BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

PERFORMANCE BENCHMARK FOR SMART TV PLATFORMS, SET-TOP BOXES AND GAME CONSOLES
VÝKONNOSTNÍ TESTY PRO CHYTRÉ TV, SET-TOP BOXY A HERNÍ KONZOLE

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR   Bc. TOMÁŠ MADAJ
AUTOR PRÁCE

SUPERVISOR   Ing. JIŘÍ HYNEK
VEDOUCÍ PRÁCE

BRNO 2018

Brno University of Technology - Faculty of Information Technology

Department of Information Systems   Academic year 2017/2018

Master's Thesis Specification

For: **Madaj Tomáš, Bc.**

Branch of study: Computer Graphics and Multimedia

Title: **Performance Benchmark for Smart TV Platforms, Set-Top Boxes and Game Consoles**

Category: Information Systems

Instructions for project work:
1. Get acquainted with the development of applications for smart TV platforms, set-top boxes and game consoles.
2. Study existing tools used for performance benchmarks of web browser cores.
3. Design a system for performance benchmarks of web browser cores used for chosen smart TV platforms, set-top boxes and game consoles. Provide proper support for maintenance of results.
4. Design a suitable set of tests.
5. Implement the designed system.
6. Evaluate applicability of the implemented system on selected devices. Focus on stability of results gathered by repetitive testing. Compare the results with the performance of web browsers running on desktop computers.

Basic references:
- HbbTv: *Resource Library - ETSI TS 102 796 Specification.* Online: https://www.hbbtv.org/resource-library/#specifications
- Open IPTV Forum: *OIPF Specifications.* Online: http://www.oipf.tv
- TIZEN Developers: *Tizen TV Web Device API Reference.* Online: https://developer.tizen.org/dev-guide/3.0.0/org.tizen.web.apireference/html/device_api/tv/index.html
- webOS TV Developer: *Luna Service API.* Online: http://webostv.developer.lge.com/api/webos-service-api/

Requirements for the semestral defense:
Items 1 through 4.

Detailed formal specifications can be found at http://www.fit.vutbr.cz/info/szz/

The Master's Thesis must define its purpose, describe a current state of the art, introduce the theoretical and technical background relevant to the problems solved, and specify what parts have been used from earlier projects or have been taken over from other sources.
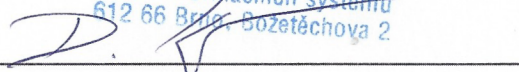
Each student will hand-in printed as well as electronic versions of the technical report, an electronic version of the complete program documentation, program source files, and a functional hardware prototype sample if desired. The information in electronic form will be stored on a standard non-rewritable medium (CD-R, DVD-R, etc.) in formats common at the FIT. In order to allow regular handling, the medium will be securely attached to the printed report.

Supervisor: **Hynek Jiří, Ing.,** DIFS FIT BUT

Beginning of work: November 1, 2017

Date of delivery: May 23, 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

Dušan Kolář
Associate Professor and Head of Department

Abstract

The purpose of this thesis is to create a tool for development of applications for certain minority platforms, primarily Smart TV and HbbTV. Those are implemented in a client-side JavaScript. Target group are hence the JavaScript developers, not the end-users. Said tool will target simplification and speed-up of development processes, mainly applications' performance tuning.

Abstrakt

Cílem této práce je vytvořit nástroj pro vývoj aplikací na určité minoritní platformy, primárně Smart TV a HbbTV. Ty jsou implementovány v klienském JavaScriptu. Cílovou skupinou jsou tedy vývojáři takových aplikací, nikoli koncoví uživatelé. Zmíněný nástroj bude mít za cíl zjednodušit a urychlit vývojové procesy, hlavně ladění výkonu aplikací.

Keywords

Smart TV, HbbTV, Tizen, WebOS, Benchmark, JavaScript, React.js, WebGL, CSS Animations

Klíčová slova

Smart TV, HbbTV, Tizen, WebOS, Benchmark, JavaScript, React.js, WebGL, CSS animace

Reference

MADAJ, Tomáš. *Performance Benchmark for Smart TV Platforms, Set-Top Boxes and Game Consoles*. Brno, 2018. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Jiří Hynek

Performance Benchmark for Smart TV Platforms, Set-Top Boxes and Game Consoles

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Mr. Ing. Jiří Hynek. The supplementary information was provided by Mautilus, s. r. o.. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

<div align="right">

. . . .

Tomáš Madaj

July 30, 2018

</div>

Acknowledgements

I would like to thank Mr. Jiří Hynek for his consultations and numerous suggestions and Mr. Petr Mazanec, CTO of Mautilus, s. r. o., for giving me a draft of the assignment in such a great detail.

Contents

Chapter 1

Introduction

Smart Television, sometimes also referred to as *Connected TV*, can be defined as a television which, besides traditional functionalities, has an internet access and is capable of providing non-linear content with interactive features. By *traditional functionalities* we understand displaying content received through conventional linear (commonly known as *live*) broadcasting media, either analogue or digital terrestrial, cable or satellite. Nowadays, the key part of Smart features is considered to be the capability to allow the user to arbitrarily choose and install additional software functionalities which he desires. From that point of view, the crucial milestone marking the birth of Smart TV was the availability of application catalogues as we know them from smartphones (Google Play or Apple App Store). Such devices from mainstream brands first came to market around 2011 with the appearance of Samsung's Smart TVs with Orsay OS and LG's Netcast OS. Today it is virtually impossible to buy a television without smart functionality.

Figure 1.1: 2017 Samsung model featuring Tizen 3.0 OS

Applications available on these platforms, however, have their uniquenesses. First, the most notable one is that the spectrum of applications is heavily dominated by services which are providing a video content. This can be both a video on demand (*VoD*) or live streams (linear broadcast); in either case, the content is received over the internet protocol. Such market share is indubitably given by the nature of these large screen devices. They, of

course, are not meant to be carried around and for decades now watching a video content has been the leisure time activity of choice for a huge part of the population globally. Nonetheless, there is a full range of applications available, much like they are in the smartphone application stores. Moreover, their visual appearance tends to be significantly different from the web or mobile client applications of the same services. The most determining cause of this effect is that the available peripherals are in most cases limited to traditional TV remote control with only a few buttons. Rarely some sort of kinetic controller is available, substituting a mouse. Therefore, the applications' GUIs must be designed in such way so that they would be fully controllable by these restricted input options. Despite sparing no effort, definitely less can be done within those limitations compared to for instance a touch screen controlled interface. Inherently, there must be a trade-off made between the number of features and retaining a certain level of simplicity and thus usage comfort.

1.1   Problems to be Solved

Just like the applications themselves, their development is also quite specific. Generally, we can say that almost all Smart TV operating systems allow running a JavaScript web application utilizing some modified web browser core with a platform-specific device API available. This goes for both the native and the HbbTV applications. The difference is that HbbTV is a standardized environment and, in theory, should be undifferentiated across all types of devices by all manufacturers. The reality, however, could not be further from this idyll. [6] [23]

By far the biggest obstacle for developers is the low-performance hardware used in these devices. It varies greatly over the range of classes of devices, roughly correlating with their price points. Also, with each generation a certain global improvement in the performance area can be observed, bringing each price category a notch above the last year's. Subjective observations done by the developers suggest that the CPUs and GPUs used in mid-end Smart TVs are roughly corresponding to those found in high-end smartphones 4–5 years ago. In the oldest or cheapest low-end devices, this problem reaches a critical level. A typical example is that the GUI animations are becoming infeasible when the device is simply lacking the performance needed to render them at some reasonable frame rate. It is virtually impossible to find out what exact hardware is used in the devices though. The manufacturers only specify the number of processor cores and rarely the size of RAM.

It is very difficult to estimate how the distribution of devices by performance in the real world looks like. This thesis is hoped to bring some insight into that area. It will be done specifically by creating a tool that will allow the developers to build a database of results collected from real devices, statistically evaluate them and to adapt their applications accordingly. Second, though probably even more important goal is to bring into the development process a new option of measuring the performance that will provide exact, reliable and stable results. Not only to evaluate the overall performance of a given device but also to test the speed of certain application components or algorithms, so that various implementations and solutions could be compared relatively quickly on multiple devices.

1.2   Outline of this Thesis

The assignment was created based on a list of requirements for such a product provided by the Chief Technical Officer of the company Mautilus, s. r. o. in Brno, Czech Republic. This

company specializes in the custom development of applications for those exact platforms mentioned earlier in the introduction and in chapter 3 which are also the subject of this thesis. The mentioned list is included in 2.1.5 . However, the development of this thesis was done completely independently and the result will be non-commercial and freely available.

Chapter 2 of this work will first bring some light into the development of applications for TV platforms and its issues, followed by a draft of features expected in the final product. Then, in chapter 3 the relevant platforms will be introduced in great detail. The research on currently existing methods and relevant tools for performance measurement is presented in chapter 4. In chapter 5 the proposed architecture of my solution will gradually be unwound by presenting a brief description of technologies that the final application will be built upon. Next chapter 6 contains the carefully selected set of tests which will be included in the resulting tool. Chapter 7 will describe the technical aspects and details of the implemented solution. At last, both the result application and sample data gathered by this application will be evaluated and commented in the 8th chapter.

One more remark I find necessary to be mentioned in the beginning: The televisions with the leading operating systems of today (Tizen and WebOS [25]) were first released only in 2015 and 2014 respectively. As a consequence of that fact, there are extremely few sources concerning this topic. The most relevant existing materials I was able to find are the two papers [9] and [8]. Hence, most of the knowledge presented here came from personal experiences of myself and other developers from commercial projects. Other significant sources were the official documentations, specifications and (device APIs) references issued by the manufacturers for the community of developers.

Chapter 2

The Development of TV Applications

This chapter is an introduction to the development processes of TV applications. The specific aspects and complications will be described and possible improvements will be discussed.

2.1 Specifics of the Development

The very first Smart TVs to enter the market between 2010 and 2011 suffered from a great number of issues. Ranging from an overload of software bugs (lacking even complete implementation of ECMAScript 5 [2]) to unbearably low performance. Moreover, back then, the smart functionality was considered as a premium feature and thus it was exclusive to the more expensive high-end series. The result of that was a very low market penetration. All these issues all combined made the 2011 generation deprecated already in 2014 and since then no native nor HbbTV applications are being actively supported on the 2011 models. Therefore, all further mentions of the „first generation" shall be referring to the devices of 2012 model line-up as this is the oldest relevant platform for development today.

The most important platforms (because of their market share [25]) are Samsung (using Tizen OS since 2015) and LG (with WebOS since 2014). Those two platforms are the primary interest of virtually all real commercial projects and they shall have the spotlight throughout this thesis as well. Their predecessors (Orsay and Netcast operating systems by Samsung and LG respectively) are the second most important thanks to the fact that historically the market share between brands had not been changing much.

„According to IHS data, the global smart TV market share was controlled by Samsung Electronics with 37.9%, followed by LG Electronics with 12.5% while Sony, Vizio, and Philips managed 8.4%, 5.9%, and 3.3% respectively." [25]

2.1.1 10-foot User Interface

The term „10-foot user interface" is used generally for all GUIs that will be primarily displayed on large screens viewed from distances of few metres, much longer than a desktop, laptop or hand-held displays. First, it must ensure good readability and visibility of all texts and control elements by making them large and contrasting enough. That is not all by far, the concept also comprehends the different peripherals which are typically represented by only a classic remote control. These limitations need to be taken into account when

designing the screen layout and position of the active (*focusable*) control elements such as buttons, slide bars, scrollable lists, etc. Complete functionalities from an existing desktop or touchscreen cannot always be transformed into a 10-foot UI. The UI often requires significant simplifications. The most crucial principle of GUI design is to always make clear which element is focused and never lose the focus. Because of the controllability by an RC with almost always only 4-directional arrows, it is highly recommended to place all the elements in an orthogonal grid, so that it would always be clear which element will receive the focus next after a user action. Nonetheless, the GUI needs to be universal enough to allow the small fraction of user base who will connect a kinetic controller, mouse or a full-layout keyboard (which is completely possible with USB peripherals) to fully utilize them as well.

2.1.2 Smart TV Quality Assurance Process

Before each application becomes available to the end-users by being listed in an app store, it must pass a Quality Assurance Process (*QA*). This is true for literally every platform with the exception of the HbbTV. Android TV applications are naturally published and distributed through the Google Play and they go through its QA. Other proprietary systems have own app stores as well as own QA handled by each vendor individually. They all have guides, checklists and documentation templates to be fulfilled beforehand. [1] The two trickiest and least obvious parts that have to be implemented correctly in every single application are error handling and multitasking (on devices that support it). The problematic part of multitasking is correctly suspending and restoring the application state, which is an especially complex procedure in players with DRM involved. Very similar procedures are applied to make instant-on functionality work correctly. The applications must be seemingly „unfrozen", unaffected by the TV being turned off to standby mode and turned on again.

Although the release date of the benchmark application developed as a part of this thesis is not yet scheduled, it sure might be useful to know what such a process includes. As it typically takes weeks, it's highly desired to pass the QA right away in the round. The key points to manage that are very well summarized in the blog post [2] by the Mautilus company.

2.1.3 Background and the Core Problems

As it has been brought up in the introduction, the main problem the developers of these applications are dealing with is the low performance of devices caused by their hardware. Insufficient memory had become less of a problem after only a couple generations of Smart TVs. However, the insufficient computational power of their CPUs and GPUs persists. Majority of them (for example even 2016 mid-range Samsung series 6 Tizen) struggle to even render a transition of few images at $60\,\mathrm{FPS}$ making even such trivial an extremely common task visibly stuttery. And that is one of the worst possible blows to the overall user experience. It is absolutely necessary to address these issues by carefully tuning the performance through various software optimizations. Now, such optimizations are typically done by finding the most efficient ways of lowering scene complexity — such with minimal

---

[1]example of Samsung: https://developer.samsung.com/tv/distribute/launch-checklist

[2]https://www.mautilus.com/blog/smarttv-app-qa-process-10-tips-to-make-it-through-on-the-first-attempt/

impact to the visual quality while causing the greatest performance improvement. Such process, of course, is always very circuitous, protracts the development significantly and thus is also very expensive.

The situation gets even worse regardless. Customers who order an application development naturally tend to require widest possible coverage of supported devices. If an optimization for the slowest devices was utilized, the product might become rather visually unappealing. Which is both completely unnecessary and undesired as the mid- and high- end devices would be left with the same scanty application, leaving their potential unfulfilled. Usual compromise of this is to target the estimated median of overall devices' performance. Of course, that is a rather suboptimal one as it results in slower devices suffering from jerky behaviour and faster ones are typically left with lower resolution than they would actually be able to handle.

The ideal, perfect solution would be to have an application that will be able to adjust its complexity (hence its performance requirements too) on runtime to precisely match and fully utilize the performance level of the device it is running on. That part of this work will most directly affect other applications developed in the future.

The global real-world distribution of performances of devices is hard to estimate. Within the sales of a single generation, it can be expected to have somewhat normal distribution shifted towards low-end. The reason behind that is that, as mentioned earlier in the introduction, the performance of a device usually roughly correlates with its price tag. And therefore it copies the number of pieces sold. The second aspect is a certain general shift in performance upwards with each new generation to come to the market. Unlike with smartphones, there is only a tiny after-market for the private trading of used television. They are likely to be used until they malfunction which makes the oldest devices back from, say, 2013 still very sizeable and significant group. In comparison, most of the 2013 smartphones have already forced their owners to an upgrade a long time ago. However, the first aspect of televisions to become outdated are the smart functionalities, application performance and application support. And as the televisions have for decades had the secondary function as displayers for external sources (historically SCART, in the last 15 years HDMI), the smart functionalities can be upgraded in this way as well. There are countless of cheap streaming sticks, dongles and boxes (like Chromecast, Roku, Amazon Fire TV, ...) which can effectively multiply the lifespan of the television by cheaply providing an up-to-date hardware and software.

2.1.4 Improvement of the Development Process

The purpose of this work is to develop a performance benchmark — a testing application that will execute performance measuring tests. It will be run directly on the device being tested. The test runner shall be automatized and the set of tests configurable in multiple ways. The core functionality is to measure the performance of device's web engine, that is the rendering core (for example WebKit, Chromium, Presto, ...) and the JavaScript engine (for example V8, JavaScriptCore, ...). That is closely followed by the need for assigning a score or rating based on the result. The score should be an absolute value normalized to a chosen device, probably the currently most wide-spread model. The reason for that is so that it would be possible to compare any of the devices tested at any time with each other. Also, it is desirable to compare them with the standard browsers run on a much more powerful desktop PCs. A common practice of the developers is to run and immediately test

the progress in a standard browser with CPU throttling enabled. This will provide them with an accurate value of the throttling ratio.

There are few more expected use cases. It is desired to collect the results for the purpose of statistical evaluation. For this, a simple server with a database solution and suitable API will be needed as well as the support in the client-side application and library. Furthermore, an optional extension of the benchmark might be a functionality similar to the jsPerf [4.1]. It would allow the developers to execute, measure and to compare the speed of arbitrary JavaScript algorithms/scripts in order to find the optimizations with a lesser effort. It should also be able to test the performance of an isolated React component, test the speed of animations etc.

2.1.5 Solution Requirements

This subsection contains the complete assignment received from the CTO of company Mautilus, s. r. o. in Brno. However, as it was already stated in the introduction chapter, the result will be non-proprietary and publicly available. This assignment served as a mere inspiration for the thesis and will not be followed in its entirety mostly because its extent is too big.

Two distinct parts are expected as the result:

- Standalone application benchmark

- Light-weight library

Standalone Benchmark

The standalone application will be used both to comfortably test new features in development and to build a database of detailed performance aspects of various devices. There are pretty much no limitations to the execution time.

- It will run an included set of tests (or its subset) in a manner similar to for example browser benchmarks Octane 2.0 [3] and Dromaeo [4].

- It will display the results of the test both graphically and log them to a developer console as well.

- It will calculate a single performance score in such manner so that it will be reasonably stable when the benchmark is run repeatedly. (Problem that the developers have been frequently encountering is that the televisions are running various tasks in the background which have a short-term but significant influence on the speed of application of interest. Therefore it is entirely possible that substantial fluctuations will be recorded throughout multiple runs.)

- Based on the result assigning the device into one of a few discrete performance classes (for example low-end, mid-end, high-end). That will allow the automatic runtime adjustments to be done in a simpler way by selecting a discrete configuration set.

- The results along with a detailed info about the device will be posted to a logging server.

---

[3] https://chromium.github.io/octane/
[4] http://dromaeo.com/

- Support of as many platforms as possible. The cross-platform compatibility will be guaranteed and limited by the use of the MAUTILUS Smart TV SDK$^{\text{TM}}$ [5.2.1] and drivers available within it. Starting with HbbTV 1.1.1, Samsung Tizen, Samsung Orsay, LG WebOS, LG Netcast, Hisense Vidaa, Panasonic Firefox, PlayStation 4, Xbox One, Google Chrome, Android TV (Philips, SONY), . . .

Library

The library should be possible to be included in any third party application. This could be achieved by incorporating it into the Mautilus Smart TV SDK$^{\text{TM}}$ [5.2.1] and that way into any other applications built on it.

It should contain only a minimal subset of the most important tests from the standalone application [2.1.5]. Thanks to that it could be run much more quickly. Typically only once during the first launch of an application after it was installed or, optionally, at each start or even multiple times. That is, however, not a concern of the library as it will be entirely in control of the third party developer. Important is to keep the launch delay as little as possible.

For the purposes of the library, several classes according to the performance level should be established. The third party applications which will include this library will then, after resolving their performance category, be able to select the corresponding configuration. Those will be manually assembled sets of functionalities switches specific to each application and each performance class.

The key properties are:

- Short execution time of the tests — ideally $\sim 2000\,\text{ms}$ at most.

- Small size — ideally under $100\,\text{kB}$.

- Beside the brief performance test it should be able to test support for various features of interest (HTML5 Canvas, CSS3, WebGL, . . .).

- The developers should be able to configure the subset of tests they will want to be used in their applications.

Examples of expected tests

- Test of DOM rendering speed.

- Speed of all types of DOM query selectors.

- DOM manipulation: speed of generating and modifying HTML elements, JavaScript-inline-style animations. Correlation of these properties to various sizes of DOM.

- Speed of mathematical operations: for example physical simulations, object collisions. Natives Math methods as well as simplified, lowered precision and integer calculations.

- Rendering speed of images in various formats, versions, compression settings, and resolutions.

- Test of rendering frame rate in a real 2D game.

- Speed of CSS2 and CSS3 animations.

- Test of array operations speed (for example flattening of nested arrays).

- Test of string operations speed including Regular Expressions.

- Test of bitwise operations speed.

- Test of 2D and 3D raycasting speed.

- Test of encryption and hash functions speed (for example AES, SHA-256, SHA-512).

- Combined tests of various operations and animations done in parallel, which would simulate common situations in a real application or game.

- HTML5 support test and canvas 2D rendering speed test.

- WebGL support test and rendering speed, Ray-Tracing renderer speed test.

- Web Workers support test and parallel operations speed test and scaling correlation.

Furthermore, I intend to add a single more test in order to also determine the current state of readiness of the WebRTC[5] technology on these platforms, where I foresee a great potential for it.

2.2   General Characteristics of the Target Platforms

There is not much of a hierarchy present in the nomenclature of the platforms discussed here, but there are some components that must be specified when referring to some subset of Smart TV platform. They will be shortly explained now. The most important is, of course, the brand or the manufacturer. Second, comes the operating system and its version. Last is the year of the release. An example: *LG WebOS 3.5 2017*. Furthermore, there can be a division between series and model groups specific to brands which will be elucidated in the corresponding sections of this chapter.

Generally speaking, pretty much all of the manufacturers update their product line annually, they rarely use multiple operating systems or even versions of them in one generation, making it a little easier to keep track of. Exceptions to this rule are:

- Samsung having both Tizen and Orsay OS in the 2015 model lineup.

- LG having both webOS and Netcast in 2014 and 2015 models.

- Philips is by far the worst case, in 2012 having used NetTV OS and Fusion OS in version 2 and 3; MKT OS in 2013 and 2014; NetTV and Android TV in 2015. Since 2016 they have abandoned all but Android TV.

Following chapter will contemplate the specific aspects of each individual platform or operating system in great depth.

---

[5]https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API

Chapter 3

TV Application Environments

This chapter elaborates on individual TV ecosystems including their application platforms and their development environments.

3.1 HbbTV

HbbTV is a technology of hybrid television. That means a combination of linear broadcast with some interactive features. Those are implemented as a JavaScript application bound to a DVB channel via its metadata. Only a URL leading to the application is included in the DVB transport stream. When a TV recognizes the special HbbTV tag (also referred to as *HbbTV signalization*) marking the availability of an application, it accesses the URL and launches the application in an overlay over the linear broadcast.

There are several versions of the HbbTV standard. Their feature set is being gradually extended. However, it also brings the issue of the highly inconsistent support on the devices. The product owners are forced to make a decision on the lowest compatible HbbTV version. That always leaves a group of users unable to access the application. However, in the devices from the early generations, it is not rare to see the HbbTV implemented incompletely or even straight-up incorrectly, especially the video player APIs. This is specifically mentioned in 7.4 . The specifications of all versions of HbbTV are publicly available in full extent. [3] [16]

Nearly all of the early generations of devices use the Opera Presto core, after 2014 a shift towards the Apple WebKit can be observed in the major brands, and latest, after 2016, Samsung and LG are switching to Chromium core with V8 JavaScript engine. The rendering core used can be easily detected from the User-Agent value. [20] [6]

3.1.1 Use Cases

HbbTV is in the vast majority of cases used as supplementary OTT (*Over the Top*) service of a TV channel. It is often referred to as *Red Button* portal. These portals can offer a wide range of applications, typical ones are EPG (*Electronic Program Guide*), weather forecast service, news, interactive advertisements, games, . . . Often such applications are already provided as part of the Smart TV OS native toolset, nevertheless, they still tend to be commonly implemented in the HbbTV portals. The reason for that restriction is that they can be completely customized to each channel's specific needs, often featuring significant extension, for example, trailers in the EPG. The biggest potential of this platform is mostly seen in the advertisement because, naturally, TV channel owners are looking for a way to
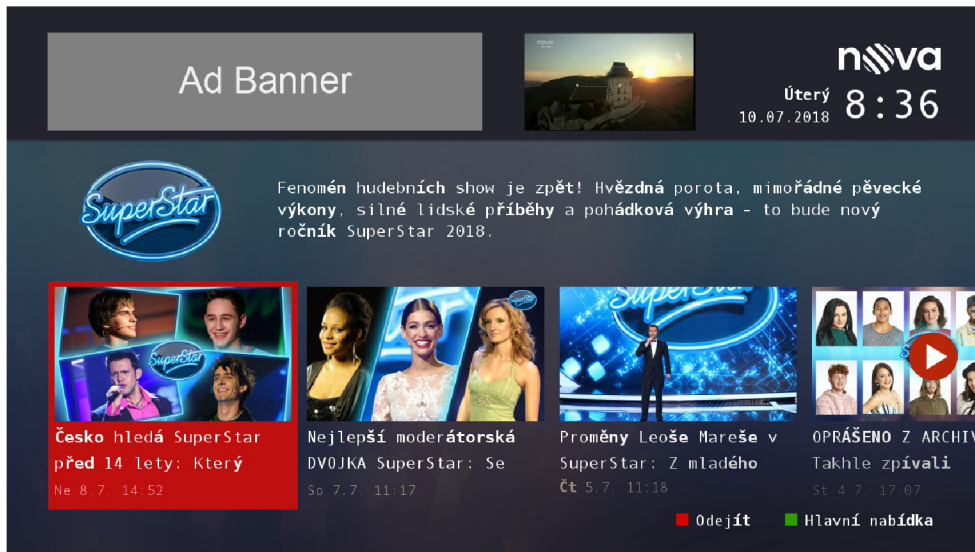
Figure 3.1: HbbTV application of Czech TV channel Nova. Thumbnail of the live broadcast is located in the header. Screenshot was taken on a set-top box VU+ ZERO running a linux-based Enigma2 OS. Font rendering inconsistency is very noticeable.

justify the development costs by monetizing these systems. Also, the interactive advertisements are perceived to be much more intriguing and thus efficient. Common practice is that an advertiser orders development of an advertisement which he then distributes to the TV companies to have it deployed. There are already at least two HbbTV advertisement management systems (intended for the TV companies) in existence.

Lately the option of *injecting* VoD (*Video on Demand*) into the linear DVB content if often mentioned for the future versions of HbbTV 2.0 . It could serve mostly for personalized advertisements.

3.1.2 Versions and Compatibility

The specifications of all versions of HbbTV are publicly available in full extent. [3] [16] The versions of HbbTV most commonly found implemented (or claimed to be implemented) are 1.1.1, 1.2.1., 1.3.1 and 1.4.1 . Version 1.5.1 appeared for the first time only in 2017 and the same applies for 2018 models. However, Samsung suggested a possibility of bringing the support for HbbTV 2.0 in future firmware update. [4] In general, the biggest new features of the latest 2.0.x versions are various possibilities of cross-platform integration and better video support including UHD resolution, high framerates, modern codecs like HEVC and VP9 and HDR colour profiles.

It can happen sometimes that certain feature is required (most frequently adaptive video streaming using MPEG-DASH) which is supported only in given HbbTV version (1.2.1 in case of MPEG-DASH). In such case, the application is disabled from even launching (except for some error message) on devices with lower HbbTV version. This significantly reduces the testing effort as those older devices may be ignored completely.

3.1.3 Development Specifics

A user interface design aspect specific for HbbTV are so-called *safe-zones*. The resolution of HbbTV application is 1280×720 px but only the area without 5% margin is defined as safe to be used because it might get cropped. It is a part of OIPF specifications [16], however, only two manufacturers really require them. Sony crops the application by 3.5% (20 px on top and bottom), Panasonic by 5% (36 px on top and bottom). This concept of safe-zones is a relic surviving in the broadcast standards from the age of CRT screens.

Another limitation is the availability of fonts. The default font family is Tiresias Screenfont. Support for custom fonts officially starts in version 1.3.1. An important word of warning: each manufacturer can implement the Tiresias font themselves and the resulting text size may vary significantly. That can cause complications when the scene layout is too tight.

Last and perhaps the most important are the controls. An application can set a *keymask* and then register key events handlers. To put it shortly, only the absolute minimum of keys should be masked out for the application. Other keystrokes should be left to be handled by the TV. For example, channel selection keys should never be used. Preventing the user to exit the application or change the channel may result in fines by the broadcast regulators and such incidents did happen so this should be taken seriously.

3.1.4 Deployment

The applications can only be launched on the TV when the tag is received as a part of the DVT channel's transport stream. Then, it can lead to the application hosted on an HTTPS server, for example in the local network. That can be achieved by creating an own DVB broadcast using, for example, the OpenCaster[1]. Although, there are some concerns about the legal aspects of this practice in some countries.

For this exact purpose, to even make testing HbbTV applications on real devices somehow feasible, Mautilus has developed a HW gadget called *HbbTV Playout kit*[2]. This tool provides a user-friendly UI allowing fairly easy set-up of a custom test channel(s) with, of course, the HbbTV signalization leading to any own test server. It is also based on OpenCaster and Raspberry Pi, all in a custom 3D printed enclosure.

3.1.5 Issues and Testing

As mentioned before, HbbTV applications are built using HTML and JavaScript and as such should be runnable in any browser. However, they are heavily dependant on the HbbTV API. There are plug-ins emulating this API available for Mozilla Firefox and Google Chrome. Still, it is recommended to access the application deployed on a local server because of the CORS restrictions (*Cross-Origin Resource Sharing*). Those should be also part of the HbbTV on real devices, however, only Sony is really using it.

„*CORS is a mechanism that uses additional HTTP headers to tell a browser to let a web application running at one origin (domain) have permission to access selected resources from a server at a different origin. A web application makes a cross-origin HTTP request when it requests a resource that has a different origin (domain, protocol, and port) than its own origin.*
For security reasons, browsers restrict cross-origin HTTP requests initiated from within

---

[1] http://www.avalpa.com/the-key-values/15-free-software/33-opencaster
[2] https://www.mautilus.com/products/hbbtv-playout-kit/

scripts. For example, `XMLHttpRequest` and the Fetch API follow the same-origin policy. This means that a web application using those APIs can only request HTTP resources from the same origin the application was loaded from unless the response from the other origin includes the right CORS headers."[3]

Also, in 3.1 it was said the oldest devices use Opera Presto browser core. Because of that, the desktop builds of Opera from before 2012 are valuable testing tools of last resort.

It is highly recommended to test the application on a widest possible range of real devices. Ideally at least each OS in each version by each manufacturer. Additionally, it has to support any device capable of receiving the relevant broadcast and with HbbTV capabilities. That includes all kinds of set-top boxes. And, those are by far the most problematic devices, which is not surprising as those are typically small manufacturers making low-cost devices in small batches. Many share the same OS core, but fragmentation is still very high. Typically, the higher-end models use Linux-based OS Enigma2 with various GUI extensions which, however, do not matter, because there is only one HbbTV plug-in available and it is compatible across all Enigma2 systems. Issues appear with small Chinese vendors who decided to implement the HbbTV themselves. These bugs are literally impossible to detect without testing on real devices and they are also extremely difficult to debug because of the lack of HbbTV developer tools, such as a remote debugger. Often even the little available, such as a remote log console is made useless when a device fails to run JavaScript at all because of the wrongly implemented parser or something else in the JavaScript engine. Relatively common is an incomplete implementation of ECMAScript [2] even in its version 5 which is fortunately rather easy to discover and fix with an appropriate polyfill.

A similar situation is with HTML and CSS support. HTML usually did not support even `table` elements on devices until 2014, likewise CSS3 was a rarity until roughly the same time. On the other hand, at some brands, some HTML5 and CSS3 functionality appeared ahead of support of the HbbTV version officially implementing it. [10] Latest devices often suffer bugs, for example, `canvas` element was dysfunctional on Samsung Tizen 2017 (in both native and HbbTV applications) for many months after their initial release but luckily, unlike most bugs, got fixed by a firmware update. Missing support for certain CSS effects is perhaps the least severe problem as it doesn't cause an application to crash.

A possible measure to reduce the testing device pool by an order of magnitude could be to the only test the devices sold or distributed in the target country. It is not rare that only very few people own some of the most problematic devices and want to use the HbbTV application. In such case, when a customer (the channel owner) receives a complaint, it is often cheaper to give the user a new device (to which they so far always agreed) than to identify the bug and implement a fix for it.

3.1.6 Safety Concerns

In February 2017 fairly serious HbbTV attack was presented at the EBU Media Cyber Security Seminar [4]. Essentially, it is performed by creating a DVB-T broadcast with a fake channel configured to fully match some real legitimate channel. The only difference is in the HbbTV tag — it is modified to lead to the attacker's application. The broadcast does not need to have high power. As long as the targeted device receives it stronger than the real broadcast, it switches to use the stronger source, in this case, the malicious one.

---

[3]https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
[4]https://youtu.be/bOJ_8QHX6OA

A property of HbbTV is that the application is always launched immediately without any indication (except for the HbbTV tag being present). The only defence against this kind of attacks is disabling the HbbTV completely. Of course, only the terrestrial DVB is vulnerable to this attack, it is virtually impossible on cable nor satellite broadcasts.

There is already McAfee Antivirus application available in the Tizen Store (only for 2017 models with Tizen 3.0) [5] but it does not mention HbbTV anywhere in the descriptions.

3.2   Samsung Tizen

Samsung uses the Tizen OS in countless variations and versions across many platforms ranging from smartphones and smartwatches to smart home appliances like fridges and washing machines. It was first featured in smart TVs in 2015 in its version 2.3, then in 2016 version 2.4, in 2017 version 3.0 [Fig. 1.1] and also in the latest 2018 models in its version 4.0. [22]

Tizen applications can be implemented in JavaScript (referred to as „*Web Applications*" by Samsung's guides), .NET, or C++ (referred to as „*Native Applications*"). The application has access to the Device API [21]. This will, however, not be used directly because of usage of the Mautilus SDK [5.2.1], which introduces an API acting as a generic abstraction layer over the vendor-specific APIs. Tizen TVs use various rendering cores and JavaScript engines with inconsistent supported feature sets and versions. The up-to-date overview can be found in [19] and [20].

The functionalities of the Tizen API are categorized into several privilege groups. [18] The Tizen ecosystem uses a system of certificates (*Author* and *Distributor*) which are linked to a Samsung account when created and are carrying its privilege level. When an application uses a functionality belonging to certain privilege group (for example DRM for the streamed video player) it has to be signed with a certificate authorized for that privilege group usage during the building and packaging process. Unsigned packages are not possible to be installed on a real device.

3.2.1   Versions and Features

This is one of very few parts that are properly documented in one place, well structured and in great detail. General overview of most important properties including supported streaming and DRM combinations [19]; Overview all models, their categorization, operating systems and their versions [22]; Web Engine Specifications (available JavaScript, CSS and HTML versions and features) [20]; Supported media formats (down to codec subversions) [17].

3.2.2   Tizen Studio

The Tizen Studio [6] is a comprehensive set of tools for developing Tizen native and Web applications. It consists of an IDE, emulators, toolchain, sample code, and documentation. Tizen Studio runs on Windows and Ubuntu, as well as macOS. It is based on the Eclipse IDE, overtaking all of its Java slowness and instability and taking them much further. Tizen

---

[5] https://www.iotgadgets.com/2017/05/samsung-partners-mcafee-brings-security-software-galaxy-s8-smart-tvs-pcs/

[6] https://developer.tizen.org/development/tizen-studio

applications can also be developed without relying on the official Tizen Studio, as long as the application complies with the Tizen packaging rules.
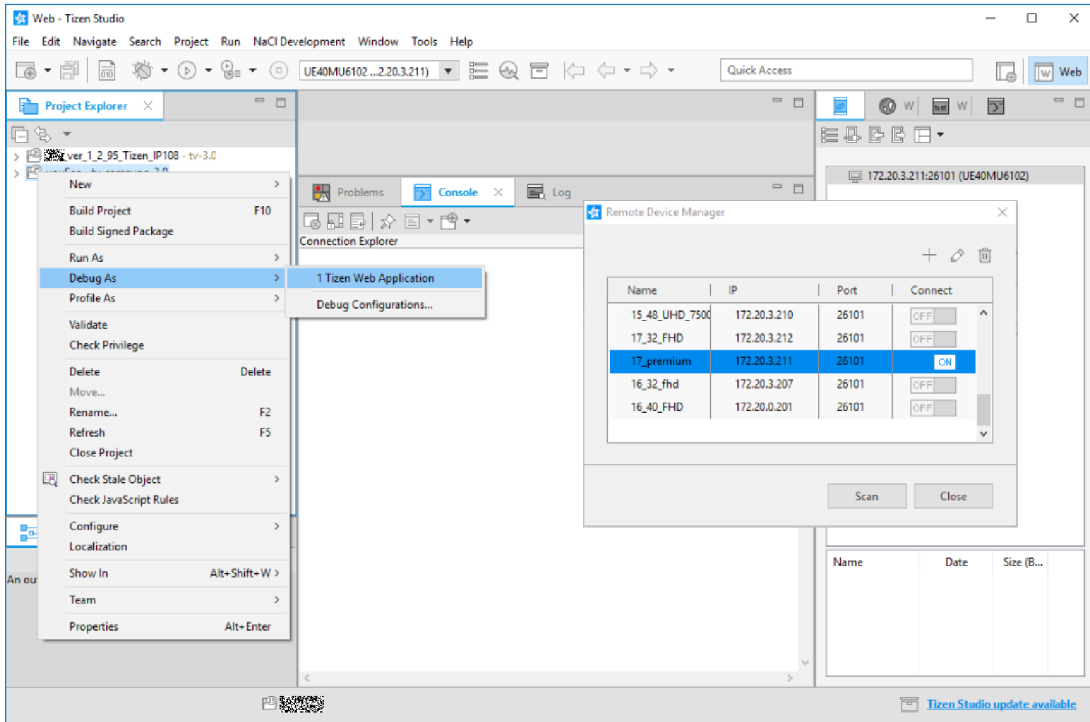

Figure 3.2: Tizen Studio in Windows. Montage: shown are the context menu for installation of project with a debugger attached and the Device Connection Manager

The predecessor of the Tizen Studio was called Tizen SDK. It was deprecated in late 2016. It is still possible to use it, but it is unable to build applications with Tizen 3.0 (2017 TV models) support.

Hidden Issues

Besides explicit requirements such as installed JDK, there is one crucially important condition which is mentioned absolutely nowhere. In order to use create and sign a project package, it is absolutely necessary to have a Windows User Account name that does not contain a space character. That would cause the packaging command chain to fail. Specifically because of one folder manipulating tool within it which seems to be running sort of emulated Unix regime. This fails when a path containing a space is used. There is no error message, only a numerical error code which is not even unambiguous to this issue. Back in 2017 when spent days dealing with this bug for the first time there was no mention of it even in Samsung/Tizen developer forums. It had been resolved by a pure guess. The WUAC username can be checked with windows command `echo %username%`. Changing the username of an existing Windows account is possible but very much recommended not to do so because it can cause tremendous issues with many other Microsoft applications (Office etc.). Creating a new Windows account is really the least complicated option.

Similarly, changing the default installation locations may cause issues with some tools within the Tizen Studio and it is advised not to change it. In all following CLI examples I

will used the default installation path (`C:\tizen-studio\`) as well as default path to the projects workspace (`C:\Users\%username%\workspace\`).

The Tizen Studio IDE has significant memory demands especially during packaging, installing and debugging. Running out of free memory had been experienced on machines with 8GB RAM installed. It can cause the mentioned operations to fail without any error message suggesting the real cause.

Last but not least, whenever a file is imported into a project in the Tizen Studio, the IDE runs a JSHint validation. When an entire large project is imported, this can take a couple minutes on a low-power dual-core CPU. This isn't the case when a `.wgt` package is imported, as those are validated during packaging (with CLI as well) and the validation after the import is skipped. This can, however be disabled in *Window > Preferences* as shown in Figure 3.3. The use case for importing entire file structures will be demonstrated later in 3.2.5.
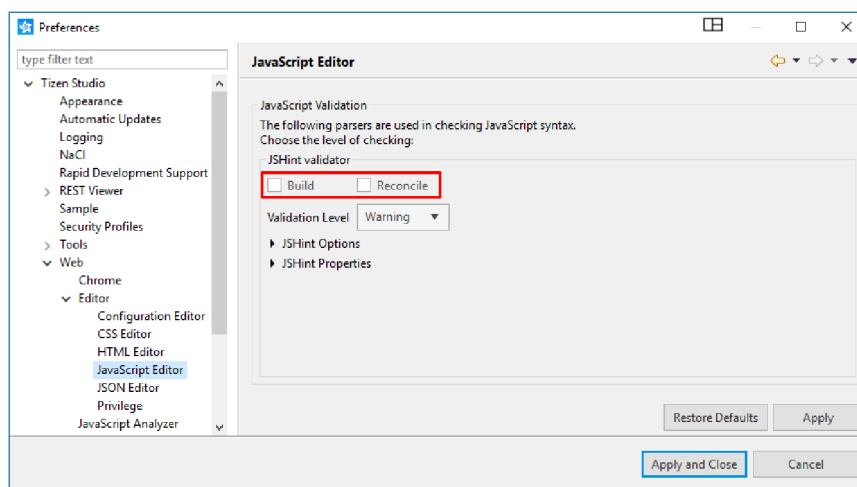
Figure 3.3: Disabling the automatic JSHint syntax validation.

Package Manager

By default, Tizen Studio installation contains only very basic tools whilst the others need to be installed additionally as extensions. That applies also to extension allowing development, packaging and installation for televisions. The utility for this purpose is the Package Manager. It is an independent tool and can be launched either from the IDE (*Tools* menu or `Alt` + `Shift ⇧` + `P`) or directly at `C:\tizen-studio\package-manager\package-manager.exe`. In the same location can be found a `package-manager.exe`, however, it seems to not be meant for a standalone use without the GUI (no documentation is available in itself or anywhere else). The list of packages recommended for development for TV is in Figure 3.4.

Tizen Studio CLI

The Tizen Studio also offers a command line interface, although it is implemented only for a somewhat limited subset of functionalities. Also, from my rather extensive testing, I found it significantly problematic in older Tizen SDK, with many functions working incorrectly or not working at all, most often with unhelpful error messages with no obvious cause. There is no complete and consistent documentation available, instead, there are only bits and

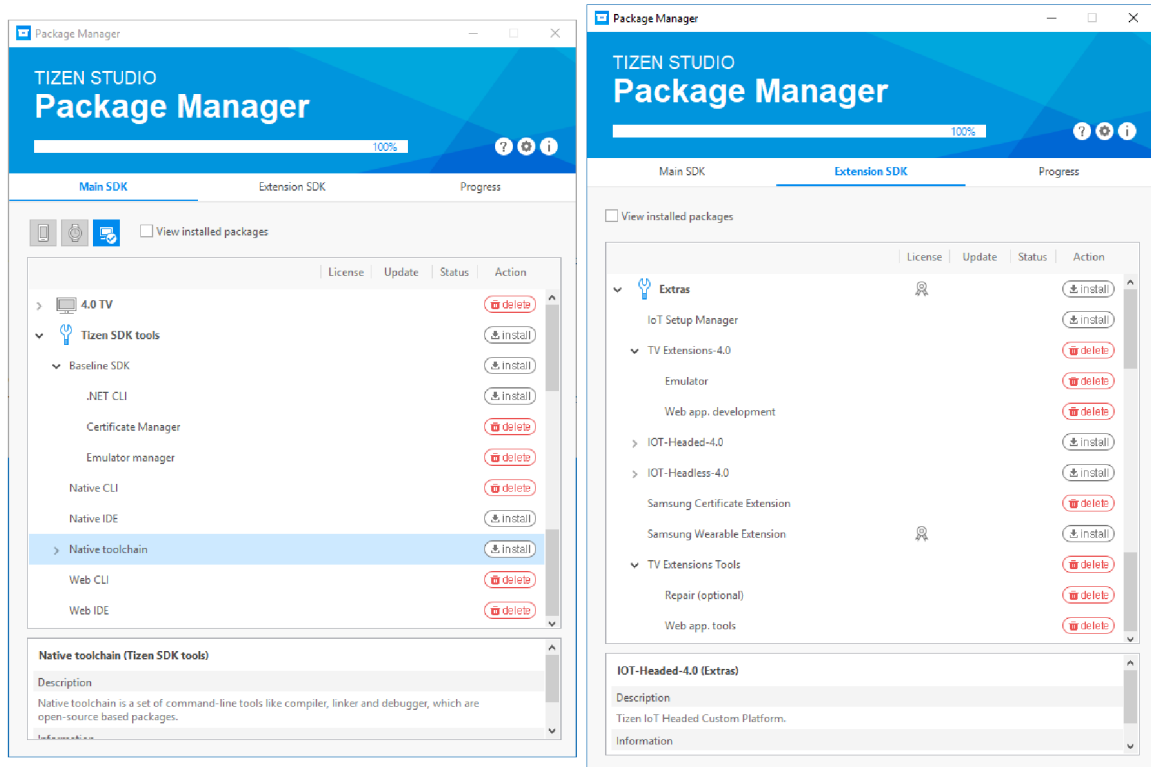Figure 3.4: Tizen Studio Package Manager GUI in Windows. Recommended set of extensions is shown as installed (those with a red delete button)

pieces of documentations, references, tutorial and examples for various subsets of features offered to be found on several Samsung and Tizen developer web locations.

CLI option of the most common actions will be demonstrated as well as the more common IDE.

3.2.3   Certificates and Security Profiles

The Certificate Manager GUI available in the Tizen Studio IDE is highly recommended over the CLI. Also, the *Samsung Certificate Extension* [3.2.2] [Fig. 3.4] is required. This Manager replaced the

As was mentioned at the beginning of 3.2, Tizen uses a system of certificates and there are two of them (*Author* and *Distributor*) needed for packaging, installing and publishing the application. In reality, a single certificate file (`.p12`) can be used in places of both of them and also across any number of applications. The reason why they're split into two is to allow better control in case two different organizations are developing and publishing a single application. The certificates represent a link to a specific Samsung account bearing a certain level of privileges.

Likewise, for working on several projects simultaneously Tizen Studio has easily switchable *Certificate Profiles* (previously called *Security Profiles*) each of which can be set up with different certificates. One of them is always set as the global default in the CLI and whenever later a profile is not specified in a command, the active global default is used.

A certificate is needed only for signing a package. Installation of a pre-made and signed `.wgt` package 3.2.4 can be done via the Tizen Studio **CLI** without a certificate. On the

contrary, when using the Tizen Studio **IDE**, a certificate is necessary because there is no discrete option for an installation. Instead, the `.wgt` package must be imported (its project space is loaded) and when being installed, it is first implicitly packaged and signed again.

GUI set-up

The steps of creating a Certificate Profile using the Tizen Studio Certificate Manager GUI are shown in Samsung's own guide[7] very clearly and in enough detail. The Certificate Manager can be found in *Tools* menu or launched by [Alt] + [Shift ⇧] + [C]. The `.p12` certificates can be created either directly in the Certificate Manager (which is recommended for the Distributor certificate as the DUIDs (Device-Unique Identification) to which it will allow installations are bound to it) or at Samsung or Tizen websites and then imported. In either case, a Samsung Developer account is a necessity.

CLI set-up

Creating a *dummy* Author certificate (useful for installation over CLI) is described in manual A.4 .

3.2.4 Packaging

The `.wgt` package is basically a renamed `.zip` archive, it can be easily looked inside. Besides the identical project structure, it contains a license file. There is also a second kind of package, a `.tmg`, which is described later in 3.2.5.

In the Tizen Studio IDE a `.wgt` package is implicitly created during installation or it can be done individually. Both options are visible in Figure 3.2 .

CLI

Both *Web CLI* and *Native CLI* extensions [3.2.2] [Fig. 3.4] are required for this for a reason unspecified by Samsung. Missing the *Native CLI* extension causes a meaningless and completely unhelpful error *„Exception in thread "main" java.lang.NoClassDefFoundError: org/tizen/core/gputil/PathUtil/"*.

The command to create a `.wgt` package (by default will be placed in the working directory) is:

```
C:\tizen-studio\tools\ide\bin\tizen.bat package -t wgt
-s <name_of_certificate_profile> -- <path_to_working_directory>
```

3.2.5 Installation on Real Devices

There are two ways to install a packaged application onto a real Tizen device. In either case, it is first needed to enable the developer mode on the TV where the application should be installed.

1. On a TV, navigate to *Smart Hub > APPS > My Apps* .

---

[7]https://developer.samsung.com/tv/develop/getting-started/setting-up-sdk/creating-certificates

2. Type sequence *12345* on the remote control or a keyboard. A window will be opened.

3. Toggle the switch *Dev mode:* to *on*.

4. Into *Host PC IP* enter the IP of the PC from which the Tizen Studio will be connected.

5. A message suggesting a reboot will appear, however, I found this not necessary if the Dev Mode had already been turned on at some point before.

Via Tizen Studio IDE

First, recommended one, is to install the application using the Tizen Studio. Now, there multiple options for how to get it in there in case it was developed in some other environment. The IDE itself is of a rather poor usability and thus there is a good chance the application will be developed in some other IDE more suitable for JavaScript projects. One option is to create a new empty project in Tizen Studio IDE and then the entire file structure can be imported into the project. This is much less comfortable and slower (especially with the JSHint validator enabled [3.2.2]). Then the project can be built, packaged and installed. Or, thanks to the Tizen Studio CLI option, the packaging can be integrated into the building chain in any task runner (Grunt, Gulp, Webpack, . . .). Afterwards the `.wgt` package can be imported into the Tizen studio (which implicitly loads it as a project because it must have been signed during packaging even via CLI) and directly installed and launched on a television.

Via Tizen Studio CLI

The installation can also be performed using the CLI only and thus also included into an automatized task chain. The process is described in manual A.3 .

From USB drive

The second way is to install the application from a USB drive. This, however requires a Samsung Seller account with Partner level privileges. That is because the `.wgt` package cannot be installed directly, it first needs to be signed using a Samsung's online USB Demo Packaging Tool [8]. There it is recertified, repackaged into a `.tmg` package (still effectively a `.zip` archive) and a separate license file is generated. Both the `.tmg` package and the `widget.license` file need to be placed in a folder named `userwidget` in the root of the drive. For for example `F:\userwidget\myApplication.tmg`.

After attaching the USB drive to a television a message in the top part of the screen will appear informing about installation start and then the second one about the installation success or failure (most often caused by certification issues). Then the application will appear in the list of installed applications and it will stay permanently installed and accessible even after unplugging the USB drive. In fact, when the TV is turned off and on again with the USB drive attached, it detects the installation package and reinstalls the application.

On Tizen 2.3 (2015) it is no longer possible to install an application from a USB drive. This can now only be performed on Tizen 2.4, 3.0 and 4.0 (2016, 2017 and 2018).

Tizen Studio **IDE** also has one crucially important feature — it allows the applications to be installed and launched on a real device with a debugger attached to them. It is

---

[8]http://seller.samsungapps.com/tv/portal/main

essentially a stripped-down version of the Chromium debugger (or at least is based on it and the Chrome DevTools remote debugging protocol [9]) with DOM inspector, Styles live editor, JavaScript debugger, console, network inspector, performance and rendering analytic tools and many more. This is shown in Figure 3.2.

3.3   LG webOS

Also, the webOS applications are using a JavaScript API [5] as a way to get access to the device's functions and control them. LG splits it into Standard Web API (normal JavaScript and HTML functionality only with player extensions) and Luna Service API for device-specific controls like the DRM, device information etc. plus a library `webOSTV.js` which implements 2 classes with methods for easily accessing the Luna Service API calls. The exact extent of supported JavaScript, HTML and CSS implementations is documented in great detail, however, it still had not been outdated for the 2018 models with webOS 4.0 (as of July 2018). [7]

Figure 3.5: LG webOS main menu. Versions 1–4 are visually indistinguishable.

3.3.1   WebOS IDE

WebOS also offers a developer IDE and toolkit named *webOS IDE* [10]. It is very similar to Tizen Studio, the reason being that it is also based on Eclipse IDE. However, unlike Tizen, the webOS ecosystem uses no certificates in the development process. The different security measures will be explained along with packaging and installation. Another difference is that webOS IDE contains a manager of applications on connected devices allowing to individually install, uninstall and run applications.

The emulators are not included in the IDE but they are virtual machines for the Oracle VirtualBox. Important is to enable PAE/NX and VT-x/AMD-V options.

---

[9]https://chromedevtools.github.io/devtools-protocol/
[10]http://webostv.developer.lge.com/sdk/tools/ide/

3.3.2 Packaging

The packaging tool offers a minification step. It is recommended for performance improvement, however, during the development, it makes debugging almost impossible. The file which must be included in each webOS project root before packaging is `appinfo.json` [B.4]

3.3.3 Installation on Real Devices

Until 2016 webOS applications were installed from USB drives. The `.ipk` package was created as an output of the webOS IDE and then it had to be signed and repackaged with a DRM on webOS TV Developer portal as *USB App Test*. This is similar to what Samsung enabled on Tizen models 2016 and newer.

Now the installation is possible only through the webOS SDK. Special Developer Mode application had been added to televisions. It can be found in the standard application menu and it requires a login with a webOS TV Developer account. A reboot is done and afterwards, a synchronization passphrase is displayed. Later on, it will have to be entered when connecting to the device in the webOS IDE. Also, a timer showing for how long the developer mode will stay enabled is displayed and a button to extend this time easily is present.

If an `.ipk` package was created by the CLI packaging tool or obtained in any other way, it can be installed immediately without the need of importing it as a project and repackaging it before installation (unlike Tizen Studio).

Another improvement over the Tizen Studio is in the *Target Configuration* (connected devices manager). In there a list of installed applications is available with options to launch any of them remotely, also with debugger connected even if it was installed any time before by someone else.

3.4 Samsung Orsay

Orsay is the Samsung's first smart TV operating system, the predecessor of the Tizen OS. It was launched with the very first generation of smart TVs in 2011 and it had last appeared in the very lowest model group *15TV_ENTRY* of 2015 [22]. It was, luckily, not a very common device as it was suffering from vastly inferior hardware performance compared to the 2015 models with Tizen OS and its other features were fairly limited as well, for example, it did not even support the HbbTV plug-in.
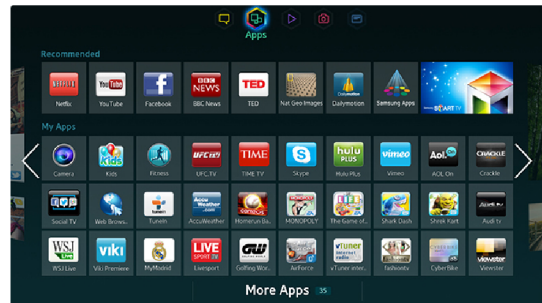
(a) Orsay 2011, 2012   (b) Orsay 2013, 2014, 2015

As for development, it only supports ES5 and partially CSS3. Sadly, unlike for the Tizen, the detailed web engine specification is not available. Although, it is safe to assume that if the Tizen 2.3 doesn't have support for something, neither does the Orsay.

3.4.1 Packaging

Orsay has a much more straightforward application development and deployment. It does not need to be signed, certified or packaged in any special/unconventional way. Only requirement are for `config.xml` [Fig. B.1] and `widget.info` [Fig. B.2] files in the root. The `config.xml` also links the application's icons.

These properties can also be conveniently exploited to further simplify the application development, mainly testing. A shell application can be created which loads its content from a predefined network location. That location can be even the actual working directory. This enables to launch the updated version without reinstallation and it even makes a hot module reload (*HMR*) possible — eliminating the need to even restart the application.

3.4.2 Installation

There are two ways to install the application to Orsay TV.

The first option is from a network folder. Typically an HTTP server (WAMP, XAMP, . . .) is used on a computer in the local network. In the root of the web server must be a folder „Widget" with a `widget.xml` [B.3] file in it. The file can contain multiple applications/widgets. The entire application needs to be zipped with `index.html` in the root of the archive. Then a special user „develop" (no password) needs to be logged in on the TV which enables am „IP Setting" and „Start User App Sync" in the menu, in the first one the IP address of the server needs to be entered. Then the „sync" can be done which will install (can take up to one minute depending on the size) all applications listed in the `widget.xml` file on the TV permanently.

The second way is to launch it from a USB drive. The `index.html` file has to be in a folder one level above the root, for example `F:\myAppName\index.html`. Rest of the project structure can be defined in any relative way in the `index.html`. The application should appear shortly after plugging in the USB drive. However, the application is not installed permanently and is only accessible as long as the USB drive is present.

3.5 LG Netcast

Netcast is the LG's first smart TV operating system, the predecessor of the WebOS. It was launched with the very first generation of smart TVs in 2011 and it had last appeared in 2014, then in the very lowest model group of 2015 (under the name *SimpleSmart*; similar to Samsung Orsay 2015 [3.4]) alongside the WebOS 1.x and WebOS 2.x.

A specific of GUI of Netcast applications is that a button Q.Menu must be implemented in any player the application might contain. It is specific for Netcast platform, they have the same HW button on the RCs, it opens a menu with some image „enhancements" settings.

3.5.1 Packaging

Similar to Samsung Orsay, there is no special package type for Netcast. The application is simply put into a special folder structure before installation. The only file that must be included in each Netcast project root before packaging is `manifest.xml` and set of icons.

3.5.2 Installation

Only option to install unpublished application on a Netcast TV is from a USB drive. The application must into a special folder structure so that the `index.html` would be located in `F:\lgapps\installed\<6-digit ID>\index.html`. After connecting a USB drive to a TV, icon of the USB drive will appear in the list of installed applications and when it is accessed, list of applications on the USB drive will appear.

3.6 Platforms of Marginal Interest

Sony had their own framework *CEB* for models 2012–2014. Later they switched to Android TV OS and there the JavaScript applications are now published on Google Play with the Android WebView wrapper used as their shell. The CEB can still be used in Android TVs but it adds an extra layer to the applications and as such is inherently less stable and worse performing. And of course, a native Java Android application is the third option.

Blu-ray players also typically feature the same OS as Smart TVs of the same vendor. They, however, never contain a DVB tuner, therefore the HbbTV is not a concern there. Furthermore, their market share is negligible, numbers sold are ever declining, customers sparsely want their applications to be released for these players, as the tiny user pool size simply does not justify the increase of development cost. Even though the OS is basically the same as the TV, behaviour in some scenarios can differ vastly and require lengthy debugging and patching.

Streaming sticks or set-top boxes like Roku TV, Google Chromecast, Apple TV or Amazon fireTV use either Android or some closed proprietary OS and do not allow to run or install JavaScript applications. These kinds of devices lack a DVB tuner too and so the HbbTV is not present either.

Satellite, Cable or Terrestrial DVB Set-top boxes are considered only in the context of HbbTV platform in this work. They mostly feature a closed proprietary OS without an option to install (complete lack of distribution system) and run JavaScript applications.

Gaming consoles (PlayStation 3 and 4, Xbox 360 and One) are also considered here, although JavaScript applications are mostly just for VoD services there. Rarely for games as they have some far better performing options for implementation of those. Neither in these can we find a DVB tuner, therefore they cannot run any HbbTV applications.

Chapter 4

Existing Solutions

For the relevant platforms, there is currently no kind of a neither complex nor partial nor single-purpose performance-measuring tools (also known as *„benchmarks“*) in existence; not even a simple test of the implemented version of ECMAScript [2]. However, there are plenty of browser performance benchmarks and tests available. Those can be a valuable source of inspiration to the composition and content of this benchmark application.

4.1 jsPerf

jsPerf[1] — „JavaScript performance playground“ — is an advanced tool for JavaScript *snippets*, algorithms, or scripts performance profiling. It is an open-source project developed under an MIT License, thus making it possible to use it or its parts in a simplified form in here. This is not focused on measurement of performance differences between different platforms or browsers.

4.2 Octane 2.0

Octane[2] is a complex browser performance benchmark. It consists of 17 different tests ranging from core JavaScript features, bit and math operations usage in cryptography, regular expressions, memory demanding operations with large objects and arrays to ray tracer, emulators, virtual machines and compilers transpiled to JavaScript. It is an open-source project under a BSD License, thus making it possible to use it or its parts in a simplified form in here.

4.3 Dromaei

Dromaei[3] is Mozilla JavaScript performance test suite. This benchmark has a vast pool of test cases, the combined running time of all of them is ≈ 30 minutes. It is an open-source project with a license enabling it to be used, modified, published, distributed, sub-licensed, and/or sold. Thus making it possible to use it or its parts in a simplified form in here.

---

[1] https://jsperf.com/
[2] http://chromium.github.io/octane/
[3] http://dromaeo.com/

4.4   BaseMark

BaseMark [4] is a highly complex benchmark, not focused on the browser's rendering core and JavaScript engine but the overall aspects of a device (such as the battery drain test). I found it not usable for this work since it is as closed commercial product and it is difficult to see even what types of tests it uses without a reverse engineering.

4.5   Acid3

Acid3 [5] is arguably the best known web test that checks a web browser's compliance with elements of various web standards and rates it with a score up to 100. It consists of 6 „buckets" [27]:

1. DOM Traversal, DOM Range, HTTP

2. DOM2 Core and DOM2 Events

3. DOM2 Views, DOM2 Style, CSS 3 selectors and Media Queries

4. Behaviour of HTML tables and forms when manipulated by script and DOM2 HTML

5. SVG, HTML, SMIL, Unicode, . . .

6. ECMAScript

   This test does not have a license stated, therefore their adoption will be avoided. Also, it only tests features support but does no performance measurements. Nonetheless, it is still strongly inspirational as many of its features intersect with the proposed test set [2.1.5].

4.6   BrowserBench

BrowserBench [6] consists of several benchmarks: JetStream, ARES-6, MotionMark, Speedometer each testing performance in different type of tasks. The tests include ES6 features, graphical animations, responsiveness (DOM changes presumably) and various advanced tasks (cryptography, raytracing, regular expressions, base64 coding, n-body simulation, . . .). Most of the source codes are available but their licensing varies, some are owned by Apple Inc. and their usability would need more thorough legal examination. However many, especially in the JetStream part are relatively easily reproducible, for example, the base64 encoding test will be included already in the first stage [6]. The most interesting one, MotionMark, which tests the DOM and Canvas graphical performance is unfortunately not open-source.

---

[4]https://web.basemark.com/
[5]http://acid3.acidtests.org/
[6]http://browserbench.org/

4.7 HTML5test

HTML5test [7] is, like the Acid3, not a performance benchmark but a test of HTML5 and ECMAScript support. It includes the very latest features such as ObjectRTC [8]. As it does no kind of performance measurements, it is of little value here.

4.8 Suitest

Suitest [9] is not a benchmarking tool. It's a comprehensive set of hardware and software tools for automatized application testing. The most unique part is the hardware allowing to connect multiple IR RC [10] transmitters (placed directly at individual devices). This allows emulated and scripted IR RC commands to be used to control the applications. It is done through a fairly simple GUI, making it possible even for testers without programming knowledge. It also provides a way of inserting a DOM inspector into the tested applications. Thanks to that, real-time internal states can be accessed and compared with expected values. It doesn't contain any JavaScript debugger. It allows massive parallelization and speed-up of a big part of the highly time-demanding quality assurance process. However, it proves close to none information about the application or device performance.

---

[7] http://html5test.com/

[8] https://ortc.org/

[9] https://suite.st/features.html

[10] Infrared Radiation Remote Control

Chapter 5

Design of the System

The standalone benchmark [2.1.5] will be an application built on the Mautilus Smart TV SDK^{TM} [5.2.1]. It can later be relatively simply packed into a single component that will be includable in any other application using the Mautilus SDK. Such library form was mentioned in the solution draft by the Mautilus company. [2.1.5]

5.1 System Architecture

A *scene* is a name for React component [5.2.2] which is loaded and rendered directly by the router. In this case, it is resolved based on the hash part of the URL. Simply put, `Router.go('`*path*`')` activates and inserts the scene which is linked with the *path* into the DOM a removes all others. The first scene is named Home. It contains device information and button to start the benchmarking. Each individual test will be a separate scene. None of the tests will allow any user interaction except for closing the application. At last, the Results scene will display the overall benchmark score, partial tests scores and it will allow to post them.

There are multiple approaches (or patterns) to benchmarking in of JavaScript applications [1]. The two basic and simplest patterns will be described here. First one is measuring the execution time of the tested code or several iterations of it. Naturally, a lower result here means higher performance. The problem is that when the test length approaches the smallest unit of time resolution in a browser, the results become useless. That can be avoided by designing the tests complicated enough so that their execution will take long enough even on faster systems in the future. The second pattern is reversed approach — setting a fixed time a counting how many iterations of a test were executed within that time period. This, on the contrary, requires the single tests iterations to have short enough execution time. It must be a small fraction of the total execution time, otherwise, the resulting score (iterations count) will not have good informational value when comparing similarly performing devices. In this case, faster devices in the future are not a concern, but the slow ones are. As it was suggested already, low scores do not allow an accurate comparison. Besides these two basic approaches, the benchmarking can be done adaptively, the test complexity can be dynamically changed between iterations based on the earlier performance to achieve consistent total execution time or some other criteria. This understandably increases the overall complexity of the benchmark framework and calculations of the results too.

I selected the first mentioned pattern for this thesis — measuring the execution time of static tasks and using it as the test score. The reason is its simplicity (with the time possibilities in mind) and better suitability for low-performing devices.

Internal collecting of the results is done through the Redux store [5.2.2]. Reducers and actions are created for posting the results. They are imported in each Test scene and called at the end of each test. They will be accessed at the end from the Results scene via selectors.

The results will be sent to Google Analytics. That is the simplest and most universal way of data gathering for statistical purposes. The module for Analytics will be prepared. The results can be optionally posted from the Results scene.

5.2 Used Technologies

This section lists all significant technologies utilized in the solution application.

5.2.1 MAUTILUS Smart TV SDK™

This SDK, developed by the Mautilus company, has the goal of allowing the developers to write only a single application which will be runnable on any platform. It achieves it by creating an overlay over the APIs specific to each platform / OS and providing one own uniform API which the application will use regardless of the platform it is running on. This principle is illustrated in Figure 5.1 . Also in case of this benchmark those properties of the Mautilus SDK will serve as basis ensuring the multi-platform compatibility.
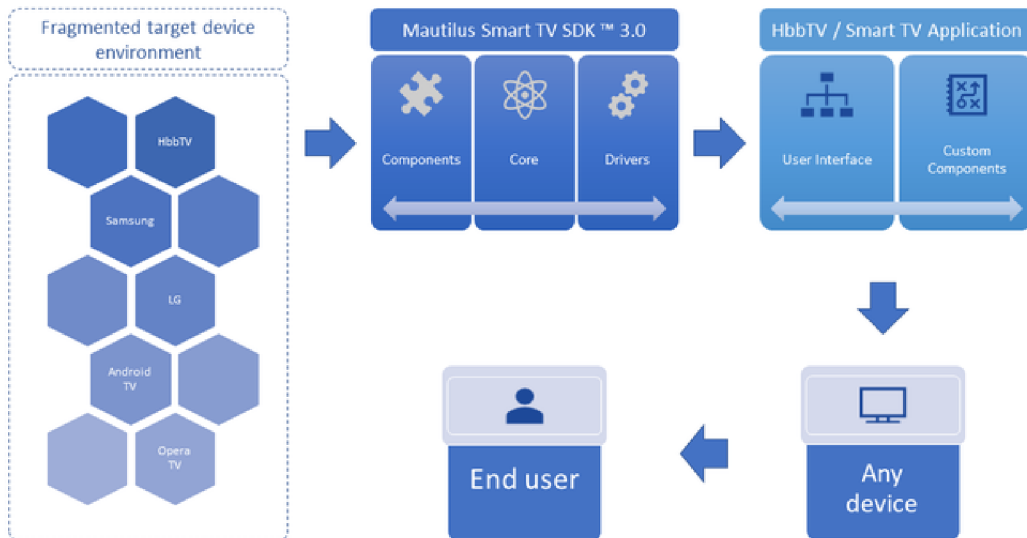
Figure 5.1: Basic high-level schema of the Mautilus Smart TV SDK™ 3.0 principles. [11]

The interlayer illustrated in Figure 5.2 is implemented and available to the applications as `Device` class. This class contains the intersection of sets of methods from all vendor-specific drivers / APIs. Those are to a great extent similar in the functionalities provided. When the SDK is initialized during the launch of an application, the device and platform are detected. The detection is done in the JavaScript runtime environment from the User-Agent value in most cases. At last, the driver for the detected platform driver is loaded. Respective `Device` class' methods are initialized or overridden by those from the driver which are using platform's native API methods.

There are other classes than the `Device` for different more specific drivers but they all work in a similar manner. These include Persistent storage drivers, DRM (*Digital Rights Management*) drivers, Controls, Inputs (SW keyboards, text fields, . . .), and others. The DRM are typically used for secured video streaming in paid SVoD (*Streaming* or *Subscription Video on Demand*) services, the most used are Microsoft PlayReady and Google Widevine (has *Classic* and newer *Modular* variants). The Mautilus Smart TV SDK^{TM} also contains a set of commonly used application components and modules, such as the internal keyboard, generic player UI, subtitles, user accounts and others.
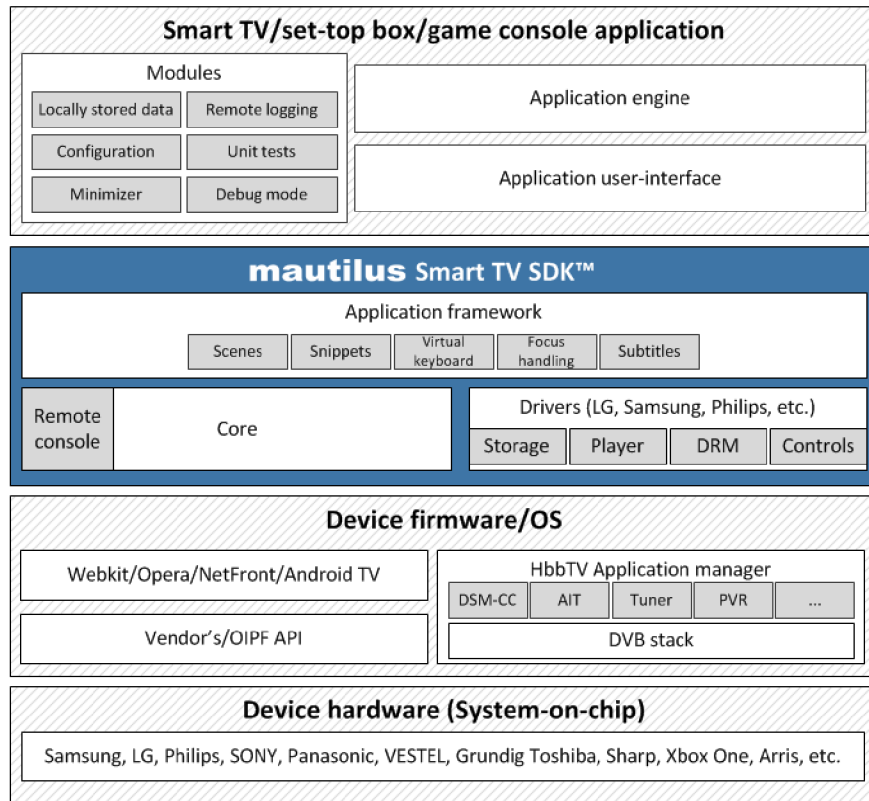

Figure 5.2: More detailed schema of the Mautilus Smart TV SDK^{TM} architecture. [13]

The Mautilus Smart TV SDK^{TM} 3.0 also comes with the *StoryBook*. The StoryBook is an application built on the SDK and it is a catalogue of components previously developed as part of various projects available for immediate use in applications. Some of these (for example the image carousel component) will be used in this benchmark too.

Enhancement of the SDK's platforms support is not the subject of this work and shall be left to the team of developers assigned to it. As of December 2017, the Mautilus SDK 3.0 is still in development, officially unreleased, and fully supports only Samsung Tizen, Samsung Orsay, LG WebOS, LG Netcast and HbbTV. Other drivers that were available in the Mautilus SDK 2.x are being or will be ported in near future. Neither the documentation for version 3.0 is completed. Documentation for version 2 is available though [14]. MAUTILUS SmartTV SDK^{TM} is available under BSD license [1]. Most of the technologies used are determined by this SDK. It is implemented as a React application and therefore using ECMAScript 2015 (*ES6*).

---

[1]Available at: https://github.com/mautilus/sdk/blob/master/LICENSE.TXT

5.2.2 React.js

React.js is a JavaScript library for building dynamic interactive user interfaces. The views are created declaratively for each state of the application and React very efficiently updates and renders just the minimal subtree of components affected when the data changes. This approach by letting the view updates be executed implicitly and automatically in the background by the React's logic, of course, makes your code more predictable and easier to debug.

React views are built from encapsulated components that manage their own state and these are then composed in multiple levels to make highly complex user interfaces. The component's logic is written in JavaScript (instead of templates), which allows to easily pass rich data through the application and keep its state out of the DOM.

React applications are written in ES6 [2] version of JavaScript, heavily utilizing classes and extensions, module exports and imports and other new capabilities of ES6. Custom components are classes extended from the `React.Component`. Each of them implements a `render()` method that takes input data and returns what to display. The input data that is passed into the component can be accessed by `render()` via `this.props`. The `render()` return value is either a `null`, string or it is defined in a *JSX* syntax. Usage of JSX [5.2.3] can be avoided by using a significantly more complex raw JavaScript code, however, this would be an inherently inefficient method. An imported component class becomes accessible and usable as a JSX tag which can be a part of the return value of `render()` method and that is exactly how the tree-like hierarchical component structure is created.

Pure Component

Pure component does only shallow comparison between `prevState` and `nextState` which generally results in fewer re-renders. This provides an option to implement/override custom highly optimized `shouldComponentUpdate()` methods leading to a speed up, especially in case of huge/deep state and props objects.

State

Components have not only the properties (`this.props`) but also State (`this.state`). Properties is a read-only object passed down when invoking a component. State is an object set by the component itself from inside by its own methods. The state object should never be changed directly (practice referred to as „*mutation*") but via the `this.setState()` method. It should be treated as if it were immutable. Thus calling (`this.setState()` does not always result in component re-render if the state was not changed or, in case of pure components [5.2.2], if the change was too deep.

Component Lifecycle

By default, when a component's state or props change, the component will re-render. If the UI depends on some other data than state (data dependencies from `props` should be copied to the `state` in `componentWillMount()`), re-rendering can be done manually by calling `forceUpdate()`. This will cause `render()` without previous `shouldComponentUpdate()`. However, for all child components normal lifecycle methods will be called, including the `shouldComponentUpdate()`. Nonetheless, React will only update the DOM if the HTML output changed. Normally, usage of `forceUpdate()` should be avoided. All updates should

be done, as intended, based on data changes in `this.props` and `this.state`. Fully acceptable way to affect (and/or optimize) updates is by using Pure Components [5.2.2].

Redux

Extremely simply put, Redux brings a `store` object which can be considered as a global state of the application. The most common pattern for store usage is the combination of reducers, actions, and selectors.

Reducers are functions which create and update parts of the store based on received (by parameters) action type and new data to be stored. Similar action types are grouped in reducers. They are not called directly from components. **Actions** are functions which dispatch a single event. Store calls a reducer function based on that event. Actions are called directly from the components. **Selectors** are functions which return pre-defined part of the store.

There is also a quite common Connect pattern. This creates a wrapper component (called *container*) mapping the actions and selectors onto `props` of the wrapped component. This is a step further towards having all updates triggered purely by data changes without creating any listeners or callbacks from the component itself.

React Motion

JavaScript animations tool with some unique properties. Surprisingly, experiments have shown that its performance is far superior to CSS3 transitions [5.2.3] in certain cases. Those are when multiple style properties (for example position and dimensions) of one element or multiple elements are being animated. It was often the case with CSS3 that these parallel atomic animations were poorly synchronized, leading to an apparent offset between their begins and ends. That is an unacceptable visual glitch for the end-users. The React Motion does not suffer from this at all.

React Performance

React used provide a set of low-level methods for performance profiling. They came within the `react-addons-perf` package (`Perf` class), however, they were deprecated in version 16 of React. They official recommendation [2] Chrome Performance tab. Another highly useful and popular React development tool is their Chrome extension [3] which brings a React's virtual DOM inspector and highlighter of components' updates.

Bluebird

Bluebird is a JavaScript Promise library. Promises have been chosen for the Mautilus SDK as the best option of asynchronicity handling and chaining and Bluebird was found to be the best Promise library, mostly because of its closeness to the native implementation of them. The async-await [4] was only standardized in ES7 (2017) [2] which is not yet being used in the Mautilus SDK build configuration. The update of babel [5.2.4] configuration to ES7 support can be expected in the near future.

---

[2] https://reactjs.org/docs/perf.html

[3] https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi

[4] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function

5.2.3 HTML5

Latest revision of the HTML [26] is supported in latest systems such as Tizen [3.2] and webOS [3.3]. However, due to the desired universality and backwards compatibility also promised by the use of Mautilus SDK [5.2.1], it needs to be transpiled to older, better supported ES5 [2]. This will be achieved by using of the Babel [5.2.4].

Canvas

Canvas is an HTML 5 element (`<canvas>`). It comes with a JavaScript API (*context*) for 2D drawing. It is also used by WebGL (also via (*context*) API) as a rendering target for GPU-accelerated 3D graphics.

Recently a bug concerning the HTML5 canvas was discovered in Samsung Tizen 2017 firmware from September 2017. Attempting to render a DOM containing a canvas element resulted in an application crash. This had already been fixed by a firmware update.

CSS3

Main interests in this work will be the Animations, Transforms (mainly the effects of 3D Transforms on the merging of the rendering layers and impact of that on performance), Transitions and Flexible Box Layout Modules [15]. Also, 3D Transforms and implicit grouping of elements into layers might have a significant influence on the performance.

JSX

JSX is a syntax extension to JavaScript. It is highly recommended to be used with React to describe the looks and internal composition and structure of the UI. JSX brings a sort of templating language capabilities into JavaScript while maintaining its full power.

Example of a variable declaration: `const element = <h2>Sample text</h2>;`. It is neither a string nor HTML. The JSX creates React *elements* which is a data type introduced by React.

WebGL

„WebGL is a cross-platform, royalty-free web standard for a low-level 3D graphics API based on OpenGL ES, exposed to ECMAScript via the HTML5 Canvas element. It stays very close to the OpenGL ES specification, with some concessions made for what developers expect out of memory-managed languages such as JavaScript. WebGL 1.0 exposes the OpenGL ES 2.0 feature set; WebGL 2.0 exposes the OpenGL ES 3.0 API.
WebGL brings plugin-free 3D to the web, implemented right into the browser. Major browser vendors Apple (Safari), Google (Chrome), Microsoft (Edge), and Mozilla (Firefox) are members of the WebGL Working Group.“ [24]

Developing each application with a bare WebGL would, however, be unfeasibly inefficient. There are a handful of WebGL libraries, SDKs and frameworks. All those that will be named below also feature an extensive collection of demos and examples some of which will be used in this benchmark application. Those will be possible to utilize as a base for the WebGL performance tests.

WebGL Libraries and Frameworks

Arguably the best and most widely used WebGL library is the **THREE.js**[5][6]. One of the reasons THREE.js is so popular is because it is so easy to get into for novices in the 3D graphics programming. It has an exceptionally good documentation too.

Physijs[7] is a physics engine plug-in for the three.js. **BabylonJS**[8] is another 3D engine based on WebGL. Its particularly interesting feature is selectable WebGL 1.0 / WebGL 2.0 renderer. It includes its own physics engine. **PixiJS**[9] is supposedly the best performing 2D WebGL renderer. A custom demo (developed as a project in course *Advanced Computer Graphics*) utilizing dynamic 3D fractals will be used here too.

5.2.4 Other Development Tools

This subsection will list technologies and tools which are important to mention because they were used extensively during the development but that are not directly used in the resulting application.

Node.js

Node server and npm are requirement of the Mautilus SDK [5.2.1] for the application building and running the development servers. For this project Node 6.9 is recommended. Newer versions may cause compatibility issues to some of the external dependencies used.

Babel

Babel[10] is a JavaScript compiler. Typically, the code is written utilizing all the features from the latest ECMAScript versions [2]. They can, however, be sparsely supported by the browsers. Then it is transpiled into an older version of ECMAScript with a better support across browsers. Source version pre-set in this case is ECMAScript 2015 (ES6) and the target version is ECMAScript 5.1. The babel outputs from higher version pre-sets have not yet been properly tested on real devices. Features of the ECMAScript 2016 (ES7) and ECMAScript 2017 (ES8) need to have explicit polyfills.

Webpack

Webpack[11] is a front-end module bundler that works great with the latest modern front-end workflows and stacks including Babel and React.js among others. It uses a module system (introduced in ES6 [2]). Webpack works best with npm (Node Package Manager).

Important analytic tool to mention here is the Webpack Visualizer[12] which interactively displays the bundle composition, hierarchy, relative and absolute sizes of the individual nodes and modules [Fig. 5.3]. Results of the analysis are exported as an HTML file. This

---

[5] https://threejs.org/
[6] https://1stwebdesigner.com/3d-javascript-libraries/
[7] https://github.com/chandlerprall/Physijs
[8] https://www.babylonjs.com/
[9] http://www.pixijs.com/
[10] https://babeljs.io/
[11] https://webpack.js.org/
[12] https://github.com/chrisbateman/webpack-visualizer

tool is available in form of an npm package [5.2.4] and it uses the famous JavaScript library D3.js [13] for the interactive sunburst graph visualisations.
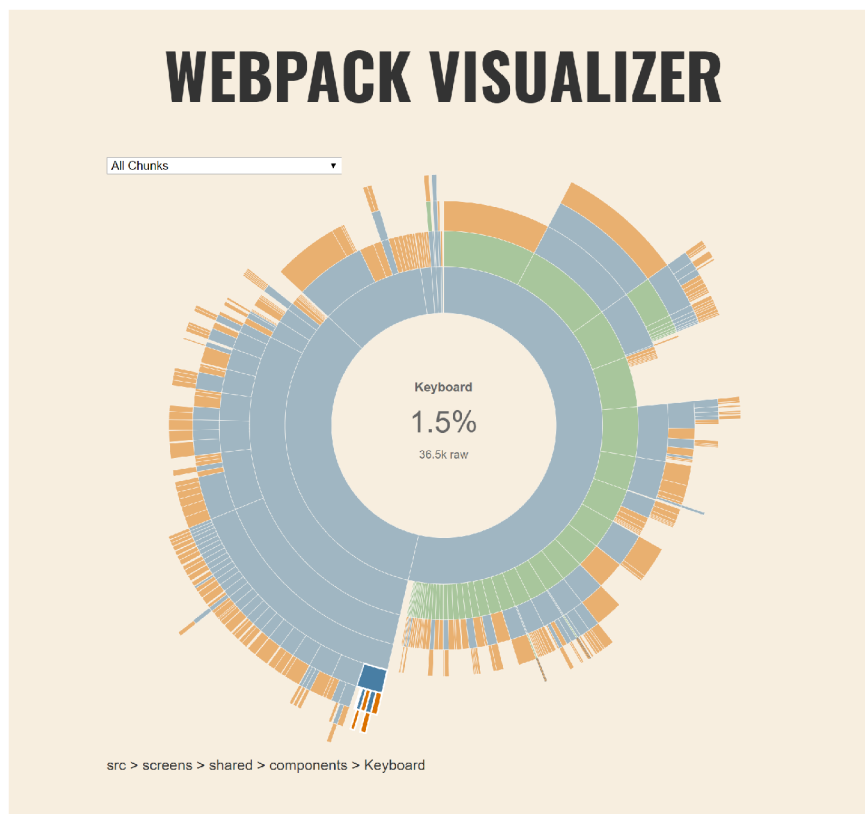

Figure 5.3: Output of the Webpack Visualizer tool. Component Keyboard is focused by mouse hover. Its subcomponents are visible in the graph, its placement in the project structure is visible both in the graph and underneath. Its percentual and absolute size is shown (including its subcomponents).

---

[13]https://d3js.org/

Chapter 6

Proposed Set of Tests

This chapter describes which tests had been selected to be included in the benchmark, the reasons for having them will be explained as well as their structure and composition.

These tests were selected so that they would represent the majority of the highly performance-demanding operations commonly found in all TV applications. First big group of such tasks are undoubtedly the visual effects and animations of user actions, second are transformations of large data structures. Typically, content catalogues of SVoD services are hundreds of kilobytes large JSON files. Those need to be split and remapped to fit the application navigational structure and UI specific for these platforms. References at each item lead to the detail of its implementation.

- Framerate during animation of a single DOM element's (`<div>`) style (position, size and opacity) by React Motion. [7.2.2]

- Framerate during animation of a single DOM element's (`<div>`) style (position, size and opacity) by CSS animations. [7.2.4]

- Test of Canvas element and its methods support.

- Framerate during the animation of a single Canvas element's (circle) position and size. [7.2.1]

- Test of WebGL support.

- Framerate of animation of a primitive 3D WebGL scene containing only a rotating cube. [7.2.5]

- Framerate of animation of a 3D WebGL scene containing increasingly complex fractals (Menger sponge or Sierpinski pyramid). [7.2.5]

- Base64 encoding and decoding [7.2.3]. This MIME content transfer encoding is very commonly used for data transitions in all kinds of services. It allows encoding any binary data as a plain alphanumerical string. Testing is done on multiple large (8 kB–16 MB) strings.

- Array flattening [7.2.6]. It is often required to have data in flat arrays. Those are such that none of their items is an array. This operation simply recursively moves all items of nested arrays into the parent array.

- Objects deep cloning and merging [7.2.7]. A property of object copying in JavaScript is that its properties, which are references to other objects, are copied as they are. That means they are still referencing the children of the original object. Therefore, it must be differentiated when the copying is recursive (*deep*). Merging is performed similarly. Two objects containing properties with identical keys will be merged by simply overwriting those properties in the destination object. Whereas during recursive (*deep*) merge, if the values of the matching properties are objects, those are recursively merged too. Native `Object.assign()` [1] method added in ES6 also does only shallow merge (can be used for shallow clone too).

---

[1] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/assign

Chapter 7

Implementation Details

7.1 Project Structure and Development Stack

The benchmark application is based on the Mautilus Smart TV SDK [5.2.1] which predetermines its project structure. The empty project contains for example ESLint configurations[1], flow[2] configurations, webpack [5.2.4] configurations for building and packaging of the application. The requirements and commands are listed in the Manual appendix [A.1] and in `/README.md` file.

Mautilus SDK contains in its core also a set of *High Order Components* (*HoC*). Which simply means a wrapper that injects some common methods and properties. These are most notably used in the `Focus Container` and `Focusable` HoCs. They are Redux / store based, they come with actions, reducers and selectors and they are the base of focus handling [2.1.1] in applications. The SDK also comes with many pre-made low-level components. They can be browsed in the StoryBook application [A.2]

All of the scenes are located in `src/scenes/` folder and the router is configured in `src/index.js`. The `src/index.js` file is the react application's root. It is the first executed script, everything is initialized there. As it was mentioned already, `HashRouter`[3] is used in this application. The application is also prepared for translations via the i18n module[4] and for device-independent time. That can be required for results collecting and it would be achieved by fetching a timestamp from a selected server and using a calculated time offset from the local device time later on. The application is also capable of displaying a throbber[5], notifications and dialog[6] windows.

The `Home` scene has a single button that starts the benchmarking. There is also a component `Info` containing information about the device it is running on. Those data are available from the `Device` driver.

The experimenting with different animation techniques and various methods of score calculation took up by far the biggest part of development time. Most of the experiments ended up unsuccessfully and could not be included in the final version of the application.

---

[1]Based on the most popular ESLint config for React projects *airbnb*: https://www.npmjs.com/package/eslint-config-airbnb

[2]Static type checker for JavaScript: https://flow.org/en/

[3]https://github.com/ReactTraining/react-router/blob/master/packages/react-router-dom/docs/api/HashRouter.md

[4]https://www.npmjs.com/package/i18n

[5]Indefinite progress animation, typically during scene loading or data downloading.

[6]Dialog window is an overlay requesting user action.

In the final version, the test scores are the measured running times of the test cases. They are collected to the Redux store. There is a reducer (`/src/reducers.js`) and action (`/src/actions.js`) for saving the individual results. Each test scene does this at the end of a test run.

Result scene displays a final score, partial scores, it has buttons for running the tests again and for posting the tests. It uses a selector (`/src/selectors.js`) for accessing the test results. The scene can be extended in many ways including various graphs, comparison with other downloaded results, posting results to other servers etc. Centralized collecting of the statistics may be done most simply and universally via Google Analytics. This driver is implemented but it was kept very general for multiple reasons. There is no demand for it yet and also because the legal side of such data collecting would have to be examined. The logs format can be adjusted in `/src/analytics.js` and the Google Analytics Tracking ID („UA-*xxxxxx*-*x*") can be set in `/config.js`.

7.2   Individual Tests

This section will describe implementational details of individual tests. Running length of each test was tuned to be between 250 ms and 3 s on a desktop PC.

7.2.1   Canvas rendering

The native `window.requestAnimationFrame()`[7] function was used for the timing of animation step size. This is the simplest way of ensuring maximal possible rendering frame rate (FPS — frames per second) without any unnecessarily frequent updates. A callback to animation-iterating function is scheduled using the `window.requestAnimationFrame(cb)`. The browser executes the registered function as soon as possible. Then, it needs to be scheduled again, typically at the end of the animation-iterating function itself. The test is ended after rendering 240 frames.

7.2.2   React Motion Animations

A `<div>` element is created. Its style (top, left, width, height, opacity) are declared within the `Motion` HoC (*High Order Component*). That pattern creates a dependency on the rotation angle stored in `this.state.anim.A`. Then any change of `this.state.anim.A` causes a component re-render with the style being animated by interpolation guided by the `Motion` HoC. The `this.state.anim.A` is updated with interval of 1 ms using the `setState()` method of `React.Component`. However, React internally schedules the state updates in groups if they are done milliseconds apart. That means not every single `setState()` causes one update of `this.state`. In this case, about 250 updates per second are performed in desktop browser.

7.2.3   Base64 Encoding and Decoding

Random string 8192 characters long is generated. Then it is encoded to Base64 and decoded back. Native JavaScript functions `atob()` and `btoa()` are used for this. There are 12 iterations of this with the string length being doubled each time. Last one is thus 16 MB large.

---

[7]https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame

7.2.4 CSS Animations

This test could not be implemented because after an extensive research I reached a conclusion that there is no way to measure rendering frame rate of a CSS animation. The CSS3 animations have configurable duration and there is an event 'animationend'. The frame rate can be seen in desktop browsers' developer tools, but it cannot be accessed from a JavaScript application. For example, MotionMark [4.6] uses a rendering loop guided by window.requestAnimationFrame() and changes element's .style.transform based on the last frame duration. The element then re-renders with the updated style in a single step. That is, however, not a CSS animation but a JavaScript one.

7.2.5 WebGL 3D rendering

WebGL test could not be finished in time because of unexpected difficulties with the integration of the THREE.js library into a React application.

7.2.6 Array Flattening

Native Array.prototype.flat() is not used because it is still only a specification proposal with experimental / candidate status and rather sparse support [8]. Lodash flattenDeep() [9] is used instead. Array is generated predictively in src/screens/Test03/data.js, reaching depth of 12 and resulting in flat array 495 000 elements long.

7.2.7 Object Deep Cloning and Merging

As there are no native functions for deep cloning or merging of Objects in JavaScript [2], lodash cloneDeep [10] and merge [11] were used. Three testing object were generated using https://www.json-generator.com (reaching 1.17 MB) and they were further manually adjusted. They are imported from src/screens/Test04/data.js, cloned and merged. .

7.3 Specific Programming Principles

In this section, some programming practices specific for development for these platforms will be presented.

Any DOM access is slow and should be done minimally. The most efficient practice to achieve this is to be pre-loading and saving references to DOM elements which will be used multiple times later. Query selectors (for example by classes) are especially slow and should be avoided completely. It is possible to use nothing but Document.getElementById() [12]. Excessive usage of HTML elements IDs is therefore necessary. jQuery library only worsens this compared to the native query methods because it adds the backwards compatibility for old browsers which slows the functions even further. Likewise, the React refs [13] should be avoided for both performance reasons and because their usage is viewed as a bad programming practice.

---

[8]https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/flat

[9]https://lodash.com/docs/#flattenDeep

[10]https://lodash.com/docs/4.17.10#cloneDeep

[11]https://lodash.com/docs/4.17.1#merge

[12]https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementById

[13]https://reactjs.org/docs/refs-and-the-dom.html

Usage of minimizers as a part of the application build chain is highly recommended. Not only as a sort of source codes protection but also as an easy way to gain a slight performance improvement. Many allow different optimization priorities (such as memory usage, code size, ...). Those options can be useful especially in HbbTV where the application is downloaded not only once, although the TVs use caching. However, this can also cause some troubles on devices with incorrect JavaScript parser implementation. For example, many DVB-S set-top boxes were found to be crashing during parsing of the source code because it was containing Object keys without quotes. The same need for minimal size goes for other assets, especially images. Services like TinyPNG [14] come in very useful here. Unfortunately, no device was yet observed to properly implement incremental loading of interlaced PNG or JPEG images.

Testing on emulators is generally not very useful because they mostly do not reflect the limitations of the real devices. This difference can be seen for example in the Samsung's specification sheets [19] (no one else publishes these specifications of emulators).

HMR (*Hot Module Reload*) is a tool which radically improves development efficiency. However, sometimes it was not possible to be used because the application used a customer's account security system. The customer refused to allow on their servers login from the unknown domains or user-agents (Smart TV platforms), therefore the application had to be run redirected within a shell application and that, of course, didn't react on HMR messages from the webpack server.

7.4   Issues Encountered During the Development

This section will list some but not all significant platform- or device-specific bugs and limitations discovered during development for these platforms. Selected are especially those directly contradicting expectations set by device specifications, references and documentations.

7.4.1   JavaScript

- `Function.prototype.bind()` is not fully supported throughout all platforms (most notably 2011 and 2012 Samsung Orsay and LG Netcast, in both native Smart TV and HbbTV environments). Even though Babel [5.2.4] transpiles the source codes to ES5.1 [2], these cases must me polyfilled explicitly.

- Multiple Philips TVs and set-top boxes are ignoring HbbTV application destroy call (or do not implement it at all). The workaround for this is preventively calling `window.close` afterwards (if an application is ended before, it will never be executed)

7.4.2   Various

- *Some* SVG images are causing applications on Tizen 2017 to crash immediately after launching before even executing any JavaScript code, making it literally impossible to debug. It had not yet been identified what in the SVG specifically is the source of this issue. Therefore the only safe way is to re-test the application every time an SVG image is inserted.

---

[14]https://tinypng.com/

- Applications on Tizen 2017 crashed when attempting to render a canvas element. This had been fixed after several months by a firmware patch.

- The firmware updates can sometimes bring in new bugs. Example from Tizen 2016: Players stopped working when a combination of SmoothStreaming and PlayReady DRM was used. This was fixed in another patch several months later.

HbbTV players

By far the most troublesome component of all applications are the players, especially when used with DRM or in HbbTV. In HbbTV players the absolutely least reliable and properly working thing is the seeking. Problems are most frequently occurring when seeking to positions close to the end or beginning of a video. On some, especially older devices, the issues can range from simply ignoring and not executing the seek, to application freezing or crashing. It is safe to say that manufacturers outright violated the OIPF and HbbTV standards [3] [16], implemented the HbbTV components and functions incompletely or incorrectly and didn't patch them even after many years.

- Philips with Android 2015 — ignoring seek do the end of a video.

- Philips with Fusion 2 2012 — application crashes when seeking to the end of a video and stops the playback after seeking to other position.

- Philips with MTK 2014 — player stops the playback after seeking to any position.

- Panasonic 2013–2015 — application crashes when seeking to the time 0.

- LG Netcast 2012–2014 — calling pause or stop on the player while having no video source attached caused the application to freeze.

- LG Netcast 2015 (also known as *SimpleSmart*) — random application crashes after any seeking.

- Samsung Tizen 2015 models suffer from failing to start a playback even though the player state changed accordingly; a workaround had to be implemented that watches whether the video position also really changes and restarts the playback until it actually does.

Chapter 8

Evaluation of the System and Results

In this chapter the real-world applicability, usefulness and target audience of the resulting tool will be discussed, as well as the results collected from it during testing.

## 8.1 Applicability Evaluation

This technical report itself can serve as an encyclopaedia for developers, both beginners and seniors, crafting applications for these platforms all across the world thanks to its English localization. After all, they are the intended target users of the benchmark application too. The test set surely needs to be extended to cover more scenarios, but even in the current state, it can provide a valuable insight into how a given device performs relative to others.

8.2 Results Measurements and Comparison

During the development of this thesis, the Mautilus had conducted own testing and published the results [12]. Apparently, this test was based on the HbbTV game engine which for compatibility reasons included both Canvas and DOM renderers. The DOM renderer should serve as a legacy option but as the results suggest, even on the very recent generations it can hugely outperform the Canvas renderer. Results measured by my own more comprehensive benchmark were achieved on different devices than Mautilus had published. Therefore the relative performance differences cannot be cross-checked.

Two televisions were acquired for the testing: Samsung UE32M5572AU (2017 Tizen 3.0) and LG 32LK6200PLA (2018 webOS 4.0). It was, however, discovered that the webOS driver in Mautilus SDK doesn't work properly on this 2018 model. After a tedious debugging session I have determined that it gets stuck during the driver initialization but I was unable to fix the bug in time. As it was said earlier, the cross-platform compatibility is secured by the Mautilus SDK and extending or fixing it is not a subject of this thesis.

Both televisions used for testing were privately purchased. Unfortunately, the truly high-end models could not be tested at this time. The reason is simply the lack of means to acquire a high-end model as those are nonlinearly more expensive. The pricing is so also due to the fact that the high-end models are only manufactured in largest screen sizes. The older models are understandably not available for purchase any longer. Also, I no longer have access to the testing device pool of the Mautilus company.
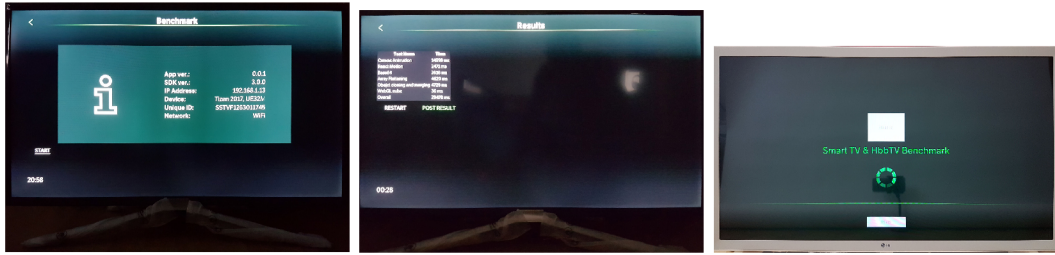
(a) Home Scene of the application on Samsung Tizen UE32M5572.

(b) Results Scene of the application on Samsung Tizen UE32M5572.

(c) Application stuck on the Splash screen because of malfunctional driver initialization on webOS 2018. LG 32LK62.

Figure 8.1: Photos of the benchmark application launched on real televisions.

The following table shows the average of 3 runs on each platform measured in milliseconds. The first platform is Google Chrome on a desktop with Intel Core i7-8550U CPU, the second is the same platform with 6× CPU throttling enabled in the Performance tab of developer tools, and the third one is Samsung UE32M5572AU TV.

Test Name	Chrome	Chrome (6× throttling)	Tizen 2017
Canvas Animation	1 669	1 706	15 063
React Motion	2 001	6 332	2 316
Base64 coding	349	2 978	2 666
Array Flattening	466	4 193	4 898
Object Merging	275	2 280	4 509
Overall	4 760	17 489	29 452

We can observe that the Canvas Animation (timed rendering of 100 frames) is limited to 60 FPS because of the usage of `window.requestAnimationFrame()`, even though it is GPU-accelerated and actual render probably take less then 16 ms. On the Tizen TV, it performs significantly worse, reaching only 7 FPS. This can be ascribed to an extremely low-power GPU. React Motion animation (timed rendering of 500 frames) does perform quite close on Tizen and the desktop as it is not highly GPU and memory demanding. Other tests are around 10 times slower on the Tizen, which can be attributed to their high memory demands.

8.3 Stability of the Results

The following table shows results of 3 runs in Google Chrome on a desktop with Intel Core i7-8550U CPU with iGPU. The scores are simply the durations of test runs in milliseconds.

Test Name	Run 1	Run 2	Run 3
Canvas Animation	1 609	1 681	1 701
React Motion	1 998	1 997	2 000
Base64 coding	376	342	350
Array Flattening	472	489	548
Object Merging	310	290	253
Overall	4 765	4 799	4 762

The following table shows results of 5 runs on Samsung UE32M5572AU TV. The scores are simply the durations of test runs in milliseconds.

Test Name	Run 1	Run 2	Run 3	Run 4	Run 5
Canvas Animation	15 082	15 025	15 135	15 111	14 998
React Motion	2 391	2 186	2 393	2 238	2 470
Base64 coding	2 639	2 558	2 649	2 594	2 616
Array Flattening	4 600	4 626	4 621	4 614	4 620
Object Merging	4 771	4 529	5 005	4 967	4 729
Overall	29 438	28 924	29 803	29 524	29 433

All tested platforms show significantly stable results. However, other older devices cannot be expected to behave similarly. As the devices are built on a relatively low-performance HW, they have very limited multitasking capabilities. This factor can be expected to be the main source of an instability of results achieved in a repetitive benchmarking. Other applications running in the background have the biggest influence. Their suspended state and thus CPU utilization is hardly estimable. Next variables are caching and memory occupancy possibly resulting in memory swapping. The difference in load time of applications between heir first and following launches is very well observable.

Also, it needs to be mentioned that devices which supporting some sort of multitasking also (in their native device APIs) typically distinguish between *killing* and *closing* the application. Closing means that the application will only be minimized and expectedly suspended. This gives users much 'snappier' experience, the device feels faster and more responsive. Not too surprisingly, the manufacturers encourage this in applications. Some (Samsung) even strictly demand this be correctly handled in order to pass their QA process [2.1.2]. Not killing the application, of course, means a possibility to switch back to it seemingly instantly and finding it in the state it was left in (again Samsung is extraordinarily strict in this matter while making it very uneasy to achieve the desired behaviour). While this application does not need to comply with those criteria until it will be distributed through the official application markets, the others still may and will influence the results of these tests.

Chapter 9

Conclusion

The goal of this thesis was to become deeply familiar with the development of applications for unusual platforms such as Smart TV, identify the biggest complications during development, design and implement a product which would improve some of the problematic aspects of development.

Analysis of both the problems and possible solutions was completed in full extent. I researched, analysed and described in a great depth the development processes and tools on the five major platforms, as well as the platforms themselves. Solution system was proposed and the implementational technologies were selected, studied, described in adequate detail, and I got practically acquainted with them. I assembled the set of tests with realistic time possibilities in mind. The draft of solution requirements provided by the Mautilus company served as a mere inspiration due to its vast extent which made it more suitable for a dedicated team of multiple developers. The benchmark application was implemented in a basic but fully functional form and tested on two major platforms (Samsung Tizen and LG webOS). Results have been gathered and evaluated along with their statistical stability. The biggest shortcoming is the missing test of CSS3 animations frame rate, as it was found technically impossible to access necessary data from a JavaScript application.

The application code is not 'polished' nor tested enough to be ready for release. It is a solid backbone allowing vast extendability mainly in terms of adding more tests into it. This technical report serves both as a manual to the application and as an introduction guide into the programming of applications for Smart TV and related platforms. Its value lies in its richness in practical experiences.

Possible future steps in development are:

- Refactoring the code and extending the test set before release.

- Finishing the GUI, for example adding a progress information during the test run.

- Making the benchmark exportable as a single component that can be simply used in other applications.

- Adding a module for downloading and displaying results posted from tests ran on other devices.

However, the most important thing will be ensuring development capacities for expansions and maintenance. That will most likely be done by an open-source release and possibly by the involvement of the Mautilus company which might have interest in this.

Bibliography

[1] Bynens, M.: *Bulletproof JavaScript benchmarks*. Personal blog. [Online; accessed 10-July-2018].
Retrieved from: https://mathiasbynens.be/notes/javascript-benchmarking

[2] Ecma International: *ECMAScript® 2017 Language Specification (ECMA-262, 8th edition, June 2017)*. Ecma Standards. [Online; accessed 2-January-2018].
Retrieved from: https://www.ecma-international.org/ecma-262/8.0/index.html

[3] ETSI, HbbTV Association: *Hybrid broadcast broadband TV Specifications*. HbbTV Resource Library. [Online; accessed 1-January-2018].
Retrieved from: https://www.hbbtv.org/resource-library/#specifications

[4] Herwig, B.: *Televizory Samsung pro rok 2018 nabízejí unikátní režim ambient a jsou připraveny i na HbbTV 2.0*. televizniweb.cz. [Online; accessed 11-June-2018].
Retrieved from:
http://www.televizniweb.cz/2018/04/televizory-samsung-pro-rok-2018-nabizeji-unikatni-rezim-ambient-a-jsou-pripraveny-i-na-hbbtv-2-0/

[5] LG Electronics: *Standard Web API*. WebOS TV Developer. [Online; accessed 27-December-2017].
Retrieved from:
http://webostv.developer.lge.com/api/web-api/supported-standard-web-api/

[6] LG Electronics: *Web Engine*. WebOS TV Developer. [Online; accessed 06-June-2018].
Retrieved from:
http://webostv.developer.lge.com/discover/webos-tv-platform/web-engine/

[7] LG Electronics: *webOS TV Web API Coverage*. WebOS TV Developer. [Online; accessed 29-January-2018].
Retrieved from: http://webostv.developer.lge.com/application/files/api_references/20170608_1496987509/

[8] Lladó, J. C.: *Developing a generic web application for Tizen TV: Case study*. October 2015. [Online; accessed 26-December-2017].
Retrieved from: https://www.researchgate.net/profile/Kruzkaya_Ordonez/publication/303345367_Actas-VI-CTVDI-IV-jAUTIcompressed/links/573dce2708ae9f741b2ff84e.pdf#page=139

[9] Matulac, J. S.: *Case Study of Tizen Operating System*. Faculty of Information and Communication Studies, University of the Philippines Open University, Laguna, Philippines 4031. January 2016. [Online; accessed 26-December-2017].

Retrieved from: https://www.researchgate.net/profile/Joemar_Matulac/
publication/292143531_Case_Study_of_Tizen_Operating_System/links/
56aa17dd08ae2df82166c290/Case-Study-of-Tizen-Operating-System.pdf

[10] Mautilus, s. r. o.: *HbbTV Development – How to Make an App Compatible on Every Device.* Mautilus Blog. [Online; accessed 4-June-2018].
Retrieved from: https://www.mautilus.com/blog/hbbtv-apps-development-how-to-make-an-app-compatible-on-every-device/

[11] Mautilus, s. r. o.: *Mautilus SDK 3.0 is here!* Mautilus Blog. [Online; accessed 27-December-2017].
Retrieved from: https://www.mautilus.com/blog/mautilus-sdk-30-je-na-svete/

[12] Mautilus, s. r. o.: *Mautilus SDK 3.0 is here!* Mautilus Blog. [Online; accessed 13-June-2018].
Retrieved from:
https://www.mautilus.com/blog/what-is-the-speed-of-your-tv/

[13] Mautilus, s. r. o.: *Mautilus Smart TV SDK^{TM}.* GitHub public repository. [Online; accessed 26-December-2017].
Retrieved from: https://github.com/mautilus/sdk

[14] Mautilus, s. r. o.: *Mautilus Smart TV SDK^{TM} API Documentation.* [Online; accessed 27-December-2017].
Retrieved from: http://smarttv.mautilus.com/SDK/#!/api

[15] Mozilla and individual contributors.: *CSS3.* MDN web docs. [Online; accessed 2-January-2018].
Retrieved from: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS3

[16] Open IPTV Forum: *OIPF Specifiacations.* [Online; accessed 1-January-2018].
Retrieved from: http://www.oipf.tv/

[17] SAMSUNG: *Media Specifications.* SAMSUNG DEVELOPERS. [Online; accessed 15-March-2018].
Retrieved from: https://developer.samsung.com/tv/develop/specifications/media-specifications

[18] SAMSUNG: *Security and API Privileges.* SAMSUNG DEVELOPERS. [Online; accessed 20-June-2018].
Retrieved from:
https://developer.tizen.org/development/training/web-application/understanding-tizen-programming/security-and-api-privileges

[19] SAMSUNG: *Smart TV General Specifications.* SAMSUNG DEVELOPERS. [Online; accessed 25-December-2017].
Retrieved from: http://developer.samsung.com/tv/develop/specifications/general-specifications

[20] SAMSUNG: *Smart TV Web Engine Specifications.* SAMSUNG DEVELOPERS. [Online; accessed 25-December-2017].
Retrieved from: http://developer.samsung.com/tv/develop/specifications/web-engine-specifications

[21] SAMSUNG: *Tizen TV Web Device API Reference*. SAMSUNG DEVELOPERS. [Online; accessed 25-December-2017].
Retrieved from: https://developer.tizen.org/development/api-references/web-application?redirect=/dev-guide/3.0.0/org.tizen.web.apireference/html/device_api/tv/index.html

[22] SAMSUNG: *TV Model Groups*. SAMSUNG DEVELOPERS. [Online; accessed 25-December-2017].
Retrieved from: http://developer.samsung.com/tv/develop/specifications/tv-model-groups

[23] SAMSUNG: *Web Runtime*. SAMSUNG DEVELOPERS. [Online; accessed 06-July-2018].
Retrieved from: https://developer.tizen.org/development/training/web-application/understanding-tizen-programming/web-runtime

[24] The Khronos Group Inc.: *WebGL Overview: OpenGL ES for the Web*. The Khronos Group Inc.. [Online; accessed 1-January-2018].
Retrieved from: https://www.khronos.org/webgl/

[25] Udin, E.: *Tizen OS set to dominate 70year*. Tizen Experts. [Online; accessed 30-December-2017].
Retrieved from: https://www.tizenexperts.com/2017/12/tizen-os-set-dominate-70-global-smart-tv-market-year/

[26] Web Platform Working Group: *HTML 5.3*. World Wide Web Consortium. [Online; accessed 29-May-2018].
Retrieved from: https://www.w3.org/TR/html53/

[27] Wikipedia contributors: Acid3 — Wikipedia, The Free Encyclopedia. 2017. [Online; accessed 2-January-2018].
Retrieved from: https://en.wikipedia.org/w/index.php?title=Acid3&oldid=816684616

Appendix A

Manual

Prerequisites are node.js [5.2.4] and optionally Tizen Studio CLI [3.2.2] and webOS TV CLI [3.3.1] to build packages for those platforms.

A.1 Build and Deployment of Mautilus Smart TV SDK™ 3.0 Application

1. `npm install`

2. `npm run build:dll`

3. (a) `npm run dev`
   (b)   i. `npm run watch <platform>`
        ii. `npm run start <platform>`
   (c) `npm run build:prod <platform>`

`<platform>` ∈ {`tizen, webos, samsung, hbbtv, playstation`}

3a deploys the application on local server with HMR, the URL will be logged in terminal.
3b deploys a local HMR server and builds a package which needs to be installed on a device.
3c builds a package which can be distributed, installed and ran anywhere.

A.2 Build and Deployment of Mautilus Smart TV SDK™ 3.0 StoryBook

1. `npm install`

2. `npm run build:dll`

3. (a) `npm run storybook:server`
   (b) `npm run storybook:build <platform>`

`<platform>` ∈ {`tizen, webos, samsung, hbbtv, playstation`}

3a deploys the storybook on local server with HMR, the URL will be logged in terminal.
3b builds a package which can be distributed, installed and ran anywhere.

A.3 Application Installation in Tizen Studio CLI

1. Enable developer mode on TV.

2. `C:\tizen-studio\tools\> sdb connect <IP_of_the_TV>:26101`
   The televisions use port 26101, other devices may vary.

3. (Optional control step) Now the connected device should be listed here:
   `C:\tizen-studio\tools\> sdb devices`
   Return format: `<serial> <state> <target>`
   `serial` should match `<IP_of_the_TV>:26101` as had been used earlier.
   `state` is supposed to be `device`.

4. Uninstallation should not usually be necessary. Following `install` function should perform the uninstallation implicitly.
   `C:\tizen-studio\tools\ide\bin\> tizen uninstall -s <serial> -p <appID>`
   `serial` is again `<IP_of_the_TV>:26101`
   `appID` string as it is stated in `project_root\configuration\config.xml`, for example `6AsvPiBNtr.appname`

5. Installation
   `C:\tizen-studio\tools\ide\bin\>`
   `tizen install -s <serial> -f <filename>.wgt -- <path_to_wgt_folder>`

6. Run the installed application:
   `C:\tizen-studio\tools\ide\bin\>`
   `tizen run -s <serial> -p <appID>`
   Note: Some 2015 models are failing to perform this command.

A.4 Creating a Certificate in Tizen Studio CLI

1. `C:\tizen-studio\tools\ide\bin\> tizen certificate -a <name>`
   `-p <password> -f <file_name>`

   Default path where it will be created:
   `C:\tizen-studio-data\keystore\author\`

2. Creating a Certificate Profile or replacing certificates of existing one:
   `C:\tizen-studio\tools\ide\bin\> tizen security-profiles add`
   `-n <certificate_name> -a <absolute_path>\author.p12 -p <password>`
   `-d <absolute_path>\distributor.p12 -pd <password>`

3. Before the first use of Tizen CLI the global default is needed to be initialized as needed so that the security profile does not need to be explicitly stated every time later:
   `C:\tizen-studio\tools\ide\bin\> tizen cli-config „default.profiles.path`
   `= C:\Users\%username%\workspace\.metadata\.plugins\`
   `org.tizen.common.sign\profiles.xml" --global`

Appendix B

Configuration Files

```xml
<?xml version="1.0" encoding="UTF-8"?>
<widget xmlns="http://www.samsung.com/">
    <cpname itemtype="string"/>
    <cplogo itemtype="string"/>
    <cpauthjs itemtype="string"/>
    <ThumbIcon itemtype="string">icons/samsung/106x087.png</ThumbIcon>
    <BigThumbIcon itemtype="string">icons/samsung/115x095.png</BigThumbIcon>
    <ListIcon itemtype="string">icons/samsung/086x070.png</ListIcon>
    <BigListIcon itemtype="string">icons/samsung/115x095.png</BigListIcon>
    <category itemtype="string"/>
    <autoUpdate itemtype="boolean">y</autoUpdate>
    <ver itemtype="string">1.0.1</ver>
    <mgrver itemtype="string"/>
    <fullwidget itemtype="boolean">y</fullwidget>
    <type itemtype="string">user</type>
    <srcctl itemtype="boolean">y</srcctl>
    <ticker itemtype="boolean">n</ticker>
    <mouse itemtype="boolean">y</mouse>
    <network itemtype="boolean">n</network>
    <childlock itemtype="boolean">n</childlock>
    <videomute itemtype="boolean">n</videomute>
    <dcont itemtype="boolean">y</dcont>
    <widgetname itemtype="string">myAppName</widgetname>
    <description itemtype="string"/>
    <width itemtype="string">1280</width>
    <height itemtype="string">720</height>
    <author itemtype="group">
        <name itemtype="string">MyName</name>
        <email itemtype="string"/>
        <link itemtype="string">www.mysite.com</link>
        <organization itemtype="string">MyCompany</organization>
    </author>
</widget>
```

Figure B.1: Samsung Orsay `config.xml` example

```
Use Alpha Blending? = Yes
Screen Resolution = 1280x720
```

Figure B.2: Samsung Orsay `widget.info` example

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<rsp stat="ok">
<list>
<widget id="ProjectID">
    <title>Project Name</title>
    <compression size="123456" type="zip"/>
    <description>Project Description</description>
    <download>http://xxx.xxx.xxx.xxx/anyPathToWidget/file.zip</download>
</widget>
</list>
</rsp>
```

Figure B.3: Samsung Orsay `widget.xml` example

```json
{
    "icon": "icons/webos/80x80.png",
    "main": "index.html",
    "bgImage": "icons/webos/bg.png",
    "title": "appName",
    "type": "web",
    "resolution": "1280x720",
    "version": "1.0.1",
    "splashBackground": "",
    "transparent": true,
    "bgColor": "#ed11ed",
    "vendor": "name",
    "largeIcon": "icons/webos/130x130.png",
    "iconColor": "#cd2c24",
    "id": "com.test.app"
}
```

Figure B.4: LG webOS `appinfo.json` example