Livrable 3.3.2

Version	**1.0**
Date	**01 septembre 2015**
Auteur(s)	**R. ALEXIA (ATEME)**
N° du Lot	**-**

Le lecteur media accessible à tous

Livrable 3.3.2 : Spécifications des mécanismes de synchronisation

Titre du projet	media4Dplayer
Abréviation	M4DP
Désignation	media4Dplayer, le lecteur media accessible à tous.
Durée du projet	De Janvier 2015 à Juin 2016 – 18 mois
Coordinateur projet	France Télévisions
Partenaires projet	FRANCE TELEVISIONS (FTV) LE GROUPE LA POSTE (LP) DOTSCREEN (DTS) ATEME (ATM) INSTITUT MINES TELECOM (TSP) LABORATOIRE CHART / LUTIN-USERLAB (UP8) PLAINE COMMUNE (PC)
Prestataires	Multimédia France Production (MFP) Holken Consultants & Partners (HC)
Organisme labellisateur	CAP DIGITAL
Financeurs	La Région Ile-de-France La BPIfrance
Titre de subvention	Fonds Unique Interministériel – FUI18

Le projet media4Dplayer, lecteur media accessible à tous.

Spécifications des mécanismes de synchronisation

Date de soumission : 01/09/2015
Version : 1.0

Objectif(s) du livrable
Spécifications des approches de synchronisation et des mécanismes d'horodatage multi-flux dans les solutions d'encodage multi-écrans live et offline

Historique	**Date**	**Modification(s)**
V 1.0	01/09/2015	création

Le projet media4Dplayer

Media4Dplayer est un projet collaboratif labellisé par le pôle de compétitivité Cap Digital et subventionné au titre du Fonds Unique Interministériel (FUI) par la région Île de France et BPIfrance. Ce projet de recherche et de développement s'inscrit dans la stratégie de Cap Digital, autour des thématiques d'accessibilité des contenus, de développement numérique et de Silver économie.

Durée de projet 18 mois : Janvier 2015 – Juin 2016

Avertissement

Les informations contenues dans ce document peuvent être sujet à modification sans préavis. Société ou noms de produits mentionnés dans ce document peuvent être des marques ou des marques déposées de leurs sociétés respectives.

Tous les droits sont réservés

Table Des Matières

1. CONTEXTE DU PROJET 6
2. INTRODUCTION 6
3. MODELES DE SYNCHRONISATION RETENUS 8
3.1. CAS D'USAGE POUR LA SYNCHRONISATION 8
3.2. SYNCHRONISATION INTRA-TERMINAL 10
4. ETUDES ET PROPOSITIONS 11
4.1. PLATEFORME WINDOWS ET MAC OSX 12
4.2. PLATEFORME ANDROID 13
4.3. VERROUS TECHNOLOGIQUES 13
4.3.1. ANDROID 13
4.3.2. SOUS-TITRES 14
4.4. EXPERIMENTATIONS 14
4.4.1. DASH.JS VERSUS HASPLAYER 14
4.4.2. ARCHITECTURE RETENUE 15
4.5. RESULTATS 15
4.5.1. LANCEMENT DES VIDEOS 16
4.5.2. SYNCHRONISATION 16
4.5.3. WEBAUDIO API 18
5. CONCLUSION 19
6. REFERENCES 20

FIGURES

Figure 1 : Principe de fonctionnement d'un player MPEG-DASH 10
Figure 2 : Architecture Desktop (Windows 8 et Mac OSX) 12
Figure 3 : Architecture Android 13

TABLEAUX

Tableau 1 : Exigences de synchronisation 9
Tableau 2 : Résultats des expérimentations de démarrage 16
Tableau 3 : Résultats des expérimentations de synchronisation 17
Tableau 4 : Résultats des expérimentations sur la WebAudio API 18

1. Contexte du projet

L'objectif du sous-programme 3.3 est la conception et le développement de mécanisme de synchronisation multiflux provenant de serveurs différents. Pour ce faire, plusieurs approches seront considérées allant du marquage des flux élémentaires à la définition de mécanismes dédiés aux diverses couches systèmes et transports considérées. L'état d'avancement des technologies Web nous fait considérer, dès le départ, des développements visant des rendus multiplateformes (HTML5, IOS, Android).

L'objectif final est d'établir une solution relativement unifiée à même de répondre aux diverses contraintes des architectures des plateformes considérées. Toutefois, les problématiques de synchronisation live (consultation de serveur d'horloge,…) et offline (dérives d'horloges intrinsèques,…) impliqueront naturellement la mise en place de solutions différentes en amont de l'horodatage considéré.

Ces mécanismes seront tout d'abord mis en œuvre en considérant la technologie de compression H.264/MPEG-4 AVC, cette dernière étant la plus largement déployée dans les écosystèmes de visualisation de contenus sur second écran. Dans un second temps, les travaux de recherche seront étendus à la nouvelle technologie de compression HEVC (High Efficiency Video Coding), capable de performances débit-distorsion deux fois supérieures à H.264/MPEG-4 AVC. Une telle technologie, poussée par le monde de l'OTT (Over The Top), nous permettra d'accroitre de manière significative l'efficacité d'encodage, et donc d'étendre l'accessibilité aux services.

2. Introduction

La démocratisation des tablettes et des smartphones permet d'offrir aux utilisateurs de nouvelles manières de consommer les services audio visuels. Sur la base d'un service principal présenté sur une TV connectée ou via une Set Top Box, de nombreux flux enrichis peuvent être proposé sur un terminal portable additionnel. Nous pouvons citer par exemple :

Sondage ou question à choix multiples relatifs au contenu courant présenté sur la TV ;

Piste audio alternative écoutée à l'aide d'un casque connecté directement sur le terminal portable en même temps que le flux vidéo principal sur la TV. Il peut s'agir d'informations supplémentaires, de la traduction dans une autre langue, d'un flux d'audio description ;

Visualisation du même contenu que celui présenté sur la TV, mais dans une vue différente ;

Présentation d'informations supplémentaires concernant le contenu principal: informations sur les acteurs au moment où ils apparaissent dans le film, publicité et lien vers un site de vente en ligne d'un produit visible à l'écran.

Tous ces cas d'usage nécessitent une synchronisation des contenus présentés sur les deux terminaux. Les exigences en termes de synchronisation varient selon les types de contenus. La valeur de 40ms est généralement utilisée dans le cas d'un flux vidéo et d'un flux audio (synchronisation labiale) ou de deux flux vidéo (synchronisation à l'image). Cependant, plusieurs éléments peuvent venir perturber cette synchronisation :

Etapes de production du contenu: dans le cas d'une transmission Over The Top (OTT), une étape supplémentaire de transcodage et de packaging est nécessaire afin de préparer les contenus selon les formats utilisés (MPEG-DASH, Microsoft Smooth Streaming, Apple http Live Streaming…) ;

Utilisation de réseaux de diffusion différents (TNT, satellite, IPTV, OTT), entraine des temps de propagation très éloignés ;

Même au sein d'un réseau IP, la diversité des chemins peut aussi entrainer des temps d'accès variables ;

Les divers éléments de la chaine de diffusion, comme les Origin Servers, les CDN et les équipements réseau, ajoutent des délais (buffering) à chaque étape.

De plus, les mécanismes de synchronisation actuels utilisent les timelines des différents flux afin de les synchroniser, or, dans des cas d'usage multi flux et multi terminaux, les timelines des différents flux ne peuvent souvent pas être mise en relation pour différentes raisons :

Les timelines des différents flux peuvent présenter des paramètres non comparables (fréquence, résolution) ;

Les différentes étapes de la production et de la distribution peuvent altérer la précision des timelines des flux ;

Les horloges des terminaux utilisés ne sont pas synchronisées entre elle et peuvent donc varier.

Les estampilles temporelles utilisées pour définir les instants de présentation des éléments unitaires des médias (image, trame audio…) sont des valeurs relatives calculées à partir d'une horloge de référence. Si la précision de cette horloge est altérée, la présentation des flux sera certainement dégradée.

De nombreux travaux ont exploré des mécanismes permettant de synchroniser des flux média en résolvant les problèmes introduits par les technologies et réseaux utilisés. Un précédent travail sur l'état de l'art (réf. [2]) reprend certains de ces travaux répondant le mieux aux exigences du projet.

L'analyse des différents cas d'usage adressés dans le cadre du projet Media4DPlayer nous a permis de mettre en évidence deux modes de synchronisation qui seront présentés dans ce document. Ensuite, nous détaillons les architectures et solutions mises en œuvre lors de nos expérimentations.

3. Modèles de synchronisation retenus

3.1. Cas d'usage pour la synchronisation

Parmi les nombreux cas d'usage rendus possibles par la présence croissante de terminaux mobiles chez les téléspectateurs, nous avons retenus ceux rentrant dans le cadre du projet (Cf.[1]).

Nous avons choisi d'adresser le cas particulier du multiple flux MPEG-DASH à synchroniser lors de la première phase du projet. Les architectures et mécanismes mis en œuvre pour répondre aux exigences de ce cas sont détaillés dans la suite du document

Dans ce cas, les différents médias (vidéos, audios…) relatifs à un même contenu sont disponibles dans des flux MPEG DASH distincts. L'agrégation et la présentation de ces flux afin de recomposer le contenu nécessite de synchroniser le téléchargement et la lecture de flux provenant de sources différentes. De plus, des flux enrichis peuvent n'être disponibles que pour des programmes spécifiques. Il est alors nécessaire de synchroniser la présentation du flux supplémentaire sur le flux principal déjà en cours de lecture.

Pour rappel, les exigences en termes de précision de la synchronisation des flux média sont décrites dans le Tableau 1 (source : (Thomas, Le Feuvre, & Singer, 2014)) ci-après :

Précision de synchronisation	Cas d'usage associés
A l'image	• Audio • Rehaussement par codage scalable
10 us – 10 ms	• Mur vidéo • Hauts parleurs réseau
10 – 100 ms	• Vidéo conférence • Jeux temps réels multi-joueurs en ligne
100 – 500 ms	• Second écran • Travail collaboratif
500 – 2000 ms	• Bascule d'un terminal à l'autre

Tableau 1 : Exigences de synchronisation

L'analyse des cas d'usage présenté précédemment (Cf.[1]) permet d'extraire plusieurs scénarios de synchronisation :

- Un flux audio ou vidéo supplémentaire doit être synchronisé avec le flux audio/vidéo principal sur le terminal principal,
- Un flux audio ou vidéo supplémentaire doit être synchronisé avec le flux audio/vidéo principal sur le terminal secondaire,
- Des informations contextuelles relatives au contenu actuellement affiché sur le terminal principal doivent être présentées sur le terminal secondaire.

Deux modèles de synchronisation sont ainsi mis en évidence, la synchronisation au sein d'un même terminal et la synchronisation entre deux terminaux (ou plus).

La première phase du projet concerne plus particulièrement la synchronisation intra-terminal, nous nous sommes donc concentrés dans ce document sur les problématiques liées à ce cas.

L'aspect inter-terminal sera abordé dans une deuxième phase du projet.

3.2. Synchronisation intra-terminal

La Figure 1 illustre le fonctionnement d'un player MPEG-DASH pour un contenu audio-visuel classique (une piste audio et une piste vidéo potentiellement à plusieurs niveaux de qualités).

A partir du manifeste, le player détermine le premier segment à télécharger pour chaque piste. Dans le cas du *file*, ces segments correspondent aux premiers segments du contenu. Par contre, dans le cas du *live*, le player détermine pour chaque piste le dernier segment disponible correspondant à l'instant le plus proche du *live* (le *live edge*).

Pour un contenu audio-visuel, les segments audio et vidéo sont alignés ; cela signifie que les indications fournies par le manifeste permettent d'associer le segment audio et le segment vidéo correspondant à un instant donné. C'est le premier niveau de synchronisation avec la granularité de la durée du segment.

Afin d'assurer ensuite une synchronisation fine, le player dispose des estampilles temporelles de chaque piste relatives à une horloge de référence commune (la Program Clock Reference : PCR).

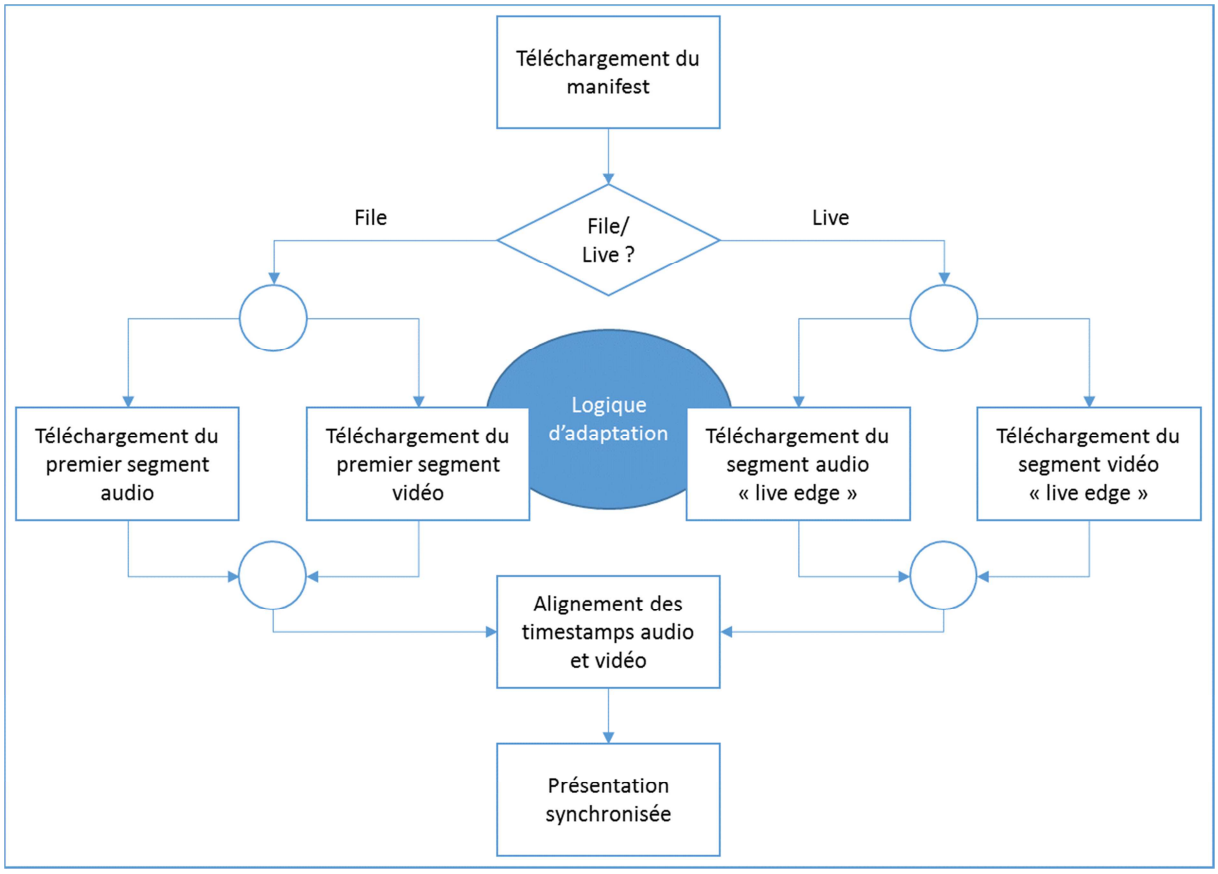

Figure 1 : Principe de fonctionnement d'un player MPEG-DASH

Le cas nominal ne pose donc pas de problème de synchronisation : tout est géré en interne par le player. Par contre, des difficultés apparaissent lorsque la diversité des flux nécessite d'instancier plusieurs players.

Par exemple, dans le cas d'un contenu composé de trois flux : un flux vidéo principal plus un flux audio principal et d'un flux vidéo LSF, deux players vidéo seront nécessaires pour présenter tous les flux disponibles. Initialement, aucun mécanisme permettant de synchroniser deux players distincts n'est présent. Les players peuvent donc être désynchronisés dès le démarrage de la lecture. Cette désynchronisation peut être due à des temps de téléchargement et donc à des délais de remplissage de buffers différents. Il est donc nécessaire dans un premier temps de s'assurer que les players démarrent la présentation des flux de manière synchronisée. De même, au cours de la lecture, rien n'assure que les players restent synchronisés surtout en présence d'actions de la part de l'utilisateur PAUSE, SEEK.

Ce qui a été réalisé lors du développement de la solution est d'ajouter une surcouche au-dessus des players, afin de les piloter de façon synchrone. De cette manière les actions de l'utilisateur (PAUSE, SEEK) peuvent être appliquées au même moment à tous les players.

4. Etudes et propositions

Au démarrage du projet, nous avons opté pour une architecture HTML5, principalement pour les raisons suivantes :

- Le potentiel de portabilité sur différentes plateformes. Android et Windows principalement,
- La possibilité d'intégrer la librairie WebAudio API, permettant de nombreuses fonctionnalités sonores liées à l'accessibilité (mixage de flux, applications d'effets sonores…),

Concernant le player MPEG-DASH nous avons choisi dash.js qui est le player de référence du Dash Industry Forum. Ecrit en Javascript, il fonctionne dans n'importe quel player HTML5 qui supporte les extensions MediaSource Extensions et Encrypted Media Extensions.

L'architecture logicielle mise en œuvre au sein d'un terminal est détaillée dans les Figure 2 et Figure 3 selon le type de terminal. En effet, les expérimentations que nous avons menées ont montré que le comportement divergeait fortement d'un type de terminal à l'autre. Nous avons donc dû définir deux architectures logicielles.

4.1. Plateforme Windows et Mac OSX

Cette architecture, décrite dans la Figure 2 concerne les terminaux de type PC (desktop ou laptop) et tablette équipés du système d'exploitation Windows version 8 ou Mac OSX.

Techniquement cela implique la mise en œuvre de trois players MPEG-DASH et d'une instance de la WebAudio API :

- Trois players dash.js
  - Vidéo principale + audio principal + sous-titres
  - Vidéo LSF
  - Audio description

- WebAudio API :
  - synthèse et synchronisation des deux flux audio : audio principal + audio description

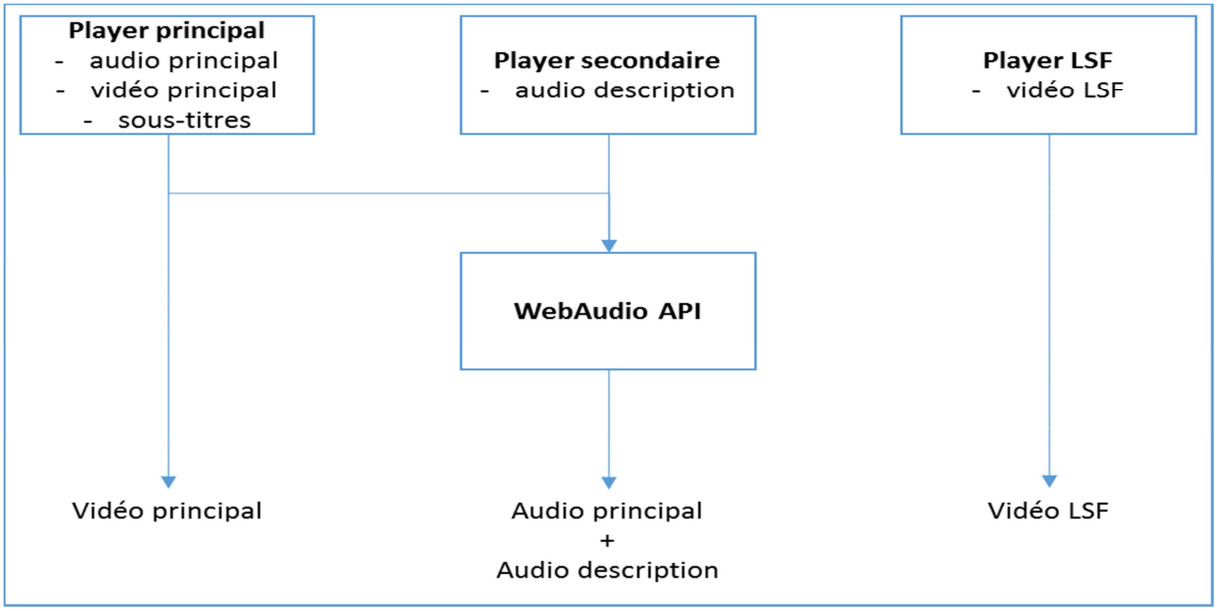

Figure 2 : Architecture Desktop (Windows 8 et Mac OSX)

4.2. Plateforme Android

Cette architecture, détaillée dans la Figure 3 concerne les terminaux smartphones ou tablettes équipés du système d'exploitation Android. Comme expliqué dans les paragraphes suivants, le comportement observé sur cette plateforme nécessitait de définir une architecture logicielle spécifique afin d'atteindre un niveau de synchronisation nominal satisfaisant :

- Un Exo player Android avec trois flux
  - Vidéo principale + audio principal
  - Vidéo LSF
  - Audio description

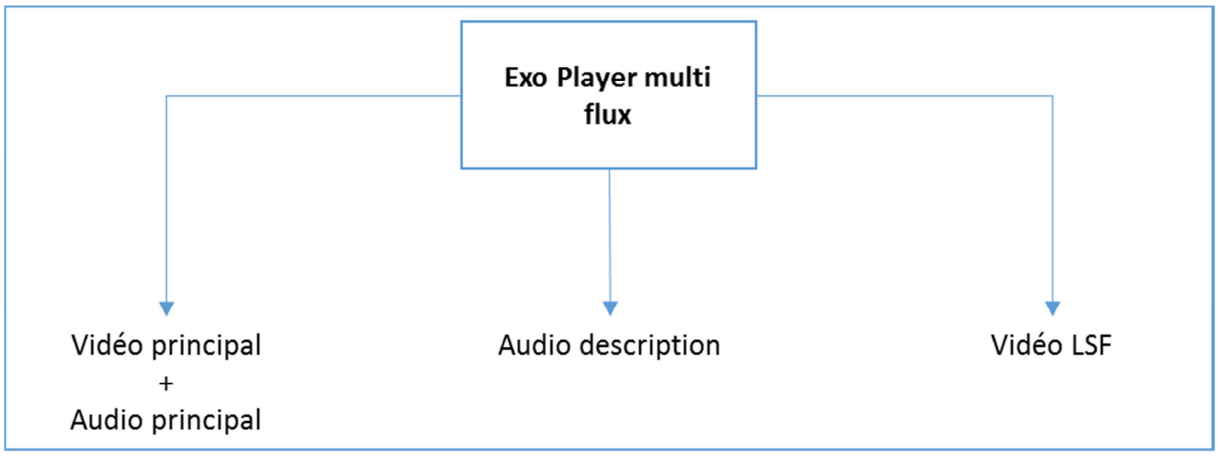

Figure 3 : Architecture Android

4.3. Verrous technologiques

Nous faisons ici la synthèse des difficultés rencontrées, ainsi que des solutions ou propositions étudiées.

4.3.1. Android

Sur Android, la synchronisation pose des problèmes importants. Nous n'avons pas réussi à synchroniser de manière satisfaisante les différents flux :

- Nous n'avons pas réussi à lancer les de manières synchrones, il y a un décalage de plus de 0,5 sec entre les 2 vidéos, sans que nous puissions le contrôler,
- Au cours du temps, nous avons observé une dérive de la désynchronisation, voir un blocage d'un des 2 players,

Par ailleurs, la WebAudio Api ne semble pas fonctionnelle sous Android. Nous n'arrivons pas à obtenir les résultats voulus.

Par contre, nous avons essayé, sous Android, une architecture plus bas niveau en instanciant le player natif (en fait la surcouche Exoplayer) et en lui fournissant les différents flux à traiter : vidéo et audio principales, vidéo LSF, Audio description.

Sur cette dernière architecture, nous n'observons plus ces problèmes de synchronisation. Néanmoins des tests complémentaires sont nécessaires pour valider définitivement cette solution.

4.3.2. Sous-titres

Concernant les sous-titres, dash.js ne supporte pas le format TTML, nous avons deux possibilités :

- Se contenter de ce que permet le format WebVTT, nous ferons une matrice de conformité entre les besoins exprimés et le format WebVTT,
- Refaire un moteur de rendu, sans avoir les évènements qui sont remontés par le player. Cela signifie donc que l'application devra analyser le moteur de sous-titre,

A l'heure actuelle, Dash.js n'envisage pas pour l'instant la possibilité d'avoir un autre moteur de rendu pour les sous-titres.

4.4. Expérimentations

4.4.1. Dash.js versus HasPlayer

Les tests réalisés avec le player dash.js ont montré que, bien que performant, ce player n'était pas le mieux adapté à nos besoins en termes de synchronisation de flux.

Suite à des échanges techniques avec un laboratoire de la société Orange, nous avons décidé de remplacer dash.js par le HasPlayer développé en interne. Ce player est intéressant sous différents aspects :

- Gestion du format TTML pour les sous-titres : les sous titres TTML sont lus et convertis au format WebVTT par le HasPlayer. Le rendu des sous-titres est directement effectué par la Balise Video,
- Fallback HLS : au besoin il est possible d'utiliser des flux au format http Live Streaming en plus des flux MPEG-DASH,

Les similitudes entres HasPlayer le dash.js permettent de passer rapidement de l'un à l'autre sur notre plateforme.

4.4.2. Architecture retenue

L'architecture cible choisie est donc un player HasPlayer avec une instanciation de la fonction MediaControler sur PC Windows ou sur tablette Windows (surface pro 3 ou 4).

Nous poursuivons quand même nos expérimentations sur les plateformes Android et Mac OSX qui seront intéressantes dans la deuxième phase du projet dans le cadre du « *companion screen* ».

4.5. Résultats

Afin de valider les architectures mises en œuvre des campagnes de tests sont menées sur les plateformes suivantes :

- **Windows 8 :**
  - Webapp Chrome Desktop,
  - Webapp Chromium,
- **Android 5+ :**
  - Webview Chromium,
  - Webapp Chrome,
  - Architecture Native,
- **Mac OSX :**
  - Webapp Chrome Desktop
  - Webapp chromium
- **IOS 8.2 :**
  - Webapp Chrome,
  - Webapp Safari,
  - Webview,

Les paragraphes suivants présentent les premiers résultats de ces tests (qui se poursuivent) en termes de fonctionnalités, de synchronisation et d'intégration.

L'environnement utilisé nous a permis de réaliser des tests en mode *file*.

Les futurs tests devront adresser le cas *live*, mais nécessitent une évolution de la plateforme afin de mettre en œuvre et d'intégrer un encodeur *live*. En effet, ce cas ajoute une dimension dynamique

avec la possibilité d'ajouter des flux secondaires uniquement pour un programme particulier. Le nombre et le type de flux ne seraient donc pas constants au cours du temps. De nouvelles problématiques pourraient donc être mises en évidence.

4.5.1. Lancement des vidéos

Les navigateurs web sur mobile (Chrome, Chromium, Safari) ne permettent pas le lancement automatique des vidéos. Une fois la page du player chargée, l'utilisateur doit effectuer une action sur une icône présentée à l'écran afin de démarrer la lecture. Cette limitation n'est présente que sur les plateformes Android lors de l'affichage du player directement dans le navigateur web.

La solution choisie est d'encapsuler la page du player dans un composant Webview disposant d'une option permettant d'activer le démarrage automatique de la lecture.

Le Tableau 2 présente les résultats des tests effectués.

Plateforme	Architecture	Résultat	Observation
PC Windows 8	Webapp Chrome	OK	/
Android 5+	Webview Chromium	OK	/
	Player natif	OK	
	Webapp chrome	KO	Car pas de contournement en Webview
Mac OSX	Webapp Chrome	OK	/
	Webapp Chromium	KO	Non creusé, mais probablement un problème de Codec ou accélération matérielle
IOS 8.2	Webapp Chrome	KO	Car pas de contournement en Webview
	Webapp Safari	KO	Car pas de contournement en Webview
	Webview	KO	Les vidéos ne se lancent pas pour une raison inconnue : pas de message d'erreur visible de la part de dash.js

Tableau 2 : Résultats des expérimentations de démarrage

4.5.2. Synchronisation

Entre le moment où l'on demande aux différentes instances du player dash.js de démarrer la lecture vidéo et l'affichage de la vidéo, plusieurs opérations sont effectuées par les players (téléchargement du manifeste, calcul des premiers segments à télécharger, téléchargement des segments, dé-encapsulation et décodage des flux, affichage). La durée de ces tâches dépend de plusieurs paramètres (débit des flux, taille des buffers et des seuils de la logique d'adaptation) et est variable d'une instance du player à l'autre. Ainsi, dès le début de la présentation, les players sont désynchronisés. Ce n'est pas le cas lors de l'utilisation du player natif car il n'y a qu'une seule instance du player).

Le Tableau 3 présente les résultats de ces tests.

Plateforme	**Architecture**	**Observation**
PC Windows 8	Webapp Chrome	Synchronisation entre 1 et 3 frames quand les flux sont en local Décalage jusqu'à 500 ms en streaming
Android 5+	Webview Chromium	Décalage jusqu'à 500 ms en streaming Le décalage n'est pas constant, il s'accentue en court de lecture
	Player natif	Aucun décalage. Synchro à la frame prêt, la synchronisation perdure avec le temps

Tableau 3 : Résultats des expérimentations de synchronisation

L'implémentation du player repose sur un certain nombre d'évènements, en particulier l'évènement CANPLAY qui indique qu'un player est prêt à jouer la vidéo (suffisamment de bufferisation…) et permet donc de démarrer la présentation effective des flux. Nous avons donc utilisé cet évènement pour affiner la synchronisation des players au démarrage et lors d'action de l'utilisateur (pause, seek…).

La fonction MediaController permet d'améliorer sensiblement la synchronisation des flux :

- Synchronisation à la frame prêt sur Surface pro 3,
- Cela semble marcher aussi en Android, mais la lecture vidéo n'est pas fluide sur un terminal Nexus 5 qui est pourtant plutôt performant. Des tests complémentaires sont en cours pour déterminer les causes de cette dégradation des performances,

La fonction Media Controller n'est encore qu'en Beta sur Chromium, cela signifie donc qu'elle peut encore avoir quelques problèmes.

4.5.3. WebAudio API

La WebAudio API permet de multiplexer plusieurs flux audio et de leur appliquer des effets sonores. Elle est particulièrement adaptée à la présentation de flux audio dédié à l'accessibilité.

Le Tableau 4 présente les résultats des tests d'instanciation de cette API.

Plateforme	Architecture	Observation
PC Windows 8	Webapp Chrome	La WebAudio API semble correctement implémentée, des tests complémentaires sont en cours
Android 5+	Webview Chromium	Il ne semble pas possible d'injecter les sons d'une source « player » vers la WebAudio API des tests complémentaires sont en cours
	Player natif	Il n'existe pas de moyen simple pour s'interfacer avec la WebAudio API
Mac OSX	Webapp Chrome	La WebAudio API semble correctement implémentée, des tests complémentaires sont en cours

Tableau 4 : Résultats des expérimentations sur la WebAudio API

5. Conclusion

Les expérimentations réalisées ont mis en évidence les problématiques liées à la synchronisation de flux au sein d'un même terminal. En effet, les différentes instances de players peuvent être désynchronisées au démarrage ou se désynchroniser au cours de la lecture. Nous avons exploré et validé des solutions permettant de corriger ou de contourner ces problèmes.

En l'état actuel, nous avons défini une architecture logicielle permettant de lire de façon synchrone (au niveau *frame*) des flux différents sur un terminal. Nous avons pu valider cette mise en œuvre sur des flux MPEG-DASH file et nous allons maintenant pouvoir étudier le cas *live*.

La première phase du projet adresse plus particulièrement les cas d'usage nécessitant une synchronisation de flux sur un seul terminal.

Dans la suite du projet nous nous intéresserons aux cas d'usage orientés second écran ou « companion screen » qui mettent en avant des exigences de synchronisation inter-terminaux.

6. Références

[1] *Proposed Exploration of "Uniform Signaling for Timeline Alignment".* Thomas, Emmanuel, Le Feuvre, Jean et Singer, David. 2014. ISO/IEC JTC1/SC29/WG11 N14644.

[2] *M4DP_D3.3.1_Etat_de_l_art_synchronisation_fine_v1.0.* ATEME. Mars 2015, media4Dplayer