalikot: Asetukset sijoitetaan omiin valikkoihinsa, jotka avataan ohjauspalkissa olevilla kuvakkeilla. Asetuksiin voi kuulua esimerkiksi tekstitysten kielen valitseminen ja ulkoasun muuttaminen sekä ääniraidan vaihtaminen.

Toteutustavasta riippumatta on tyypillistä, että käyttöliittymä näytetään muutaman sekunnin ajan, kun käyttäjä esimerkiksi kelaa videota. Tämän jälkeen käyttäjälle näytetään toiston uusi sijainti sekä ilmoitetaan, että sovellus on vastaanottanut painalluksen.

3.5 Yleisimmät haasteet

Toimivan HbbTV-videosoittimen kehittäminen vaatii alustan haasteiden ymmärtämistä ja ratkaisemista. Nämä haasteet liittyvät laitteiden monipuolisuuteen ja

suorituskykyyn. HbbTV-yhteensopivia laitteita on useilta eri valmistajilta ja ne vaihtelevat ominaisuuksiltaan, suorituskyvyltään sekä toteutusten laadultaan. Laitteiden selainten, koodekkituen ja rajapintojen toteutuksissa on usein pieniä, mutta merkittäviä eroja, jotka tulee ottaa huomioon sovellusta kehitettäessä. Yksi ominaisuus voi toimia täydellisesti yhdellä laitteella, mutta puutteellisesti tai ei ollenkaan toisella. Tämän takia sovelluksiin täytyy usein tehdä vararatkaisuja ja testata niiden toimivuus kattavasti eri laitteilla. Televisioiden suorituskyvyssä on myös suuria eroja laitteiden välillä. Monimutkaiset JavaScript-operaatiot tai animaatiot voivat hidastaa käyttöliittymän vasteaikoja tai jopa kaataa koko sovelluksen. Tämän vuoksi on tärkeää minimoida resurssien käyttö. (HbbTV Association n.d.e.)

Virheenkäsittely on videotoistossa erittäin tärkeää. Verkkoyhteys voi hetkellisesti hidastua tai jopa katketa, tai palvelin voi lähettää vääränlaista dataa. Hyvin rakennetun soittimen täytyy käsitellä nämä tilanteet hallitusti. Sen sijaan, että soitin pysähtyy ja näyttää vain mustan ruudun, sen tulisi esimerkiksi yrittää ladata epäonnistunut videosegmentti uudelleen, siirtyä seuraavaan segmenttiin tai ilmoittaa virheestä käyttäjälle selkeästi.

4 VIDEOSOITTIMEN TEKNINEN TOTEUTUS

4.1 Toteutuksen lähtökohdat

Tässä opinnäytetyössä esiteltävän videosoittimen tavoitteena on havainnollistaa aiemmissa luvuissa käsiteltyjä teknologioita, versioita ja menetelmiä. Toteutuksessa on tärkeää huomioida kaikki HbbTV-standardin versiot, jotta videosoitin toimisi mahdollisimman monella laitteella. Työssä esitellään soittimen toteutus käyttäen kolmea eri rajapintaa, jotka kattavat kaikki nykyiset HbbTV-versiot. Kaikkien rajapintojen tukemisella varmistetaan mahdollisimman hyvä toiminnallisuus eri sukupolvien laitteilla. Tavoitteena on luoda arkkitehtuuri, jonka avulla soittimen versio voidaan valita automaattisesti laitteen version perusteella. Videosoittimeen toteutetaan sille yleisimmät ja tärkeimmät ominaisuudet:

- Videon lataaminen ja toiston aloittaminen.
- Toiston ohjaus, kuten tauotus ja kelaus.
- Tekstitysten näyttäminen.
- Ääniraitojen vaihtaminen.

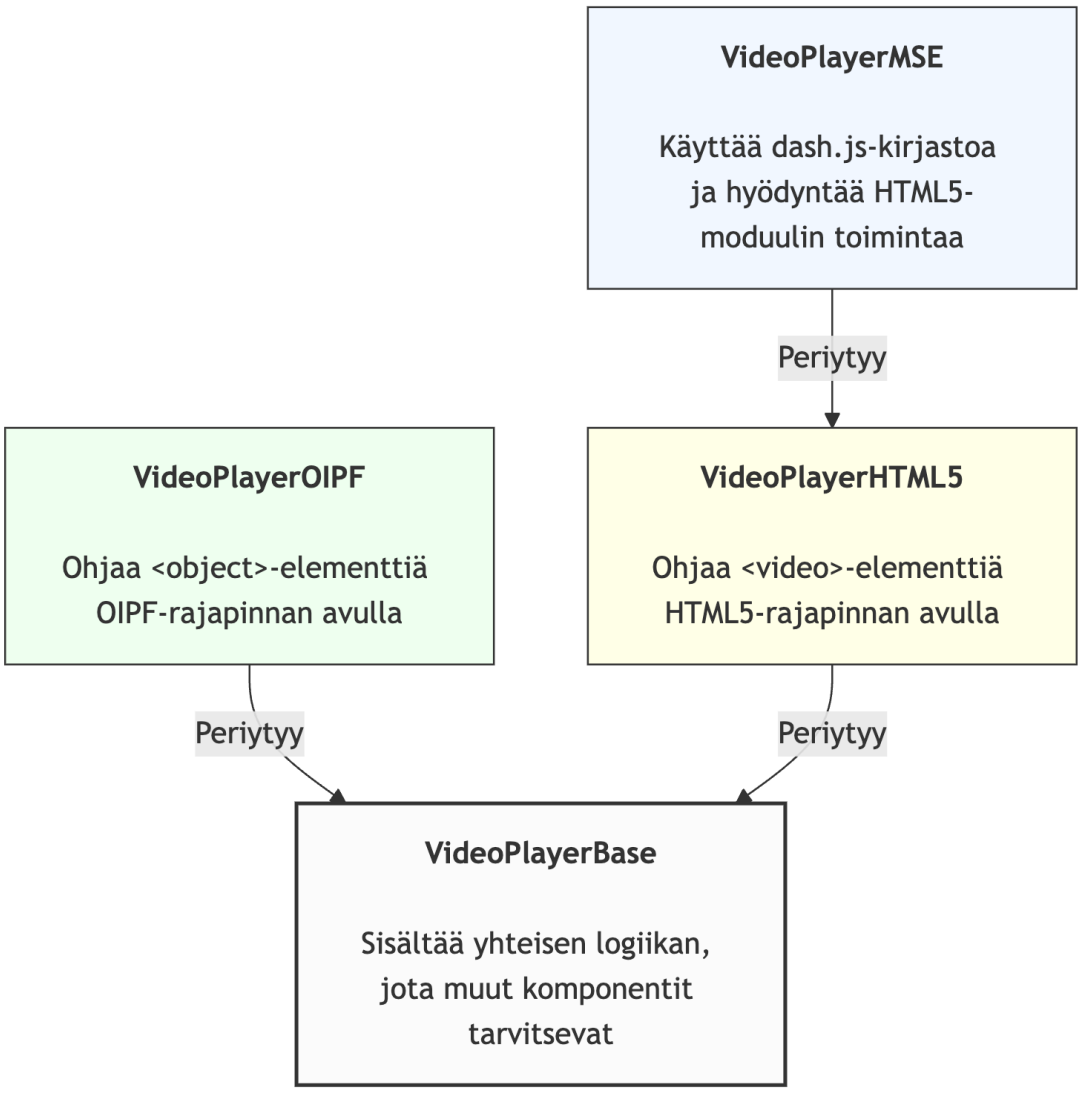
4.2 Modulaarinen ohjelmistoarkkitehtuuri

Modulaarinen ohjelmistoarkkitehtuuri helpottaa ylläpidettävän HbbTV-videosoittimen rakentamista. Videosoittimen on lähes aina toimittava usean sukupolven laitteissa, jotka tukevat eri teknologioita, joten sen toiminta kannattaa jakaa pienempiin teknologiakohtaisiin moduuleihin. Vaikka vanhemman OIPF-rajapinnan käyttäminen on mahdollista myös uusissa laitteissa, sen täydellinen toiminta ei ole aina taattua. Tässä luvussa esitellään videosoittimen arkkitehtuuri, jonka tavoitteena on hallita HbbTV-ympäristön hajanaisuutta ja luoda selkeä, helposti ylläpidettävä koodipohja.

4.2.1 Arkkitehtuurin osat

Arkkitehtuurissa soittimen yhteinen logiikka erotetaan teknologiakohtaisista toteutuksista. Tämän avulla kaikkia soittimen toimintoja, kuten käyttöliittymän päivittämistä ja näppäinkomentojen käsittelyä, ei tarvitse lisätä jokaiseen moduuliin erikseen. Tämä rakenne parantaa koodin ylläpidettävyyttä, mikä puolestaan helpottaa uusien ominaisuuksien lisäämistä tulevaisuudessa. Kuviossa 1 on esitelty videosoittimen arkkitehtuuri, joka koostuu seuraavista komponenteista:

- `VideoPlayerBase`: Tämä sisältää kaiken yhteisen logiikan, jota soittimen muut komponentit tarvitsevat. Tähän kuuluu esimerkiksi käyttöliittymän päivitys soittimen tilan mukaan ja muut metodit, joita sovelluksen muu osa käyttää. Se ei kuitenkaan sisällä koodia videon toiston hallintaan. Soittimen kaikki muut komponentit rakennetaan tämän päälle.
- `VideoPlayerOIPF`: Tämä moduuli sisältää OIPF-rajapintaa käyttävän toteutuksen. Se käyttää vanhempien HbbTV-laitteiden `<object>`-elementtiä ja toteuttaa toiminnot, kuten videon toisto ja tauotus, OIPF-rajapinnan vaatimilla tavoilla.
- `VideoPlayerHTML5`: Tämä moduuli on suunniteltu HbbTV 2.x -laitteille, jotka käyttävät `<video>`-elementtiä. Se ohjaa videon toistoa HTML5-rajapinnan avulla.
- `VideoPlayerMSE`: Tämä on edistynein moduuli, joka on suunniteltu HbbTV 2.x -laitteille, joissa MSE-rajapinta on tuettu ja missä on testauksen aikana todettu ongelmia muiden versioiden kanssa. Tämä moduuli periytyy `VideoPlayerHTML5`-moduulista ja käyttää dash.js-kirjastoa videon toistamiseen.



KUVIO 1. Videosoittimen arkkitehtuuri.

4.2.2 Version automaattinen valinta

Oikean soitinversion valitseminen tehdään HbbTV-version perusteella. Valinta-prosessi suoritetaan sovelluksen käynnistyessä ja sen tavoitteena on valita, mitä versiota soittimesta käytetään. Logiikka valinnalle on yksinkertainen ja se voi edetä esimerkiksi seuraavasti:

1. Ensin tarkistetaan, onko kyseessä televisio, jonka videotoistossa tiedetään olevan ongelmia. Jos testauksen aikana on havaittu ongelmia, sille voidaan pakottaa käyttöön jokin tietty, toimivaksi todettu versio. Laite voidaan pakottaa käyttämään mitä vain kolmesta versiosta.
2. Jos kyseessä on HbbTV 2.x -laite, käytetään HTML5-versiota.
3. Jos laite on HbbTV 1.5 tai vanhempi, käytetään OIPF-versiota.

4.2.3 Soitinkirjastojen hyödyntäminen

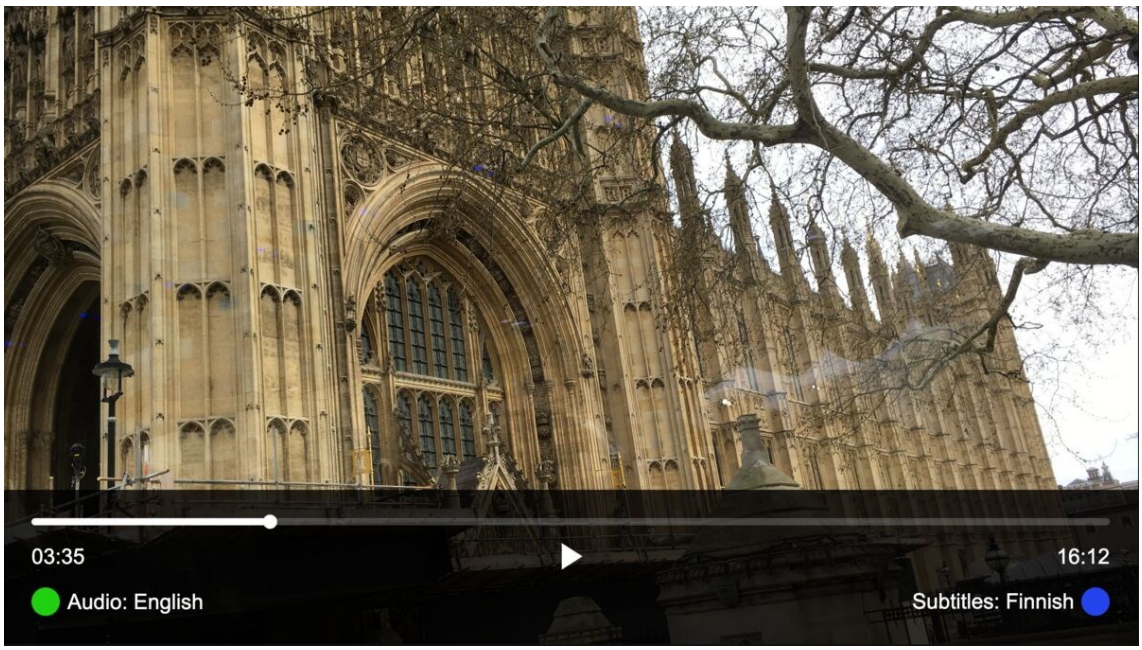
Videosoittimen toteuttaminen alusta alkaen käyttämällä MSE-rajapintaa on monimutkaista. Toteutus vaatii manifestitiedoston käsittelyä, videosegmenttien lataamista, laatutason vaihtoa verkkoyhteyden mukaan sekä monia muita toimintoja. Tämän toteuttaminen itse olisi vaativaa ja virhealtista.

Tämän vuoksi on yleistä ja suositeltavaa hyödyntää valmiita avoimen lähdekoodin soitinkirjastoja. Kirjastot, kuten dash.js ja Shaka Player, hoitavat kaiken monimutkaisen logiikan. Ne hyödyntävät MSE-rajapintaa ja tarjoavat selkeän rajapinnan videon hallintaan. Tämä nopeuttaa kehitysprosessia huomattavasti ja samalla parantaa myös lopputuloksen toimivuutta. Näitä kirjastoja ei ole selain- tai HbbTV-ympäristöissä automaattisesti, joten ne täytyy lisätä osaksi sovellusta tai ladata sovelluksen käynnistyksen yhteydessä. Tässä opinnäytetyössä käytetään dash.js-kirjastoa.

4.3 Käyttöliittymän rakentaminen

Tässä luvussa käsitellään HbbTV-sovelluksen käyttöliittymän rakentamista. Aiheeseen kuuluu sovelluksen HTML-rakenne ja ulkoasun määrittely CSS:llä.

Opinnäytetyössä esiteltävä soitin sisältää yksinkertaisen käyttöliittymän, joka näytetään videon päällä käyttäjän ohjatessa soitinta. Kuva 1 esittää tämän käyttöliittymän, kun se on näkyvissä. Käyttöliittymässä on aikajana, joka kuvaa toiston edistymistä ja kestoa sekä painikkeet ääniraidan ja tekstityksen kielen vaihtamiselle. Käyttöliittymä on suunniteltu olemaan yksinkertainen ja se tulee näkyviin vain tarvittaessa, jotta se ei häiritse katselukokemusta.



KUVA 1. Videosoittimen käyttöliittymä.

4.3.1 Sovelluksen HTML-rakenne

HbbTV-sovellukset perustuvat web-teknologioihin ja niiden HTML-rakenteen on noudatettava tiettyjä standardin vaatimuksia, jotta televisio tunnistaa sovelluksen oikein ja aktivoi tarvittavat rajapinnat. Dokumentin alkuun on asetettava `DOCTYPE`-määritys, joka viittaa vanhempaan HbbTV-standardiin. Tämän lisäksi `content-type`-metatieto täytyy määritellä yhteensopivuuden varmistamiseksi. Toinen pakollinen osa HTML-rakennetta on `application/oipfApplicationManager`-tyyppinen objekti, jonka avulla sovellus kommunikoi television kanssa. Tämän objektin kautta JavaScript-koodi saa pääsyn olioon, jota käytetään esimerkiksi sovelluksen elinkaaren hallintaan. Yksi tärkeimmistä metodeista on `show()`, jolla sovellus näytetään käynnistymisen jälkeen. HbbTV-sovellukset käynnistetään oletuksena piilotettuina ruudun välkkymisen estämiseksi, joten tämän metodin kutsuminen on usein yksi ensimmäisistä asioista, jonka sovellus tekee. (HbbTV Association n.d.f.)

Kuva 2 näyttää esimerkin HbbTV-sovelluksen rakenteesta, joka yhdistää nämä vaatimukset. Siinä on standardin vaatimat elementit sekä `app`-komponentti, johon lisätään videosoitin sekä käyttöliittymä myöhemmin JavaScriptin avulla.

```
<!DOCTYPE html PUBLIC "-//HbbTV//1.1.1//EN" "http://www.hbbtv.org/dtd/HbbTV-
1.1.1.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>HbbTV-videosoitin</title>
    <meta http-equiv="content-type" content="application/vnd.hbbtv.xhtml+xml;
utf-8" />
    <link rel="stylesheet" href="style.css" />
    <script type="text/javascript" src="main.js"></script>
  </head>
  <body>
    <div>
      <object
        type="application/oipfApplicationManager"
        id="applicationManager"
        width="0"
        height="0"
      ></object>
    </div>

    <div id="app"></div>
  </body>
</html>
```

KUVA 2. HbbTV-sovelluksen rakenne.

4.3.2 Ulkoasun määrittely

HbbTV-sovellusten ulkoasu määritellään verkkokehityksen tavoin CSS-teknologialla. Televisiot käyttävät usein vanhoja versioita selaimista, minkä vuoksi niiden CSS-tuki voi olla vuosia jäljessä nykyisistä standardeista. Tämän takia uusimpien ominaisuuksien toimintaan ei voi luottaa.

Sovelluksen rakentaminen vaatii usein elementtien tarkkaa asettelua ja vanhempien ominaisuuksien käyttöä, jotta käyttöliittymä näyttää samalta eri laitteilla. Toinen tärkeä asia on käyttää etuliitteitä, joiden avulla selainten valmistajat ovat ottaneet käyttöön kokeellisia ominaisuuksia (MDN Web Docs n.d.c). Koska selaimet HbbTV-ympäristössä perustuvat vanhoihin versioihin, on tärkeää käyttää `webkit`-etuliitettä monien CSS-arvojen, kuten animaatioiden, kanssa. Kaikkien tarvittavien etuliitteiden manuaalinen hallinta on hankalaa ja virhealtista. Tämän vuoksi on suositeltavaa hyödyntää automaattisia työkaluja, kuten Autoprefixeriä.

Se on työkalu, joka analysoi CSS-koodia ja lisää siihen automaattisesti tarvittavat etuliitteet, jotta koodi toimii määritellyissä selainten versioissa (PostCSS n.d.).

4.4 Sovelluksen toimintalogiikan toteuttaminen

Tässä luvussa keskitytään sovelluksen tekniseen toteutukseen. Luvussa käsitellään sovelluksen alustamista, käyttäjän komentoihin reagoimista sekä videosoittimen tärkeimpien toimintojen toteutusta eri soitinmoduuleilla.

4.4.1 Sovelluksen alustaminen ja ohjaaminen

Sovellus tulee alustaa oikein, jotta se voi reagoida käyttäjän komentoihin. Alustukseen sisältyy HbbTV-rajapintojen käyttöönotto, kaukosäätimen näppäinten kuuntelu sekä sovelluksen näyttäminen. Alustus tehdään tyypillisesti heti sovelluksen latauduttua. Ensimmäinen vaihe on hakea sovelluksen hallintaan tarkoitettu `Application`-olio, jonka avulla pääsee käsiksi HbbTV-alustan toimintoihin.

Seuraavaksi sovelluksen on ilmoitettava televisiolle, mitä näppäinkomentoja se tarvitsee. Tämä tehdään käyttämällä rajapintaa, joka hyödyntää numeroarvoa haluttujen näppäinryhmien määrittämiseen. Taulukko 1 esittää standardin määrittelemät arvot näppäinryhmille.

TAULUKKO 1. Näppäinryhmien arvot ja niiden käyttötarkoitus (Open IPTV Forum 2012, 60).

Vakio	Arvo	Käyttötarkoitus
RED	0x1	Punaisen painikkeen tunnistus
GREEN	0x2	Vihreän painikkeen tunnistus
YELLOW	0x4	Keltaisen painikkeen tunnistus
BLUE	0x8	Sinisen painikkeen tunnistus
NAVIGATION	0x10	Navigointi (nuolet, OK, takaisin)
VCR	0x20	Mediapainikkeet (play, pause, stop, kelaus)
SCROLL	0x40	Sivun vieritys ylös/alas
INFO	0x80	Infopainike
NUMERIC	0x100	Numeropainikkeet 0-9
ALPHA	0x200	Kirjainpainikkeet (a-z ja vastaavat)
OTHER	0x400	Muut painikkeet, joita ei luokitella yllä oleviin

Nämä arvot lasketaan yhteen ja yhteenlaskettu luku kertoo televisiolle, mitkä painallukset sen täytyy välittää sovellukselle. Kuva 3 havainnollistaa, miten sovellus määrittelee sen tarvitsemat näppäinryhmät.

```
// Haetaan sovellusobjekti
const appManager = document.getElementById('applicationManager');
const app = appManager.getOwnerApplication(document);

// Haetaan keyset-objekti ja asetetaan näppäinryhmät
const keyset = app.privateData.keyset;
keyset.setValue(0x01 + 0x02 + 0x04 + 0x08 + 0x10 + 0x20 + 0x100);

// Näytetään sovellus
app.show();
```

KUVA 3. Sovelluksen tarvitsemien näppäinryhmien määrittäminen.

Kaukosäätimen painikkeille on määritelty HbbTV-standardissa vakiot, kuten `VK_RIGHT` ja `VK_ENTER`, jotka ovat saatavilla `KeyEvent`-oliossa (ETSI TS 102 796 V1.2.1 2012, 54). Tämä olio on ainoastaan HbbTV-yhteensopivissa laitteissa eikä tietokoneen selaimissa, joten sovelluksen testaamisen helpottamiseksi on järkevää määrittää vakiot niin, että ne toimivat molemmissa ympäristöissä. Kuva

4 esittää tavan, jossa ensin yritetään lukea arvo `KeyEvent`-oliosta, mutta jos sitä ei ole saatavilla, muuttujalle annetaan oletusarvo, joka vastaa tietokoneen näppäimistön arvoa.

```
function getKeyCode(keyName, defaultValue) {
  if (typeof KeyEvent !== 'undefined' && typeof KeyEvent[keyName] !== 'unde-
fined') {
    return KeyEvent[keyName];
  }
  return defaultValue;
}

// Esimerkkejä vakioiden määrittelystä
const VK_RIGHT = getKeyCode('VK_RIGHT', 39);
const VK_ENTER = getKeyCode('VK_ENTER', 13);
```

KUVA 4. Esimerkki näppäinkoodien määrittelystä.

Reagointi kaukosäätimen näppäinten painalluksiin tapahtuu erillisen tapahtuman kuuntelijan avulla. Kuten kuvasta 5 nähdään, sovellus vertaa vastaanotettua näppäinkoodia aiemmin määriteltyihin vakioihin ja kutsuu painiketta vastaavaa toimintoa, kuten videon tauottamista tai kelaamista. On myös tärkeää, että sovellus estää painikkeen oletustoiminnon niille painikkeille, jotka se käsittelee itse.

```
// Lisätään tapahtuman kuuntelija
document.addEventListener('keydown', handleKeyDown);

function handleKeyDown(event) {
  event.preventDefault();

  // Kutsutaan oikeaa toimintoa painetun näppäimen perusteella
  switch (event.keyCode) {
    case VK_PLAY:
      player.play();
      break;
    case VK_PAUSE:
      player.pause();
      break;
    case VK_RIGHT:
      player.seek(30);
      break;
  }
}
```

KUVA 5. Näppäintapahtumien kuunteleminen ja käsitteleminen.

4.4.2 Soitinelementin luominen

Sovelluksen alustamisen jälkeen luodaan varsinainen soitinelementti. Kuten luvussa 4.2.2 kuvattiin, sovellus on ensin päätellyt laitteen version perusteella oikean soitinmoduulin. Tämän jälkeen luodaan kyseinen moduuli ja kutsutaan metodia, joka vastaa soitinelementin luomisesta.

OIPF-toteutuksessa luodaan uusi `<object>`-elementti JavaScriptin avulla. Tälle elementille asetetaan tarvittavat attribuutit ja se lisätään osaksi sivun rakennetta. HTML5-toteutus on hyvin samanlainen, mutta `<object>`-elementin sijaan luodaan `<video>`-elementti. MSE-toteutus puolestaan ei luo omaa elementtiään, vaan se hyödyntää HTML5-moduulia ja liittää dash.js-kirjaston siihen. Kuva 6 havainnollistaa soitinelementin luomista eri moduuleissa.

```
class VideoPlayerOIPF extends VideoPlayerBase {
  createPlayer() {
    // Luodaan uusi object-elementti
    const videoObject = document.createElement('object');
    videoObject.setAttribute('id', 'video');
    videoObject.setAttribute('type', 'video/mp4');

    // Lisätään elementti sivulle ja tallennetaan viittaus siihen
    document.getElementById('videoContainer').appendChild(videoObject);
    this.player = videoObject;
  }
}

class VideoPlayerHTML5 extends VideoPlayerBase {
  createPlayer() {
    // Luodaan uusi video-elementti
    const videoElement = document.createElement('video');
    videoElement.setAttribute('id', 'video');

    // Lisätään elementti sivulle ja tallennetaan viittaus siihen
    document.getElementById('videoContainer').appendChild(videoElement);
    this.player = videoElement;
  }
}

class VideoPlayerMSE extends VideoPlayerHTML5 {
  createPlayer() {
    // Kutsutaan ensin perityn luokan metodia, joka luo video-elementin
    super.createPlayer();

    // Alustetaan dash.js ja liitetään se video-elementtiin
    const dashjsPlayer = window.dashjs.MediaPlayer().create();
```

```
    dashjsPlayer.initialize();
    dashjsPlayer.setAutoPlay(true);
    dashjsPlayer.attachView(this.player);

    // Tallennetaan viittaus dash.js-olioon
    this.dashjsPlayer = dashjsPlayer;
  }
}
```

KUVA 6. Soitinelementin luominen eri moduuleissa.

4.4.3 Videon lataus ja tilanhallinta

Kun soitinelementti on luotu, seuraava vaihe on ladata siihen video. Sovellus kutsuu aina samannimistä metodia, mutta sen tekninen toteutus on eri soitinmoduuleissa erilainen. Kuva 7 havainnollistaa, miten video ladataan eri moduuleissa.

```
class VideoPlayerOIPF extends VideoPlayerBase {
  loadVideo(url) {
    this.source = url;
    this.player.onPlayStateChange = () => this.handlePlayStateChange();
    const mimeType = this.getMimeType(url);
    this.player.setAttribute('type', mimeType);
    this.player.data = url;
    this.play();
  }
}

class VideoPlayerHTML5 extends VideoPlayerBase {
  loadVideo(url) {
    this.source = url;
    const mimeType = this.getMimeType(url);
    this.player.setAttribute('type', mimeType);
    this.player.src = url;
    this.play();
  }
}

class VideoPlayerMSE extends VideoPlayerHTML5 {
  loadVideo(url) {
    this.source = url;
    const mimeType = this.getMimeType(url);
    if (mimeType === 'application/dash+xml') {
      this.player.pause();
      this.player.removeAttribute('src');
      this.player.removeAttribute('type');
      this.player.load();
      this.dashjsPlayer.attachSource(url);
```

```
    } else {
      this.dashjsPlayer.attachSource(null);
      super.loadVideo(url);
    }
  }
}
```

KUVA 7. Videon lataaminen eri moduuleissa.

Kuten kuvasta 7 nähdään, OIPF- ja HTML5-toteutuksissa videon lataaminen on yksinkertaista. Molemmissa malleissa on tärkeää asettaa oikea mediatyyppi, jotta laite osaa toistaa videota. MSE-pohjainen toteutus puolestaan valitsee mediatyypin perusteella, käytetäänkö dash.js-kirjastoa vai ei. Jos video on DASH-muodossa, se käyttää dash.js-kirjastoa. Jos video taas on tavallinen MP4-tiedosto, se käyttää HTML5-moduulin logiikkaa. Kaikissa malleissa myös tallennetaan videon osoite muistiin myöhempää käsittelyä varten.

Videon lataamisen jälkeen soittimen tilaa täytyy hallita tapahtumien perusteella. Eri moduulien tehtävänä on käsitellä niiden rajapintojen lähettämät tapahtumat. OIPF-malli perustuu yhteen tapahtuman kuuntelijaan, joka ilmoittaa soittimen tilan numerona. Taulukkoon 2 on listattu eri tilat ja niiden arvot. HTML5-mallissa jokaista tapahtumaa varten täytyy olla oma kuuntelija. MSE-versio ei tarvitse omaa toteutustaan, sillä se hyödyntää HTML5-luokan kuuntelijoita.

TAULUKKO 2. OIPF-soitinelementin tilojen arvot (Open IPTV Forum 2012, 184–185).

Tila	Arvo	Kuvaus
stopped	0	Toisto on pysähtynyt
playing	1	Sisältöä toistetaan
paused	2	Toisto on tauotettu
connecting	3	Muodostetaan yhteyttä
buffering	4	Sisältöä puskuroidaan
finished	5	Toisto on päättynyt
error	6	Toistossa on tapahtunut virhe

Arkkitehtuurin tarkoituksena on varmistaa, että kaikki toiminnot tekevät saman asian. Kuten kuvasta 8 nähdään, luokat eivät sisällä käyttöliittymän päivityksen logiikkaa, vaan ne kutsuvat `VideoPlayerBase`-luokassa määriteltyjä metodeja. Tämän avulla vähennetään koodin toistoa ja varmistetaan yhdenmukainen toiminnallisuus moduulien välillä.

```
class VideoPlayerBase {
  onStatePlaying() {
    this.isLoading(false);
    this.updateProgressBar();
    this.startProgressInterval();
  }

  onStatePaused() {
    this.isLoading(false);
    this.updateProgressBar();
    this.clearProgressInterval();
  }

  // muut tilat...
}

class VideoPlayerOIPF extends VideoPlayerBase {
  // Yksi kuuntelija, joka tarkistaa tilan
  handlePlayStateChange() {
    switch (this.player.playState) {
      case 1:
        this.onStatePlaying();
        break;
      case 2:
        this.onStatePaused();
        break;
      // muut tilat...
    }
  }
}

class VideoPlayerHTML5 extends VideoPlayerBase {
  createPlayer() {
    // elementin luonti kuten aiemmin...

    // Lisätään kuuntelijat tilan muutoksille
    this.player.addEventListener('playing', () => this.onStatePlaying());
    this.player.addEventListener('pause', () => this.onStatePaused());

    // muut tilat...
  }
}
```

KUVA 8. Tilanhallinta eri moduuleissa.

4.4.4 Toiston ohjaus

OIPF- ja HTML5-mallien välillä on pieniä eroja videon käynnistämisessä ja tauottamisessa. MSE-moduuli ei tarvitse omaa toteutustaan, vaan se hyödyntää jälleen HTML5-luokan toimintaa.

OIPF-rajapinta ei sisällä erillisiä metodeja toistolle ja tauotukselle, vaan molemmat toiminnot tehdään yhdellä `play()`-metodilla, jolle annetaan parametrina nopeus. Videon toisto käynnistetään asettamalla nopeudeksi 1 ja se tauotetaan asettamalla nopeudeksi 0. HTML5-mallissa sen sijaan käytetään kahta eri metodia, `play()` käynnistää toiston ja `pause()` tauottaa sen. Kuva 9 havainnollistaa, miten nämä toteutetaan ja miten ne hyödyntävät yhteistä `showUI()`-metodia.

```
class VideoPlayerBase {
  showUI(seconds) {
    // Näytetään käyttöliittymä ja poistetaan vanha ajastin
    this.elements.playerUI.classList.add('show');
    this.elements.playerUI.classList.remove('hide');
    clearTimeout(this.hideUITimer);

    // Asetetaan ajastin piilottamaan käyttöliittymä, jos aika on määritelty
    if (seconds) {
      this.hideUITimer = setTimeout(() => {
        this.hideUI();
      }, seconds * 1000);
    }
  }

  hideUI() {
    this.elements.playerUI.classList.add('hide');
    this.elements.playerUI.classList.remove('show');
  }
}

class VideoPlayerOIPF extends VideoPlayerBase {
  play() {
    this.player.play(1);
    this.showUI(5);
  }

  pause() {
    this.player.play(0);
    this.showUI();
  }
}
```

```
class VideoPlayerHTML5 extends VideoPlayerBase {
  play() {
    this.player.play();
    this.showUI(5);
  }

  pause() {
    this.player.pause();
    this.showUI();
  }
}
```

KUVA 9. Toiston käynnistäminen ja pysäyttäminen.

Kuten kuvasta 9 nähdään, molemmat toteutukset kutsuvat samaa `VideoPlayerBase`-luokassa määriteltyä `showUI()`-metodia. Tämän metodin avulla näytetään käyttöliittymä käyttäjälle ja piilotetaan se automaattisesti ajastimen avulla. On yleistä ja suositeltavaa näyttää käyttöliittymä hetkellisesti videon käynnistyksen yhteydessä, piilottaa se toiston ajaksi ja jättää se näkyviin tauotettaessa.

4.4.5 Videon kelaus

Videon käynnistämisen ja tauottamisen lisäksi videon kelaaminen on yksi videosoittimen oleellisimmista toiminnoista. Videon kelaaminen vaatii tiedon nykyisestä sijainnista sekä videon kestosta. OIPF- ja HTML5-rajapinnat käyttävät näitä tietoja eri muodoissa, joten ne täytyy normalisoida. Tämä tehdään `VideoPlayerBase`-luokan `time()`-metodissa, joka on esitetty kuvassa 10. Metodi tarkistaa, mitkä arvot soitinelementissä on saatavilla ja palauttaa niiden perusteella videon keston ja nykyinen sijainnin sekunteina.

```
class VideoPlayerBase {
  time() {
    let duration = 0;
    let position = 0;
    let progress = 0;

    if (this.player.playTime) {
      // OIPF (millisekunteina)
      duration = this.player.playTime / 1000;
      position = this.player.playPosition / 1000;
```

```
    } else if (this.player.duration) {
      // HTML5 (sekunteina)
      duration = this.player.duration;
      position = this.player.currentTime;
    }

    // Varmistetaan, että ei palauteta NaN-arvoja
    if (isNaN(duration)) duration = 0;
    if (isNaN(position)) position = 0;

    if (position > 0 && duration > 0) {
      progress = position / duration;
    }

    return { duration, position, progress };
  }
}
```

KUVA 10. Metodi, joka normalisoi videon edistymisen tiedot.

Kelaukseen liittyvä yhteinen logiikka, kuten uuden sijainnin laskeminen, on lisätty `VideoPlayerBase`-luokan `seek()`-metodiin. Videon lopullinen kelaaminen toteutetaan kuitenkin teknologiakohtaisesti, sillä niiden toteutukset eroavat jälleen toisistaan. Kuva 11 esittää, miten yhteistä `seek()`-metodia ja teknologiakohtaisia toteutuksia käytetään yhdessä.

```
class VideoPlayerBase {
  seek(seconds) {
    const { position, duration } = this.time();
    const newPosition = position + seconds;

    // Estetään kelaus videon rajojen yli
    if (newPosition < 0) {
      this.performSeek(0);
    } else if (newPosition > duration - 5) {
      this.performSeek(duration - 5);
    } else {
      this.performSeek(newPosition);
    }

    this.showUI(5);
  }
}

class VideoPlayerOIPF extends VideoPlayerBase {
  performSeek(newPosition) {
    // OIPF käyttää millisekunteja
    this.player.seek(newPosition * 1000);
  }
}
```

```
class VideoPlayerHTML5 extends VideoPlayerBase {
  performSeek(newPosition) {
    // HTML5 käyttää sekunteja
    this.player.currentTime = newPosition;
  }
}
```

KUVA 11. Videon kelaaminen eri moduuleissa.

Tässä toteutuksessa monimutkainen ja yhteinen logiikka erotetaan moduulien rajapintojen toteutuksista. Toteutuksessa hyödynnetään aiemmin luotua `time()`-metodia, jonka avulla videon tiedot ovat aina samassa muodossa laskutoimituksia varten. Kelauksessa on huomioitava, että toiston uusi sijainti ei mene alle nollan sekunnin. Vastaavasti ongelmien välttämiseksi on tärkeää estää videon kelaaminen ihan sen loppuun asti.

4.4.6 Toiston edistymisen seuranta

Soittimen tietoja on päivitettävä säännöllisesti toiston aikana. Tämä toteutetaan ajastimella, joka päivittää tietoja sekunnin välein. Tämä logiikka on jaettu useaan eri metodiin, jotka vastaavat ajastuksen hallinnasta ja käyttöliittymän päivittämisestä.

Kun videon toisto alkaa, soittimen täytyy käynnistää ajastin, joka kutsuu käyttöliittymän päivityksestä vastaavaa metodia. Ajastin kannattaa myös pysäyttää aina, kun video tauotetaan tai lopetetaan. Tämä estää sovellusta suorittamasta turhia operaatioita, kun videon sijainti ei muutu. Ajastimen pysäyttäminen on tärkeää varsinkin sellaisissa tilanteissa, joissa sen sisällä suoritetaan muuta raskaampaa logiikkaa.

Käyttöliittymän päivitykseen tarkoitettu metodi kutsuu ensin luvussa 4.4.5 esiteltyä `time()`-metodia saadakseen tiedon videon nykyisestä sijainnista ja kestosta. Näiden tietojen perusteella sovellus tekee käyttöliittymään tarvittavat päivitykset, kuten näyttää toiston nykyisen sijainnin. Kuva 12 havainnollistaa tätä päivityslogiikkaa.

```
class VideoPlayerBase {
  startProgressInterval() {
    // Poistetaan vanha ajastin ja käynnistetään uusi
    this.clearProgressInterval();
    this.progressUpdateInterval = setInterval(() => {
      this.updateProgressBar();
      // Tekstitysten ajastusta tarkistetaan myös säännöllisesti
      this.subtitles.update();
    }, 1000);
  }

  clearProgressInterval() {
    clearInterval(this.progressUpdateInterval);
  }

  updateProgressBar() {
    // Haetaan nykyinen sijainti ja kesto
    const { position, duration, progress } = this.time();

    // Päivitetään toiston tilaa kuvaavat elementit
    const percent = progress * 100;
    this.elements.progressBar.style.width = `${percent}%`;
    this.elements.currentTimeDisplay.textContent = this.formatTime(position);
    this.elements.durationDisplay.textContent = this.formatTime(duration);
  }
}
```

KUVA 12. Käyttöliittymän päivittäminen ajastimen avulla.

4.4.7 Tekstitysten toteutus

Tekstitysten lataamiselle ja näyttämiselle on monia erilaisia toteutustapoja. DASH-manifesti voi esimerkiksi sisältää tiedon tekstitystiedostoista ja HTML5:n `<track>`-elementti on suunniteltu näyttämään niitä videoiden aikana. Näissä toteutustavoissa voi olla laitekohtaisia eroja ja ne saattavat toimia virheellisesti laitteen toteutuksesta riippuen. Yhdenmukaisen toiminnallisuuden saavuttamiseksi kaikilla laitteilla luotettavin ratkaisu on toteuttaa tekstitysten lataaminen ja näyttäminen JavaScriptillä. Tämä tarkoittaa sitä, että sovellus itse lataa tekstitystiedoston, käsittelee sen ja näyttää tekstitykset oikeaan aikaan.

JavaScript-toteutuksessa erillinen `Subtitles`-luokka lataa ensin tekstitystiedoston internetistä ja käsittelee sen. Tiedoston käsittelyyn sisältyy ajastusten ja

tekstirivien muuntaminen sellaiseen muotoon, jota sovellus pystyy helposti hyödyntämään. Tekstitystiedostojen käsittely on tässä esimerkissä jätetty pois ja toiminnallisuutta simuloidaan käyttämällä sovellukseen valmiiksi määriteltyä esimerkkidataa. `VideoPlayerBase`-luokasta kutsutaan sekunnin välein `update()`-metodia ja kuten kuvasta 13 nähdään, se tarkistaa videon nykyisen sijainnin ja etsii sitä vastaavan tekstitysrivin. Jos tekstitystiedostosta löytyy tekstityksiä kyseiselle ajanhetkelle, se asettaa ne erilliseen `div`-elementtiin, joka näytetään videon päällä. Tämä toteutustapa on teknisesti hieman vaativampi toteuttaa, mutta sen avulla tekstitykset toimivat ja näyttävät samalta kaikilla laitteilla. Tämän mallin avulla tekstitysten ulkoasua on myös helppo muuttaa. Koska tekstitykset ovat omassa `div`-elementissään, niiden ulkoasu määritellään CSS:n avulla. Tämä mahdollistaa sen, että sovellukseen tai soittimeen voidaan lisätä asetuksia esimerkiksi tekstitysten koon ja värin vaihtamiseen.

```
class Subtitles {
  constructor(player) {
    this.player = player;
    this.element = document.getElementById('subtitles');
    this.currentLanguageIndex = 0;
    this.isVisible = true;

    // Käytetään esimerkkidataa
    // Oikeassa sovelluksessa data ladattaisiin ja käsiteltäisiin tässä
    this.data = {
      en: [{ start: 0, end: 3, text: '0s-3s (EN)' } /* ... */],
      fi: [{ start: 0, end: 3, text: '0s-3s (FI)' } /* ... */],
    };
    this.languages = Object.keys(this.data);
  }

  // Päivitetään näytettävä tekstitysrivi
  // Tätä metodia kutsutaan sekunnin välein ajastimesta
  update() {
    if (!this.isVisible) {
      this.element.textContent = '';
      return;
    }
    const { position } = this.player.time();
    const currentData = this.data[this.getCurrentLanguage()];

    // Etsitään oikea tekstitysrivi nykyisen ajan perusteella
    const sub = currentData.find((sub) => sub.start <= position && sub.end >=
position);
    this.element.textContent = sub?.text ?? '';
  }
```

```
  // Metodi tekstityksen kielen vaihtamiseen
  // Tätä kutsutaan kaukosäätimen painalluksesta
  switchLanguage() {
    this.currentLanguageIndex = (this.currentLanguageIndex + 1) % this.lan-
guages.length;
    this.update();

    // käyttöliittymän päivitys...
    this.player.showUI(5);
  }

  // Metodi tekstitysten näkyvyyden vaihtamiseen
  toggle() {
    this.isVisible = !this.isVisible;
    this.update();
  }

  // Palauttaa nykyisen tekstityksen kielen
  getCurrentLanguage() {
    return this.languages[this.currentLanguageIndex];
  }
}
```

KUVA 13. JavaScript-pohjainen tekstitysjärjestelmä.

4.4.8 Ääniraidan vaihtaminen

Ääniraidan vaihtaminen on toteutettava jokaiselle moduulille erikseen. Tätä varten jokaiseen moduuliin tehdään `switchAudio`-niminen metodi, jota sovellus kutsuu. OIPF-malli käsittelee videon eri komponentteja numeroina, jotka on esitetty taulukossa 3.

TAULUKKO 3. OIPF-komponenttien numerot (Open IPTV Forum 2012, 207).

Arvo	Kuvaus
0	Videokomponentti
1	Audiokomponentti
2	Tekstityskomponentti

Tieto ääniraidoista OIPF-mallissa on luotettavasti saatavilla ainoastaan videon ollessa käynnissä. Muissa tiloissa tieto voi olla saatavilla, mutta se ei ole taattua. Tämä tarkoittaa sitä, että ääniraidan voi vaihtaa vain, kun video on käynnissä.

(Open IPTV Forum 2012, 207–209.) Ääniraidan vaihtaminen sisältää monta vaihetta, jossa ensin haetaan saatavilla olevat raidat, asetetaan ne pois päältä ja vasta sitten aktivoidaan uusi raita. Kuva 14 kuvaa ääniraidan vaihtamista OIPF-moduulissa.

```
class VideoPlayerOIPF extends VideoPlayerBase {
  switchAudio() {
    if (this.player.playState !== 1) return;
    // Haetaan audiokomponentit käyttämällä arvoa 1
    const audioComponents = this.player.getComponents(1);
    if (!audioComponents || audioComponents.length <= 1) return;

    const currentTrackIndex = this.currentAudioTrackIndex ?? 0;
    const nextTrackIndex = (currentTrackIndex + 1) % audioComponents.length;
    for (let i = 0; i < audioComponents.length; i++) {
      this.player.unselectComponent(audioComponents[i]);
    }
    this.player.selectComponent(audioComponents[nextTrackIndex]);

    // Tallennetaan uuden raidan numero muistiin
    this.currentAudioTrackIndex = nextTrackIndex;

    // käyttöliittymän päivitys...
    this.showUI(5);
  }
}
```

KUVA 14. Ääniraidan vaihtaminen OIPF-moduulissa.

HTML5-toteutus on hyvin samanlainen. Myös tässä käydään läpi kaikki raidat ja asetetaan ne pois päältä ennen kuin uusi raita aktivoidaan. Kuva 15 esittää tämän toiminnallisuuden.

```
class VideoPlayerHTML5 extends VideoPlayerBase {
  switchAudio() {
    const tracks = this.player.audioTracks;
    if (!tracks || tracks.length <= 1) return;

    let currentTrackIndex = -1;
    for (let i = 0; i < tracks.length; i++) {
      if (tracks[i].enabled) {
        currentTrackIndex = i;
        tracks[i].enabled = false;
      }
    }

    const nextTrackIndex = (currentTrackIndex + 1) % tracks.length;
    tracks[nextTrackIndex].enabled = true;
```

```
    // käyttöliittymän päivitys...
    this.showUI(5);
  }
}
```

KUVA 15. Ääniraidan vaihtaminen HTML5-moduulissa.

MSE-toteutuksessa käytetään dash.js-kirjaston omia metodeja. Kuten kuva 16 esittää, toteutus käyttää dash.js-kirjastoa ainoastaan, jos kyseessä on DASH-video. Jos kyseessä on MP4-video, toteutus hyödyntää HTML5-moduulia, joka tukee MP4-videoiden ääniraitojen vaihtamista. dash.js-kirjastoa käytettäessä ei tarvitse asettaa kaikkia ääniraitoja ensin pois päältä, vaan haluttu ääniraita voidaan aktivoida suoraan.

```
class VideoPlayerMSE extends VideoPlayerHTML5 {
  switchAudio() {
    if (this.getMimeType(this.source) !== 'application/dash+xml') {
      super.switchAudio();
      return;
    }

    const availableTracks = this.dashjsPlayer.getTracksFor('audio');
    if (!availableTracks || availableTracks.length <= 1) return;

    const currentTrack = this.dashjsPlayer.getCurrentTrackFor('audio');
    const currentTrackIndex = currentTrack ? availableTracks.indexOf(cur-
rentTrack) : -1;
    const nextTrackIndex = (currentTrackIndex + 1) % availableTracks.length;
    const nextTrack = availableTracks[nextTrackIndex];

    this.dashjsPlayer.setCurrentTrack(nextTrack);

    // käyttöliittymän päivitys...
    this.showUI(5);
  }
}
```

KUVA 16. Ääniraidan vaihtaminen MSE-moduulissa.

4.5 Havaintoja testauksesta

HbbTV-kehitykseen liittyy usein odottamattomia haasteita, jotka johtuvat laitteiden välisistä eroista. Standardin tarkka noudattaminen ei takaa toimivaa sovellusta. Tässä luvussa käsitellään testausvaiheessa tehtyjä havaintoja ja esitellään menetelmiä, joilla niitä voidaan hallita.

4.5.1 Yhteensopivuusongelmat

Testauksessa tuli esiin useita tilanteita, joissa laitteiden toiminnallisuus ei vastannut niiden ilmoittamaa HbbTV-versiota. Esimerkiksi MSE- ja HTML5-moduulit toimivat joillakin HbbTV 1.5 -laitteilla, vaikka version perusteella niiden ei olisi pitänyt toimia. Vastaavasti virallisesti tuettu ominaisuus saattoi toimia virheellisesti.

Ääniraidan vaihtamisen toiminta vaihteli paljon eri laitteiden välillä. Testauksessa varsinkin HbbTV 1.5 -laitteilla oli useita ongelmia tämän kanssa. Joillain HbbTV 1.5 -laitteilla OIPF-version ääniraidan vaihto toimi ongelmallisesti, kun taas MSE-toteutus toimi ongelmitta. Joillakin laitteilla vaihto tapahtui välittömästi ja toisilla se aiheutti videon lyhyen pysähtymisen. Joskus vaihto kesti jopa useita sekunteja, jolloin video jatkoi toistoa hetken vanhalla ääniraidalla.

Videoiden koodekeilla oli myös vaikutusta toimintaan ja väärän koodekin käyttö saattoi kokonaan estää videon käynnistymisen. Lisäksi testauksessa huomattiin, että HLS-suoratoisto toimi yllättävän monella laitteella, vaikka HbbTV-ympäristö ei tue sitä virallisesti.

4.5.2 Laitekohtaiset säännöt

Laitteiden välisiä yhteensopivuusongelmia voidaan hallita hyödyntämällä selainympäristöissä olevaa `navigator.userAgent`-arvoa. Myös HbbTV-standardi määrittelee tuen `navigator`-oliolle. HbbTV-standardin mukaan tämän arvon pitäisi aina sisältää HbbTV-versionumero. Standardi myös suosittelee edellä

mainitun arvon sisältävän muita tietoja, kuten valmistajan ja mallin nimen, ohjelmistoversion sekä selaimen version. Kaikkia tietoja ei välttämättä ole mukana arvossa, joten sen sisältö vaihtelee eri laitteiden välillä paljon. (ETSI TS 102 796 V1.2.1 2012, 45, 75.)

Tätä tietoa voidaan hyödyntää laitekohtaisten sääntöjen määrittämiseen. Säännöt määritellään erillisessä tiedostossa, joka lisätään sovellukseen tai ladataan sen käynnistyessä. Tiedosto voi sisältää sääntöjä esimerkiksi tietyille malleille tai valmistajille. Sovellus tarkistaa käynnistyksen yhteydessä, onko sille määritetty sääntöjä.

Jos laitteelle on määritetty sääntöjä, sovelluksen toimintaa voidaan muuttaa. Yksi käyttötarkoitus kyseisille säännöille on laittaa pois päältä ominaisuuksia, joiden tiedetään aiheuttavan ongelmia tietyillä laitteilla. Esimerkiksi kaikilta tietyn valmistajan vuosimallin laitteilta voidaan estää ääniraidan vaihtaminen kokonaan. Toinen käyttötarkoitus on pakottaa jokin laite käyttämään tiettyä versiota soittimesta. Jos esimerkiksi HbbTV 2.0 -laitteella on testauksen aikana havaittu ongelmia HTML5-moduulin kanssa, se voidaan pakottaa käyttämään joko OIPF- tai MSE-moduulia.

4.5.3 Kehitys- ja testausmenetelmät

Videosoittimen kehitystä hidastaa huomattavasti toiminnallisuuden jatkuva testaaminen oikeilla televisioilla. Kehitystä voidaan kuitenkin nopeuttaa hyödyntämällä MSE-moduulia. Sen lisääminen sovellukseen on melko yksinkertaista, sillä se hyödyntää lähes täysin jo olemassa olevan HTML5-moduulin logiikkaa.

MSE-moduulin yksi suurimmista eduista on sen kyky toimia tietokoneen selaimessa. Sen avulla käyttöliittymän päivitystä, tapahtuman käsittelyä ja tekstityksiä pystyy testaamaan selaimen kehitystyökaluilla. Tämä nopeuttaa kehitystä merkittävästi, sillä jokaisen pienen muutoksen jälkeen sovellusta ei tarvitse testata oikealla televisiolla. MSE-moduuli on myös hyödyllinen monimutkaisempien

ominaisuuksien lisäämisessä ja testaamisessa. Esimerkiksi mainosten lisääminen soittimeen ei ole laitekohtainen toiminnallisuus, mutta se vaatii kattavaa testausta, mikä olisi erittäin hidasta ja vaikeaa ilman selaimen työkaluja.

Televisiolla testattaessa on vain harvoin pääsy selaimen kehitystyökaluihin. Yksinkertainen ratkaisu on näyttää tietoja ruudulla `div`-elementissä kaikkien muiden elementtien päällä. Tämän avulla saadaan tietoa sovelluksen toiminnasta. Siinä voidaan näyttää esimerkiksi muuttujien arvoja, virheviestejä tai muita tärkeitä tietoja.

Edellä mainitut tavat nopeuttavat kehitystä, mutta sovellusta täytyy silti testata kattavasti eri laitteilla. Kuten on todettu aiemmin, yksi kehityksen suurimmista haasteista on laitehajanaisuus. Laadun varmistamiseksi sovellusta on aina testattava useilla eri valmistajien ja vuosimallien laitteilla. Kuvassa 17 näkyy videosoittimen testaukseen käytettyjä televisioita.

KUVA 17. Videosoittimen testaukseen käytettyjä televisioita (Kuva: Miikka Ylisiurunen).

5 JOHTOPÄÄTÖKSET JA POHDINTA

Opinnäytetyön tarkoituksena oli analysoida ja esitellä HbbTV-yhteensopivan videosoittimen toteutusta ja siihen liittyviä erityispiirteitä. Yksi työn tärkeimpiä johtopäätöksiä on, että HbbTV-ympäristön laitehajanaisuuden hallintaan tarvitaan modulaarinen ohjelmistoarkkitehtuuri. Työssä esitelty arkkitehtuuri, jossa soittimen yhteinen logiikka erotetaan erillisistä teknologiakohtaisista moduuleista, osoittautui erinomaiseksi tavaksi käsitellä standardien vaatimia erilaisia tekniikoita.

Toinen merkittävä havainto oli, että HbbTV-laitteen versionumero ei takaa kaikkien ominaisuuksien täydellistä toimintaa, joten laitekohtaiset erot ja puutteet ovat yleisiä. Toimivan sovelluksen kehittäminen vaatii testausta useilla eri valmistajien ja vuosimallien laitteilla. Lisäksi työ osoitti, että kehitystyökalut ja -prosessit, kuten polyfillit ja transpilointi, ovat erittäin hyödyllisiä ylläpidettävän ja toimivan sovelluksen kehityksessä. Työssä esiteltiin myös tapoja testauksen ja yhteensopivuusongelmien helpottamiseen.

Työ onnistui sille asetetussa tavoitteessa hyvin. Työn tulokset toimivat ohjeena ja johdatuksena HbbTV-ympäristöön. Lisäksi kokeneemmatkin kehittäjät voivat käyttää saatuja tuloksia oppaana sovellusten kehittämisessä. Työssä esitetyt toteutukset perustuvat virallisiin standardeihin ja ne on todennettu toimiviksi useilla eri HbbTV-laitteilla. Tulosten avulla voidaan kehittää uudelleenkäytettävä soitinkomponentti, joka nopeuttaisi tulevien projektien kehitystä.

Jatkokehityksenä videosoittimeen voisi lisätä useita uusia ominaisuuksia ja sen toiminnallisuutta voisi laajentaa. Erinomaisia jatkokehitysaiheita ovat esimerkiksi DRM-suojattujen videoiden ja live-lähetysten toistaminen sekä mainoskatkojen lisääminen soittimeen.

LÄHTEET

Babel. n.d. What is Babel?. Verkkosivu. Viitattu 18.8.2025. https://babeljs.io/docs/

Dash Industry Forum. n.d. dash.js. Verkkosivu. Viitattu 8.9.2025. https://github.com/Dash-Industry-Forum/dash.js

ETSI. 2015. ETSI releases new specification for HbbTV 2.0. ETSI Newsroom 16.11.2015. Viitattu 13.8.2025. https://www.etsi.org/newsroom/news/1026-2015-11-etsi-releases-new-specification-for-hbbtv-2-0

ETSI TS 102 796 V1.1.1. 2010. Hybrid Broadcast Broadband TV. European Telecommunications Standards Institute. Viitattu 9.8.2025. https://www.etsi.org/deliver/etsi_ts/102700_102799/102796/01.01.01_60/ts_102796v010101p.pdf

ETSI TS 102 796 V1.2.1. 2012. Hybrid Broadcast Broadband TV. European Telecommunications Standards Institute. Viitattu 9.8.2025. https://www.etsi.org/deliver/etsi_ts/102700_102799/102796/01.02.01_60/ts_102796v010201p.pdf

ETSI TS 102 796 V1.3.1. 2015. Hybrid Broadcast Broadband TV. European Telecommunications Standards Institute. Viitattu 10.8.2025. https://www.etsi.org/deliver/etsi_ts/102700_102799/102796/01.03.01_60/ts_102796v010301p.pdf

ETSI TS 102 796 V1.4.1. 2016. Hybrid Broadcast Broadband TV. European Telecommunications Standards Institute. Viitattu 11.8.2025. https://www.etsi.org/deliver/etsi_ts/102700_102799/102796/01.04.01_60/ts_102796v010401p.pdf

ETSI TS 102 796 V1.5.1. 2018. Hybrid Broadcast Broadband TV. European Telecommunications Standards Institute. Viitattu 11.8.2025. https://www.etsi.org/deliver/etsi_ts/102700_102799/102796/01.05.01_60/ts_102796v010501p.pdf

ETSI TS 102 796 V1.6.1. 2021. Hybrid Broadcast Broadband TV. European Telecommunications Standards Institute. Viitattu 30.8.2025. https://www.etsi.org/deliver/etsi_ts/102700_102799/102796/01.06.01_60/ts_102796v010601p.pdf

Francis, A. 2023. Adaptive Bitrate Streaming (ABR): What is it & How Does it Work? [2023 Update]. Bitmovin 22.9.2023. Viitattu 20.8.2025. https://bitmovin.com/blog/adaptive-streaming

HbbTV Association. 2025. Major milestone: HbbTV reaches 100 million homes in Europe. HbbTV Association News 27.3.2025. Viitattu 11.9.2025. https://www.hbbtv.org/news-events/major-milestone-hbbtv-reaches-100-million-homes-in-europe/

HbbTV Association. n.d.a. Common mistakes to avoid. Verkkosivu. Viitattu 10.8.2025. https://developer.hbbtv.org/guide/hints-tips/common-mistakes-to-avoid/

HbbTV Association. n.d.b. HbbTV 2.0.1 explained. Verkkosivu. Viitattu 9.8.2025. https://developer.hbbtv.org/guide/introduction/hbbtv-2-0-1-explained/

HbbTV Association. n.d.c. HbbTV 2.0.2 explained. Verkkosivu. Viitattu 9.8.2025. https://developer.hbbtv.org/guide/introduction/hbbtv-2-0-2-explained/

HbbTV Association. n.d.d. HbbTV 2.0.3 explained. Verkkosivu. Viitattu 30.8.2025. https://developer.hbbtv.org/guide/introduction/hbbtv-2-0-3-explained/

HbbTV Association. n.d.e. HbbTV application testing and verification. Verkkosivu. Viitattu 10.8.2025. https://developer.hbbtv.org/guide/testing-and-certification/hbbtv-application-testing-and-verification/

HbbTV Association. n.d.f. Hello World application. Verkkosivu. Viitattu 17.9.2025. https://developer.hbbtv.org/guide/getting-started/hello-world-application/

HbbTV Association. n.d.g. Introduction to AIT and its role in HbbTV. Verkkosivu. Viitattu 15.8.2025. https://developer.hbbtv.org/guide/launching-hbbtv-applications-from-a-broadcast-channel/introduction-to-ait-and-its-role-in-hbbtv/

HbbTV Association. n.d.h. Overview. Verkkosivu. Viitattu 9.8.2025. https://www.hbbtv.org/overview/

HbbTV Association. n.d.i. Overview of the HbbTV architecture. Verkkosivu. Viitattu 9.8.2025. https://developer.hbbtv.org/guide/introduction/overview-about-hbbtv-architecture/

HbbTV Association. n.d.j. The A/V Control object. Verkkosivu. Viitattu 12.8.2025. https://developer.hbbtv.org/references/media-playback-apis/the-a-v-control-object/

HbbTV Association. n.d.k. What is HbbTV? Verkkosivu. Viitattu 9.8.2025. https://developer.hbbtv.org/guide/introduction/what-is-hbbtv/

MDN Web Docs. n.d.a. Polyfill. Verkkosivu. Viitattu 18.8.2025. https://developer.mozilla.org/en-US/docs/Glossary/Polyfill

MDN Web Docs. n.d.b. The Video Embed element. Verkkosivu. Viitattu 10.8.2025. https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/video

MDN Web Docs. n.d.c. Vendor prefix. Verkkosivu. Viitattu 29.9.2025. https://developer.mozilla.org/en-US/docs/Glossary/Vendor_Prefix

Mueller, C. 2022. MPEG-DASH (Dynamic Adaptive Streaming over HTTP). Bitmovin 28.2.2022. Viitattu 15.8.2025. https://bitmovin.com/blog/dynamic-adaptive-streaming-http-mpeg-dash

Mux. n.d. A Guide to HTTP Live Streaming (HLS): Overview, definition, and considerations. Verkkosivu. Viitattu 15.8.2025. https://www.mux.com/articles/a-guide-to-http-live-streaming-hls-overview-definition-and-considerations

Open IPTV Forum. 2012. Release 1 Specification, Volume 5 - Declarative Application Environment, V1.2. Viitattu 5.10.2025. https://www.oipf.tv/docs/OIPF-T1-R1-Specification-Volume-5-Declarative-Application-Environment-v1_2-2012-09-19.PDF

PostCSS. n.d. Autoprefixer. Verkkosivu. Viitattu 29.9.2025. https://github.com/postcss/autoprefixer

Samsung Developer. n.d. Web Inspector for HbbTV. Verkkosivu. Viitattu 16.9.2025. https://developer.samsung.com/smarttv/develop/tools/additional-tools/web-inspector-for-hbbtv.html

Shaka Project. n.d. Shaka Player. Verkkosivu. Viitattu 8.9.2025. https://github.com/shaka-project/shaka-player

Tizen. n.d. Managing UI Component Focus. Verkkosivu. Viitattu 23.8.2025. https://docs.tizen.org/application/native/guides/ui/efl/component-focus/

Vite. n.d. Why Vite. Verkkosivu. Viitattu 20.10.2025. https://vite.dev/guide/why