# Position Paper: Challenges for enabling targeted multi-screen advertisement for interactive TV services

for the Fourth W3C Web and TV Workshop

Christopher Krauss, Louay Bassbouss, Stefan Pham,
Stefan Kaiser, Stefan Arbanowski, Stephan Steglich
Fraunhofer Institute for Open Communication Systems
Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany
{firstname}.{lastname}@fokus.fraunhofer.de

February 3, 2014

The Fraunhofer Institute FOKUS[1] Competence Center Future Applications and Media FAME[2] researches and develops in the fields of interactive smart and social media services, cross-platform applications as well as personalized service offerings. All technical solutions comply with applicable international standards, such as DASH IF, the Open IPTV Forum, HbbTV and W3C. In this context, the theme of the FAME organized 4th FOKUS Media Web Symposium[3] on May 8 - 9, 2014 is "Internet delivered Media and Multi-screen Technologies and Standards".

In this position paper we will address the challenges and missing standards for enabling targeted multi-screen advertisement for interactive TV services. Furthermore, we will outline and discuss the limitations and difficulties of building cross-platform and multi-screen TV services. We will also propose potential solutions for closing the gap between the web and TV domains based on the following use case:

Alice and Bob are watching a Formula 1 race on the TV. The broadcaster offers a service which allows users watch the race from different camera perspectives, where the TV displays the perspective of the main camera and a companion device may be used to display the perspective of the on-board camera of a selected driver (user can select which driver). During commercial breaks, the broadcast stream on the TV shows advertisements for a certain brand of footwear. At same time, the on-board camera stream will be interrupted on each companion device and a companion app related to the advertisement displayed on the TV will be launched. Alice and Bob retrieve personalized advertisements, in this case Alice's companion app displays women's shoes and Bob's companion app men's shoes. Through this app the user can for example chose to bookmark products displayed in the commercial break that he is interested in buying later and may even get a discount. After the commercial break ends, the main perspective of the Formula 1 race continues on the TV and the on-board camera perspective on the companion devices.

The rest of this paper is structured as following: Section 1 deals with the application layer of connected TV and introduces best practices for the development of TV applications. Section 2 addresses the multi-screen aspect and the missing APIs for connecting companion devices to TV. Section 3 looks at W3C media extensions which provide functionalities for interoperable adaptive streaming and content protection.

## 1 Opportunities and Challenges for Developing interactive TV services

The FOKUS team is currently working on different technologies and services for customers from the whole value added chain of hybrid TVs. The multithek[4], for instance, is Germany's first HbbTV-based interactive TV portal that allows users access different catch-up TV and interactive video offerings from one central DVB-Service. Moreover different IP-based video live streams are delivered via broadband,

[1]http://www.fokus.fraunhofer.de/en/fokus/index.html
[2]http://www.fokus.fraunhofer.de/go/fame
[3]http://www.fokus.fraunhofer.de/go/mws
[4]http://www.multithek.de/

while the access point for each stream is still the regular broadcast service. An HbbTV-enabled and connected TV device will automatically start this application and run the service after switching to the according channel. As this approach allows even smaller enterprises to launch TV services and offer it to a mass-market at lower costs, they need tools that assist them in developing TV services or that convert their existing content to TV compliant offerings.

The multithek approach leverages providers to publish their own IP-based stream services on current TV and second screen devices. The development of TV applications differ in various usability and technology driven aspects from regular web services. From a usability perspective TV applications require a special interaction paradigm resulting from the 10 foot user-to-display-distance and handling with a remote control or other interactive control approaches. Technical aspects in contrast are complex and mostly manufacturer depend.

While standards, such as OIPF DAE[5], CE-HTML[6], HbbTV[7] and W3C[8], allow services to be developed once and published on all devices, TV propriety solutions are mostly based on common web technologies (e.g. HTML, JavaScript, CSS) with some manufacturer specific extensions, such as Samsung SmartTV SDK[9], SmartTV SDK (by Philips, Sharp and Band&Olufsen – formerly known as NetTV)[10] and SmartTV Alliance SDK[11]. Our experiences in developing TV services result in 10 suggestions for general TV development and a differentiation of general in-app logic and technology dependent code:

1. The Mime-Type dictates the browser interpreter. That is why you shall first define the Mime- and Doc-Type before starting the implementation. Weigh up possible target devices first.

2. Most apps need initialization procedures with general app logic and technology depend code which shall be encapsulated from the rest.

3. The usage of common design pattern, such as Model-View-Controller, Observer, Adapter and Facade Pattern leverage the re-usability of your app code.

4. Apps benefit from ways to differentiate, manage, show and hide as well as focus and blur views. Ideally, these views can be controlled from a central view controller, called scene manager. The scene manager can be used to initialize scenes independently and control states of all scenes.

5. In order to avoid interruption times when switching scenes – e.g. HbbTV displays the broadcast stream for a short time when changing the URL – use a container displaying a background and load different scenes as content into the container.

6. Apps shall have routines to internally go back in history (or in hierarchy), exit the app to external app portals, broadcast services or other entry points. While the in-app history is mostly app logic, the exiting logic is mostly technology dependent.

7. Apps can listen to a general set of input events, especially key events, (number, color, arrow, OK and back keys), but also technology specific key events (e.g. volume, channel up/down) as well as some additional events (e.g. gesture and voice control). That is why the input layer shall be abstracted from the interaction execution layer in order to reuse existing control code for the same interaction type.

8. Respect the ten-foot user experience. Title-Safe, Action-Safe and Overscan area as well as the alignment and size of elements, such as texts, input elements etc., shall be designed to be recognized from a greater distance. Web- and TV-safe colors shall be the first choice. Gradients and half transparencies in contrast have to be used carefully as they differ on various devices.

9. Interactive elements (buttons, input fields, etc.) shall have predefined states. For example: only one element shall be highlighted at the same time, others can be toggled, inactive or even indicate they can be selected. Moreover navigation elements shall know about their actions, such as an OK-action and the next elements to highlight.

[5]Open IPTV Forum Release 1 specification, volume 5 (V1.2): "Declarative Application Environment"

[6]CEA-2014 revision A: "Web-based Protocol and Framework for Remote User Interface on UPnP Networks and the Internet (Web4CE)".

[7]HbbTV specification 1.5 – ETSI TS 102 796 v1.2.1

[8]W3C Working Draft 19 November 2009: "XMLHTTPRequest"; W3C, XML Schema Part 2: "Datatypes Second Edition"; W3C Recommendation (July 2002): "Exclusive XML Canonicalization - Version 1.0".

[9]http://www.samsungdforum.com/

[10]http://www.yourappontv.com/

[11]http://www.smarttv-alliance.org/

10. Most TV sets can only display one video or audio element at a time. That is why the initialization of multiple video objects shall be avoided.

Most issues arise, as the TV market is still fragmented and the browser implementations of single manufactures differ from each other. An IDE (integrated development environment) for all TV technologies based on standard web technologies may support service providers in respecting these general and manufacturer specific TV characteristics. It shall offer frameworks for different technologies, assist the users with tips and wizzards in developing apps and can offer GUIs for the easier alignment of graphical elements. BBC's TV Application Layer (TAL)[12] is a good starting point for building such an IDE that allows an app be developed once and compiles it for the different TV and portal offerings.

In contrast, some web elements would benefit even more from a common standard dictating just one way of implementation. The implementation of audio/video elements, for instance, even differ when using standardized web technologies on different platforms (e.g. HTML-Object-Tag vs. HTML5-Video-Object vs. proprietary solutions). Using only one object for all platforms offering access to all needed functions, parameters as well as sync and stream events, via standardized APIs would leverage the current TV application development a lot. These APIs must make it possible to load broadcast, broadband and local audio/video content.

Referring to the bigger world of multiple screens: A lot of service providers have already aggregated their own media files as well as according metadata and host them on web servers. In most cases common or proprietary Content Management Systems (CMS) assist in administering this data for distributed web services. New Content Management Systems or extensions of existing ones would enable an automatic porting to different target platforms, such as regular web pages, mobile pages and mobile apps as well as TV applications. These target devices are still diverse in operation system, browser or Runtime Environment and interpreter settings. Thus, developments for specific target devices still require specific implementations – even for the communication layer in between. How about connecting companion devices to the first screen?

## 2 Multi-screen for the web

Multi-screen applications require developers to think about additional aspects that do not necessarily have to be addressed when building single-screen applications. This includes discovery, synchronization, adaptation and distribution of content, security and alignment of user preferences all across different operating systems. Multi-screen applications are entering the market with force. The main drivers of this trend are connected TV sets. Almost every modern TV or set-top-box can be controlled with a smartphone app provided by the manufacturer or extended with additional functionality by installing new applications. While using web technologies has proved to be a cost-efficient method for building apps that run on multiple platforms, they currently lack the features that make platform-specific products that are utilizing multiple screens like AirPlay[13] or Wifi Display[14] so successful. This mainly includes direct device-to-device discovery and communication, which due to the traditional client-server architecture of web applications is implemented indirectly by either requiring applications on different devices to be actively registered at run-time or implementing a manual coupling process, e.g. using alpha-numeric or QR codes. Depending on the application model, centralized or distributed deployment, media and state synchronization may also pose a challenge to developers. Looked at individually, none of these challenges are unsolvable today. There is however no common baseline, no language support or framework of conventions to aid developers that need to build multi-screen applications for the web. We therefore propose in this section a potential API which will enable the introduction of multi-screen capabilities in traditional web applications with minimal effort. Back to the use case we already described at the beginning of this paper, to realize the type of service described using web technologies we can identify two main requirements related to the multi-screen aspect:

1. Notification and Launch of companion web apps: There is a strong requirement for a platform independent mechanism for notifying companion devices to join a session started on the TV. This will enable the user agent running on the TV to send invitations to companion devices in order to launch a web application and join the session running on the TV. The notification mechanism should abstract from the mechanisms used to discover and pair the TV and the companion devices. Existing discovery and pairing technologies like UPnP, Bonjour, etc. can be used in the background.

[12]http://fmtvp.github.io/tal/index.html
[13]https://www.apple.com/airplay
[14]http://www.intel.com/widi

2. App to App Communication: After launching the companion app, an offline and secure bidirectional communication channel between the TV and the companion application should be established. Both application parts will be able to exchange and synchronize content until the connection is closed.

Therefore, we propose an API which addresses these two requirements as described in listing 1. The API introduces two new event types "*DeviceConnected*" and "*DeviceDisconnected*" which are triggered in case a companion device is connected or disconnected. After a companion device is connected, the TV application can send a request to it in order to launch a specific web application. In case the user of the companion device accepts the request, the companion web app will be launched and the success callback will be triggered in the TV application by passing a message port (part of the HTML5 Web Messaging API[15]) as input parameter. The TV application can use the *port.postMessage* and *port.onmessage* to exchange data with the companion screen App.

```
window.addEventListener("DeviceConnected",function(evt){
  var device = evt.device;
  var req = new LaunchRequest();
  req.target = device;
  req.url = "http://example.com/cs.html";
  req.onsuccess = function(port){
    port.onmessage = function(evt){
      // handle message
    }
    port.postMessage("say hello");
  };
  req.onerror = function(err){/* handle error */};
  req.send();
});
window.addEventListener("DeviceDisconnected",function(evt){
  // handle disconnection e.g. update UI of connected devices
});
```

Listing 1: Multi-screen API proposal

Regarding the companion App, there are no additional requirements since the messaging is done over the HTML5 Web Messaging API and the *window.onmessage* can be used to receive messages from the TV app as depicted in listing 2.

```
window.onmessage(function(evt){
  if (evt.origin == 'http://example.com' && evt.data == 'say hello') {
    evt.origin.postMessage("hello");
  }
});
```

Listing 2: Companion Web App

Besides the basic communication using Web Messaging API, the app provider can upgrade to other existing Web technologies such as WebSockets[16] or WebRTC[17] if needed. In this case, the TV and companion apps use the message port as signaling channel in order to exchange a common session ID and establish a connection to a WebSocket server or to exchange device connection metadata in order to establish a WebRTC channel (SDP offer/answer, etc.).

## 3 DASH and DRM

According to Cisco's recent Visual Networking Index, Internet-delivered video traffic keeps increasing compared to global consumer IP traffic. The numbers state that video traffic will increase from 57% in 2012 to 69% in 2017, video-on-demand traffic will triple by 2017 and similarly, Internet video to TV will continue to grow at a rapid pace. Moreover the increasing success of HbbTV clearly shows the market trend towards OTT. Services that broadcast their program over broadband will benefit from HTTP adaptive streaming techniques(e.g. MPEG DASH). In our previously introduced use case, the broadcast signal on the TV could be replaced by a broadband channel, which could then be easier synchronized with companion devices.

The standards ISO MPEG Live Dynamic Adaptive Streaming over HTTP (DASH), ISO MPEG DRM-interoperable Common Encryption (CENC), HTML5 Media Extensions under development in the W3C - Media Source Extensions (MSE), Encrypted Media Extensions (EME) and Web Crypto Extensions form the future of Internet-delivered media. With TV and other screens supporting HTML5 browsers, interoperable consumption of adaptive premium video content can already be achieved today.

[15]http://www.w3.org/TR/webmessaging/#message-ports
[16]http://www.w3.org/TR/websockets
[17]http://www.w3.org/TR/webrtc

"dash.js" is an open source project that serves as a reference client implementation for MPEG DASH playback in Web browsers. For this purpose it takes advantage of the HTML5 Media Extensions. IE11 and Chrome have already implemented these, thus are supported by "dash.js". Nevertheless potential incompatibilities between browsers need to be considered. The advancement and constant enhancement of these W3C standards is essential for the adoption of these standards by other platforms (e.g. TV) and more browsers. Implementation feedback is also important to standardization. FOKUS supports the development of "dash.js", as it promotes non-proprietary browser-based MPEG DASH players. This is a subset of features that we have implemented using "dash.js":

- ad-insertion
- multi-screen media synchronization
- improved adaptive switching algorithms
- MPEG DASH live playback from broadcast sources
- interoperable content protection

Ad-insertion is one of the main requirements for OTT business models. We have implemented various ways to do server- or client-side ad-insertion as well as leveraged MPEG DASH "tools" such as multiple periods, XLink or DASH events for proof of concept purposes. Ad-tracking (e.g. using VAST) is also considered in the different methodologies.

Multi-screen media synchronization can also be easily realized as an extension to "dash.js". Our synchronization framework, based on the previously described messaging mechanism between TV and companion devices, leverages WebSockets and WebRTC for signaling between the peers. Further, it supports the creation of so called "synchronization groups".

A reliable adaptive switching algorithm is crucial to an MPEG DASH player. It guarantees an optimal media stream quality depending on the current bandwidth and at the same time avoids buffer underflows or long startup times, which is a complex challenge. Most algorithms are limited to measuring the throughput or current buffer level to determine the best possible media quality. High-resolution clocks and timers could lead to more accurate measurements of throughput or buffer level and more performance indicators/hardware APIs (e.g. CPU, average available bandwidth etc.) in the browser might help to improve algorithm accuracy and thus, lead to a better user experience.

Adoption of MPEG DASH live playback in hybrid TV environments (e.g. HbbTV) will lead to a better user experience (due to adaptiveness) compared to HTTP "chunked" streams , which are in practice today. With our end-to-end implementation, we are able to transcode DVB signals to IP-multicast and then to live MPEG DASH.

Another important aspect of OTT media consumption is content protection. DRM interoperability can be solved by utilization of the common encryption standard (CENC) and the application of common encryption to the proposed HTML5 Encrypted Media Extensions (EME). It will enable the same media stream to be consumed, for example, on a phone or tablet, a set-top-box or a smart TV - protecting the content owner's rights while enabling interoperable media consumption even on open source browsers. "dash.js" comes with support for these standards. With the introduction of 4K content, new security requirements may arise (e.g. cryptography) and need to be considered for the respective standards.

All of these features can be easily implemented and deployed across HTML5 platforms. Thus, "dash.js" enables developers to build rich TV applications.

## 4 Conclusion

As identified in this paper, we see various challenges for developing interactive TV services, connecting TV and companion devices and adaptive streaming in the browser. Furthermore, we proposed an API for building multi-screen applications using Web technologies. As proof of concept for these challenges, Fraunhofer FOKUS offers an end-to-end prototype implementation called FAMIUM[18], a collection of software components for early technology evaluation and interoperability testing, including an implementation of the proposed multi-screen API. The components of the FAMIUM framework dealing with browser-based DASH and DRM are *FAMIUM.transcoder* which converts existing media (on-demand or live content) to DASH-compliant content, while the *FAMIUM.player* (based on dash.js) is a Web-based DASH and DRM-capable media player. For encrypted content, FAMIUM.cenc relies on underlying DRM platforms that can be accessed via dedicated Content Decryption Modules (CDM).

[18] http://www.fokus.fraunhofer.de/go/famium