ESTUDI DEL PROTOCOL HTTP LIVE STREAMING D'APPLE I LA IMPLEMENTACIÓ D'UN CLIENT PER A L'IPHONE

Jordi Núñez Vera

Prefaci 8

Objectiu del projecte 8

Guia de lectura 8

1 Streaming de continguts audiovisuals 11

1.1 Què entenem per streaming? 11

1.2 On-Demand en vers Live-Streaming 11

1.3 Solucions actuals 13

1.3.1 RTP/RTSP 13

1.3.2 Adobe Flash Media Streaming Server 14

1.4 L'era dels smartphones i la mobilitat 15

1.4.1 Streaming adaptatiu a Internet 15

1.4.2 Implementacions actuals 17

1.4.3 Beneficis del streaming adaptatiu 19

2 HTTP Live Streaming (HLS) 22

2.1 Objectiu 22

2.2 Arquitectura 22

2.2.1 Component de servidor 24

2.2.2 Component de distribució 27

2.2.3 Component de client 27

2.3 El fitxer d'índex 28

2.3.1 Primera versió del protocol. Paraules clau. 29

2.3.2 Exemples 33

2.4 El servidor i el client segons el protocol 35
2.4.1 El procés del servidor 35
2.4.1.1 Generar el fitxer d'índex. 36
2.4.1.2 Qualitats variables 37
2.4.2 El procés del client 38
2.4.2.1 Reproducció els segments 38
2.4.2.2 Actualització del fitxer d'índex 39
2.4.2.3 Llistes de reproducció variables 40
2.5 Evolució i millores del protocol 40
2.5.1 Múltiples canals d'àudio i vídeo 42
2.5.2 Descàrrega de conjunts de bits 45
2.5.3 I-Frames 46
2.5.4 Versions el protocol 48
2.6 Estandardització i implantació 48
3 Programació d'aplicacions per a iOS 51
3.1 Objectiu d'aquest capítol 51
3.2 Primers passos 51
3.2.1 Recursos necessaris 51
3.2.2 Coneixements necessaris 53
3.2.2.1 Objective-C 53
3.2.2.1.1 Sintaxis 54
3.2.2.2 Xcode 58

3.2.2.3 Cocoa Touch 58
3.2.3 Dos paradigmes importants: OOP i MVC 58
3.2.3.1 Programació Orientada a Objectes 59
3.2.3.2 Model-Vista-Controlador 59
3.2.3.2.1 Vista (View) 60
3.2.3.2.2 Controlador (Controller) 61
3.2.3.2.3 Model 65
3.2.4 Diferències en la programació per a iOS 66
3.2.4.1 Una sola aplicació activa. 67
3.2.4.2 Tot en una sola finestra 67
3.2.4.3 Accés limitat 67
3.2.4.4 Temps de resposta limitat 68
3.2.4.5 Mida limitada de pantalla 68
3.2.4.6 Recursos limitats 68
3.2.4.7 No hi ha Garbage Collector, però ... 69
3.2.4.8 Nous recursos 69
3.2.4.9 Maneres diferents d'interactuar 70
3.2.4.10 Limitacions imposades per Apple 70
3.3 La primera aplicació 71
3.3.1 Creació del projecte 71
3.3.2 Dissenyar la vista 74
3.3.3 Escriure el codi del controlador 75

3.3.4 Implementar els mètodes d'origen de dades 76

3.3.4 Compilar i provar l'aplicació 79

3.3.5 Implementar els mètodes de delegat 80

3.3.6 Accedir als objectes de les views de l'aplicació 82

3.3.7 Enllaçar esdeveniments dels controls 84

3.4 Conclusió 85

3.5 Altres opcions 85

4 Client HLS en una aplicació per l'iPhone. 87

4.1 Objectiu del projecte 87

4.2 Aspectes a tenir en compte sobre aquesta aplicació 87

4.3 Eines i tecnologies emprades 87

4.4 Requisits de l'aplicació 88

4.5 El Servidor 88

4.5.1 Preparació dels continguts 89

4.5.1.1 Eines d'Apple per l'HLS 89

4.5.1.1.1 Media Stream Segmenter 89

4.5.1.1.2 Media File Segmenter 90

4.5.1.1.3 Media Stream Validator 90

4.5.1.1.4 Variant Playlist Creator 90

4.5.1.1.5 Metadata Tag Generator 90

4.5.1.2 Creació dels segments i dels fitxers d'índex. 91

4.5.1.3 Llista de vídeos accessibles 95

4.5.1.4 Millora dels fitxers generats per a múltiples servidors 97
4.6 El client 98
4.6.1 Configuració de l'aplicació. 100
4.6.2 Inici de l'aplicació 103
4.6.2.1 Model de dades 104
4.6.2.2 L'obtenció de les dades 105
4.6.2.3 Fitxers d'índex estàtics 111
4.6.3 - Reproducció didàctica de l'HLS 111
4.6.3.1 Procés de selecció del següent segment a descarregar 112
4.6.3.2 Traces de les operacions 114
4.6.3.3 Disseny de la interfície visual del reproductor 115
4.6.4 Característiques afegides 118
4.6.4.1 Enllaços locals a fitxers d'índex 118
4.6.4.2 Reproducció amb el reproductor del framework Cocoa 120
4.6.4.3 Generació de gràfiques 122
4.6.4.4 Compartir els resultats per correu electrònic 124
4.6.4.5 Diversos algoritmes en el moment d'escollir el següent segment 124
4.7 Solucions tecnològiques 127
4.7.1 Descàrrega d'arxius 127
4.7.1.1 Descàrrega única 130
4.7.1.2 Descàrregues múltiples 133
4.7.1.3 Catàleg de pel·lícules 135

4.7.1.4 Generador de l'estructura interna d'una pel·lícula 138
4.7.2 Reproductor dels segments 139
4.7.3 Disseny de la interfície gràfica 141
4.7 Millores en l'aplicació 144
5 Conclusions i observacions personals 147
5.1 Conclusions i observacions sobre el protocol HTTP Live Streaming 147
5.2 Desenvolupament per l'iOS 148
A1 Enllaços i bibliografia 151
A1.1 Enllaços 151
A1.2 Bibliografia 152
A2 Taula de reproductors i servidors compatibles 153
A2.1 Clients 153
A2.2 Servidors 154
A3 Traça del reproductor didàctic 156

Prefaci

En aquest document es descriuen les tasques dutes a terme dins d'una de les vessants d'un projecte del Departament d'Enginyeria Telemàtica, del professor Josep Pegueroles Vallès, i que va servir a la vegada per a la realització del meu Projecte Final de Carrera.

Objectiu del projecte

L'objectiu del projecte és, per una banda, l'estudi del protocol HTTP Live Streaming d'Apple. Aquest és un protocol de *streaming* de vídeo i àudio adaptatiu; que varia la tassa de bits del *stream* en dependència de la capacitat del medi pel qual es transmet. D'altra banda mostra el desenvolupament d'un client d'aquest protocol per a l'iPhone i, aprofitant el desconeixement inicial d'aquesta plataforma, la descripció dels passos necessaris per a desenvolupar aplicacions per a l'iOS, així com una introducció a les tecnologies emprades. L'aplicació està enfocada a l'ensenyament del funcionament del protocol HTTP Live Streaming i a la demostració didàctica de la seva capacitat adaptativa.

El projecte es completa amb altres tasques realitzades per companys del Departament d'Enginyeria Telemàtica amb aplicacions per a altres plataformes, com ara Android

Guia de lectura

La present memòria comença amb una introducció on es situa el protocol HTTP Live Streaming dins la historia del *streaming* i del consum de continguts multimèdia. En aquesta part es fa una breu introducció a l'*streaming* adaptatiu amb una visió de les seves capacitats i bondats. Al mateix temps es descriuen els principals protocols que, com l'HTTP Live Streaming, tenen la capacitat de ser adaptatius i es comparen amb l'estudiat.

Al segon capítol s'entra en detall en la descripció del protocol HTTP Live Streaming. Es descriu com està definit, es detalla la seva especificació i s'observen cadascuna de les seves característiques. En aquest capítol, el lector podrà veure l'evolució d'aquest protocol en el temps i les noves característiques que s'han anat afegint per tal d'enriquir l'experiència final del client.

El tercer capítol, aprofitant les hores dedicades a l'aprenentatge de la programació per l'iPhone, serveix com a introducció per a qualsevol programador nouvingut al món d'Apple i a "l'ecosistema" de dispositius d'iOS. Es tracta d'un conjunt de passos a seguir i de recursos per convertir-se en programador d'aquest sistema operatiu per a dispositius mòbils. Al final hi ha un petit programa d'aprenentatge pas a pas per a poder desenvolupar una primera aplicació per a l'iPhone, on es veu la manera de programar i de fer sevir els recursos del sistema.

El quart capítol descriu l'aplicació desenvolupada per a aquest projecte. Es descriuen tots els requeriments que s'han definit al començament i com s'han implementat cadascun d'ells. Des de la preparació dels continguts de servidor fins a la reproducció en el client d'aquests. Després es descriuen noves funcionalitats afegides que enriqueixen l'aplicació i l'ajuden a assolir el seu objectiu, que és facilitar l'aprenentatge del funcionament del protocol. Abans d'acabar, s'analitzen algunes parts interessants del desenvolupament, alguns problemes que s'han trobat durant la programació de l'aplicació i la manera en que s'han solucionat. I finalment s'assenyala el camí per a seguir per tal de millorar l'aplicació, amb la descripció en paral·lel d'algunes mancances d'aquesta.

Al darrer capítol es troben les observacions personals de l'autor sobre el protocol estudiat i sobre l'experiència de programar per primera vegada amb el sistema iOS.

El document és acompanyat d'un conjunt d'annexos necessaris per a complementar el projecte.

- Capítol 1 -

STREAMING DE CONTINGUTS AUDIOVISUALS

1 Streaming de continguts audiovisuals

A començaments dels anys 90 els ordinadors personals van començar a ser capaços de reproduir àudio i vídeo. Les primeres maneres de fer-ho eren a través de mitjans físics, com per exemple des d'un CD-ROM, o descarregant el fitxer des d'un servidor remot i desant-lo en el disc dur per poder reproduir-los després. El principal problema era el conflicte entre la mida dels fitxers que s'havien de descarregar i l'ample de banda limitat que hi havia en aquella època.

Durant els primers anys del segle XXI, les xarxes i les comunicacions van experimentar un gran increment de l'ample de banda. Els nous protocols, les noves tecnologies emprades a les comunicacions juntament amb les millores en els algoritmes de compressió i els avenços en potència computacional dels ordinadors personals van donar lloc a l'aparició de sistemes d'*streaming*.

La tecnologia d'*streaming* va aparèixer l'any 1995 amb el llançament de RealAudio 1.0.

1.1 Què entenem per *streaming*?

El terme *streaming* es fa servir per descriure la transmissió de dades d'una manera continuada amb la reproducció gairebé simultània de les dades descarregades a mesura que aquestes són rebudes. A diferència de la manera antiga on primer es descarrega totalment l'arxiu i després es reprodueix, la possibilitat de reproduir el contingut quan aquest comença a rebre's és el principal avantatge de l'*streaming*. L'usuari ja no ha d'esperar a que finalitzi una descàrrega que podria trigar hores depenent de la mida del fitxer.

1.2 On-Demand en vers Live-Streaming

Una de les diferents agrupacions sobre les classes d'streaming que podem trobar és la diferencia entre l'*streaming* sota demanda (*On-Demand*) i el *Live-Streaming*. En el cas del primer, *On-Demand*, els arxius han estat gravats prèviament i comprimits. Els arxius estan guardats en un servidor i són descarregats per diferents clients quan aquests els demanen. Hi ha

centenars de pàgines i serveis d'*streaming On-Demand*: YouTube, Microsoft Video, Vimeo, etc. són alguns exemples ben coneguts per tots nosaltres.

D'altra banda tenim el *Live-Streaming* (contingut en temps real). En aquest cas el contingut audiovisual es capturat, comprimit i transmès al mateix temps. Aquesta última modalitat dóna la possibilitat de retransmetre esdeveniments en temps real: competicions esportives, emissions de ràdio, conferències, etc. Per contrapartida, l'*streaming* en temps real requereix una gran potència computacional i, sovint, dispositius (*hardware*) dedicats.

Qualsevol servei d'*streaming* ofereix la possibilitat de pausa, rebobinar o, en el cas de que no sigui Live-Streaming, anar a moments posteriors de la reproducció, com qualsevol reproducció de continguts locals.

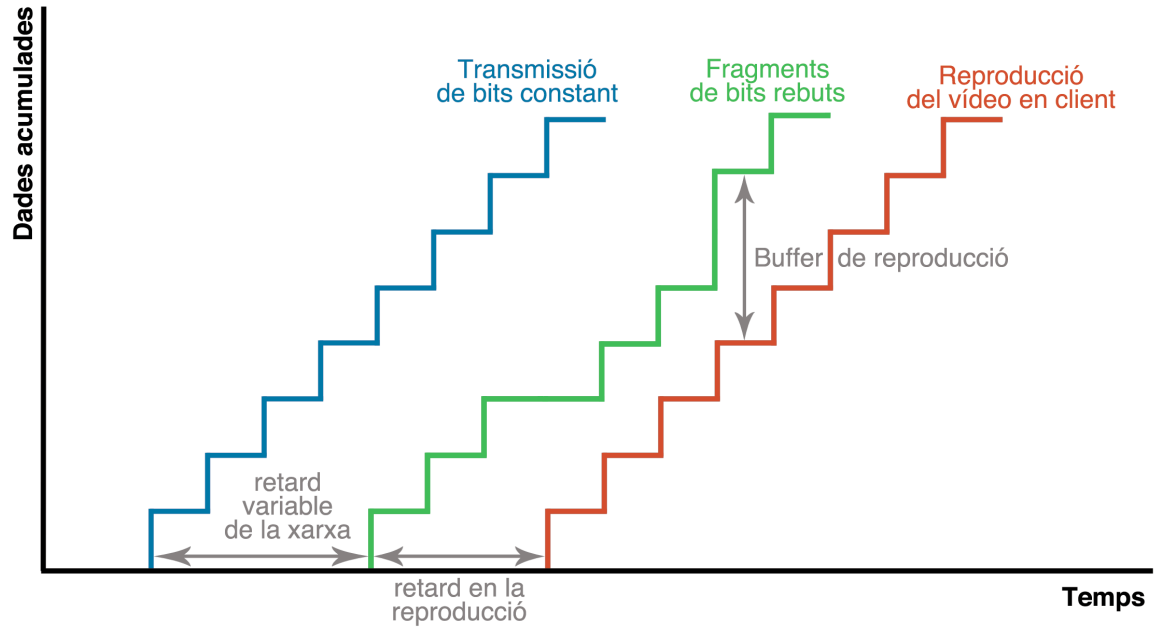

Fig 1.1 - **On-demand streaming**

En la figura 1 podem veure com funciona l'*streaming* sota demanda. En l'eix vertical hi ha l'acumulació de dades transmeses i en l'horitzontal el temps emprat per realitzar les operacions. D'esquerra a dreta: el primer que es veu és la transmissió constant de dades. Després les dades

són rebudes pel client amb diferents ritmes degut a les variacions del retard introduït per la xarxa. Degut a aquestes variacions hi ha d'haver un *buffer* de reproducció que introdueix un retard en la visualització del contingut.

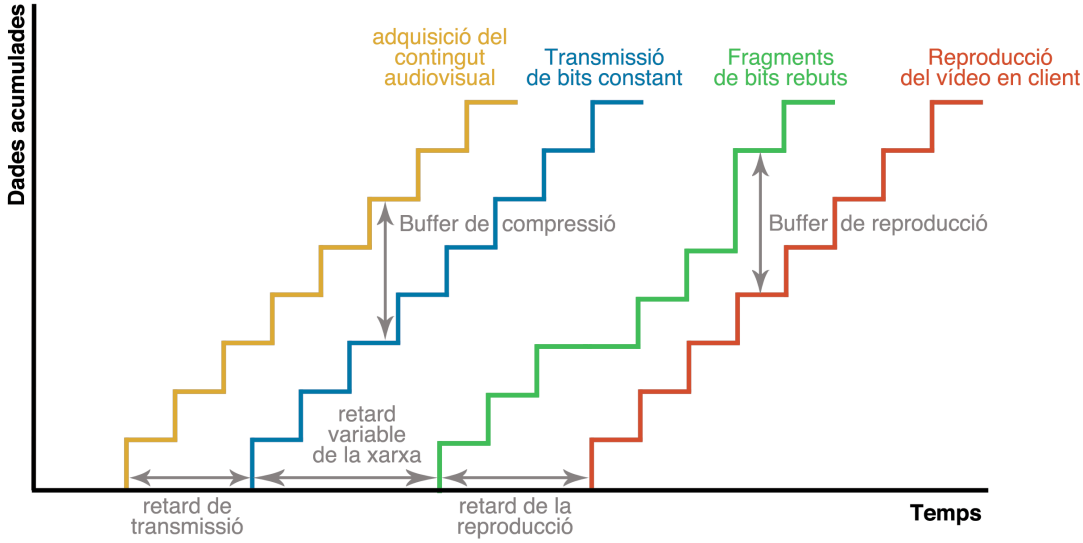

Fig 1.2 - **Live streaming**

En la figura 2 tenim el Live-Streaming. La diferència és que aquest afegeix un nou pas, ja que abans de transmetre les dades, primer s'han de capturar i comprimir. Aquest procés afegeix un nou retard degut al *buffer* de compressió. Un cop comprimit, el procés que segueix és el mateix que en el cas de sota demanda.

1.3 Solucions actuals

Actualment hi ha dos solucions àmpliament esteses en el mercat, aquestes són el RTP/RTSP i l'Adobe Flash Media Streaming Server. Tot i que aquestes tecnologies queden fora de l'àmbit d'aquest treball, cal fer una petita introducció d'aquestes per poder ubicar l'HLS (HTTP Live Streaming) en el context on es troba.

1.3.1 RTP/RTSP

El Real Time Streaming Protocol (RTSP) va ser creat originalment per Netscape i Real a finals dels anys 90. És un protocol de control per a controlar el servidor d'*streaming* de continguts. El

protocol es fa servir per establir i controlar sessions entre *end-points*. El client té comandaments de reproducció i pausa per facilitar un control de la reproducció dels fitxers audiovisuals des del servidor. RTSP va ser publicat com RFC 2326 el 1998.

La transmissió de l'*streaming* en sí no forma part del protocol RTSP. Alguns servidors de RTSP fan servir el Real-Time Protocol (RTP) per a la transmissió dels continguts però molts fabricants fan servir protocols de comunicacions propis. Per exemple, el servidor de RTSP de RealNetworks fa servir el seu protocol d'*streaming* RDT.

Un dels punts febles d'aquest protocol és l'ús per defecte del port 554, massa sovint bloquejat pels tallafocs. Generalment, les passarel·les dels proveïdors d'Internet per a llars tenen aquest port bloquejat.

1.3.2 Adobe Flash Media Streaming Server

El Flash Media Server (FMS) és un servidor propietari de dades i continguts audiovisuals d'Adobe Systems (originalment un producte de Macromedia). Aquest pot servir tant vídeo sota demanda com contingut en temps real. El portal de vídeos més gran del món, YouTube, el fa servir per proveir vídeos sota demanda.

El servidor de Flash Media Server treballa com a concentrador central amb aplicacions basades en Flash que s'hi connecten fent servir el Real Time Messaging Protocol (RTMP). El servidor pot enviar i rebre dades dels diferents usuaris connectats que tinguin instal·lat el reproductor de Flash.

Alguns dels inconvenients de fer servir aquesta solució són:

- El servidor Flash Media Streaming Server es propietari i suposa un cost de 995 US$.
- Els usuaris han de tenir instal·lat el reproductor de Flash. Però avui en dia ja no és un problema, donatque la majoria dels usuaris ja el tenen.

- El reproductor de Flash no està del tot ben implementat en algunes plataformes. Les implementacions per sistemes Linux i Mac OS tenen problemes de sobrecàrrega de la CPU i, en dispositius portàtils, requereix un elevat consum de les bateries. Aquesta és la raó per la qual Apple va decidir no implementar el reproductor de Flash en els seus dispositius portàtils. Actualment Adobe també a decidit no proveir una implementació del reproductor Flash per a les noves versions d'Android. Aquests són indicis de la tendència a no fer servir tan aquesta solució en el futur.

1.4 L'era dels smartphones i la mobilitat

Amb els avenços actuals en matèria de telefonia mòbil, la quantitat d'usuaris amb telèfons intel·ligents (smartphones) equipats amb sistemes operatius com Android, iOS o el nouvingut al mercat Windows Phone, ha crescut de forma exponencial als darrers anys. La gran capacitat computacional d'aquests dispositius els permet ser uns grans consumidors de continguts multimèdia i dóna lloc a l'aparició d'un nou problema: la gran variació d'ample de banda que pateixen les connexions d'aquests dispositius quan es mouen per diferents ambients i xarxes.

Al tenir una gran variació de l'ample de banda, els serveis de *streaming* convencionals queden molt exposats a patir grans pauses en la reproducció de continguts. Aquest problema ha propiciat l'aparició del que anomenem sistemes de *streaming* adaptatiu. La característica principal d'aquests sistemes és que serveixen contingut audiovisual amb un tassa de bits variable segons l'ample de banda.

1.4.1 Streaming adaptatiu a Internet

Si abans hem comentat que els sistemes més estesos al fer *streaming* multimèdia utilitzen protocols com el RTP amb RTSP, quan parlem d'*streaming* adaptatiu gran part d'aquests sistemes estan basats en HTTP i preparats per treballar eficientment sobre grans xarxes distribuïdes, com Internet.

Els protocols IP RTSP, RTP o RTMP per a Flash són molt més eficients en quan a consum d'ample de banda en el servidor i el client, però ho són a canvi d'una més gran complexitat en els

sistemes del servidor i amb elevats requeriments de CPU, memòria, bateries, etc. en els dispositius.

D'altre banda, amb la recent aparició de l'iOS i la oposició de Steve Jobs (Apple) a donar suport a Flash en els seus dispositius, així com la decisió final sobre el *codec* a fer servir en l'especificació HTML5 (per a l'etiqueta <video>), s'ha popularitzat l'adopció de protocols de distribució de vídeo basats en HTTP.

L'*streaming* adaptatiu funciona mitjançant la detecció de l'ample de banda de l'usuari i la capacitat computacional en temps real, i l'ajust de la qualitat del *stream* de vídeo d'acord amb els paràmetres detectats. Aquest sistema requereix l'ús d'un dispositiu que codifiqui una única font de vídeo en diverses, cadascuna amb una tassa de bits diferent. El reproductor del client escull quina de les diferents qualitats descarregar depenent dels recursos disponibles. El resultat d'aquest procés és que no hi ha gairebé pauses a la reproducció ("*buffering*"). El començament de la reproducció és molt ràpid i ofereix una bona experiència tan per a usuaris amb pocs recursos, com per a usuaris amb dispositius potents i un elevat ample de banda.

Més específicament, i seguint les implementacions que s'estan fent servir actualment, el *streaming* adaptatiu és un mètode de *streaming* de vídeo sobre HTTP on el contingut és codificat en diverses fonts a tasses de bits diferents, cada una d'aquestes es segmentada en petites parts de pocs segons. Al client se l'informa d'aquestes diferents opcions o qualitats i dels seus segments per mitjà d'un manifest. Quan el client comença el procés de reproducció, es descarrega primer el segment de tassa de bits menor. Si el client detecta que la velocitat de descàrrega és prou gran per a poder consumir els segments d'una qualitat més elevada, el segment següent el descarrega amb la qualitat més gran possible. Més endavant, si el client troba que la velocitat de descàrrega per a un segment és més petita que la tassa de bits d'aquest, això voldrà dir que la velocitat de la xarxa s'ha deteriorat i escollirà la descàrrega del següent segment a una qualitat inferior; la primera inferior a la tassa de bits de la xarxa o la més baixa possible. La mida dels segments pot variar segons de les diferents implementacions, però normalment es troba entre 2 i 10 segons.

1.4.2 Implementacions actuals

El concepte de *Streaming* adaptatiu és una creació de Move Networks, Inc. ("Move"), una empresa internacional de serveis de streaming de canals de televisió per Internet. El concepte de Move està en desenvolupament i es fa servir, sobretot, a tres grans empreses: Microsoft, Adobe Systems i Apple.

Tots fan servir el mateix *codec,* el MPEG-4 H.264, també conegut com AVC (Advanced Video Coding). El H.264 és un dels formats més emprats per codificar, comprimir i distribuir vídeo d'alta definició. En el cas de l'àudio, aquest és codificat en format AAC (Advanced Audio Coding).

Existeixen tres implementacions principals: **HTTP Live Streaming** d'Apple, **Smooth Streaming** de Microsoft i **HTTP Dynamic Streaming** de Adobe. Totes tres fan servir MPEG-4 H.264 com a font d'entrada. I totes tres fan servir una arquitectura i un flux de treball similar:

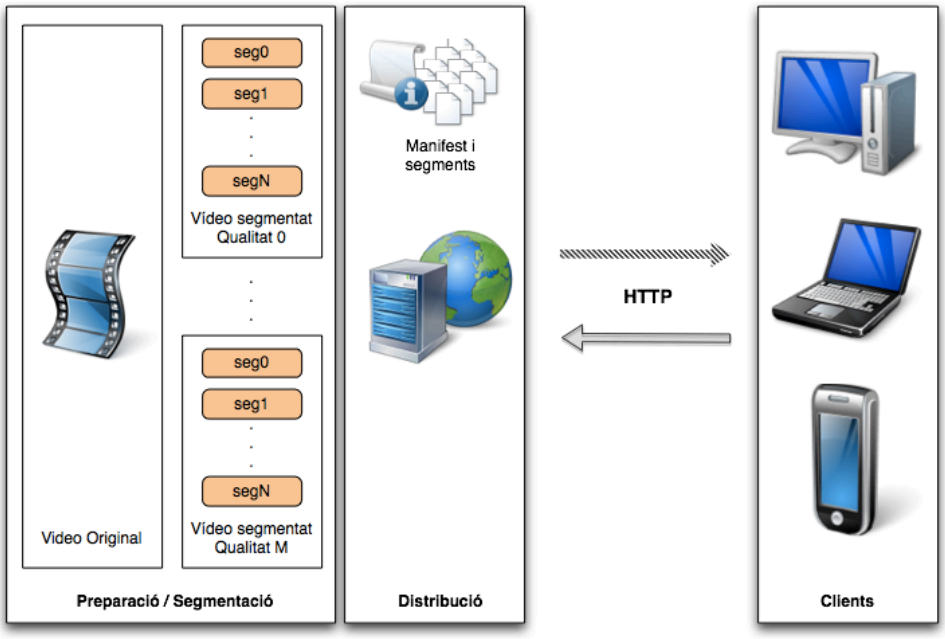

Fig 1.3 - **Parts d'un sistema d'*streaming* adaptatiu sobre HTTP**

La primera part es basa en la preparació dels continguts. En aquesta primera fase partim d'un vídeo d'origen on es fa servir un codificador (*encoder*) per obtenir diferents qualitats de vídeo

(diversos *streams* amb diferents tasses de bits), codificades en MPEG H.264 amb l'àudio en AAC. Després s'afegeix un petit programa anomenat *segmenter,* que divideix els diferents *streams* en petits trossos de la mateixa durada.

Un cop dividits els trossos com a fitxers, es distribueixen mitjançant servidors HTTP tradicionals (IIS, Apache, etc.) juntament amb un manifest que inclou tota la informació necessària per a que el client sigui capaç de realitzar el procés de reproducció adaptativa. Aquest fitxer indica quines són les diferents fonts de vídeo i les seves qualitats (tasses de bits), així com la direcció dels diferents segments i el seu ordre. Segons el tipus d'implementació, el fitxer pot variar de format i pot contenir molta més informació.

Els clients es descarreguen el manifest i descarreguen també per HTTP els diferents segments que necessiten. El client ha de ser capaç de saber decidir quins segments s'ha de descarregar i de presentar-los en un *stream* de vídeo continuat.

En el cas de l'**HTTP Live Streaming** (HLS), els codificadors prenen el vídeo i l'àudio d'entrada, els codifiquen en H.264 i AAC, i els retornen encapsulats en un arxiu MPEG-2 TS (Transport Stream ".ts"). Després, el segmentador (*segmenter)* divideix l'arxiu .TS en una serie de petits fixers d'uns 10 segons de durada (també anomenats *chunks*). El servidor web també conté i manté el manifest, en aquest cas un fitxer amb extensió .M3U8, que conté la llista de fitxers .TS en què s'ha dividit el vídeo original. La direcció d'aquest arxiu .M3U8 es publica al servidor web. Aquest procés es repeteix per a cadascuna de les qualitats en les que es decideixi emetre el vídeo. Tot aquest procés es descriu amb més detall al capítol següent.

L'**HTTP Dynamic Streaming** és més o menys el mateix, però es diferencia de l'HLS en alguns punts. L'etapa del *segmenter* la fan eines pròpies de l'Adobe Flash Media Server, té suport per a contenidors FLV i l'usuari no es descarrega completament el fitxer per a veure el vídeo com a l'HLS. Generalment, el client reprodueix el vídeo fent servir el component Flash.

En el cas de l'**HTTP Smooth Streaming** de Microsoft, la diferència es troba a l'etapa del *segmenter*, en aquest cas són eines pròpies del IIS (Internet Information Services) les quals s'encarreguen de les tasques de codificació i segmentació. També suporta Silverlight com a contenidor de vídeo.

Tots tres solucions suporten *streaming* en directe (Live Streaming), seguint el mateix fluc de treball o *workflow*.

1.4.3 Beneficis del *streaming* adaptatiu

Els primers beneficiats de l'*streaming* adaptatiu són les grans empreses d'entreteniment i de serveis audiovisuals. L'espai per poder emmagatzemar grans quantitats de dades, vídeo en aquest cas, creix exponencialment. Les xarxes de lliurament de continguts CDN (Content Delivery Networks) i els proveïdors de vídeo poden oferir als clients una experiència superior. La tecnologia d'adaptar la tassa de bits, que al cap i la la fi és el que fa l'*streaming* adaptatiu, requereix codificadors més senzills; simplifiquen tot el procés i ofereixen uns resultats superiors.

Els CDN distribueixen normalment els continguts per a tots els usuaris d'Internet, la qual cosa requereix escal·labilitat. El CDN rep el contingut ja segmentat i preparat i el replica a tots els seus servidors 'Edge' de caché. La petició de l'usuari és redirigida al servidor 'Edge' més proper. L'us de l'*streaming* adaptatiu basat en HTTP fa que el servidor Edge tan sols hagi de tenir un únic servidor HTTP, la llicencia del qual és pràcticament gratuita i redueix les despeses de llicències, sobretot si ho comparem els les llicencies de servidors de continguts audiovisuals (per exemple el servidor de Adobe Media Streaming) d'alt cost. El cost del CDN per a fer streaming sobre HTTP és similar al cost del CDN de caché de pàgines web; la diferència es troba en la quantitat de capacitat necessària per poder emmagatzemar els vídeos en les diferents qualitats, que és força elevada.

Val a dir que, com que el client s'ha de descarregar temporalment el fitxer de vídeo - el segment - aquest sistema no és tan robust com els altres davant la pirateria de continguts.

Els usuaris i consumidors dels continguts servits mitjançant un streaming adaptatiu obtenen la millor qualitat possible donat que la reproducció s'adapta als recursos disponibles en cada moment. A més, el cost de CPU i de memòria és menor. D'altra banda, el client s'ha de descarregar completament cada segment per a poder veure'l. En l'*streaming* normal només es descarrega la part que es veu. No obstant, aquesta afirmació no és vàlida per a l'HTTP Dynamic Streaming d'Adobe.

Per tant, els mètodes d'*streaming* adaptatiu sobre HTTP milloren les despeses per als editors, encara que generin un tràfic de dades més elevat per al client final.

Com a referència important, podem esmentar diversos emissors de vídeo d'Estats Units, com la ABC, Netfix o Hulu que estan adoptant l'HLS en les aplicacions per iPad. A Anglaterra, la BBC està fent proves d'HD HTTP Streaming. I a Espanya, el Grupo Prisa ja fa servir també l'*encoder* Envivio per a produir HLS.

- Capítol 2 -

HTTP LIVE STREAMING

2 HTTP Live Streaming (HLS)

HTTP Live Streaming (també conegut per les seves sigles HLS) és un protocol d'*streaming* de continguts audiovisuals basat en HTTP i creat per Apple Inc., que forma part de l'ecosistema format per QuickTime X i iOS.

2.1 Objectiu

Apple va desenvolupar el HLS amb la finalitat de poder reproduir vídeos i música en *streaming* per a la versió 3.0 del seu sistema operatiu per dispositius mòbils, l'iOS. Aquest suporta tant la reproducció d'elements prèviament emmagatzemats com la de retransmissions en directe. L'objectiu principal és aconseguir un protocol capaç de fer *streaming* de continguts a l'iPhone, l'iPod Touch, l'iPad o a l'Apple TV, i de fer-ho sense necessitat de cap *software* especial en els servidors. El protocol està definit per treballar sobre HTTP, per poder ser accessible des de qualsevol lloc, sense haver de preocupar-se per problemes de tallafocs o servidors *proxy*; a diferència dels protocols basats en UDP com el RTP. Ja que està realitzat sobre HTTP, l'HLS permet servir els vídeos o àudios fàcilment a través de xarxes de distribució de continguts (Content Delivery Networks, CDN). A més, si cal, es pot fer en mitjans xifrats i amb autenticació.

Tot i que el sistema d'Apple ha estat creat enfocat a la reproducció del vídeos en els seus dispositius, donat que no està lligat a cap *software* especial (ni propietari), ni per part del servidor ni pel part del client, permet el desenvolupament d'eines pròpies per fer servir aquest protocol i implementar-lo en multitud de sistemes.

2.2 Arquitectura

Conceptualment l'HLS consisteix en tres parts: un component de servidor, un component de distribució i un component de client.

El **component de servidor** és el responsable d'aconseguir els *streams* multimèdia d'origen i codificar-los digitalment, encapsular-los en formats definits pel protocol i preparar-los per a la seva distribució.

El **component de distribució** és un simple servidor web estàndard. És el responsable d'acceptar les peticions de l'usuari i lliurar els continguts i els arxius associats degudament preparats a l'usuari. Per a una distribució a gran escala d'aquests continguts es poden fer servir xarxes de servidors Edge o altres xarxes de distribució de continguts.

El **component del client** és el responsable de determinar el recurs que s'ha de demanar, descarregar aquest recurs i llavors reproduir els arxius descarregats de tal manera que els pugui presentar a l'usuari com un *stream* continuu. El *software* necessari per fer-ho esta inclòs en tots els dispositius d'Apple amb la versió 3.0 o superior del sistema operatiu o en ordinadors amb el navegador Safari 4.0 o superior. També són capaços de reproduir-ho altre sistemes com es pot veure en l'annex A2.

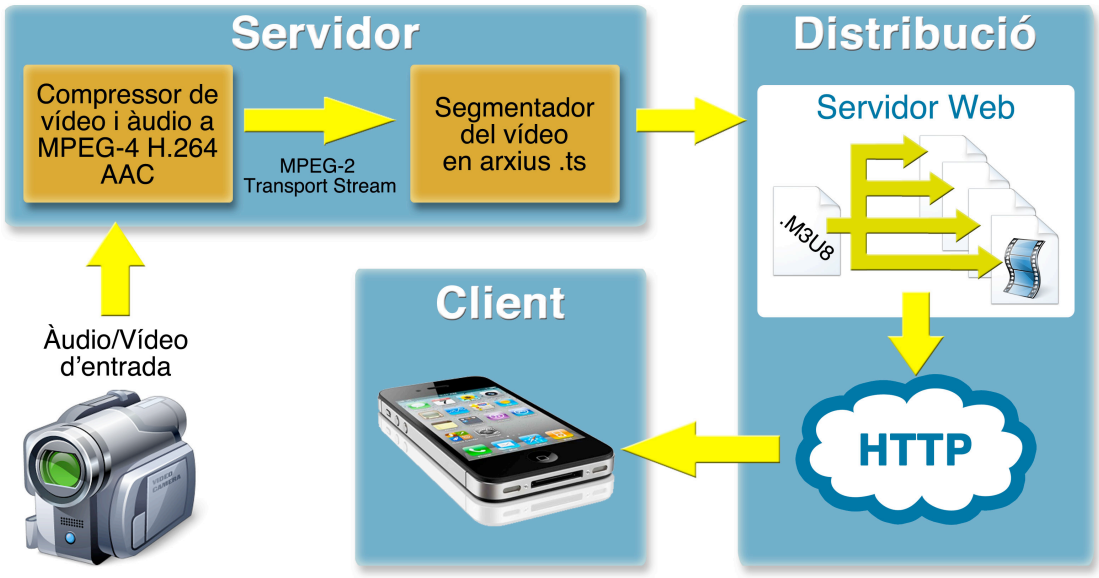

Fig. 2.1 - **Configuració bàsica d'un sistema HLS**

L'entrada al sistema pot ser la captura d'una càmera o gravadora d'àudio o un fitxer prèviament guardat. Normalment es comprimeix i codifica en format MPEG-4 (el vídeo en H.264 i l'àudio en

AAC) i encapsula en una trama de transport MPEG-2 Transport Stream (TS). El resultat es divideix en petits segments i guardat en sèries d'un o més fitxers amb l'extensió *.ts*. Això és típicament realitzat per una eina dedicada, com per exemple l'*Stream Segmenter* d'Apple. A partir d'ara ens referirem a aquest component *software* com a ***segmenter***.

Els *streams* que només contenen àudio poden ser sèries de fitxers d'àudio MPEG elementals o AAC amb capçaleres ADTS o MP3.

El *segmenter* també ha de crear un fitxer d'índex [punt 2.3]. Aquest fitxer conté la llista dels fitxers .ts i altres dades addicionals (*metadata*). És un fitxer amb extensió .M3U8, una llista de reproducció. La direcció URL del fitxer és accessible per l'usuari, el qual demana els fitxers indexats en la seqüència.

2.2.1 Component de servidor

El primer que hem de tenir és una eina per convertir el fitxer de vídeo, o la captura que estem fent d'aquest, al format definit pel protocol: MPEG-4 H.264 i AAC. Aquesta eina pot ser un programa del sistema o, a nivells més professionals, una màquina específica, un component *hardware* dedicat i construït per realitzar aquest tipus d'operacions. Aquesta eina s'anomena *encoder*. Un cop convertit, l'*encoder* ha d'encapsular el resultat en una trama de transport en format *MPEG-2* [1] *Transport* Layer.

És molt important que l'*encoder* no modifiqui en absolut les dimensions del vídeo (s'ha de mantenir la proporció d'aquest) o el tipus de *codec* en totes les qualitats en les que es vol distribuir el contingut. Una variació d'aquests paràmetres entre les diferents qualitats faria

[1] No s'ha de confondre el MPEG-2 Transport Stream amb la compressió de vídeo en MPEG-2. El que anomenem Transport Stream és una mena de paquet, un contenidor de dades en un format especificat, que es pot fer servir per emmagatzemar diferents tipus de compressions. En aquest cas, tan sols aquells encapsulats en una capa MPEG-2 Transport Stream amb el contingut comprimit en H.264 i AAC són vàlids dins de l'especificació del HLS. Si només hi ha àudio, aquest podria ser encapsulat en una capa de transport MPEG-2, o trames bàsiques en MPEG, o directament en format AAC amb capçaleres ADTS o en format MP3.

impossible que la reproducció fos continua amb segments de diferents qualitats. Donaria lloc a salts en l'àudio i el vídeo i no seria viable per a la implementació de les versions més avançades del protocol, on entren en joc els anomenats I-Frames [punt 2.5.3].

Per a realitzar i preparar el vídeo per a posar-lo a disposició mitjançant l'HLS, segons l'especificació (enllaç 1 de l'annex A1), el vídeo d'origen només s'ha de comprimir en el format especificat, segmentar-lo i preparar un arxiu .M3U8 amb la llista dels diferents arxius .TS seqüencialment. Però amb només una qualitat de sortida el client no tindria opcions per a escollir en el moment de crear un sistema adaptatiu i no tindria sentit realitzar aquest protocol. Així doncs, el procés realitzat per l'*encoder* s'ha de repetir amb diferents qualitats del vídeo d'origen. Això dóna com a resultat un fitxer .TS per a cada qualitat. El fitxer de qualitat més alta té una mida d'arxiu molt més elevada que el de qualitat inferior, per tant més bits per a transmetre en un mateix espai de temps. El client pot escollir així entre diferents opcions per adaptar-se l'ample de banda en cada moment.

Per altra banda es necessita una altra eina per dividir el vídeo comprimit en petits segments i guardar-los en diferents arxius amb l'extensió .TS: el *segmenter*.

Hi ha dos tipus de *segmenters*: el ***stream segmenter*** i el ***segmenter* d'arxiu**. El ***stream segmenter*** és un procés (generalment un programa) que llegeix la informació d'un *stream* de transport de la xarxa local (una càmera digital o IP) i la divideix en series de petits fitxers de la mateixa duració. Tot i que cada segment està en un fitxer, donat que aquests han estat creats a partir d'un vídeo continuu, si és reprodueixen l'un darrera l'altre obtenim el vídeo complet. Aquests fitxers són guardats amb l'extensió .TS.

L'*stream segmenter* també crea un manifest, un fitxer que conté referències als diferents segments. Cada cop que el *segmenter* completa un segment, actualitza el fitxer afegint la referència a aquest últim segment i elimina les referències als segments que ja no existeixen. Aquest fitxer serveix per saber on es troben els diferents segments i si estan accessibles. També pot servir per xifrar cada un dels segments i crear una clau com a part del procés. Aquest fitxer

d'índex és guardat amb extensió .M3U8. El fitxer i la seva creació es descriu amb més detall en els punts 2.3 i 2.4.1.1.

Si el contingut ja està guardat i codificat fent servir els *codecs* suportats, es pot utilitzar el ***segmenter* d'arxiu** (*file segmenter*) per encapsular el vídeo d'origen en un *stream* i dividir-lo en petits segments de la mateixa duració. El *file segmenter* fa servir els fitxers locals d'àudio i vídeo i els prepara per a ser consumits via HTTP Live Streaming. Aquest *segmenter* realitza les mateixes tasques que l'*stream segmenter*, però fa servir fitxers com a entrada en comptes de *streams.* L'altre diferencia està en el fitxer M3U8; mentre que l'*stream segmenter* va afegint les referències als segments creats a mesura que els captura i crea, el *file segmenter* crea el .M3U8 amb una sola operació, ja que aquest és estàtic i no dinàmic, els segments sempre estan accessibles i tots es troben en el fitxer.

Els fitxers .M3U8, anomenats també fitxers d'índex o manifests, són creats (normalment) per mateix *segmenter* i guardats com a llista de reproducció .M3U8. L'extensió .m3u [2] es fa servir per a les llistes de fitxers MP3.

Un exemple molt senzill d'un fitxer d'índex: si el vídeo té una duració de 30 segons exactes i el *segmenter* ha creat segments de 10 segons (sense cap mena de xifrat):

```
#EXTM3U8
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:1
#EXTINF:10,
http://servidorweb.com/video1/segment0.ts
#EXTINF:10,
http://servidorweb.com/video1/segment1.ts
#EXTINF:10,
http://servidorweb.com/video1/segment2.ts
#EXT-X-ENDLIST
```

[2] Donat que el fitxer d'índex pot tenir una extensió .m3u i el sistema HLS suporta fitxers .mp3, el client ha de ser compatible amb el típic format de les llistes de reproducció de MP3 que es fan servir per fer streaming d'àudio a Internet.

El fitxer d'índex també pot contenir URLs per les claus de xifrat i referències a altres fitxers d'índex per a diferents amplades de banda. La informació emmagatzemada en els fitxers .M3U8 i el seu format estan documentats al punt 2.3.

2.2.2 Component de distribució

El sistema de distribució és un servidor web o un sistema de *cache* que serveix els fitxers de vídeo o àudio i els fitxers d'índex al client mitjançant HTTP. No es necessita cap mena de mòdul dedicat per servir el contingut i, normalment, el sistema pot estar en funcionament amb molt poca configuració. Només s'ha d'indicar al servidor quina mena de fitxer servirà.

La configuració recomanada es limita a especificar l'associació de MIME-Type als fitxers .M3U8 i .TS.

Extensió del fitxer	*MIME Type*
.M3U8	application/x-mpegURL o vnd.apple.mpegURL
.TS	video/MP2T

En el cas de voler fer *streaming* en temps real, l'*stream segmenter* va actualitzant el fitxer .M3U8 amb ña referència als nous segments i s'han de configurar els valors del TTL (time-to-live) del servidor per als fitxers .M3U8 per aconseguir el *cache* desitjat.

2.2.3 Component de client

El primer que fa el component de client és descarregar el fitxer d'índex, basat en una URL identificadora de l'*stream* a reproduir. El fitxer d'índex especifica la localització dels segments disponibles, les claus de xifrat (si n'hi ha), i els *streams* alternatius disponibles. Per a un *stream* seleccionat (una qualitat de vídeo), el client es descarrega cadascun dels segments en seqüència. Un cop el client ha descarregat prou dades, aquest comença a reproduir els diferents segments d'una manera continua, com si es tractés d'un sol fitxer.

El client és el responsable de buscar les claus de xifrat, autenticar-se o presentar una interfície d'usuari per autenticar-se, i desxifrar els fitxers quan sigui necessari. També és responsable d'escollir l'*stream* necessari per adaptar la reproducció a l'ample de banda de l'usuari en cada moment.

El procés continua fins que el client troba la paraula clau `#EXT-X-ENDLIST` en el fitxer d´índex. Si no hi ha la paraula clau significa que el fitxer forma part d'un *streaming* en temps real. Durant un *streaming* en temps real, el client ha d'anar descarregant noves versions del fitxer d'índex periòdicament fins que trobi la paraula clau que indica que ja no hi ha més segments: `#EXT-X-ENDLIST`.

Tot aquest procés del client està documentat en el punt 2.4.2.

2.3 El fitxer d'índex

El fitxer d'índex o *playlist*, amb extensió .M3U8m és una extensió del *M3U Playlist (Extended M3U)* al qual s'hi afegeixen noves etiquetes (paraules clau). La informació que conté aquest fitxer és tota la necessària per a que un client, un cop interpretat el contingut d'aquest arxiu, sigui capaç de desenvolupar tot el procés necessari per l'streaming adaptatiu HLS.

El *M3U Playlist* és un fitxer de text que conté diverses línies individuals. Cada línia va separada pel caràcter LF o pel caràcter CR seguit de LF. Cada línia pot ser un enllaç, una línia en blanc o una línia que comença amb el caràcter '#'. Les línies en blanc són ignorades i no poden haver espais en blanc si no estan definits dins d'un element clau.

En el cas de la línia de l'enllaç, aquest ha de fer referència a un segment .ts o bé a un altre fitxer d'índex. Aquests enllaços poden ser relatius; en aquest cas la URI completa de l'enllaç es construirà fent servir la direcció de la *playlist* que la conté. Cada segment o fitxer d'índex està definit per dues línies: una amb una sèrie de paraules clau que el defineixen i l'altre amb l'enllaç al segment o a la *playlist*.

Les línies que comencen amb el caràcter '#' poden ser tant comentaris com paraules clau. La diferència depèn de quins caràcters segueixen al '#'. Si la línia comença amb #EXT es tracta d'una paraula clau. Totes les altres són línies amb comentaris i són ignorades pel client en el moment d'interpretar el contingut del fitxer d'índex.

Algunes de les paraules clau necessiten d'un conjunt d'atributs que vindran definits per parells de valors separats per comes sense espais en blanc.

El conjunt de paraules claus definides pel protocol ha anat augmentant a mesura que el protocol ha anat evolucionant.

2.3.1 Primera versió del protocol. Paraules clau.

Les primeres paraules clau que veurem són les necessàries per crear un sistema HTTP Live Streaming bàsic. En apartats posteriors es descriuen noves paraules clau. Aquestes han estat afegides en diverses revisions del protocol i afegeixen noves funcionalitats al protocol HLS bàsic.

EXTM3U

Tot fitxer *Extended M3U* ha de començar amb aquesta paraula clau. Aquest paraula clau s'ha de situar en la primera línia del fitxer perquè aquest sigui vàlid.

```
#EXTM3U
```

EXTINF

La paraula clau EXTINF especifica la duració d'un segment i tan sols aplica al segment al qual fa referència l'enllaç de la línia següent. Cada enllaç a un segment ha d'estar precedit d'aquesta paraula clau. El format de la paraula clau és el següent:

```
#EXTINF: <duració>, <títol>
```

El primer atribut, 'duració', indica el temps exacte del segment en segons, arrodonit al número sencer més proper. El segon atribut és opcional i serveix per posar-li un títol al segment.

EXT-X-TARGETDURATION

Aquest atribut especifica la duració màxima d'un segment. La duració indicada en l'atribut anterior EXTINF ha de ser igual o inferior que la indicada en aquest. Aquesta paraula clau tan sol apareix un sol cop i s'aplica a la totalitat del fitxer d'índex. El seu format és el següent:

```
#EXT-X-TARGETDURATION: <segons>
```

L'atribut 'segons' ha ser un número sencer.

EXT-X-MEDIA-SEQUENCE

Cadascun dels enllaços als segments d'una llista de reproducció té un número de seqüència. L'atribut EXT-X-MEDIA-SEQUENCE índica quin és el número de seqüència del primer enllaç que apareix en el fitxer. Els números de seqüència dels altres segments s'assignen de forma consecutiva. Aquest atribut només pot aparèixer un cop a cada fitxer d'índex o pot no ser-hi; en aquest cas al primer segment se li hauria d'assignar el número zero.

```
#EXT-X-MEDIA-SEQUENCE: <número>
```

La variable "número" ha de ser un número sencer positiu.

Aquesta paraula clau entra en joc quan es fa servir un *stream segmenter*. Ja que es tracta d'un *streaming* en temps real que actualitza el fitxer d'índex cada cert temps afegint enllaços a segments nous i eliminant-los a segments antics, és necessari que el client pugui saber quin és el número de seqüència del primer segment per poder seguir el procés adequadament.

EXT-X-KEY

Els segments poden estar xifrats i aquesta paraula clau indica cóm desxifrar-los. Si EXT-X-KEY no apareix al fitxer d'índex, vol dir que els segments no estan xifrats. Aquesta paraula clau s'aplica a cadascun dels enllaços a segments que es troben entre dos EXT-X-KEY. El seu format és:

```
#EXT-X-KEY: METHOD=<mètode>, URI=<enllaç>
```

El primer dels dos atributs indica el mètode de xifrat emprat. Actualment només hi ha dos valors possibles; NONE o AES-128. En el cas de NONE, l'altre atribut ja no apareix, donat que no existeix cap xifrat.

Si el mètode de xifrat és AES-128, vol dir que els segments estan xifrats segons l'anomenat Advanced Encryption Standard (AES) amb una clau de 128 bits i PKCS7 d'esquema d'emplenament (PKCS7 *padding*). Si el mètode no és NONE, llavors hi ha d'estar l'atribut URI, que indica la manera d'obtenir la clau de desxifrat mitjançant un enllaç.

El format de la clau de xifrat i la forma de xifrar els segments queden fora de l'àmbit del projecte (enllaços 1 i 2 de l'annex A1).

EXT-X-PROGRAM-DATE-TIME

Aquesta paraula clau associa un segment a un instant de temps i s'aplica només al primer enllaç que la precedeix. El seu format és:

```
#EXT-X-PROGRAM-DATE-TIME: <instant de temps>
```

L'atribut "instant de temps" segueix la norma ISO/IEC 8601:2004 (`YYY-MM-DDThh:mm:ssZ`) i pot indicar també la zona horària. Un exemple:

```
#EXT-X-PROGRAM-DATE-TIME: 2012-07-09T12:04:48.031+02:00
```

Aquesta paraula clau es fa servir per poder associar el contingut que s'està reproduint en el temps, per exemple, el contingut que s'està gravant (en el cas d'una emissió en temps real).

EXT-ALLOW-CACHE

Per poder indicar si el client pot o no guardar els segments descarregats per poder reproduir-los més endavant es fa servir la paraula clau EXT-ALLOW-CACHE. Només pot aparèixer un sol cop al fitxer d'índex i s'aplica a tots els segments d'aquest.

```
#EXT-ALLOW-CACHE: <YES|NO>
```

Si l'atribut és YES o la paraula clau no hi és, el client pot desar els segments descarregats. Si el valor es NO, el client ha d'esborrar els segments un cop reproduïts.

EXT-X-ENDLIST

Aquesta paraula clau indica que ja no hi ha més segments. Pot aparèixer en qualsevol punt del fitxer, encara que normalment es troba al final, però només pot aparèixer un sol cop al fitxer d'índex.

A l'*streaming* de continguts sota demanda, donat que tots els segments es troben al mateix fitxer d'índex, EXT-X-ENDLIST es troba dins del fitxer. En el cas de *streaming* en temps real, on el fitxer d'índex es va actualizant a mesura que passa el temps, només apareix a la darrera versió d'aquest i indica al client que ja no cal que actualitzi el fitxer més cops, doncs ja disposa dels enllaços a tots els segments.

```
#EXT-ENDLIST
```

EXT-X-STREAM-INF

Així com la paraula clau EXTINF dóna informació (duració i títol) sobre el segment enllaçat que la precedeix , la paraula clau EXT-X-STREAM-INF dóna informació sobre el fitxer d'índex enllaçat que el precedeix.

Hi ha dos tipus de llistes de reproducció M3U8, aquelles que contenen els enllaços a segments d'un vídeo (o àudio) i aquelles que contenen una sèrie de enllaços a llistes de reproducció (fitxers d'índex). Aquestes últimes són les que contenen la paraula clau EXT-X-STREAM-INF i serveixen per indicar les diferents opcions (o qualitats) per a un mateix contingut.

Tot i que el sistema funcionaria perfectament amb un sol fitxer d'índex amb la llista de segments, només hi hauria una qualitat i el client no tindria opcions per adaptar-se a un ample de banda variable.

```
#EXT-X-STREAM-INF: BANDWIDTH=<ampla de banda>, PROGRAM-ID=<identificador>,
CODECS=<codecs>
```

Els diferents atributs defineixen l'*stream* enllaçat (.m3u8) que el precedeix.

L'ampla de banda és un número decimal que representa els bits per segon del vídeo codificat en els segments del *stream*. Aquest és l'atribut que el client consulta per escollir la *playlist* d'on agafar el segment següent. BANDWIDTH és un atribut obligatori.

PROGRAM-ID es opcional, el valor pot ser tant un número decimal com sencer i és tan sols un identificador per al contingut audiovisual que contenen els segments. Com els diversos enllaços son fitxers d'índex del mateix contingut però amb qualitats diferents, l'identificador serà el mateix per a totes les línies EXT-X-STREAM-INF d'un fitxer d'índex. Així que es tracta d'un valor informatiu.

L'últim d'aquests tres atributs és CODECS. És una cadena de text entre dobles cometes que conté la llista de *codecs* fets servir en la compressió del contingut del segment. Si el fitxer conté vídeo o àudio serà un sol *codec*. Si el segment conté vídeo i àudio, en seran dos, separats per una coma. Per exemple:

```
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=330768,CODECS="mp4a.40.5, avc1.4d401e"
```

Cadascuna de les línies EXT-X-STREAM-INFO hauria de incloure l'atribut CODECS, però pot ser omès.

2.3.2 Exemples

Arribats a aquest punt, ja tenim totes les paraules clau definides per poder crear els fitxers d'índex necessaris per desenvolupar un sistema HTTP Live Streaming.

Per poder veure més clarament aquest funcionament i la informació que rep un client quan es descarrega aquest tipus de fitxer, analitzarem un petit exemple:

```
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:10,
fileSequence0.ts
#EXTINF:10,
fileSequence1.ts
#EXTINF:10,
fileSequence2.ts
#EXTINF:4,
fileSequence4.ts
#EXT-X-ENDLIST
```

L'arxiu comença amb la paraula clau EXTM3U i el número de seqüència del primer segment és 0, tal i com ens informa la línia amb la paraula clau EXT-X-MEDIA-SEQUENCE. També conté la paraula EXT-X-ENDLIST al final del fitxer. Amb aquestes tres dades el client sap que es tracta d'un *straeming* sota demanda o, en el cas de ser en temps real, ja disposa de tots els segments que formen el contingut. Al no haver-hi cap línia amb la paraula clau EXT-X-KEY no hi ha xifrat. La paraula clau EXT-X-TARGETDURATION indica que la duració màxima d'un segment és de 10 segons i les línies EXTINF ens indiquen la duració dels diferents segments. L'adreça de cada segment és relativa, així que es farà servir l'adreça on es troba el fitxer d'índex com a base per formar les adreces de cada segment: si el fitxer .m3u8 es troba a *http://servidor.com/video1/qualitat1/prog_index.m3u8*, el primer segment es troba en *http://servidor.com/video1/qualitat1/fileSequence0.ts*.

Si tenim un sistema adaptatiu, aquest fitxer d'índex dóna la informació i localització dels segments d'una de les qualitats disponibles. Però com sap el client quines alternatives, en àmbit de qualitat, en té? Aquí entra en joc l'altre tipus de fitxer d'índex: les llistes de reproducció variables.

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=330768,CODECS="mp4a.40.5, avc1.4d401e"
http://servidor.com/video1/qualitat1/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1204619,CODECS="mp4a.40.2, avc1.42e015"
http://servidor.com/video1/qualitat2/prog_index.m3u8
```

```
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=2588007,CODECS="mp4a.40.2, avc1.4d401e"
http://servidor.com/video1/qualitat3/prog_index.m3u8
```

Aquest fitxer d'índex, que és el 'pare' de l'anterior, informa de les tres alternatives disponible en termes de qualitat. La primera alternativa està codificada a 330.078 kbps, la segona a 1.204.619 bps i la tercera a 2.588.007 bps. El client, en el moment d'escollir d'on descarregar el segment següent, pot escollir una d'aquestes tres alternatives. Mitjançant l'atribut BANDWIDTH i la mesura que hagi fet de l'ample de banda del que disposa, accedirà al fitxer d'índex[3] que escolleixi per a buscar el següent segment.

En aquest cas, el client té tres franges d'ample de banda. Si l'ample de banda disponible (BW) varia, escollirà una de les 3:

BW < 1.204.619 bps	1.204.619 bps < BW < 2.588.007 bps	2.588.007 bps < BW

Si el client mesura un ample de banda inferior a 330.768 bps la reproducció no serà continua ja que la qualitat menor és superior a l'ample de banda disponible.

2.4 El servidor i el client segons el protocol

En aquest punt es descriu com el servidor ha de generar el fitxer d'índex i els segments i com el client s'ha de descarregar els segments per a reproduir-los.

2.4.1 El procés del servidor

La forma de codificació del vídeo o l'àudio no entra dins del contingut del protocol i queda fora de l'abast del projecte. El servidor directament obté un *stream* continuu correctament codificat i encapsulat en una capa de transport MPEG-2 que conté el vídeo o l'àudio.

El servidor ha de ser capaç de dividir aquest *stream* en segments individuals d'una duració igual o menor a una constant definida. Aquesta constant de duració màxima de segment ha de fer

[3] Els tres fitxers d'índex que apareixen tenen exactament el mateix contingut. El mateix nombre de segments i amb les mateixes característiques. La única dada que pot ser diferent és l'adreça d'enllaç de cadascun dels segments.

que els segments obtinguts continguin, com a mínim, un o dos fotogrames Intra (I-Frames) [Punt 2.5.6]; en la pràctica serà un mínim de 2 segons de duració. Llavors, com més petit sigui aquest segment, més ràpidament s'adaptarà el client als canvis d'ample de banda. Un valor típic d'aquest valor constant és de 10 segons.

El servidor també ha de crear una llista de reproducció (*playlist*) dels segments, el que anomenem el fitxer d'índex. Aquest fitxer ha de seguir el format especificat més a dalt al punt 2.3. Aquest fitxer ha d'incloure enllaços als segments creats que hagin de ser accessibles pel client, i per a cada segment ha d'existir la línia EXTINF que indica la duració d'aquest. El fitxer d'índex generat ha de ser accessible pel client a través un enllaç web.

Aquest fitxer ha de tenir la paraula clau EXT-X-TARGETDURATION i el seu valor ha de ser igual o superior al valor d'EXTINF de qualsevol segment. A més, aquest fitxer hauria de tenir, tot i que no és obligatori, l'etiqueta EXT-X-VERSION [punt 2.5], que indica la versió més baixa del protocol que compleixen tant el servidor, com el fitxer d'índex i els segments audiovisuals. Totes dues etiquetes no poden ser mai modificades.

El servidor no hauria de canviar el fitxer d'índex si es tracta d'un *streaming* sota demanda, però si es tracta d'una retransmissió en viu, el servidor ha d'actualitzar el fitxer seguint unes restriccions.

2.4.1.1 Generar el fitxer d'índex.

El servidor pot actualitzar el fitxer d'índex afegint enllaços a segments o afegint la línia EXT-X-ENDLIST. Si el fitxer no inclou la línia que indica que no hi ha més segments, el servidor no pot actualitzar el fitxer sense afegir cap segment. Si el servidor vol esborrar enllaços a segments, el fitxer d'índex ha de tenir la línia EXT-X-MEDIA-SEQUENCE per indicar quin número de seqüència té el primer segment del fitxer, i actualitzar el valor de la paraula clau cada cop que elimina un segment.

Si s'elimina la referència a un segment, aquest ha de romandre accessible durant el temps resultant de la suma dels temps de reproducció de tots els segments que hi havia a la versió amb més duració del fitxer d'índex que el contenia.

Cada versió del fitxer d'índex ha de ser accessible, al menys durant la meitat del valor de EXT-X-TARGETDURATION i com a màxim 1,5 vegades aquest valor. Si, per exemple, EXT-X-TARGETDURATION val 10 segons, el fitxer d'índex ha de ser accessible entre 5 i 15 segons. Si es vol eliminar el fitxer d'índex, els segments d'aquest han de romandre accessibles, com a mínim durant el temps total dels segments segons la última versió d'aquest fitxer.

Si els segments han de ser xifrats, és el servidor l'encarregat de proporcionar un enllaç mitjançant la paraula clau EXT-X-KEY que permeti als clients autoritzats obtenir la clau de desxifrat. El servidor s'encarrega de xifrar els segments amb la clau corresponent. El mecanisme de xifrat queda fora de l'abast el projecte, però hi ha més informació en els enllaços 1 i 2 de l'annex A1.

2.4.1.2 Qualitats variables

El servidor pot oferir diversos fitxers d'índex per proveir diferents codificacions, qualitats, d'un mateix recurs. Si és així, el servidor ha de generar un fitxer d'índex amb els enllaços a cada una de les llistes de reproducció indicant la qualitat o codificació de cada una d'elles. Aquest fitxer d'índex ha de ser accessible pel client per tal de poder canviar d'una qualitat a una altre dinàmicament. Aquests fitxers s'anomenen llistes de reproducció variables (*variant playlists*).

Els fitxers d'índex variables han de contenir la paraula clau EXT-X-STREAM-INF per a cada qualitat amb la informació de l'ample de banda del *stream.*

Cada una de les diverses codificacions d'un mateix recurs, cada una de les qualitats, ha de complir unes restriccions. Totes han de tenir el mateix contingut, els segments de cadascuna ha de tenir la mateixa duració i les llistes han de tenir la mateixa duració màxima de segment. Si es fa servir la paraula clau EXT-X-PROGRAM-DATE-TIME, ha de tenir el mateix valor en totes les codificacions, per a que el client pugui sincronitzar els diferents *streams*. Un darrer requisit, molt important, és que totes les codificacions de vídeo, si tenen l'àudio incorporat, han de tenir la mateixa codificació en tots els *streams*; llavors els clients poden canviar de codificació sense cap "*glitch*" (só estrany de molt curta durada i molt molest).

2.4.2 El procés del client

El primer pas que ha de fer el client és obtenir l'enllaç de la llista de reproducció. Això dependrà de l'arquitectura del sistema; un enllaç web, un servei web, una base de dades, etc.

El client ha de descarregar-se el fitxer d'índex i, si es tracta d'una llista de reproducció variable, descarregar-se cadascun dels fitxers d'índex de cada codificació. Un cop obtinguts tots els fitxers, el client té tota la informació necessària per començar a reproduir el recurs amb HLS.

El client s'ha d'assegurar de què els fitxers compleixen amb el format indicat en el punt 2.3 i que aquests comencen amb la paraula clau EXTM3U. Si els fitxers contenen l'atribut EXT-X-VERSION, el client s'ha d'assegurar que el seu sistema és compatible amb la versió. Si no ho és, no ha d'intentar fer servir aquestes llistes. Si hi ha alguna paraula clau o atribut que no reconeix, el client ha de ser capaç d'ignorar-los.

Si la llista de reproducció conté la paraula clau EXT-X-MEDIA-SEQUENCE, el client ha d'assumir que els diferents segments poder tornar-se inaccessibles quan passi el temps equivalent a la suma dels temps de tots els segments de la llista de reproducció: si la llista conté tres segments de 10 segons cadascun, aquests poden tornar-se inaccessibles al cap de 30 segons.

2.4.2.1 Reproducció els segments

El client ha d'escollir quin segment ha de descarregar i reproduir en primer lloc. Per aconseguir una reproducció fluida i correcta, el client ha de reproduir els segments en l'ordre que apareixen en la llista de reproducció. El client ha de ser capaç de reinicialitzar el *parser* i el *decoder* abans de reproduir un segment al qual s'hagi aplicat la paraula clau EXT-X-DISCONTINUITY [punt 2.5].

Si la llista de reproducció conté la paraula clau EXT-X-MEDIA-SEQUENCE, el client l'ha de tenir en compte per calcular el número de seqüència dels segments del arxiu i escollir el segment correcte.

Si el segment està xifrat, el client ha de fer servir la clau i les dades proporcionades per la paraula clau EXT-X-KEY per a desxifrar el segment. El procés de desxifrat i les metodologies emprades queden fora de l'àmbit del projecte (enllaços 1 i 2 de l'annex A1).

El client ha de procurar de carregar els segments abans de què els necessiti per assolir una reproducció fluida i compensar les variacions temporals i d'ample de banda que poden haver-hi. A més, si el fitxer d'índex conté la paraula clau EXT-X-ALLOW-CACHE amb NO, el client ha d'eliminar el segment descarregat un cop reproduït. D'altra banda, si EXT-X-ALLOW-CACHE no està en el fitxer o el seu valor és YES, el client pot guardar els segments per a futures reproduccions.

Si el fitxer d'índex conté informació de dia i hora enllaçada als segments mitjançant l'etiqueta EXT-X-PROGRAM-DATE-TIME, el client pot mostrar-ho a l'usuari i fer-ho servir com a informació quan l'usuari vulgui buscar un instant en concret del recurs, però la reproducció no ha de dependre en cap moment d'aquests valors, han de ser merament informatius.

2.4.2.2 Actualització del fitxer d'índex

Si el fitxer d'índex no conté la paraula EXT-X-ENDLIST, el client l'ha d'anar recarregant buscant noves versions fins que aparegui aquesta paraula clau. És important que el client no recarregui la llista de reproducció amb massa freqüència, doncs hi ha un temps mínim estipulat per a la recàrrega d'aquest fitxer quecomença a comptar des del moment en que el client comença la càrrega del fitxer. Aquest temps mínim per a la primera recàrrega del fitxer és la durada de l'últim segment d'aquest, indicada en la línia EXTINF. Si en recarregar el fitxer el client no troba cap canvi, s'ha d'esperar la meitat de la duració màxima de segment per a tornar-ho a intentar.

Per reduir la càrrega dels servidor, el client no hauria de recarregar llistes de reproducció que no estigui fent servir. Fins i tot, si el client canvia de llistes dins d'un fitxer d'índex variable, només actualitzarà la llista de la codificació que necessiti descarregar.

2.4.2.3 Llistes de reproducció variables

L'especificació del protocol no limita ni descriu la metodologia que el client ha d'aplicar per escollir segments de diferents llistes de reproducció de qualitats variables. L'usuari és lliure d'aplicar i decidir com millor cregui l'elecció de la llista per adaptar-se al mitjà.

Tot i això, en la documentació per a programadors que publica Apple, es donen una sèrie de consells, dels quals en reprodueixo alguns:

- Començar sempre amb el segment de la llista de reproducció de qualitat més baixa, ja que el client no ha descarregat res per conèixer la velocitat de la xarxa.
- Descarregar el segment següent tenint en compte la velocitat de descàrrega de l'últim segment descarregat.
- Tenir en compte el temps de processament dels arxius i el retard provocat pel descodificador i el desxifrat.

2.5 Evolució i millores del protocol

Apple ha anat millorant el protocol i ha ampliat les seves característiques i funcionalitat. A cadascuna de les versions del document que descriu el protocol s'han anat afegint noves paraules claus o nous atributs a paraules antigues. Actualment, el document és un *Internet Draft* [punt 2.7], i això vol dir que les diverses versions del protocol poden variar en qualsevol de les revisions.

Durant el temps de redacció d'aquest projecte, el protocol es troba en la quarta versió. La darrera revisió, la 8na, data del 23 de Maig de 2012. No s'ha de confondre la versió del protocol amb la versió (revisió) del document que el descriu.

Els canvis introduïts per Apple en les primeres revisions del protocol no han estat gaire importants; la major part són millores en el xifrat i l'ampliació de la descripció d'algunes parts del protocol.

En la segona revisió del protocol es va afegir una nova paraula clau, EXT-X-DISCONTINUITY. Aquesta indica al client que entre el segment anterior a la línia i el següent hi ha hagut algun de canvi important al de format del fitxer o a la seva codificació.

Fins ara, el vídeo o l'àudio havia de partir d'un mateix origen segmentat en diferents fitxers; aquesta nova paraula clau dóna lloc a la possibilitat d'encadenar dos vídeos diferents en un de sol. Un dels usos pot ser el d'afegir un anunci durant una reproducció.

En la tercera revisió del protocol es defineix la versió 2 d'aquest. Per poder indicar al client la versió del protocol implementada en el servidor i que pugui decidir si el seu sistema és o no compatible, s'afegeix una nova paraula clau: EXT-X-VERSION, on s'indica la versió per mitjà d'un atribut.

En la segona versió del protocol s'afegeix un nou atribut a la paraula clau EXT-X-KEY. L'atribut és IV i es tracta d'un valor sencer o hexadecimal que representa un vector d'inicialització que es fa servir amb la clau obtinguda amb l'atribut URI. Tan el client com el servidor han de ser compatibles amb la versió 2 del protocol per fer servir aquest atribut.

En la tercera revisió s'afegeix la paraula clau EXT-X-PLAYLIST-TYPE. Aquesta serveix per indicar si el fitxer d'índex és o no actualitzat pel servidor i, si ho és, de quina manera. Hi ha tres possibilitats: que la paraula clau no hi sigui, això indica que la *playlist* pot ser modificada pel servidor en qualsevol moment eliminant i afegint segments. Si la paraula clau hi és, aquesta té un atribut que pot ser EVENT o VOD. Si és EVENT vol dir que el servidor no pot eliminar cap segment, però si afegir-ne. Si és VOD, vol dir que el servidor no pot modificar per res el fitxer d'índex, ni s'afegiran ni s'esborraran segments.

En la cinquena revisió del protocol és defineix la versió 3 d'aquest. La duració de cada segment definida en la paraula clau EXTINF pot ser un número decimal, fins a la versió 2 havia de ser un nombre sencer.

És a la setena revisió del protocol on es defineixen les tres grans millores que porta la versió 4 del protocol. La primera millora és la possibilitat de separar l'àudio i el vídeo d'un mateix recurs o enllaçar més d'una pista d'àudio o vídeo a un mateix recurs. La segona ofereix la possibilitat d'indicar quin és el conjunt de bits que es vol descarregar dins del segment enllaçat. I la tercera afegeix el concepte de I-FRAME al context HLS. Amb aquesta nova funcionalitat, el client pot

desenvolupar un sistema on l'usuari pot desplaçar-se d'una manera ràpida i precisa pel contingut, reproduint el vídeo des del punt que necessiti.

2.5.1 Múltiples canals d'àudio i vídeo

Sense entrar en detalls en les paraules clau necessàries i en la seva utilització, la possibilitat de poder separar l'àudio del vídeo i de tenir més d'un canal d'àudio per a un mateix vídeo dóna lloc a diversos avantatges.

El primer d'aquests és la possibilitat del client de donar a l'usuari l'opció d'escollir quin canal de àudio vol escoltar; a més podrà canviar el canal d'àudio durant la reproducció. És pot crear el contingut amb diferents canals d'àudio per a diferents llengües o afegir canals amb comentaris del director o complements semblants.

Fins a la versió 4 del protocol, per servir un contingut visual amb diferents canals d'àudio s'havia de generar el contingut tantes vegades com canals necessitéssim.

Un sol vídeo en 5 qualitats diferents; la més baixa només àudio, i les altre augmentant la qualitat del vídeo, totes elles agrupades ocupen una gran quantitat de disc. Si volem tenir aquest mateix contingut en 4 idiomes diferents, la quantitat d'espai al disc és multiplica per 4. Això, per empreses dedicades l'*streaming* de grans quantitats de vídeos, suposa la contractació de 4 vegades més capacitat d'emmagatzemament, és a dir, una despesa 4 cops més elevada.

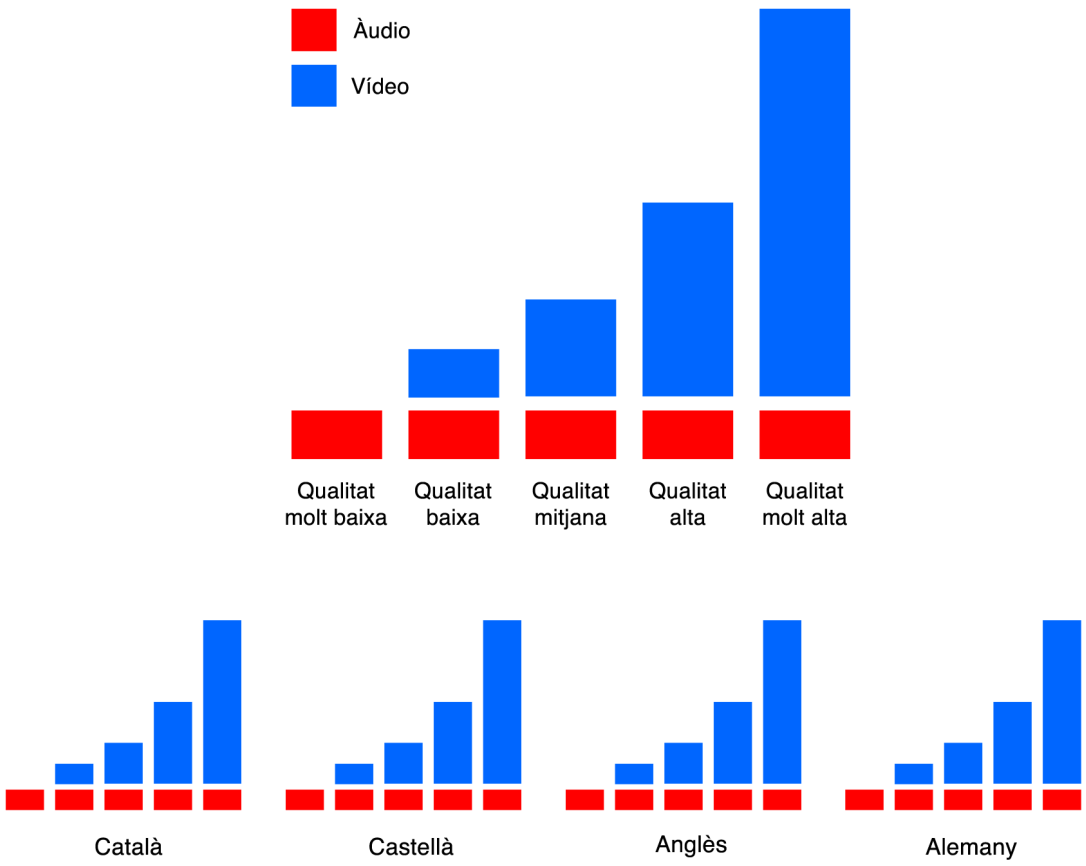

Fig. 2.2 - **Quantitat d'espai necessari abans de la versió 7 del protocol**

A partir de la versió 4, amb la possibilitat de separar l'àudio del vídeo i enllaçar més d'un canal d'àudio a un mateix vídeo, la quantitat d'espai necessari per a poder emmagatzemar un vídeo en 4 idiomes diferents es redueix dràsticament, així com es redueixen també les despeses generades per aquests nous canals.

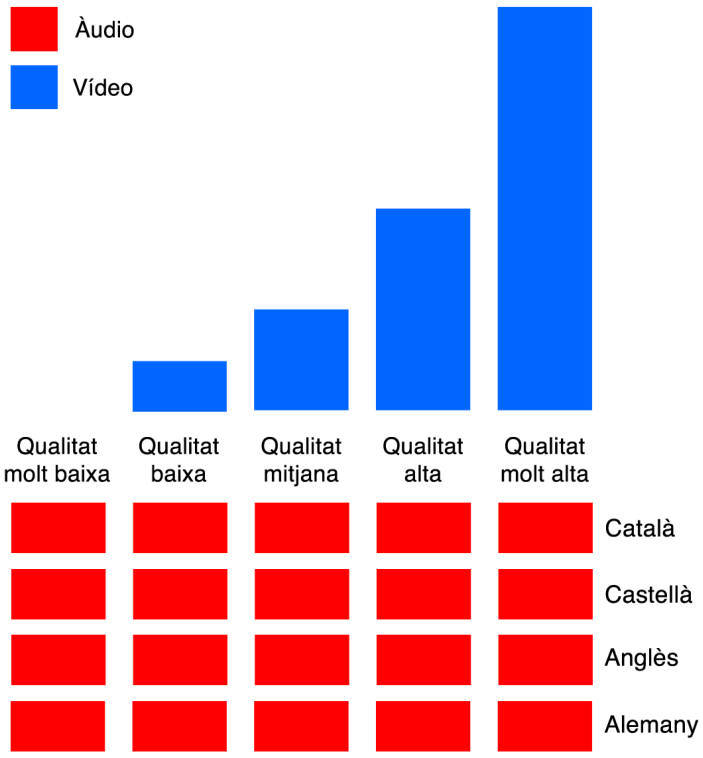

Fig. 2.3 - **Quantitat d'espai necessari a partir de la versió 7 del protocol**

El client ha de descarregar dos segments, un d'àudio i un altre de vídeo; la quantitat de dades descarregades serà una mica més elevada degut a que són dos fitxers, però aquesta petita addició de dades és mínima i no afecta el rendiment total del sistema. També ha de ser capaç d'ajuntar, o reproduir alhora, el segment que pertoqui d'àudio amb el segment de vídeo de la qualitat seleccionada i de descarregar tots dos segments.

Aquesta nova funcionalitat dóna també la possibilitat d'enllaçar més d'un vídeo a un recurs i pot serveix, per exemple, per donar diferents vistes d'una escena des de càmeres alternatives.

Un exemple de fitxer d'índex per a tenir diversos canals d'àudio:

```
#EXTM3U
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="a",NAME="Català",DEFAULT=YES,AUTOSELECT=YES,
LANGUAGE="ca",URI="main/cat-audio.m3u8"
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="a",NAME="Castellà",DEFAULT=NO,  AUTOSELECT=NO,
LANGUAGE="es",URI="main/esp-audio.m3u8"
```

```
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="a",NAME="Anglès",DEFAULT=NO,  AUTOSELECT=NO,
LANGUAGE="en",URI="main/eng-audio.m3u8"
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="a",NAME="Alemany",DEFAULT=NO,  AUTOSELECT=NO,
LANGUAGE="de",URI="main/deu-audio.m3u8"
#EXT-X-STREAM-INF:BANDWIDTH=1280000,CODECS="...",AUDIO="a"
low/video-only.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=2560000,CODECS="...",AUDIO="a"
mid/video-only.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=7680000,CODECS="...",AUDIO="a"
hi/video-only.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=65000,CODECS="mp4a.40.5",AUDIO="a"
main/eng-audio.m3u8
```

En cas de què es volgués tenir diferents canals de vídeo, l'atribut AUDIO es substituiria per l'atribut VIDEO i els enllaços a les *playlists* pertinents. L'atribut LANGUAGE només apareix si es tracta de canals en diferents idiomes. Si hi hagués un canal extra, aquest no apareixeria.

2.5.2 Descàrrega de conjunts de bits

En versions anteriors, després de la paraula clau EXTINF seguia l'enllaç al segment. Ara es pot afegir una nova paraula clau EXT-X-BYTERANGE i es pot indicar a partir de quin bit del segment comença la descàrrega i quina quantitat de bits s'ha de descarregar. Un exemple seria aquest:

```
#EXTINF:4.12,
#EXT-X-BYTERANGE:9400@376
segment1.ts
```

En aquest cas indiquem que el segment següent té una duració de 4,12 segons, però que només cal descarregar-se 9400 bits començant des del bit 376.

La principal utilitat d'aquesta millora és la d'unificar tots els segments en un sol fitxer .ts i després accedir a diferents parts d'aquest per a cadascun dels segments. Com que només hi ha un fitxer en lloc de tots els segments, hi ha molts menys arxius per a gestionar en el servidor i les xarxes de distribució de continguts, i es millora també la gestió dels *caches* dels servidors.

Aquesta nova funcionalitat també es fa servir als I-Frames, que s'expliquen al punt següent.

2.5.3 I-Frames

Quan un usuari vol avançar ràpidament el vídeo que s'està reproduint en un sistema de streaming HLS, el client es troba amb un problema. El sistema del client no pot mostrar les imatges del vídeo ràpidament ja que aquest no és capaç de descarregar-se els segments amb tanta velocitat. L'usuari pot ser informat de l'instant de temps al qual vol desplaçar-se o s'està desplaçant, però no de la imatge aproximada que hi ha en aquell instant. Apple ha afegit en la versió 4 del protocol una nova funcionalitat que soluciona aquesta necessitat. La solució és informar al client d'on es troben els fotogrames principals en cada segment. Aquests fotogrames principals s'anomenen I-Frames.

Però, què és un I-Frame? Els vídeos encapsulats en una trama de transport MPEG-2 per al protocol HLS estan codificats i comprimits en H.264. En un vídeo comprimit hi ha multitud de fotogrames comprimits i n'hi ha uns de més comprimits que d'altres. Els més comprimits contenen molt poca informació i els reproductors es basen en la informació dels fotogrames anteriors per poder mostrar en pantalla tot un fotograma. Els menys comprimits contenen tota la informació del fotograma i són els anomenats I-Frames o Intra Frames. Aquesta nova funcionalitat crea nous fitxers d'índex per a cada qualitat on només hi ha informació sobre els punts on es troben aquests fotogrames principals i la seva mida, fent servir la nova paraula clau EXT-X-BYTERANGE i indicant que el fitxer d'índex és un fitxer d'informació de I-Frames.

En el moment de crear el fitxer d'índex, el servidor ha de afegir noves línies per a cadascuna de les qualitats que apuntin a altres fitxers d'índex on estan definides les posicions dels I-Frames.

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=1280000
baixa/audio-video.m3u8
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=86000,URI="baixa/iframe.m3u8"
#EXT-X-STREAM-INF:BANDWIDTH=2560000
mitjana/audio-video.m3u8
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=150000,URI="mitjana/iframe.m3u8"
...
```

En aquest cas hi ha, per a cada qualitat, una línia amb una nova paraula clau EXT-X-I-FRAME-STREAM-INF on s'indica la qualitat dels fotogrames i l'enllaç al fitxer d'índex on són definits.

A partir de cada segment es recuperen les posicions i duracions dels Intra Frames i és genera la llista en un fitxer d'índex.

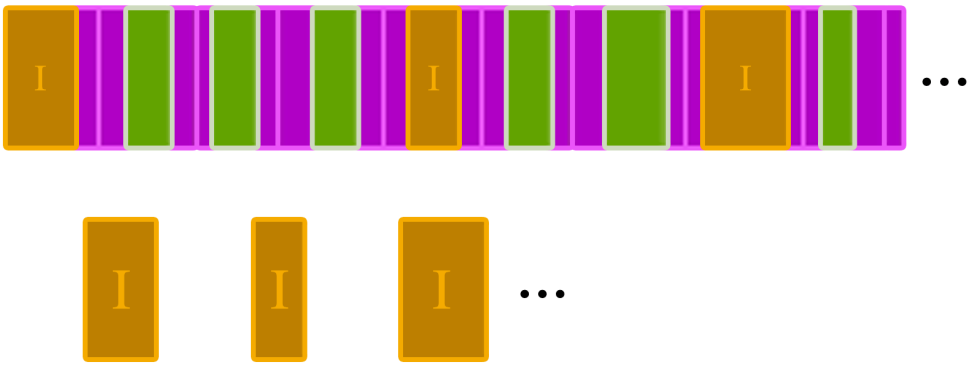

Fig. 2.4 - **Selecció dels fotogrames Intra (I-Frames)**

Cadascun dels fitxers d'índex de I-Frames té un format semblant a aquest:

```
#EXTM3U
#EXT-X-VERSION:4
#EXT-X-I-FRAMES-ONLY
...
#EXTINF:4.12,
#EXT-X-BYTERANGE:9400@376
segment1.ts
#EXTINF:3.56,
#EXT-X-BYTERANGE:7144@47000
segment1.ts
#EXTINF:3.82,
#EXT-X-BYTERANGE:10340@1880
segment2.ts
...
```

Hi ha una nova paraula clau EXT-X-I-FRAMES-ONLY que indica que aquest fitxer només conté referències a I-Frames. L'atribut de EXTINF és la duració del I-Frame i la paraula clau EXT-X-BYTERANGE indica la posició i la mida en bits del fotograma dins del segment enllaçat per la línia següent.

Un cop definits tots els fotogrames *Intra*, el client els descarrega i, quan l'usuari avança ràpidament cap endavant, mostra els fotogrames un darrera l'altre. Un cop l'usuari escull un punt des d'on reproduir, si el sistema no té descarregat el segment on es troba el fotograma hi ha una temps d'espera per a la descàrrega del segment. Si ja té el segment, la reproducció és instantània.

2.5.4 Versions el protocol

Al començament del punt 2.5 s'ha descrit com les diferents revisions de l'*Internet Draft* que descriu el protocol han definit fins a 4 versions del protocol HTTP Live Streaming.

Servidor, fitxer d'índex, segments generats i client, tots quatre han de ser compatibles amb una versió del protocol per a poder fer-lo servir. A la següent taula es mostren les funcionalitats i les paraules clau o atributs que apliquen en cada nova versió.

Paraules clau i atributs	*Versió*
Atribut IV a la paraula clau EXT-X-KEY	2+
L'ús d'un decimal per a la duració d'un segment a la paraula clau EXTINF	3+
Les paraules clau següents: • EXT-X-BYTERANGE • EXT-X-I-FRAME-STREAM-INF • EXT-X-I-FRAMES-ONLY • EXT-X-MEDIA Atributs VIDEO i AUDIO a la paraula clau EXT-X-STREAM-INF	4+

2.6 Estandardització i implantació

Apple ha documentat l'HTTP Live Streaming com a un *Internet Draft (Individual Submission)*, que és el primer pas en el procés d'estandardització per la IETF (Internet Engineering Task Force), per a convertir-se en un estàndard internacional. Tot i que la primera versió del document es va presentar l'1 de Maig del 2009 i que Apple ha anat afegint petites millores i actualitzacions

durant d'aquests anys, sembla que la IETF o Apple no han fet cap pas per a l'estandardització del protocol.

Un *Internet Draft* (I-D) és un document de treball publicat per la IETF. Aquesta mena de documents representen el primer estat per on passen molts dels protocols i projectes abans de ser estandarditzats. Un I-D és un document que descriu com funciona, en aquest cas un protocol, però sense arribar a tenir una definició completa, és a dir que encara no està tancat i pot evolucionar i canviar, encara s'està treballant en ell.

No és gaire adequat que es prengui com a referència un *Internet Draft* per al desenvolupament d'un projecte o aplicació d'una tecnologia. Aquests tan sols són vàlids durant sis mesos i són després actualitzats amb noves versions més evolucionades que poden variar molt. En el punt 2.6 hem vist com en diferents revisions del document s'han canviat diverses parts i especificacions de les versions del protocol.

Com a protocol independent d'un *software* propietari o d'una tecnologia dedicada, i tot i ser tant sols un I-D, hi ha moltes empreses i serveis importants que estan fent servir aquest protocol de *streaming* en tecnologies i serveis actuals. Empreses com Microsoft, que han implementat el protocol en els serveis del seus servidors d'Internet IIS Media Services; o Google, al seu sistema operatiu per mòbils, Android, que suporta HLS des de la versió 3.0 Honeycomb. També Adobe Systems ha actualitzat el seu servidor de *streaming* Adobe Flash Media Server per suportar HLS, i el sistema operatiu web de HP, webOS, suporta HLS des de la versió 3.0.5. Hi ha molt més exemples d'empreses i sistemes que suporten i implementen serveis fent servir HTTP Live Streaming a l'annex A2.

Una de les moltes restriccions que han de complir les aplicacions per al sistema iOS té a veure, no obstant, amb aquest protocol. Malgrat de ser un *Internet Draft,* tota aplicació que realitzi *streaming* de continguts, si el vídeo o el contingut a reproduir té una duració superior a 10 minuts o més de 5 MB de dades, s'ha d'implementar l'HTTP Live Streaming. A més, si l'aplicació es pot fer servir també a xarxes mòbils, hi haurà d'haver al menys un *stream* de baixa qualitat a 64 Kbps. Si no es compleixen aquestes restriccions, l'aplicació no passa els filtres i no serà apte per a la botiga d'aplicacions d'Apple.

- Capítol 3 -

PROGRAMACIÓ D'APLICACIONS PER A iOS

3 Programació d'aplicacions per a iOS

Gran part del temps dedicat a aquest projecte s'ha consumit aprenent a desenvolupar aplicacions en el sistema operatiu per a dispositius mòbils d'Apple, l'iOS. Aquest capítol intenta donar una petita introducció a aquest món, una orientació per a programadors que, com jo, entren de nou en aquest ecosistema d'Steve Jobs.

3.1 Objectiu d'aquest capítol

En aquest capítol el lector no trobarà una informació exhaustiva per desenvolupar aplicacions per a l'iPhone o l'iPad. Per aconseguir això calen llibres sencers (més d'un). L'objectiu és donar al lector una punt d'entrada, uns coneixements inicials que facilitin la obtenció de tots els recursos necessaris per a crear aplicacions per als dispositius d'Apple i una visio global d'aquests sistemes.

3.2 Primers passos

Per a poder desenvolupar aplicacions per a iOS, el programador ha de tenir uns recursos i uns coneixements bàsics.

3.2.1 Recursos necessaris

El primer que un programador ha de comprovar per començar a desenvolupar aplicacions per a iOS és si disposa de les eines per fer-ho. Apple proporciona un SDK i uns portals per a desenvolupadors accessibles per a tothom, però les eines de desenvolupament que proporciona només funcionen en MAC OS X i només els ordinadors d'Apple disposen d'aquest sistema operatiu. Així que el primer requisit és tenir un Mac Mini o Pro, un iMAc o un MacBook.

Per poder desenvolupar aplicacions necessitem el SDK i les eines que aquest ens proporciona. Per tenir accés a aquestes eines, el programador s'ha de donar d'alta dins del programa de desenvolupadors d'Apple a https://developer.apple.com. Hi ha quatre modalitats.

La primera modalitat és gratuïta. Apple l'anomena **Online Developer Program**, i va dirigida a tot aquell que vulgui aprendre, explorar i programar per a iOS, però sense cap més objectiu que aquest. Amb aquesta modalitat, el programador té accés a totes les eines i SDKs disponibles

però no pot publicar aplicacions en el mercat d'Apple ni registrar dispositius per executar en un dispositiu real les aplicacions que hagi desenvolupat[4].

Amb aquesta subscripció, el programador pot executar les aplicacions en un simulador. Tot i que aquest simulador cada cop s'apropa més a la representació real del dispositiu, el resultat final no es pot fonamentar en el seu funcionament o no al simulador. Una aplicació que funciona bé al simulador pot no respondre correctament en el dispositiu o a l'inrevés. A més, el simulador no té vibració i no reacciona a certs moviments o gestos.

Aquest programa tampoc dóna accés a les versions beta del sistema operatiu, ja que només es poden instal·lar als dispositius registrats com a dispositius de test.

Una altre programa és l'anomenat **Standard Developer Program**. Aquest té un cost de 99$ l'any i permet publicar aplicacions al *App Store* i registrar dispositius per provar-les en ells. A més, aquest programa permet fer distribucions *ad hoc*; permet al desenvolupador instal·lar l'aplicació en 100 dispositius diferents. Standard Developer Program és la solució general per a la major part de desenvolupadors.

L'opció següent s'adreça a empreses grans. **Enterprise Developer Program** costa 299$ l'any i va adreçada a companyies de 500 empleats o més. Aquest programa no dóna accés al mercat d'aplicacions d'Apple sinó que està enfocat a la producció d'aplicacions d'àmbit intern de l'empresa.

L'últim programa, **University Developer Program**, està enfocat a les universitats, per tal d'animar a aquestes a integrar el sistema de desenvolupament d'Apple als seus plans d'estudis i aconseguir que surtin enginyers preparats i formats en l'ecosistema d'Apple. El programa permet al professor crear equips de treball de fins a 200 estudiants i tenir accés total al iPhone

[4] Segons la documentació d'Apple en el *Online Developer Program* té accés a totes les eines, el SDK i tots els recursos. Però en el transcurs del desenvolupament d'aquest projecte he pogut descobrir que no és així. Hi ha certes eines i recursos que només són accessibles per a les subscripcions de pagamant.

SDK. Els estudiants poden compartir les seves aplicacions amb els companys i professors del programa. És la institució la que pot publicar aplicacions en la *App Store*.

Dins de l'enllaç anterior, https://developer.apple.com, també hi ha accés a tota la documentació d'Apple sobre el SDK, les tecnologies, vídeos de presentacions, exemples i multitud d'informació ben catalogada i organitzada per poder tenir accés a tot el necessari per a programar en els seus sistemes.

Un cop el programador està donat d'alta en algun d'aquests programes té accés a tota la informació i al portal de desenvolupadors. En aquest portal, si el programa ho permet, és on el propietari de la llicencia pot gestionar els diferents dispositius de test, els altres programadors del seu equip, les llicencies per distribució *ad hoc*, etc.

3.2.2 Coneixements necessaris

El programador necessita conèixer el paradigma de programació que envolta l'ecosistema dels productes d'Apple. A més, necessita saber programar amb el llenguatge adoptat per la companyia, l'Objetive C. Necessita conèixer els SDKs dels que disposa i les possibilitats i limitacions dels sistemes Apple; no és el mateix desenvolupar una aplicació per a un MacBook que per un iPod Touch, les diferències de *hardware* són evidents, però també ha de conèixer les limitacions que imposa la pròpia Apple per a les aplicacions en cadascuna de les plataformes i les limitacions pròpies dels SDKs dels quals disposa.

3.2.2.1 Objective-C

L'Objective-C és un llenguatge orientat a objectes creat com un *superconjunt* de C per implementar un model d'objectes similar l'Smalltalk. Va ser creat per Brad Cox i la corporació StepStone a l'any 1980. Al 1988 va ser adoptat com a llenguatge de programació de NeXTSTEP (El sistema operatiu de NeXT Computer, Inc.) i al 1992 va ser alliberat sota llicència GPL per al compilador GCC. Actualment es fa servir com a llenguatge principal de programació per a Mac OS X, iOS i GNUstep.

Si es compara amb el C/C++, la sintaxis és molt similar, així que si el programador té experiència amb llenguatges orientats a objectes com el C++ o el Java, li resultarà fàcil adaptar-se a la nova sintaxis.

3.2.2.1.1 Sintaxis

Objective C està construït per sobre de C, de fet és un *superconjunt* de C. Això vol dir que es pot compilar un programa en C amb el compilador d'Objective-C o es pot escriure codi C d'una classe en Objective-C.

El típic "Hola Mundo" en Objective-C seria:

Codi 3.1 - **Hola Mundo en Objective-C**

```
#import <stdio.h>

int main(int argc, const char * argv[])
{
    printf("Hola Mundo\n");
    NSLog(@"Hello, World!");
    return 0;
}
```

Hi ha dos línies que diferencien el codi anterior amb un codi en C. La primera diferencia és la primera línia amb el #import enlloc del `#include` de C. La segona és la línia del mètode `NSLog:`, aquest és un mètode que fa el mateix que el `printf` de c (en la línia anterior) però que afegeix informació de l'instant de temps en que s'executa; és un mètode propi de Objective-C. També es pot veure que el mètode `printf` segueix funcionant tot i ser purament C.

La sintaxis d'objectes d'Objective-C ve derivada de *Smalltalk*. Tota la sintaxis que no té relació amb la programació orientada a objectes és C bàsic, i la part de la programació orientada a objectes és una implementació similar a la missatgeria de *Smalltalk*.

Missatges

El model de programació orientada a objectes d'Objective-C es basa en enviar missatges a les instàncies dels objectes. Això vol dir que és diferent del model de programació de C++ i la diferència és semànticament important. En Objective-C no es criden mètodes (tot i que de vegades es diu així per comoditat i per ajudar a entendre el concepte), si no que s'envien

missatges. La diferència radica en com s'executa el codi al que es refereix el nom del missatge o mètode.

En un llenguatge com el C++, el nom del mètode està, en la majoria de casos, lligat a una secció de codi en la classe objectiu pel compilador. Però en el cas de l'Objective-C o del *Smalltalk*, el missatge segueix essent simplement un nom; aquest és resolt en temps d'execució. La principal conseqüència és que el missatge passat pel sistema no te cap mena de comprovació de tipus; l'objecte al qual s'adreça el missatge (el receptor) no garanteix que respongui al missatge i, si no ho fa, simplement l'ignora i retorna un punter nul.

Enviar el missatge `method` a l'objecte apuntat pel punter `obj` requeriria el següent codi en C++:

```
obj->method(parameter);
```

En canvi, en Objective-C:

```
[obj method:parameter];
```

És important denotar que, tot i que els dos estils de programació tenen les seves virtuts i les seves mancances, la POO de C++ permet l'herència múltiple i una ràpida execució gràcies al lligam en temps de compilació, però aquest no suporta lligams dinàmics per defecte. En canvi, en Objective-C es permet que els missatges no tinguin implementació i tampoc és necessari que un objecte sigui definit en temps de compilació. Tot i això, degut a la sobrecàrrega en la interpretació dels missatges, un missatge en Objective-C pren, en el millor dels casos, tres vegades més temps d'execució que una crida a un mètode virtual en C++.

Interfícies i implementacions

Objective-C requereix que la interfície i la implementació d'una classe estiguin en blocs de codi separats. Per convenció, la interfície es posa en un arxiu de capçalera *.h* i la implementació en un arxiu de codi *.m*.

Interfície

La interfície d'una classe està normalment definida en l'arxiu de capçalera i, per convenció, aquest arxiu porta el mateix nom que la classe. Una interfície per a la classe MyClass hauria de ser com en el Codi 3.2, i trobar-se en el arxiu *MyClass.h*.

Codi 3.2 - **Definició de la interfície d'una classe**

```
@interface MyClass : NSObject
{
    // instance variables
}
+ (void)classMethod1;
+ (int)classMethod2;
+ (NSString *)classMethod3:(MyClass *)param1;

- (void)instanceMethod1:(int)param1 :(double)param2;
- (id)instanceMethod2WithParameter:(NSData *)param1 andOtherParameter:(int)param2;
- (void)instanceMethod3;
@end
```

Els símbols + indiquen que els mètodes són de la classe, no cal crear una instància de la classe per executar-los. Els símbols - indiquen que són mètodes d'instància. Els mètodes de classe no tenen accés a les variables d'instància.

Els paràmetres dels mètodes van definits amb el caràcter : seguit de la definició del tipus del paràmetre seguit per el nom que aquest té. És un bon costum afegir un text descriptiu abans de cada paràmetre.

Implementació

La interfície tan sols defineix els mètodes i variables de la classe però no la seva implementació, aquesta sol estar en un altre arxiu amb el mateix nom però amb l'extensió *.m*.

Codi 3.3 - **Implementació d'una interfície**

```
@implementation MyClass

- (id)init
{
    self = [super init];
    if (self) {
        //Init vars
    }
    return self;
}
```

```
+ (void)classMethod1
{
    //Implementation code here
}

+ (int)classMethod2
{
    int numToreturn = 0;
    //Implementation code here
    return numToreturn;
}

+ (NSString *)classMethod3:(MyClass *)param1
{
    //Implementation code here
    return @"Done";
}

- (void)instanceMethod1:(int)param1 :(double)param2
{
    //Implementation code here
}

- (id)instanceMethod2WithParameter:(NSData *)param1 andOtherParameter:(int)param2
{
    //Implementation code here
    [self instanceMethod1:10 :10.5];
    return nil;
}

- (void)instanceMethod3
{
    [self instanceMethod2WithParameter:nil andOtherParameter:5];
}

@end
```

En la implementació dels mètodes definim el codi que executarà cadascun d'ells.

Instàncies

Un cop una classe està totalment definida ja es poden crear instàncies d'aquesta. El primer que s'ha de fer és reservar l'espai de memòria per al objecte i després inicialitzar-lo. Una instància no és completament funcional fins que aquests dos passos no s'han dut a terme. Es poden executar en una sola línia de codi:

```
MyClass *obj = [[MyClass alloc] init];
```

En el codi d'implementació de la classe, el programador ha de definir el contingut del mètode `init`: on s'inicialitzen les variables de la classe.

Hem vist una mica per sobre els canvis, les diferències i la sintaxis que defineixen l'Objetcive-C. Aprendre Objective-C i veure tota la informació d'aquest requeriria més d'un llibre i queda fora de l'abast del projecte. Més endavant, en els punts següents, veurem altres tecnologies i característiques de l'Objective-C.

3.2.2.2 Xcode

L'eina més important per a desenvolupar aplicacions per a iOS s'anomena Xcode. Xcode és un entorn de desenvolupament on trobarem les eines necessàries integrades dins d'un mateix marc de treball. Xcode inclou eines per a crear i depurar el codi, compilar aplicacions, dissenyar interfícies i gestionar el flux de treball. També conté eines per analitzar el rendiment de les aplicacions, gestionar els recursos i monitoritzar el codi.

Actualment Xcode es pot descarregar gratuïtament des de la botiga d'aplicacions d'Apple per Mac OS.

3.2.2.3 Cocoa Touch

Cocoa és el nom del framework que permet el desenvolupament d'aplicacions natives per a Mac OS X. El el cas de iOS s'anomena Cocoa Touch. El framework està totalment integrat en l'entorn de programació Xcode. La API de més alt nivell de Cocoa permet que el programador pugui afegir animacions, treball en xarxa i una interfície amb una aparença igual a les aplicacions natives amb molt poques línies de codi.

Cocoa fa servir un patró de disseny anomenat MVC, Model Vista Controlador (Model View Controler). Els Models encapsulen les dades de l'aplicació, les Vistes presenten les dades i les editen, i el Controlador gestiona la lògica entre aquests dos elements. Amb aquesta separació de tasques, l'aplicació és més fàcil de dissenyar, d'implementar i de mantenir.

3.2.3 Dos paradigmes importants: OOP i MVC

La programació per a iOS es centra en dos paradigmes importants: la programació orientada a objectes (OOP) i el patró de disseny MVC (Model Vista Controlador). El SDK està dissenyat per suportar aquests conceptes en les aplicacions. Per a poder fer això s'han introduït els delegats

(Controladors), classes visuals personalitzades (Vista) i mètodes d'accés i gestió de dades (Model).

3.2.3.1 Programació Orientada a Objectes

Com s'ha dit abans, l'Objective-C està basat en el *Smalltalk*, un dels llenguatges orientats a objectes històricament importants. Els llenguatges orientats a objectes fan servir els conceptes d'encapsulació i d'herència per a construir classes de codi re-utilitzables, amb interfícies externes públiques i una implementació interna privada. Les aplicacions es construeixen a partir de petites peces independents que es combinen per a construir una de més gran; igual que les peces de LEGO.

Una pseudo-herència múltiple és una part important del paradigma de la programació orientada a objectes. Les classes en Objective-C poden heretar comportaments i dades des de més d'un pare. Prenent per exemple la classe *UITextView*. Aquesta és tant text com una vista (*view*). Com altres classes *view*, aquesta pot aparèixer per pantalla, se li pot definir una mida, una posició, canviar els colors i l'aparença, etc. Al mateix temps, també hereta les característiques de text; el programador pot canviar-li la font, la mida, el color de la font, etc. Obective-C i Cocoa Touch combinen aquests comportaments en una sola classe fàcil de fer servir.

Per a veure totes les característiques de la programació orientada a objectes hi ha llibres sencers i queda fora de l'àmbit del projecte.

3.2.3.2 Model-Vista-Controlador

El MVC separa la manera en com un objecte es mostra per pantalla del seu comportament. Un botó en pantalla (la vista) no té cap significat intrínsec; és tan sols un botó que l'usuari pot pressionar. El control de vista (*view controller*) que gestiona el botó actua com a intermediari; connecta les accions que l'usuari fa amb aquest botó amb els mètodes en l'aplicació, els quals defineixen el Model. L'aplicació guarda dades i respon a les interaccions amb aquest botó produint una resposta cap a l'usuari.

El patró de disseny MVC està molt ben explicat i descrit en el document escrit al 1988 per Glenn Krasner i Stephen Pop anomenat "*A cookbook for using the model-view controller user interface paradigm in Smalltalk-80*" (enllaç 3 de l'annex A1).

Cada un dels elements del MVC és independent. El programador podria canviar el botó per un interruptor i no hauria de tocar el codi de la part del Model o del Contolador, tan sols ha canviat l'aparença visual. També pot ser al revés, l'aparença no es toca però si les accions del Controlador o del Model. Cada un dels components pot ser actualitzat d'una manera independent dels altres.

El paradigma del MVC en iOS es separa en les següents categories:

- **Model** - Els mètodes del Model obtenen dades a través de protocols, els quals, obtenen les dades implementant els mètodes designats pel Controlador.
- **Vista** - Els components de les vistes (*views*) són tots fills de la classe *UIView* i són assistits i gestionats pel seu controlador (UIViewController)
- **Controlador** - El comportament del controlador es implementat a través de tres tecnologies: *delegates*, *target actions*, and *notifications.*

Tots tres elements formen la columna vertebral del paradigma de programació de MVC.

3.2.3.2.1 Vista (*View*)

Cocoa Touch construeix totes les vistes a través de dues classes *UIView* i *UIViewController*. Aquestes dues classes, i les que heretin d'elles, són les encarregades de definir i col·locar tots els elements per pantalla.

Les vistes pinten coses per pantalla i *UIView* representa la classe més abstracta possible de tipus vista. Gairebé totes les classes d'interfície d'usuari són descendents de *UIView* o del seu pare, *UIResponder*. Exemples importants de la classe *UIView* són *UITextView*, *UIImageView*, *UIAlertView*, etc. Hi ha una classe que hereta de *UIView* anomenada *UIWindow*, que proporciona una base on s'arrela tota l'aplicació.

Les vistes estan organitzades en forma d'arbre. El programador pot afegir una vista directament a la *UIWindow* o dins d'una altra vista. A més, cada vista afegida es solapa a les afegides anteriorment, com si es tractés d'una mena de capes.

Els controladors de vista (*view controllers*) ajuden al programador a gestionar les vistes i interactuar amb elles. Són els encarregats de rotar la pantalla quan l'usuari gira el dispositiu, canvia la mida de les vistes (si aquestes estan ben enllaçades), etc.

Addicionalment hi ha dos controladors especials heretats d'*UIViewController*; *UINavigationController* i *UITabBarController*, que gestionen el moviment entre diferents vistes automàticament. El control de navegació habilita la navegació entre vistes d'una manera suau, animada i transparent per al programador, desplaçant una vista cap a una altra. Aquest controlador recorda quina vista hi havia primer i proporciona automàticament un botó per a tornar a la vista anterior sense cap línia de codi.

El controlador *UITabBarController* habilita el canvi entre diferents instàncies de vistes fent servir una barra amb botons per a cada vista, només cal que el programador associi una vista a un botó i li posi una icona i un títol, la gestió de les vistes és automàtica.

3.2.3.2.2 Controlador (*Controller*)

Quan a Apple es dissenyen elements interactius com taules o desplaçadors (*sliders*), no es té idea de com els farà servir el programador. Les classes són deliberadament generals. Amb MVC no hi ha cap mena de significat en la programació quan es seleccionada una fila d'una taula o es prem un botó. El programador és qui desideix quin comportament ha de tenir cada acció. El SDK proveeix diverses maneres amb les quals les classes genèriques de Cocoa Touch poden comunicar-se amb les classes generades per els programadors. Les tres maneres o tecnologies més importants són: *delegation*, *target-action* i *notifications*.

Delegation

Moltes classes que hereten de *UIKit* fan servir *delegation* per a cedir la responsabilitat de respondre a les accions del usuari. Quan un programador programa la resposta a un delegat

d'un objecte, li està indicant al programa que no faci cas de la acció, que serà el seu propi codi que se n'ocuparà.

Un bon exemple de *delegation* és la classe *UITableView*. Quan l'usuari toca una fila d'una taula, l'*UITableView* no té cap mena de codi de resposta a aquesta acció, tan sols una resposta visual per a indicar a l'usuari conforme s'ha tocat la fila. En comptes de respondre a l'acció, la classe consulta el seu delegat - normalment una classe *view controller* o la aplicació en sí (*main application*) - i li passa l'acció de selecció de fila a través del mètode de delegat. Això habilita al programador a definir la resposta a l'acció com més li convingui, d'una manera totalment independent a com estigui implementada la classe *UITableView*.

El mètode de delegat de *UITableView* anomenat `tableView: didSelectRowAtIndexPath:` n'és un exemple. El model del programador agafa el control del mètode i implementa la reacció a l'acció de selecció de fila. El programador pot mostrar un menú, navegar a una altra vista o posar una marca de selecció a la fila. La resposta depèn totalment de la implementació del programador.

Per a definir el delegat d'un objecte es pot fer assignant un valor a la propietat *delegate* del objecte o fent servir el mètode `setDelegate:`. Això indica a l'aplicació que ha de redirigir les accions del usuari cap al delegat. Per indicar que l'objecte pot reaccionar a les accions del usuari, el programador ha d'indicar a la declaració de la classe quin protocol implementa la aquesta. El protocol defineix els mètodes de delegat, les interaccions de l'usuari en aquest cas. El Codi 3.1 mostra com una classe que hereta de *UIViewController* implementa els mètodes de delegat per a una vista *UITableView*. La classe *MyTableController* és l'encarregada d'implementar tots els mètodes de delegats requerits de *UITableView*.

Xcode permet llistar tots els mètodes de delegat, tant obligatoris com opcionals. A més, els mètodes de delegat no estan limitats a les classes d'Apple. El programador pot definir els seus propis protocols i fer que les classes puguin tenir delegats i fer-los servir. El Codi 3.1 crea un protocol anomenat `MyProtocol` i li defineix dos mètodes de protocol. També declara una instància anomenada `myDelegate`. Quan es faci servir aquest objecte s'hauran d'implementar els dos mètodes del protocol. Els protocols són grans eines d'Objective-C i permeten crear classes de client que garanteixen el suport a totes les funcionalitats requerides per la classe primària.

Codi 3.4 - **Definir i afegir declaracions de mètodes de protocol a una definició de classe**

```
@protocol MyProtocol <NSObject>
- (void) methodOne: (NSString *)text;
- (void) methodTwo: (id)sender;
@end

@interface MyTableController : UIViewController
    <UITableViewDelegate, UITableViewDataSource>
{
    UIView          *contentView;
    UITableView     *subView;
    id<MyProtocol> myDelegate;
    id              targetVar;
    SEL             actionMethod;
}
...
@end
```

Target-action

Target-action és una forma de més "baix nivell" per redirigir interaccions amb l'usuari. Cocoa Touch ho fa servir casi exclusivament per a fills de la classe *UIControl*. Amb les *target-action* el programador indica al control que contacti amb un objecte en concret quan l'usuari realitza una acció. Per exemple, el programador especifica quin mètode de quin objecte s'ha de cridar quan l'usuari prem un botó.

El en Codi 3.2 tenim un bon exemple. Aquest petit tros de codi defineix un instància de la classe *UIBarButtonItem*, una mena de botó que hi ha a les barres de navegació de les aplicacions. El codi defineix que l'objecte destí és ell mateix (l'objecte on està definit el codi) i que la acció (el mètode a utilitzar) és `@selector(GoToNext:)`. Quan es prem el botó, aquest envia el missatge `GoToNext:` a l'objecte (executa el mètode).

Codi 3.5 - **Exemple de target-action**

```
UIBarButtonItem *buttonNext = [[UIBarButtonItem alloc] initWithTitle:@"Next"
        style:UIBarButtonItemStyleBordered
        target:self
        action:@selector(GoToNext:)];
```

Llavors, la classe que està creant la instància ha de tenir definit un mètode amb el nom de *`GoToNext:`*. Però, en comparació amb els delegats i els protocols obligatoris, no hi ha cap

garantia que mètode `GoToNext:` estigui definit a temps de compilació. És el programador el responsable d'assegurar-se que així sigui[5].

Notifications

Addicionalment als delegats i al *target-action*, iOS fa servir una altra manera de comunicar interaccions amb l'usuari entre el model del programador i la vista. Les notificacions (*Notifications)* permeten que els objectes d'una aplicació es comuniquin entre ells o amb altres aplicacions del sistema. Podríem dir, fent una analogia amb les xarxes de telecomunicacions, que les notificacions emeten una mena de missatge de *broadcast* a tothom qui escolti: "He canviat", "He començat a fer alguna cosa" o "He acabat".

Altres objectes poden estar escoltant i esperant aquesta mena de missatges per reaccionar a ells o no. El primer que ha de fer el programador és registrar l'objecte en el Centre de Notificacions d'Apple i indicar-li quina mena de notificacions ha d'esperar.

La classe més important en aquest context és la *NSNotificationClass*. Un objecte es pot subscriure a certes notificacions i, quan les rebi, rebrà un objecte que, a part del nom de la notificació, també pot contenir dades. Això dóna una gran flexibilitat al sistema.

Generalment el programador registra el *UIViewController* com a observador al centre de notificacions (*Notification Center*). El programador indica quin mètode executar quan rep una notificació. En el Codi 3.3 es registra un objecte perquè rebi totes les notificacions del sistema i mostra per la consola la informació de les diferents notificacions que va rebent. Reacciona a totes les notificacions perquè al paràmetre *Name* no hem indicat res, això indica que no volem escoltar cap notificació en concret.

---

[5] Per codificar una *target-action* en classes pròpies, el programador ha d'afegir una variable de tipus *id*(qualsevol objecte) com a *target* i una variable acció de tipus *SEL*(selector de mètode). Al Codi 3.1 estan definides com a *targetVar* i *actionMethod* respectivament. En temps d'execució es fa servir `performSelector: withObject:` per a executar el mètode indicat amb els paràmetres que es necessitin (o `nil` si no en necessita).

Codi 3.6 - **Exemple de target-action**

```
(void)viewDidLoad
{
    [super viewDidLoad];
     // Do any additional setup after loading the view, typically from a nib.

    [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(trackAllNotificactions:) name:nil object:nil];
}

- (void)trackAllNotificactions: (NSNotification *) notif
{
    NSLog(@"%@", [notif name]);
    NSLog(@"%@", [notif object]);
    NSLog(@"%@", [notif userInfo]);

}
```

3.2.3.2.3 Model

L'encarregat d'escriure tot el codi que defineix el que entenem per Model dins del MVC és el programador. Ell és qui crea tots els mètodes cridats pel Controlador i proveeix la implementació de tots els delegats. Per a aplicacions relativament simples, els detalls del Model solen estar codificats dins de la mateixa definició de la classe Controlador (que hereta de *UIViewController*).

Hi ha una part del SDK que sí que aporta una ajuda en quan a la definició del Model: es tracta dels *Data Sources*. Aquests ajuden a omplir els objectes del *UIKit* de dades amb contingut propi.

Orígen de dades (*Data Source*)

Un *Data Source*, un origen de dades, és qualsevol objecte que proveeix a algun altre objecte dades quan aquest ho demana. Alguns objectes de la interfície d'usuari són contenidors sense cap mena de contingut. Quan a aquests objectes se'ls assigna un origen de dades, el programador habilita l'objecte visual per a que agafi aquestes dades les mostri per pantalla. Normalment es parla de "dades" provinents d'una base de dades, d'un fitxer local o d'un servei web, d'entre altres. *UITableView* i *UIPickerView* són dues de les poques classes de Cocoa Touch que suporten o requereixen *data sources*.

Els orígens de dades són com delegats, en els quals el programador ha d'implementar els seus mètodes en altres objectes, típicament el *UITableViewController* propietari de la taula. La

diferència es troba en què, en aquest cas, creen o proveeixen objectes en lloc de reaccionar a accions de l'usuari.

El Codi 3.4 mostra un mètode típic de *data source* que retorna una cel·la d'una taula per la fila indicada. Com en altres mètodes d'origen de dades, aquesta manera de programar facilita la separació entre la manera de gestionar les dades i la manera de mostrar-les per pantalla. La manera de declarar mètodes d'origen de dades propi és la mateixa que amb els delegats, a partir de la definició de protocols.

Codi 3.7 - **Un mètode d'origen de dades omple una vista**

```
- (UITableViewCell *)tableView:(UITableView *)tableView
         cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *cellId = @"cellIdentifier";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:cellId];

    if (!cell)
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:cellId];

    cell.textLabel.text = [tableData objectAtIndex:[indexPath row]];
    return cell;
}
```

3.2.4 Diferències en la programació per a iOS

iOS és el sistema operatiu mòbil d'Apple. Originalment va ser creat per al telèfon mòbil d'Apple, l'iPhone i ara s'implementa també a l'iPod Touch, l'iPad i a l'Apple TV (Apple no permet la instal·lació de l'iOS en dispositius de tercers). Tot i ser petits, la potència computacional d'aquests dispositius és molt gran i, per poder treure'ls-hi partit, Apple ha dissenyat un sistema operatiu molt potent i un "kit de desenvolupament de software" SDK (*Software Development Kit*) amb totes les eines necessàries per a dissenyar, implementar i realitzar aplicacions.

Tota la interfície de control de iOS es basa en una manipulació directa fent servir gestos multitàctils (desplaçaments, pessics, tocs, etc.). Tots els elements de control són botons, interruptors, desplaçadors, etc. amb una resposta immediata i una interfície fluida. A més, els dispositius tenen sensors de proximitat, magnetòmetres (brúixoles), sensors de moviment, GPS, etc. que fan que el sistema reaccioni a certes accions del usuari. El SDK dóna accés a tots

aquests controls i sensors per a crear aplicacions amb una gran capacitat de interaccionar amb l'usuari.

iOS és un sistema que deriva del Mac OS que, alhora, està basat en Darwin BSD, així que es tracta d'un sistema Unix. Però tot i això, es tracta d'un sistema operatiu per a dispositius mòbils, la qual cosa comporta uns canvis en la manera de programar, dissenyar i gestionar recursos respecte a les aplicacions d'escriptori.

3.2.4.1 Una sola aplicació activa.

En iOS pot estar activa una sola aplicació a al mateix temps. A partir de l'iOS 4, les aplicacions poden córrer en *background* després de què l'usuari premi el botó 'Home' del dispositiu. Però tan sols poden fer una sèrie d'accions limitades i s'han de programar específicament. Les accions permeses per a aplicacions en *background* són: la reproducció de música (local o en *streaming*), VoIP, seguir informat de la localització (GPS), seguir comunicant-se amb accessoris externs al dispositiu, comunicació amb accessoris per *Bluetooth* o, si la aplicació és un Newsstand, pot seguir descarregant i processant les revistes o diaris que necessiti.

3.2.4.2 Tot en una sola finestra

Els sistemes operatius d'ordinadors de sobretaula o portàtils permeten que s'executin diferents programes al mateix temps i, a més, que cadascun d'aquests gestioni més d'una finestra a la vegada. L'iOS dóna a les aplicacions una sola finestra per a treballar. Tota la interacció de l'usuari es fa a través d'aquesta finestra i la seva mida està limitada per les dimensions de la pantalla del dispositiu.

3.2.4.3 Accés limitat

Els programes d'escriptori solen tenir permisos per accedir a tots els recursos que l'usuari tingui també permisos. En iOs això no és així, iOS restringeix molt l'accés a cadascuna de les aplicacions. Aquestes tan sols poden llegir i escriure dins de la part de l'iOS que s'ha creat per a cada aplicació, anomenada *sandbox*. Dins de la *sandbox* l'aplicació por desar els documents, les preferències, i totes les dades que es necessiten.

Les aplicacions també tenen altres limitacions: no poden accedir a ports amb numeració baixa in poden realitzar cap operació que necessiti privilegis d'administrador.

3.2.4.4 Temps de resposta limitat

Quan es fa servir el dispositiu, l'iOS necessita ser ràpid i proporcionar respostes ràpides a l'usuari. Quan aquest realitza una acció necessita una resposta ràpida; la mateixa rapidesa s'espera de les aplicacions de tercers. Quan una aplicació és executada, aquesta s'ha d'obrir, s'han de carregar les dades i preferències, i la vista principal ha de ser visible a la pantalla el més aviat possible.

En qualsevol moment de l'execució de l'aplicació, l'usuari pot prémer el botó de 'Home' i l'aplicació ha de ser capaç de desar tot el que calgui i sortir en molt poc temps. Si l'aplicació tarda més de 5 segons el procés és finalitzat pel sistema, hagi o no acabat. Nos obstant això, a partir de l'iOS 5 s'ha afegit una nova API al framework que dóna la possibilitat de demanar més temps per realitzar aquest tipus de gestions.

3.2.4.5 Mida limitada de pantalla

Tot i que els nous iPhone i iPad compten amb pantalles de resolució enorme i de gran qualitat, la mida de les pantalles als dispositius és limitada i les interfícies d'usuari s'han de crear tenint en compte aquestes limitacions.

3.2.4.6 Recursos limitats

Qualsevol programador amb anys d'experiència riuria amb la idea de 'recursos limitats' davant en 256MB de RAM i 8GB de disc dur, però sí són limitats. Desenvolupar aplicacions per a iOS no és el mateix que intentar escriure codi per una a màquina de 48KB de memòria. No obstant, donada la naturalesa gràfica de l'iOS i tot el que es capaç de fer, trobar-se sense memòria disponible no és gaire difícil.

Actualment els dispositius d'Apple tenen 256MB, 512MB o 1GB de RAM i compten amb processadors de més d'un nucli amb grans capacitats gràfiques. Però gran part d'aquesta memòria es dedica al *buffer* de pantalla i altres processos del sistema. Normalment no queda

més de la meitat de la memòria lliure per fer-la servir. Si afegim que podem tenir aplicacions actives executant certs processos en *background*, la memòria disponible pot ser considerablement inferior.

Tot i això, sembla que es segueix tenint una quantitat decent de memòria per un a dispositiu tan "petit", però hi ha un altre factor a tenir en compte en termes de memòria disponible en iOS. Els sistemes operatius moderns com Mac OS X agafen espais del disc dur que no es fan servir i els destinen a crear el que s'anomena *swap file*; fan servir aquests troços de disc dur com si fossin memòria RAM. El *swap file* permet a aplicacions seguir executant-se tot i que aquestes necessitin més memòria RAM de la que hi ha disponible al sistema. Però iOS no escriu dades volàtils al *swap file*, com les d'una aplicació. Al final, la quantitat de memòria disponible per a una aplicació es limita a la quantitat de memòria física que no està fent-se servir al dispositiu iOS.

3.2.4.7 No hi ha *Garbage Collector*, però ...

Cocoa Touch fa servir Objective-C. No obstant això, una de les característiques d'aquest llenguatge no està disponible per iOS, però sí per Mac OS; Cocoa Touch no suporta el *Garbage Collector*. La necessitat de gestionar manualment la memòria ha estat una dificultat afegida per a molts dels nous programadors, sobretot per a aquells que venen d'altres llenguatges que sí que tenen una gestió automàtica de la memòria.

Aquesta carència s'ha solucionat amb la versió d'Objective-C suportada a partir de l'iOS 5. L'iOS 5 introdueix una nova característica anomenada *Automatic Reference Counting* (ARC), la qual es desfà de la necessitat de gestionar manualment la memòria dels objectes d'Objective-C.

3.2.4.8 Nous recursos

Hem vist que Cocoa Touch no té totes les característiques del Cocoa, però si que en té algunes que no hi són en la versió d'escriptori.

- El SDK de l'iOS proveeix eines per determinar la localització geogràfica del dispositiu (ja sigui per GPS o per triangulació amb antenes mòbils o amb xarxes Wifi) fent servir el SDK de *Core Location*.

- Molts dels dispositius amb iOS tenen càmeres incorporades i galeries de fotografies, el SDK proveeix mecanismes que permeten a les aplicacions accedir tant a la càmera com a les galeries d'imatges.
- Els dispositius amb iOS també tenen acceleròmetres i giroscopis incorporats. El SDK permet als desenvolupadors accedir a les dades d'aquests sensors i detectar quan el dispositiu es mou i, fins i tot, cóm es mou per a interactuar amb ell.

3.2.4.9 Maneres diferents d'interactuar

Dos accessoris que els dispositius no tenen són un teclat físic i un ratolí. Això vol dir que la manera d'interactuar de l'usuari amb el dispositiu és totalment diferent que amb aplicacions d'escriptori. Afortunadament, la majoria d'aquestes interaccions són gestionades pel sistema. Per exemple, si una aplicació té un camp de text, l'iOS sap que ha de mostrar el teclat virtual i el programador no s'ha de preocupar[6].

Addicionalment, el framework proveeix de les APIs necessàries per a gestionar gestos més elaborats de l'usuari amb molt poques línies de codi.

3.2.4.10 Limitacions imposades per Apple

Hi ha un gran nombre de limitacions que no existeixen per manca de recursos. Apple vol que totes les aplicacions del seu mercat compleixin una sèrie de requisits i normes. Algunes d'aquestes normes són per protegir l'usuari i assegurar un nivell de privacitat per a les dades que tracten les aplicacions. Altres normes són per mantenir un nivell d'experiència d'usuari dins del seu sistema. També hi ha normes per no permetre certs tipus d'aplicacions que podrien ferir la sensibilitat d'alguns usuaris, i d'altres que protegeixen, senzillament, els interessos d'Apple.

Les normes i restriccions per a publicar un aplicació a l'*App Store* d'Apple han estat sovint motiu de discussió i noticia en el món tecnològic. Es poden trobar al portal de desenvolupadors d'Apple accedint amb el compte de desenvolupador.

---

[6] Els dispositius actuals permeten connectar teclats externs mitjançant Bluetooth o el mateix connector del dispositiu, els quals donen una experiència d'escriptura superior i deixen més espai a la pantalla per treballar.

3.3 La primera aplicació

Ara veurem com es crea una aplicació per l'iPhone fent servir algunes de les eines esmentades. L'aplicació tan sols mostrarà una taula per pantalla i la omplirà amb una llista de paraules. L'objectiu és presentar l'entorn de desenvolupament Xcode, passar per tots els diferents passos per crear una aplicació i mostrar algunes de les tecnologies que s'han descrit en punts anteriors.

Donem per fet que el programador ja s'ha donat d'alta en algun dels programes d'Apple i s'ha descarregat les eines necessaries: l'Xcode i l'SDK per iOS.

3.3.1 Creació del projecte

El primer pas és executar Xcode. Un cop s'obre, apareix una pantalla inicial on podem obrir els nostres projectes o crear-ne un de nou, entre altres coses. Crearem un nou projecte.

La següent pantalla és important. En aquesta pantalla (Figura 3.1), Xcode ens dóna a escollir entre un conjunt de plantilles preparades per a desenvolupar diferents tipus de projectes. Un cop escollim quin tipus de projecte volem, Xcode crearà les carpetes i els arxius necessaris d'acord amb la selecció per a estalviar-nos feina.

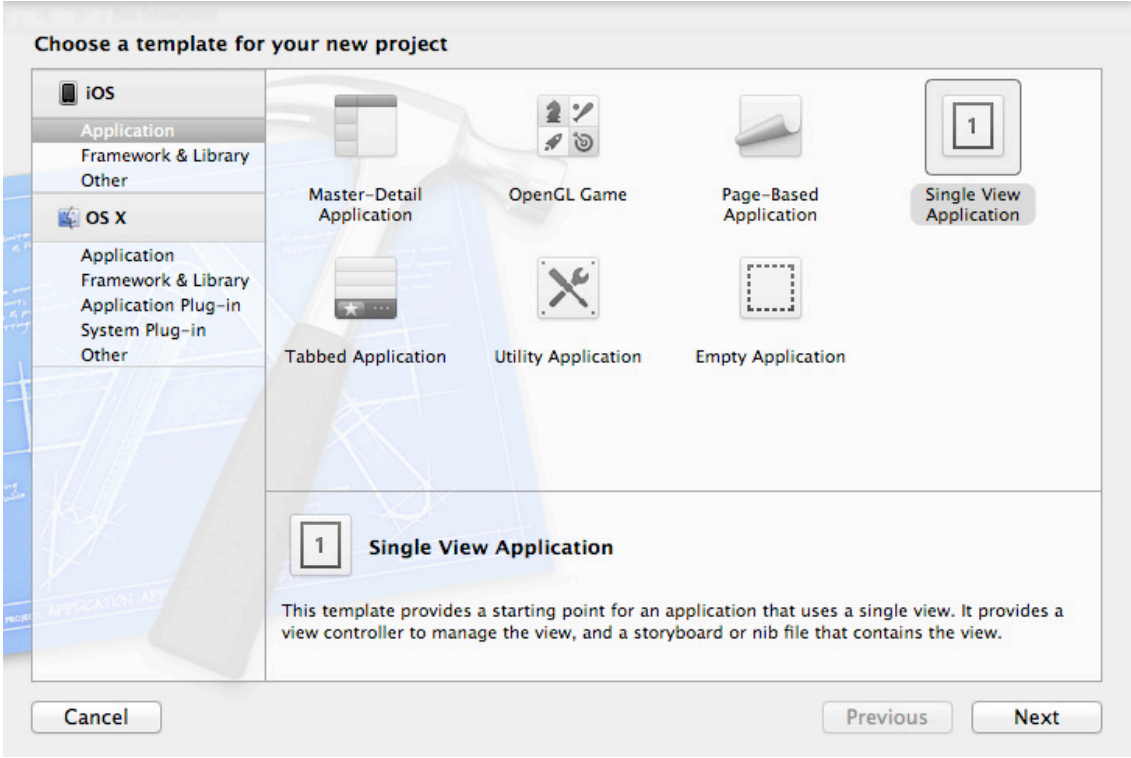

Fig. 3.1 - **Pantalla de selecció de plantilla d'Xcode**.

El primer que veiem a la esquerra és la selecció de la plataforma on voldrem desenvolupar el nostre programa; escollim iOS i *Application*. A la dreta seleccionem el tipus de projecte que volem crear; el projecte "Single View Application" ens va perfecte per el que nosaltres busquem. A l'hora d'aprendre va bé provar les diferents plantilles per veure quina quantitat de codi i arxius genera cadascuna, per tenir-ho present en el moment en que ho necessitarem.

En el següent pas hem de donar-li un nom al projecte, l'anomenarem "SimpleTable". Mirant els altres camps el primer que potser ens crida l'atenció és "Class Prefix": aquest camp ens deixa definir un prefix per a totes les classes que creem en el nostre projecte. Això és de gran ajut ja que evita la creació de duplicats. Ens assegurarem que no hi ha cap altre classe d'una altre llibreria amb el mateix nom i, a més, ens identificarà totes les nostres classes. Més endavant veurem com les classes dels diferents *frameworks* interns de Cocoa també tenen prefixos. Posarem "PFC" com a prefix.

Com a dispositiu seleccionarem "iPhone" i deixarem marcada la opció "*Use Automatic Reference Counting*" i desmarcades les altres dues. Aquesta opció que hem marcat farà que sigui el propi compilador l'encarregat de gestionar la memòria. Finalitzem escollint en quina carpeta del sistema volem guardar el nostre projecte.

Un cop fet això hauríem de tenir davant una pantalla semblant a la figura 3.2.

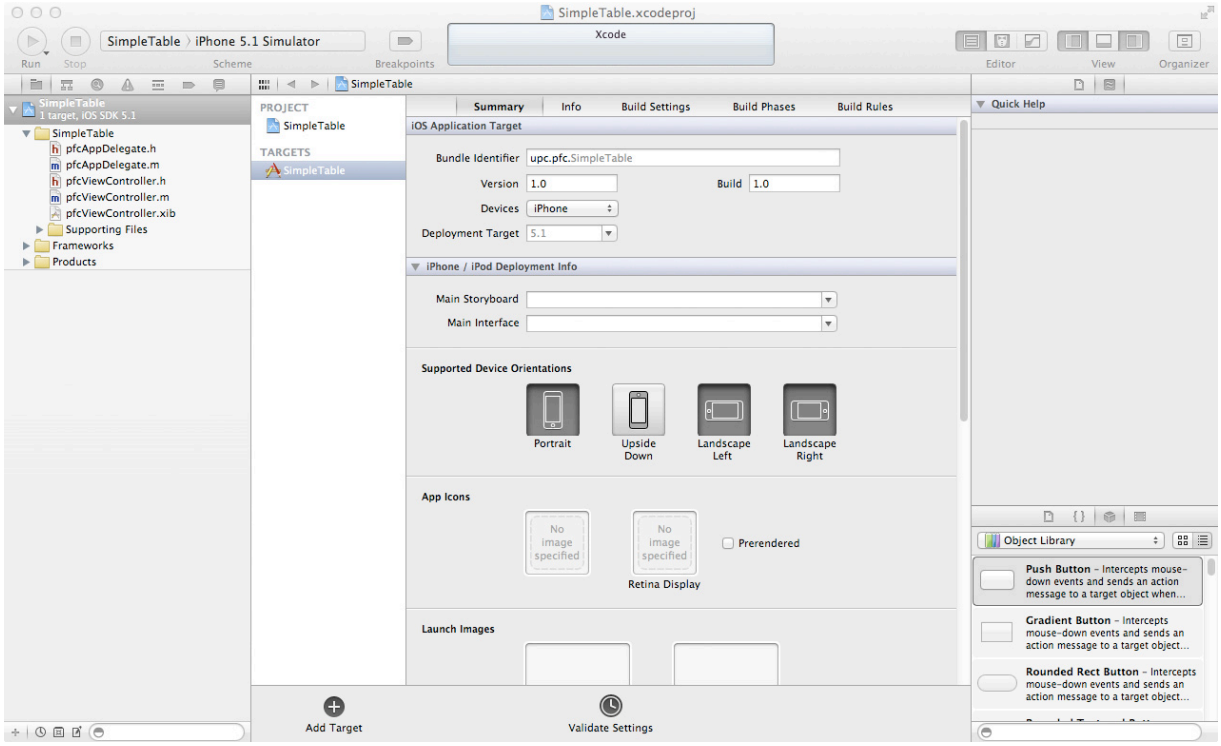

Fig. 3.2 - **Plantilla creada per Xcode per a "Single View Application"**

Xcode té tres seccions ben marcades. A l'esquerra trobem la navegació pels diferents fitxers del projecte. Escollint entre les petites icones que hi ha a la capçalera de la dreta canviem la visualització: arxius del projecte, símbols, errors, etc. A la dreta hi ha un menú que, segons el que tenim seleccionat, s'adapta per mostrar les eines que necessitem. Si estem dissenyant, mostrarà les propietats dels elements que seleccionem; si estem escrivint codi, ens informarà de les classes que toquem i dels mètodes que utilitzem, etc. A sota d'aquest, sempre visible, tenim el seleccionador de components visuals. En aquest petit llistat trobem tots els components que podem fer servir per a dissenyar les interfícies. Aquest llistat s'anomena llibreria d'objectes. Per acabar, al mig tenim l'àrea de treball, on programarem i dissenyarem.

3.3.2 Dissenyar la vista

En els arxius de projecte, expandim el projecte *SimpleTable* i la carpeta *SimpleTable*. Seleccionem l'arxiu que ens ha creat l'Xcode anomenat PFCViewController.xib. Els arxius *.xib* són de disseny; en ells dissenyarem d'una manera visual i molt intuïtiva com volem que sigui l'aparença de la nostra aplicació. Al seleccionar-lo, l'entorn de treball central canviarà i mostrarà una vista (la pantalla de l'iPhone) buida. Aquesta vista ens mostra també la típica barra on es troba l'indicador de bateria de l'iPhone; així tindrem sempre visible com es veu i la mida que té el nostre disseny. Ara agafem de la llibreria d'objectes un *Table View* (veure figura Fig. 3.3), l'arrosseguem i el deixem anar sobre la vista.

El *Table View* hauria d'agafar la mida de la vista automàticament, tal i com volem. Les vistes estan dissenyades per agafar tota l'amplada i tota l'alçada de la pantalla tenint en compte si hi ha barres de navegació, barres d'eines o altres elements.

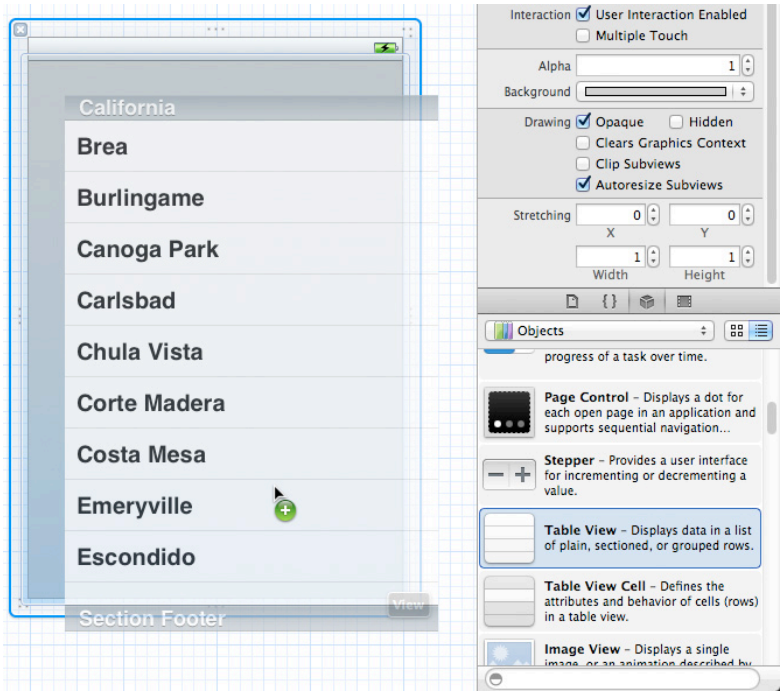

Fig. 3.3 - **Afegir un *Table View* a la vista**

La taula hauria d'estar seleccionada, si no, la seleccionarem amb un simple *click* del ratolí. A la dreta veurem les propietats de la taula. A dalt, si ens posem a sobre de les petites icones

apareix una petita etiqueta que ens mostra per a què serveixen. La sisena icona ens mostra l'inspector de connexions (*Show Connector Inspector*); seleccionem-la. Els dos primers connectors que hi ha es diuen *dataSource* i *delegate*. Seguint la imatge de la figura 3.4, seleccionem el petit cercle a la dreta de cadascun d'ells i l'arrossegem fins a la icona més alta de les tres que hi ha a l'esquerra de l'àrea de disseny. Veure'm que aquestes icones representen el *File's Owner*, el *First Responder* i la *View* que estem dissenyant. Hem connectat el *dataSource* i el *delegat* al *File's Owner*. El que estem fent amb això és convertir la nostra classe controlador en el delegat i en l'origen de dades de la taula.

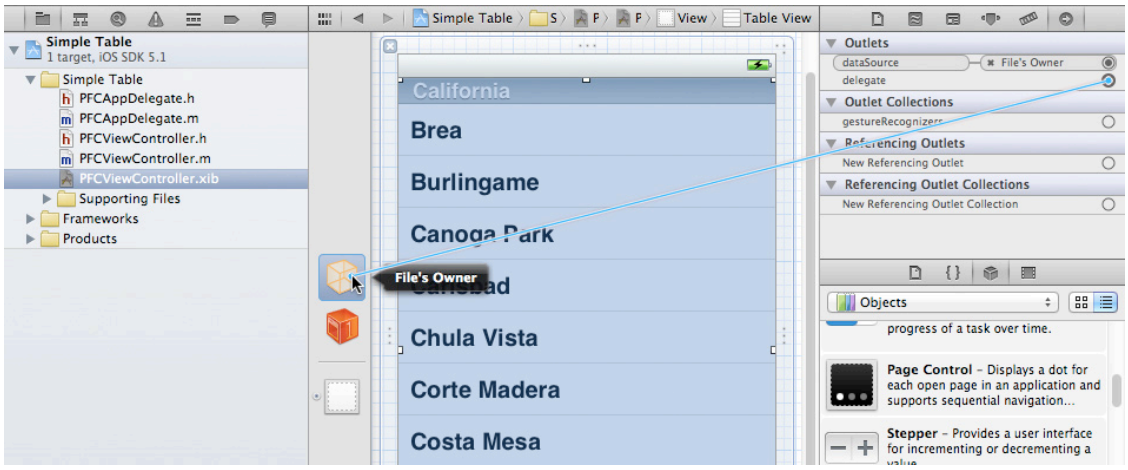

Fig. 3.4 - **Connexió del delegat i l'origen de dades al nostre controlador**

Un cop fet això podem salvar l'arxiu i ja estem llestos per escriure una mica de codi.

3.3.3 Escriure el codi del controlador

Obrim el fitxer de capçalera del controlador: *BIDViewController.h*, i afegim el següent codi.

Codi 3.8 - **Definició dels protocols del *UITableView que* implementarà el controlador**

```
#import <UIKit/UIKit.h>

@interface PFCViewController : UIViewController
    <UITableViewDelegate, UITableViewDataSource>

@property (strong, nonatomic) NSArray *listData;

@end
```

El que hem fet es indicar que la nostre classe PFCViewController implementa els dos protocols necessaris per actuar com a delegat i com a origen de dades a la taula, i també hem declarat una llista que contindrà les dades que es mostraran a la taula.

Salvem els canvis. Ara anem a implementar el controlador. Seleccionem l'arxiu anomenat *PFCViewController.m* i afegim el següent codi al principi del fitxer.

Codi 3.9 - **Implementació del controlador**

```
#import "PFCViewController.h"

@implementation PFCViewController

@synthesize listData;

- (void)viewDidLoad
{
    [super viewDidLoad];
     // Do any additional setup after loading the view, typically from a nib.

    NSArray *array = [[NSArray alloc] initWithObjects:
                      @"Hola", @"Món", @",", @"Què", @"tal", @"estàs", @"?"
    self.listData = array;
}

- (void)viewDidUnload
{
    [super viewDidUnload];
    // Release any retained subviews of the main view.
    self.listData = nil;
}

...
```

El primer que fem és implementar el mètode que s'executa quan la vista és carregada (`viewDidLoad:`). Aquí tan sols inicialitzem la llista amb les dades amb les quals s'omplirà la taula. En una aplicació real,aquesta part podria obtenir les dades d'un fitxer, d'un servei web, d'una base de dades, etc.

3.3.4 Implementar els mètodes d'origen de dades

Ara hem d'implementar els mètodes de delegat i d'origen de dades dels protocols que hem indicat en la definició de la classe en el fitxer *.h*. Si no ho fem i provem de compilar el projecte, aquest compila sense problemes. Però si intentem executar-ho, l'aplicació es tanca amb un error:

```
Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: '-
[PFCViewController tableView:numberOfRowsInSection:]: unrecognized selector sent to
instance 0x6a2c1a0'
```

Quan indiquem a la taula que el nostre controlador és el seu delegat i origen de dades, aquesta envia el mètode d'origen de dades `tableView:numberOfRowsInSection:` al nostre controlador, però donat que aquest no l'ha implementat, això provoca un error. Implementarem doncs els mètodes.

Codi 3.10 - **Implementació dels mètodes del protocol d'origen de dades**

```
...

#pragma mark -
#pragma mark Table View Data Source Methods

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    return [listData count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
         cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *cellId = @"cellIndentifier";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:cellId];
    if (cell == nil) {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
                                      reuseIdentifier:cellId];
    }

    NSUInteger row = [indexPath row];
    cell.textLabel.text = [listData objectAtIndex:row];

    return cell;
}

@end
```

Fins aquí hem implementat dos mètodes: tots venen definits pel protocol `UITableViewDataSource`. El primer dels dos, `tableView:numberOfRowsInSection:`, el fa servir la taula per preguntar quantes files hi ha en una secció en particular. Si aquest mètode no s'implementa (aquest no és obligatori) retorna 1 per defecte. Nosaltres només mostrem una secció i aquesta té tantes files com elements té la nostra llista.

El següent mètode és una mica més complicat i el mirarem amb més detall. El mètode `tableView:cellForRowAtIndexPath:` és cridat per la taula cada cop que intenta dibuixar una fila.

El segon paràmetre és una instància de `NSIndexPath`, un objecte fet servir per les taules per empaquetar una secció i una fila en un mateix objecte. Per saber la fila o la secció només cal cridar el seu mètode `row` o `section`; tots dos retornen un nombre sencer. El primer paràmetre, `tableView`, és una referència a la taula que fa la petició; això ens permet crear classes que actuen com a origen de dades per múltiples taules.

Després definim una instància d'una cadena estàtica:

```
static NSString *cellId = @"cellIndentifier";
```

Aquesta cadena es fa servir com a clau per a representar el tipus d'una cel·la d'una taula. La nostra taula només fa servir un sol tipus.

Una taula tan sols pot mostrar unes poques files a la vegada en la petita pantalla d'un iPhone; però, tot i això, la taula pot tenir moltes més files. Si pensem que cadascuna de les files està representada per un objecte de tipus *UITableViewCell*, una sub-classe de *UIView*, vol dir que cada una de les files pot contenir altres vistes al seu interior. Amb taules molt grans, això pot suposar que la taula ha de mantenir la informació de totes les files i les seves vistes internes encara que no s'estiguin mostrant per pantalla. Això consumeix molta memòria innecessàriament. Afortunadament, les taules en iOS no funcionen d'aquesta manera.

Un cop una cel·la surt de la pantalla es posa en la cua de cel·les que estan disponibles per a ser re-utilitzades. Si el sistema es troba baix de memòria, la taula es desfarà de les cel·les d'aquesta cua però, si el sistema té memòria disponible per a aquestes cel·les, es mantindran en memòria per si es volen tornar a fer servir.

Cada cop que una cel·la surt de la pantalla és bastant normal que una altre aparegui per l'altre cantó. Si aquesta nova cel·la pot fer servir una d'aquestes que acaba de sortir, el sistema evita sobre-carregar la memòria creant i destruint constantment aquestes vistes. Per aprofitar aquest avantatge, al crear la cel·la demanem primer a la taula si té alguna cel·la del tipus que necessitem, llesta per ser re-utilitzada.

```
UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:cellId];
```

Però és possible que la taula no en tingui cap, per exemple al començament. Llavors ho hem de comprovar i crear-ne una si la taula no ens ha donat cap.

```
if (cell == nil) {
    cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
                                   reuseIdentifier:cellId];
}
```

Ja tenim la cel·la que podem retornar a la taula per a que aquesta la faci servir; però primer la hem d'omplir amb informació correcta. Mostrar text en una fila és un tasca molt comuna, així que l'objecte *UITableViewCell* ja té per defecte una propietat anomenada *textLabel* que nosaltres podem fer servir per mostrar text. Només cal agafar la cadena de text correcta de la nostra llista i fer-la servir. Hi haurà tantes files com elements tinguem en la nostra llista, així que si sabem quina fila estem dibuixant, sabrem quin element de la nostra llista necessitem. Això ho podem saber amb el paràmetre *indexPath*. I un cop tenim la cel·la llesta, la retornem.

```
NSUInteger row = [indexPath row];
cell.textLabel.text = [listData objectAtIndex:row];

return cell;
```

3.3.4 Compilar i provar l'aplicació

Si compilem i executem el codi (*Product->Run*) s'executarà el simulador de l'iPhone i podrem veure el resultat (Figura 3.5).

Fig. 3.5 - **Resultat en el simulador**

3.3.5 Implementar els mètodes de delegat

Tot i que hem definit la nostra classe controlador com a delegat de la taula amb el protocol `UITableViewDelegate`, no hem implementat cap els mètodes de delegat d'aquest. L'aplicació funciona perquè, al contrari que el protocol d'origen de dades, el protocol de delegat de taula no té cap mètode obligatori.

Anem a implementar un dels mètodes de delegat més comuns d'un protocol de taula: el mètode que llançarà la taula quan l'usuari seleccioni una fila. Per a fer-ho afegim el codi 3.11 al final de la implementació de la classe.

Codi 3.11 - **Implementació dels mètodes de protocol de delegat**

```
...

#pragma mark -
#pragma mark Table View Data Source Methods

- (void)tableView:(UITableView *)tableView
        didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSString *selectedWord = [listData objectAtIndex:[indexPath row]];
    NSString *title = @"Fila seleccionada";
    NSString *message = [NSString stringWithFormat:
                         @"Has seleccionat la paraula '%@'", selectedWord];
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:title
                                                    message:message
                                                   delegate:nil
                                          cancelButtonTitle:@"Vale!"
                                          otherButtonTitles:nil];

    [alert show];
}
@end
```

El mètode `tableView:didSelectRowAtIndexPath:` es llança cada cop que l'usuari selecciona una fila. Com en el cas dels mètodes d'origen de dades, un dels paràmetres que rep és la referència a la taula en la que s'ha produït l'acció, l'altre ens indica quina fila ha seleccionat l'usuari. El següent codi mostra un *popup* de sistema, indicant quina paraula hi havia a la fila seleccionada.

Si tornem a executar l'aplicació i seleccionem una de les files, el resultat hauria de ser semblant al de la Figura 3.6.

Fig. 3.6 - **Mostra el *popup* al seleccionar la fila.**

3.3.6 Accedir als objectes de les *views* de l'aplicació

Hi ha una acció molt comú en el moment de desenvolupar aplicacions que no queda coberta amb aquest exemple i que veurem ara. En el dissenyador d'Xcode podem arrossegar molts controls de la llibreria i crear la interfície d'usuari que vulguem. Però, cóm s'accedeix des del controlador als diferents controls que hem col·locat a la pantalla? Anem a veure-ho.

Crearem una referència a la taula a la classe controlador. Per a fer-ho, primer hem de seleccionar l'arxiu .xib de disseny. Un cop obert, anem a les opcions que hi ha a dalt a la dreta (Fig. 3.7). Hi ha tres grups de botons: el primer es diu *Editor*, el següent *View* i el tercer *Organizer*. El primer canvia la forma de visualitzar l'àrea de treball, el segon oculta o ensenya els panells i el tercer obre una eina anomenada *Organizer* on es pot, per exemple, gestionar característiques del simulador i dels dispositius, obrir la documentació d'Apple, etc.

En les opcions *Editor*, seleccionem la del mig. Això fa que l'àrea de treball es divideixi en dos; a l'esquerra es segueix veient l'arxiu que teniem obert, a la dreta es veu un altre fitxer que, si

Xcode pot, serà l'arxiu relacionat amb el de l'esquerra. En aquest cas Xcode haurà obert la definició del nostre controlador, el fitxer *.h*. Si la pantalla no és prou gran a és una bona idea ocultar el panell de la dreta amb les opcions de *View*, ja que ara no el farem servir i es guanya espai.

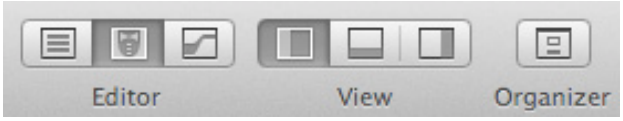

Fig. 3.7 - **Opcions de visualització de la interfície d'Xcode**

Un cop tenim les dues finestres obertes; a l'esquerra el disseny i a la dreta la definició de la classe *PFCViewController*, clicarem sobre la taula per assegurar-nos que està seleccionada. Ara, mantenint pressionada la tecla **crtl,** arroseguem la taula cap a al panell de la dreta, just a sota de la línia que defineix la variable `listData`, tal i com es veu a la figura 3.8.

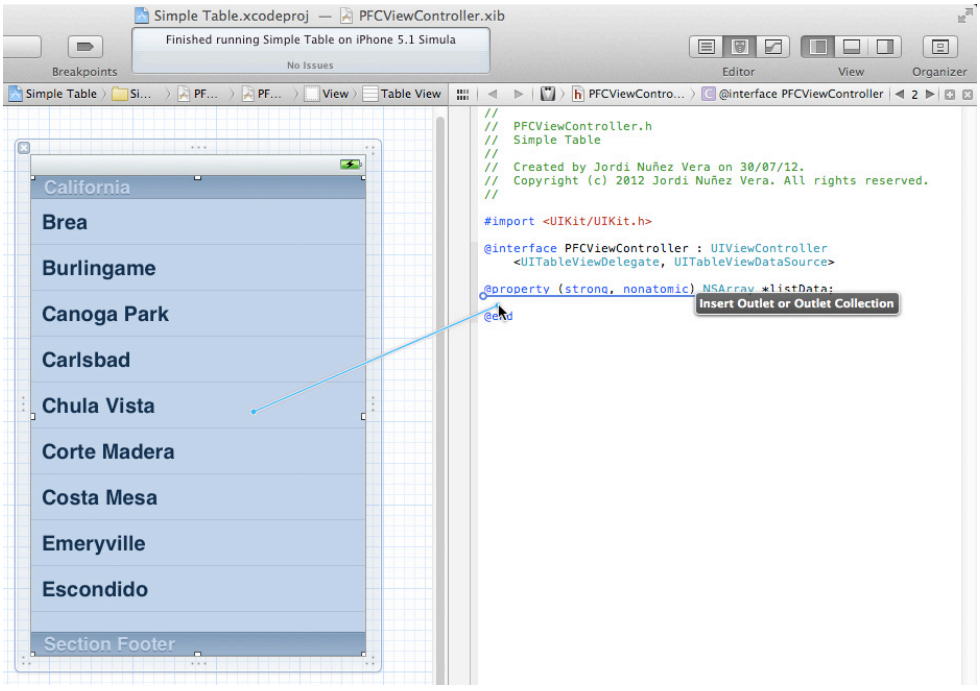

Fig. 3.8 - **Seleccionem la taula i l'enllacem al controlador**

Sortirà un petit *popup* amb diferents opcions (Fig. 3.9). La primera opció és el tipus de connexió, seleccionem *Outlet*. Un *Outlet* és qualsevol variable que està enllaçada amb un control visual. Li

donarem un nóm, *myTable*, i deixarem les demés opcions com estan. Acceptem amb el botó *Connect*.

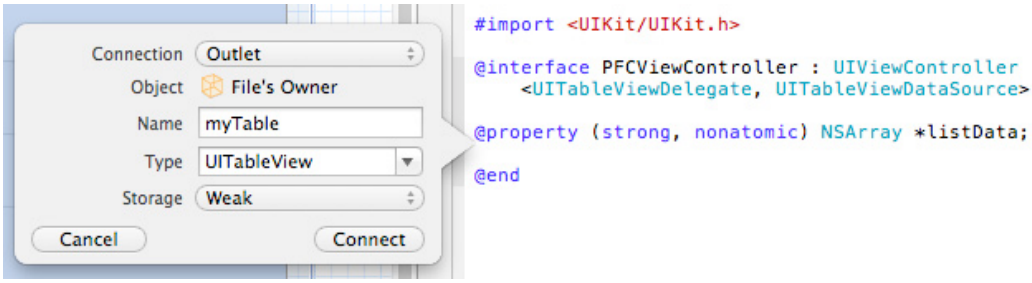

Fig. 3.9 - **Opcions d'enllaç d'un *Outlet* o *Action***

Aquesta acció crearà una línia de codi sota la declaració de la llista de dades, *listData*. Si ens fixem hi ha un punt a l'esquerra de la línia que està ple de color gris. Si esborrem aquesta línia i la escrivim a mà, apareixerà el mateix punt però buit; això indica que s'ha creat un *Outlet* però encara no s'ha enllaçat a cap control visual (s'ha de seguir el mateix procediment anterior per crear l'enllaç). També veiem que aquesta variable té el modificador *IBOutlet*, que indica que fa referència a un objecte que hereta de *UIView*.

```
@property (weak, nonatomic) IBOutlet UITableView *myTable;
```

Ara ja tenim una variable pública en el nostre objecte controlador que fa referència al control de pantalla. En qualsevol part del codi podem accedir a aquesta variable i editar el control de pantalla.

3.3.7 Enllaçar esdeveniments dels controls

Si en comptes d'haver estat una taula haguéssim fet el mateix amb un botó, al primer camp del *popup* de creació d'enllaç anterior *Connection*, hi hauria també l'opció *Action*. Aquesta opció, en comptes de crear una variable enllaçada amb el control, crea un mètode enllaçat a l'esdeveniment del control que s'hagi seleccionat en el *popup*. D'aquesta manera es poden codificar les accions que s'han d'executar quan, per exemple, es fa *clic* en un botó.

3.4 Conclusió

Amb aquesta petita introducció a l'entorn de desenvolupament d'aplicacions per iOS i al ecosistema Apple espero haver donat la informació necessària per a poder començar a endinsar-se en aquest món.Dsenvolupar aplicacions per l'iPhone no es pot aprendre en poc més de 30 pàgines amb un exemple d'una aplicació amb una taula, però aquí estan les eines, els coneixements i les referències a alguns recursos per donar el primer pas per fer-ho.

3.5 Altres opcions

Actualment el mercat de telèfons intel·ligents està dominat per dues grans plataformes on desenvolupar aplicacions: iOS i Android. Per darrera d'aquestes tenim plataformes com Blackberry o Windows Phone, però el volum de desenvolupadors i aplicacions per aquestes plataformes queda eclipsat per les dues primeres.

Amb aquesta diversitat i la potència i versatilitat del nou estàndard HTML5, la programació multi-plataforma està prenent forma a Internet. Actualment hi ha diverses empreses desenvolupant entorns de treball i *frameworks* que permeten al programador desenvolupar aplicacions multi-plataforma. Amb aquestes solucions, no només es pot desenvolupar aplicacions per l'iOS sense haver de tenir un ordinador d'Apple, sinó que l'aplicació es pot exportar de tal manera, que s'obté el mateix producte per a diferents plataformes: iOS, Android, Windows Phone, Blackberry, etc.

Algunes d'aquestes solucions són, per posar alguns exemples: Phonegab, Marmalade, ParticleCode, Corona SDK, Unity, etc. No entrarem en detall en cap d'elles, però és important veure com aquest mercat està creixent d'una manera exponencial i molt ràpidament.

- Capítol 4 -

CLIENT HLS EN UNA APLICACIÓ PER L'iPHONE

4 Client HLS en una aplicació per l'iPhone.

4.1 Objectiu del projecte

L'objectiu d'aquesta part del projecte és implementar part del protocol HTTP Live Streaming en una aplicació per a l'iPhone. El que s'espera d'aquesta aplicació és que sigui un exemple didàctic del protocol més que una implementació real d'aquest; on es pugui veure clarament el seu funcionament i caràcter adaptatiu.

4.2 Aspectes a tenir en compte sobre aquesta aplicació

Per a poder oferir un caràcter didàctic i centrar-se en la facultat adaptativa del protocol, aquest no està desenvolupat en l'aplicació seguint totes les restriccions i normes marcades dins la documentació oficial d'Apple. Tot i això, la implementació segueix la descripció bàsica del protocol, amb l'omissió de parts com el xifrat de dades, la verificació de versió o les limitacions en la codificació dels continguts.

Tenint en compte que es tracta d'un client per iPhone i seguint el camí del capítol anterior, es descriuen les solucions emprades per a la programació del client i el seu funcionament en les parts més interessants i complicades.

4.3 Eines i tecnologies emprades

Tot el desenvolupament de l'aplicació s'ha realitzat en un Macbook 13" d'Apple amb el sistema operatiu MAC OS X Lion i fent servir les eines d'Apple: Xcode 4 i iOS 5 SDK. El servidor web utilitzat per al desenvolupament de l'aplicació ha estat un servidor Apache instal·lat en la mateixa màquina de desenvolupament. Per a les proves i l'entorn de producció s'han utilitzat dos servidors, un Apache en un servidor cedit per la universitat i un ordinador amb Windows 7 Professional i el servidor de web de Microsoft Internet Information Server 7.

El mòbil de proves és un iPhone 4.

4.4 Requisits de l'aplicació

- L'aplicació s'ha de descarregar una llista amb els diferents *streams* de vídeo d'un servidor. Aquesta llista serà un fitxer en XML que contindrà l'enllaç al fitxer d'índex .m3u8 de cadascun dels vídeos.
- L'usuari ha de poder seleccionar diferents orígens de dades.
- L'usuari ha de poder afegir un origen de dades propi.
- L'aplicació ha de ser capaç de mostrar la llista de vídeos per pantalla perquè l'usuari pugui escollir el vídeo a reproduir.
- L'aplicació, un cop mostrada la llista per pantalla, ha de ser capaç de descarregar els fitxers d'índex de cada vídeo i analitzar la informació que aquests contenen d'una manera asíncrona.
- Quan l'usuari selecciona un vídeo, l'aplicació ha de mostrar la informació d'aquest, així com la informació del seu fitxer d'índex d'un manera didàctica, de manera que es vegi clarament com estan formats els fitxers d'índex i el seu contingut.
- L'aplicació ha de funcionar amb vídeos sota demanda.
- L'aplicació ha de ser capaç de reproduir el vídeo d'una manera didàctica, on es vegi clarament cada pas del protocol, la descàrrega de segments i l'adaptació d'aquest a l'ample de banda disponible.
- L'usuari ha de tenir la possibilitat de simular variacions en l'ample de banda disponible a fi de poder veure la capacitat adaptativa del protocol.
- L'aplicació ha de generar un traça de totes les accions que fa el protocol i donar a l'usuari l'opció de visualitzar-la un cop acabada la reproducció.

4.5 El Servidor

Per poder realitzar l'aplicació, provar-la i que funcioni cal disposar d'una font de dades. Es necessiten una sèrie de vídeos comprimits en diferents qualitats, segmentats i els fitxers d'índex necessaris per a cadascun d'aquests. Després cal un fitxer que contingui els enllaços als diferents fitxers d'índex d'aquests vídeos i la descripció d'aquests. Tots aquests fitxers hauran d'estar en un (o més) servidors web i ser accessibles per a l'usuari final.

4.5.1 Preparació dels continguts

Un cop disposem de diversos vídeos comprimits en diferents qualitats necessitem segmentar-los en petits fragments i generar els fitxers d'índex seguint l'especificació del protocol HLS. Per a realitzar aquesta tasca és necessari un segmentador o *segmenter* [Cap. 2.2.1]; un *software* que sigui capaç d'agafar aquests vídeos comprimits en H.264, encapsular-los en trames MPEG-2 TS i segmentar-los en trossos d'uns 10 segons.

Apple proporciona un conjunt d'eines que ajuden a preparar els continguts d'un servei de HTTP Live Streaming. Aquestes eines són un *stream segmenter*, un *segmenter* de fitxers, un validador de fitxers d'índex, un generador d'etiquetes id3 i un generador de llistes de reproducció adaptatives.

Aquestes eines són accessibles a tots els desenvolupadors d'Apple que no estiguin en el programa gratuït i es poden trobar a l'apartat de descàrregues del portal per desenvolupadors *connect.apple.com*.

4.5.1.1 Eines d'Apple per l'HLS

Per a l'aplicació que es vol desenvolupar tan sols es necessiten el *segmenter* de fitxer i el generador de llistes de reproducció adaptatives. Tot i això, aquestes són les eines que ofereix Apple:

4.5.1.1.1 Media Stream Segmenter

El `mediastreamsegmenter` és una comanda que agafa un *stream* codificat en MPEG-2 Transport Stream com a entrada i produeix una sèrie de fitxers d'igual duració a partir d'aquest, adequats per a fer-los servir en HTTP Live Streaming. Aquesta comanda també és capaç de generar el fitxer d'índex (o *playlist*), xifrar les dades, produir claus de xifrat, optimitzar els fitxers reduint la sobrecàrrega i crear els fitxers necessaris per generar automàticament múltiples alternatives de *stream* (fitxers d'índex adaptatius). Per exemple:

```
mediastreamsegmenter –s 3 –D –f /Library/WebServer/Documents/stream 239.4.1.5:20103
```

Aquest exemple captura un *stream* en viu de la xarxa des de la direcció `239.4.1.5:20103` i crea els diferents segments i els fitxers d'índex. Els fitxers d'índex contenen una llista dels tres segments actuals (`-s 3`). Aquests segments són esborrats després de fer-los servir (`-D`). I, per últim, els segments generats i els fitxers d'índex són desats en el directori `/Library/WebServer/Documents/stream`.

4.5.1.1.2 Media File Segmenter

La comanda `mediafilesegmenter` agafa un fitxer de vídeo com a entrada, l'encapsula en una trama MPEG-2 TS i genera segments de la mateixa durada per al HTTP Live Streaming. La comanda també genera els fitxers d'índex i les claus de xifrat (si s'escau). El comportament d'aquesta ordre és molt semblant a l'anterior, però treballa amb fitxers de disc en comptes de *streams*.

4.5.1.1.3 Media Stream Validator

En aquest cas, la comanda `mediastreamvalidator` examina els fitxers d'índex, els fitxers adaptatius i els segments en un servidor, i realitza unes proves per veure si realment estan correctament definits i creats per fer-los servir amb el protocol HLS.

4.5.1.1.4 Variant Playlist Creator

Per a crear el fitxer d'índex "master", on estan definides les diferents qualitats del vídeo i els seus fitxers d'índex, Apple ha creat la comanda `variantplaylistcreator`. Per a poder fer servir aquesta comanda, el *segmenter* s'ha d'invocar amb l'argument `–generate–variant–playlist`; que fa que a part dels segments i del fitxer d'índex, el *segmenter* creï un altre fitxer amb la informació necessària per al *Variant Playlist Creator*.

4.5.1.1.5 Metadata Tag Generator

La comanda `id3taggenerator` genera les etiquetes id3. Aquestes etiquetes poden ser tant escrites en un fitxer com inserides en els diferents segments.

És pot trobar molta més informació sobre la utilització d'aquestes comandes a l'enllaç 2 de l'annex A1 o executant la comanda `man + [nom de la comanda]` en el terminal de Mac OS X.

4.5.1.2 Creació dels segments i dels fitxers d'índex.

En aquest punt es veurà el procediment dut a terme, pas per pas, amb la creació de totes les dades necessàries per preparar un vídeo destinat a ser reproduït per la nostra aplicació mitjançant HTTP Live Streaming.

El primer que tenim és el vídeo comprimit en H.264 i l'àudio en AAC. I tenim diverses qualitats d'aquest. En el cas que veurem tenim tres qualitats: una baixa (*low*), una mitjana (*medium*) i una alta (*high*).

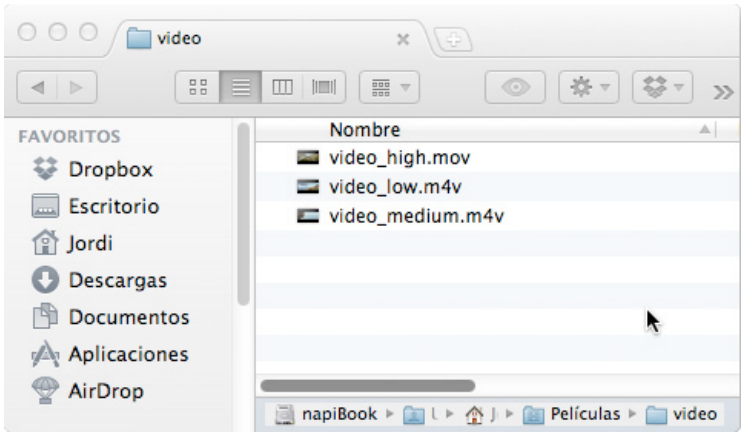

Figura 4.1 - **Fitxers de vídeo codificats en H.264 en diferents qualitats**

En el terminal:

```
macbook:video Jordi$ ls
video_high.mov          video_low.m4v           video_medium.m4v
```

Per crear els segments de la qualitat baixa i el seu fitxer d'índex pertinent fem servir el *Media File Segmenter*:

```
macbook:video Jordi$ mediafilesegmenter -b http://localhost/HLS/video/low -f /Users/Jordi/Sites/HLS/video/low video_low.m4v -I -iframe-index-file none
```

Amb aquesta comanda es segmenta el fitxer `video_low.m4v` en tots els segments necessaris (per defecte, de 10 segons de duració) que es desen al directori `/Users/Jordi/Sites/HLS/video/low`

(ha d'existir abans d'executar la comanda). També és genera el fitxer d'índex en el mateix directori que, en no indicar-se el nom de fitxer serà per defecte *prog_index.m3u8*. S'indica també la ruta base per a cadascun dels segments amb l'argument `-b http://localhost/HLS/video/low`; els diferents enllaços als segments contenen aquesta ruta acabada amb el nom del segment (van des de *fileSequence0.ts a fileSequence22.ts*). Hi ha dos arguments més: el primer, `-I`, que és la versió curta de `-generate-variant-playlist`, crea un fitxer al directori del vídeo origen anomenat *video_low.plist*. El fitxer *.plist* conté la informació necessària perquè, un cop segmentades totes les qualitats, es pugui crear el fitxer d'índex adaptatiu. El segon argument, `-iframe-index-file none`, serveix perquè el *segmenter* no creï un fitxer d'índex dels I-Frames [Cap. 2.5.3].

El fitxer d'índex generat és el següent:

Fitxer prog_index.m3u8 - **Fitxer d'índex dels segments de video_low.m4v**

```
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:3
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-PLAYLIST-TYPE:VOD
#EXTINF:10,
http://localhost/HLS/video/low/fileSequence0.ts
#EXTINF:10,
http://localhost/HLS/video/low/fileSequence1.ts
#EXTINF:10,
http://localhost/HLS/video/low/fileSequence2.ts
...
#EXTINF:10,
http://localhost/HLS/video/low/fileSequence20.ts
#EXTINF:10,
http://localhost/HLS/video/low/fileSequence21.ts
#EXTINF:8.48,
http://localhost/HLS/video/low/fileSequence22.ts
#EXT-X-ENDLIST
```

Com es pot veure, inclou totes les característiques descrites als capítols anteriors i conté 22 segments d'una duració màxima de 10 segons. Per defecte es genera un fitxer d'índex

compatible amb la versió 3 del protocol. Hi ha arguments en la comanda executada que fan que la duració de cadascun dels segments sigui un número sencer perquè sigui compatible amb la primera versió del protocol, però no cal aplicar aquesta restricció per al nostre client.

El contingut del fitxer d'informació .plist per a generar les llistes adaptatives és el següent:

Fitxer video_low.plist - **Informació per a generar llistes adaptatives**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
	<key>audioCodecInfo</key>
	<string>mp4a.40.5</string>
	<key>avgBitRate</key>
	<real>265128.71148459386</real>
	<key>maxBitRate</key>
	<real>289626.41509433964</real>
	<key>videoCodecInfo</key>
	<string>avc1.4d401e</string>
	<key>videoDimensions</key>
	<dict>
		<key>Height</key>
		<real>225</real>
		<key>Width</key>
		<real>400</real>
	</dict>
</dict>
</plist>
```

La informació que el *segmenter* desa en aquest fitxer és la que es necessitarà per a crear els diferents atributs de l'etiqueta `#EXT-X-STREAM-INF`; l'ample de banda, els *codecs* d'àudio i vídeo, les dimensions, etc. És un fitxer amb extensió *.plist*, que és una extensió del format XML per a la definició de llistes i diccionaris.

Aquest procés s'ha de realitzar per a cadascuna de les qualitats del vídeo, i al final d'obté el següent model d'arxius a la carpeta del servidor web:

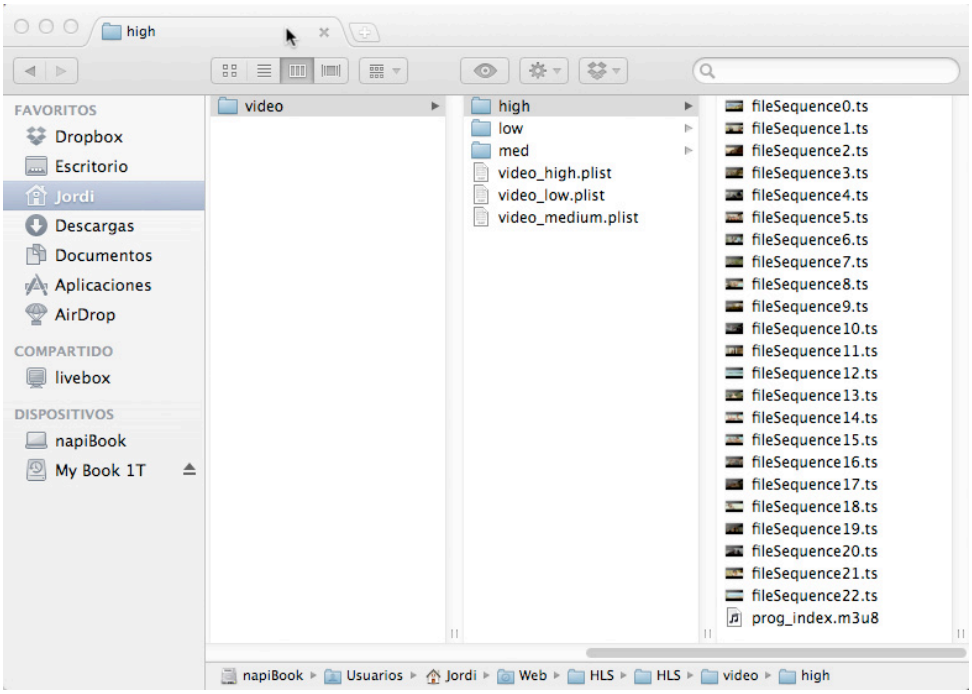

Fig. 4.2 - **Fitxers segmentats i llestos per generar una llista adaptativa**

Els fitxers .plist es desen a la carpeta on es troba el vídeo d'origen; aquí s'han mogut a la carpeta mare "video".

El següent pas és generar el fitxer d'índex "master" que contindrà les diferents qualitats. Per fer-ho s'utilitza l'eina anomenada *Variant Playlist Creator*.

```
macbook:video Jordi$ variantplaylistcreator -o all.m3u8 http://localhost/HLS/video/low/prog_index.m3u8 video_low.plist http://localhost/HLS/video/med/prog_index.m3u8 video_medium.plist http://localhost/HLS/video/high/prog_index.m3u8 video_high.plist
```

Amb aquesta comanda es genera el fitxer d'índex variable. El que indiquen els diferents arguments utilitzats és: el nom i localització del fitxer (`-o all.m3u8`), les parelles de valors on s'indica la URL, el fitxer d'índex d'una qualitat i la informació d'aquest en el fitxer .plist pertinent.

Al final s'obté un fitxer d'índex "master" anomenat all.m3u8 amb aquest contingut:

Fitxer all.m3u8 - **Fitxer d'índex "master" que defineix les diferents alternatives**

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=265129,CODECS="mp4a.40.5, avc1.4d401e"
http://localhost/HLS/video/low/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=2781119,CODECS="mp4a.40.2, avc1.4d401e"
http://localhost/HLS/video/med/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=7258815,CODECS="mp4a.40.2, avc1.64001f"
http://localhost/HLS/video/high/prog_index.m3u8
```

El vídeo ja està llest per posar-ho a disposició mitjançant el protocol HTTP Live Streaming. Aquest mateix procediment s'ha d'aplicar a tots els vídeos i les seves qualitats que s'hagin de servir mitjançant el protocol HLS.

4.5.1.3 Llista de vídeos accessibles

El que es necessita ara és una forma de donar accés als usuaris als diferents vídeos. Per a fer-ho hi ha diverses solucions. Es podria realitzar una pàgina web on els diferents vídeos es podrien visualitzar mitjançant HTML 5 i la nova etiqueta "video". Una altra solució seria fer servir un servei web, on els clients poguessin accedir i que els hi donés la informació de cadascun dels vídeos i els enllaços a aquests. Com a últim exemple, si aquests vídeos formessin part d'un sistema més complex, es podria integrar la informació dels enllaços als fitxers d'índex i de les dades dels vídeos dins d'una base de dades i que aquesta informació formés part d'un sistema més estructurat.

Per a la nostre aplicació, i per simplificar el procés d'accés, es fa servir un sistema més senzill. Al servidor es desarà un fitxer XML amb la llista dels diferents vídeos i la informació d'aquests. El fitxer XML conté la següent informació:

Fitxer movies.xml - **Llista dels enllaços als diferents vídeos i la seva informació**

```
<?xml version="1.0" encoding="UTF-8"?>
<movies>
    <movie id="1">
        <title>Justice - D.A.N.C.E.</title>
        <m3u8>http://localhost/HLS/JusticeDance/justiceDance.m3u8</m3u8>
```

```
        <thumbnail>http://localhost/HLS/JusticeDance/justiceDANCE.png</thumbnail>
    </movie>
    <movie id="2">
        <title>Anunci Estrella 2012</title>
        <m3u8>http://localhost/HLS/Estrella2012/estrella2012.m3u8</m3u8>
        <thumbnail>http://localhost/HLS/Estrella2012/estrella2012.jpg</thumbnail>
    </movie>
    <movie id="3">
        <title>For The Birds</title>
        <m3u8>http://localhost/HLS/ForTheBirds/forTheBirds.m3u8</m3u8>
        <thumbnail>http://localhost/HLS/ForTheBirds/forTheBirds.jpg</thumbnail>
    </movie>
    <movie id="4">
        <title>Partly Cloudy</title>
        <m3u8>http://localhost/HLS/PartlyCloudy/partlyCloudy.m3u8</m3u8>
        <thumbnail>http://localhost/HLS/PartlyCloudy/partlyCloudy.jpg</thumbnail>
    </movie>
    <movie id="5">
        <title>Summits Of My Life</title>
        <m3u8>http://localhost/HLS/SummitsOfMyLife/summitsOfMyLife</m3u8>
        <thumbnail>http://localhost/HLS/SummitsOfMyLife/summitsOfMyLife.jpg</thumbnail>
    </movie>
    <movie id="6">
        <title>Apple Basic Stream</title>
                <m3u8>https://devimages.apple.com.edgekey.net/resources/http-streaming/
examples/bipbop_4x3/bipbop_4x3_variant.m3u8</m3u8>
    </movie>
    <movie id="7">
        <title>Apple Advanced Stream</title>
                <m3u8>https://devimages.apple.com.edgekey.net/resources/http-streaming/
examples/bipbop_16x9/bipbop_16x9_variant.m3u8</m3u8>
    </movie>
</movies>
```

Com es pot veure, es tracta d'una llista de vídeos amb diferents elements per cadascun d'ells. A part de l'enllaç al fitxer d'índex "master" m3u8 també hi ha l'enllaç a un fitxer d'imatge, per a tenir una captura que defineixi el vídeo i un títol per al vídeo. A part, també es guarda un identificador. En aquest cas, en estar treballant en local, totes els enllaços es refereixen a *localhost*.

Podem notar que els últims dos vídeos del fitxer XML estan en servidors remots. Aquests dos són els dos vídeos que dóna Apple per a provar el protocol i que també s'han afegit a la llista.

Aquest és el fitxer principal que el client es descarregarà i amb el qual tindrà accés a la biblioteca de vídeos del nostre servidor de HTTP Live Streaming. L'enllaç a aquest fitxer és el que anomenarem "origen de dades".

4.5.1.4 Millora dels fitxers generats per a múltiples servidors

En l'entorn de treball del què s'ha disposat en aquest projecte es necessitaven diferents orígens de dades per poder provar l'aplicació en local, en un servidor remot i sobre xarxes mòbils. Els continguts generats són els mateixos en els tres entorns o servidors web, però varia la ruta base d'aquests en cadascun dels casos.

Una solució seria copiar tots els fitxers en els diferents servidors i editar, en cadascun dels fitxers d'índex, la ruta base dels enllaços. Però seguint l'especificació del protocol, no és necessari que els enllaços tinguin la ruta sencera als fitxers, amb una ruta relativa n'hi ha prou. La ruta sencera es construeix a partir de la ruta del fitxer d'índex i de les rutes relatives dels enllaços del seu interior.

Si editem els fitxers d'índex "master" de tal manera que quedin d'aquesta manera:

Fitxer all.m3u8 - **Fitxer d'índex variables amb rutes relatives**

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=265129,CODECS="mp4a.40.5, avc1.4d401e"
low/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=2781119,CODECS="mp4a.40.2, avc1.4d401e"
med/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=7258815,CODECS="mp4a.40.2, avc1.64001f"
high/prog_index.m3u8
```

i els fitxers d'índex de cadascuna de les diferents qualitats d'un recurs d'aquesta:

Fitxer prog_index.m3u8 - **Fitxer d'índex dels segments amb rutes relatives**

```
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:3
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-PLAYLIST-TYPE:VOD
#EXTINF:10,
fileSequence0.ts
#EXTINF:10,
fileSequence1.ts
#EXTINF:10,
fileSequence2.ts
...
#EXTINF:10,
fileSequence20.ts
#EXTINF:10,
fileSequence21.ts
#EXTINF:8.48,
fileSequence22.ts
#EXT-X-ENDLIST
```

tan sols caldrà editar el fitxer XML amb les rutes correctes per a cada servidor, amb l'única tasca de copiar tots els fitxers sense haver d'editar-los.

Per poder servir correctament els vídeos en els servidors web, aquests s'han de configurar indicant el MIME TYPE dels fitxers .m3u8 i .ts tal i com està explicat en el Cap. 2.2.2.

4.6 El client

En la implementació del client entren en joc les especificacions esmentades en el Punt 4.4. S'ha desenvolupat un client per el dispositiu iPhone d'Apple. Aquest es descarrega el fitxer XML que hi ha al servidor, analitza cadascun dels fitxers d'índex de les pel·lícules i mostra l'usuari la llista d'aquestes. Llavors, l'usuari pot veure cada pel·lícula per separat on es mostra com estan formats els fitxers d'índex i quines opcions pot realitzar. Si el format es vàlid, l'usuari pot reproduir la pel·lícula de forma didàctica amb un reproductor propi desenvolupat amb aquesta

finalitat. Un cop reproduïda, de forma parcial o en la seva totalitat, l'usuari pot veure les traces generades per l'última reproducció.

La figura 5.3 mostra tot aquest comportament de l'aplicació i les accions que l'usuari pot dur a terme.

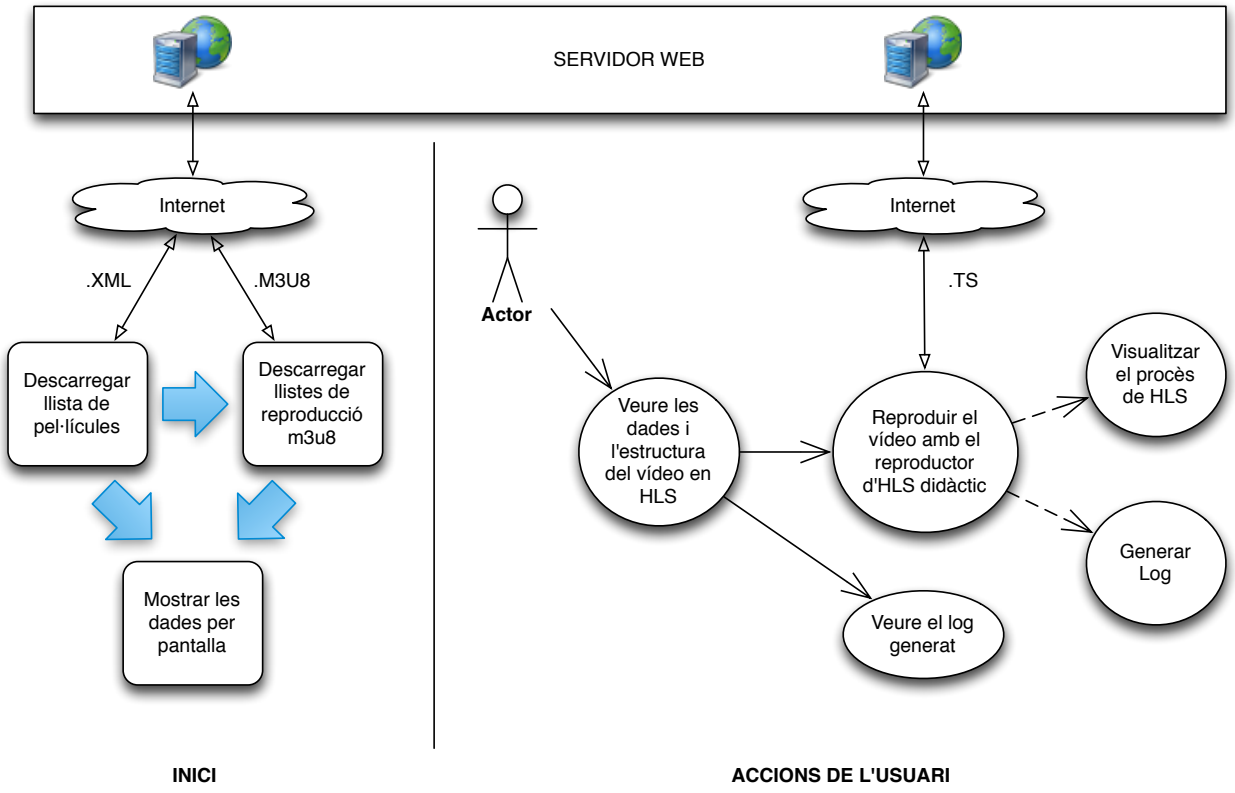

Fig. 4.3 - **Esquema de funcionament del client d'HLS desenvolupat**

En la primera part, que defineix l'inici de l'aplicació, es realitzen una sèrie de processos els quals han estat desenvolupats per a executar-se d'una manera asíncrona. La descàrrega dels diferents fitxers d'índex i l'anàlisi de cadascun d'aquests es realitzen asíncronament. Fins i tot la descàrrega de les imatges (*thumbnails*) de cada pel·lícula és asíncrona. Un dels objectius d'aquesta primera part ha estat conèixer i provar diferents mètodes de programació asíncrona amb el framework de Cocoa, aconseguint així una experiència d'usuari bona sense bloquejar innecessàriament l'aplicació.

La segona part, adreçada a les accions que l'usuari pot realitzar s'han fet servir complements de tercers per a poder realitzar les tasques requerides: un framework per a la creació de gràfiques (punt 4.6.4.3) i un servidor HTTP executat dins del client (punt 4.7.2).

En el següents punts es veuen en més detall aquestes solucions i el perquè del seu ús.

4.6.1 Configuració de l'aplicació.

Un dels requisits de l'aplicació és que l'usuari ha de poder escollir entre diferents orígens de dades i afegir-ne un de propi. Una de les possibilitats que dóna el framework d'Apple és la definició de paràmetres de configuració per a les aplicacions. Aquests paràmetres són accessibles des de l'aplicació però s'editen des de les pantalles de configuració del sistema operatiu. En la Figura 4.4 es pot veure com un cop definits els paràmetres de configuració d'una aplicació, aquesta apareix en la llista d'aplicacions que poden ser configurades des de l'apartat d'ajustos del sistema.

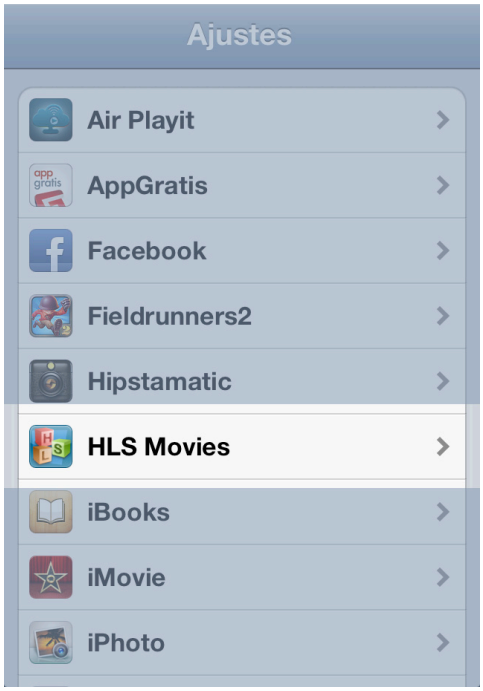

Fig. 4.4 - **Icona de configuració de l'aplicació**

Per a fer-ho en XCode tan sols hem d'editar un arxiu .plist on definim cadascuna de les opcions. Cada opció pot tenir fills (depèn del tipus d'opció), pot ser un valor de vertader o fals, un valor numèric, text, etc. Aquesta varietat d'opcions ens permet crear la configuració dels paràmetres de l'aplicació i la visualització d'aquests està totalment integrada en el sistema.

Si donem un cop d'ull a la interfície per a la creació dels paràmetres de configuració (Figura 4.5) podem veure com es defineixen aquests paràmetres: en aquest cas es tracta de la definició de les opcions de la nostra aplicació en referència a l'origen de dades.

Key	Type	Value
▼iPhone Settings Schema	Dictionary	(1 item)
▼Preference Items	Array	(7 items)
▶Item 0 (Group – Movies Server)	Dictionary	(2 items)
▼Item 1 (Multi Value – Movies'	Dictionary	(6 items)
Type	String	Multi Value
Title	String	Movies' Feed
Identifier	String	feeds
Default Value	String	http://napibook.home/~Jordi/hls/movies.xml
▼Titles	Array	(4 items)
Item 0	String	localhost (napiBook)
Item 1	String	napibook.home
Item 2	String	casal
Item 3	String	lewis.upc.es
▼Values	Array	(4 items)
Item 0	String	http://localhost/~Jordi/hls/movies_loc.xml
Item 1	String	http://napibook.home/~Jordi/hls/movies.xml
Item 2	String	http://casal.upc.edu/~napi25/moviesCasal.xml
Item 3	String	http://lewis.upc.es/HLS/moviesLewis.xml
▼Item 2 (Group – Alternative	Dictionary	(2 items)
Type	String	Group
Title	String	Alternative Feed
▼Item 3 (Toggle Switch – Enable	Dictionary	(4 items)
Type	String	Toggle Switch
Title	String	Enable Alt. Feed
Identifier	String	isUseAlternativeFeed
Default Value	Boolean	NO
▼Item 4 (Text Field – Alt. Feed)	Dictionary	(4 items)
Type	String	Text Field
Title	String	Alt. Feed
Identifier	String	altFeed
Default Value	String	

Fig. 4.5 - **Creació dels paràmetres de configuració**

Podem veure com per sota del punt 'mare' anomenat *Preference Items* trobem una sèrie de punts. Aquests són els paràmetres de primer nivell que trobarem en la configuració i que podem veure en la Fig. 4.6. El primer punt és tan sols un títol, una capçalera de grup, amb el text

"Movies Server". El segon defineix un element que es format per diversos valors a seleccionar, una mena de desplegable, amb el text "Movie's Feed" per títol. Després es defineixen els diferents elements disponibles: primer es defineix una llista amb els títols que es mostren a l'usuari i després una llista amb els valors que tenen aquests. Aquests són els diferents orígens de dades que l'usuari pot seleccionar. Com veiem, tots ells apunten a l'arxiu .xml que se suposa que conté la llista de vídeos en el format esperat. Amb l'identificador definit en el camp *Identifier,* el programador por accedir al valor del paràmetre seleccionat des de l'aplicació.

El següent element torna a ser una capçalera de grup amb el títol "Alternative Feed", aquest cop, però, té dos elements; el primer és un element de valor vertader o fals i el següent un camp de tipus text on l'usuari pot escriure el que vulgui. Amb aquests dos paràmetres l'aplicació compleix un altre requeriment: l'usuari pot introduir un origen de dades propi i activar-lo o no amb el primer paràmetre de vertader o fals. En la figura 4.6 es pot veure com el sistema operatiu de l'iPhone mostra aquestes opcions.

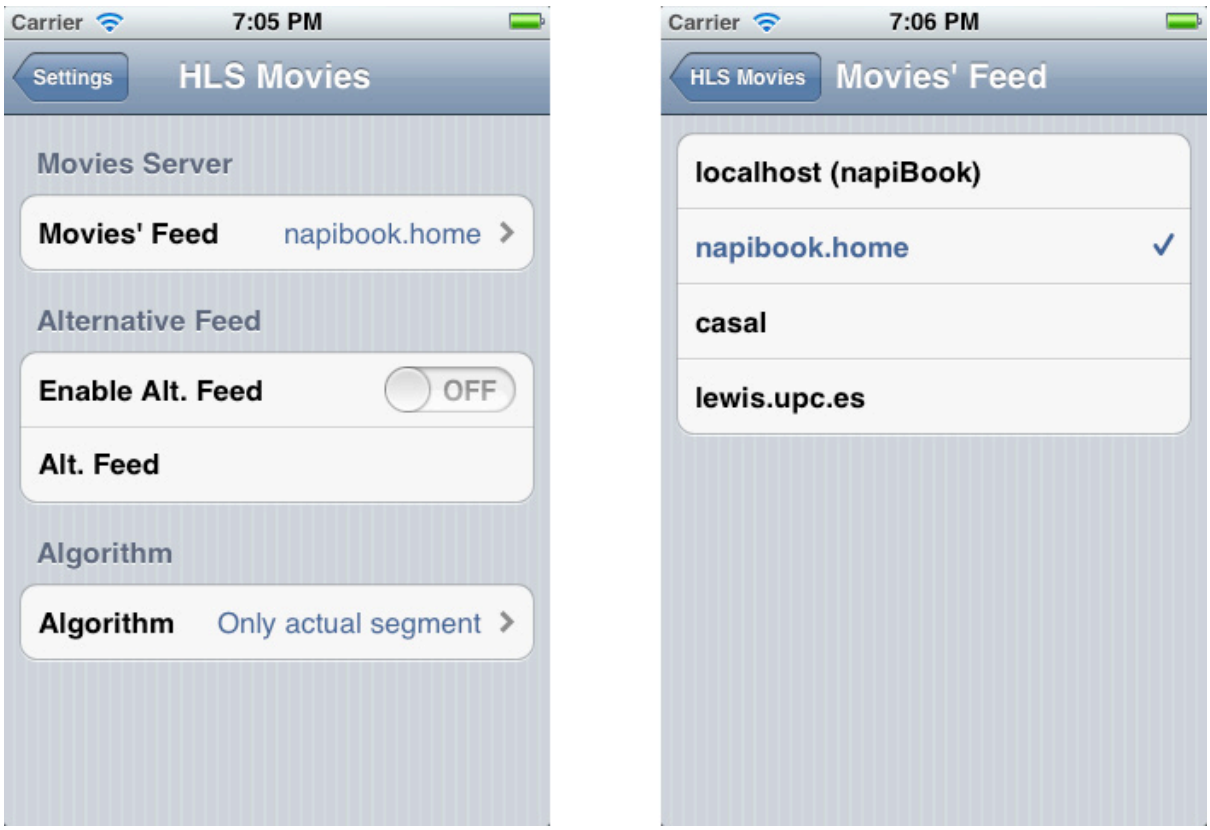

Fig 4.6 - **Visualització de les opcions en el iOS**

A l'esquerra es veuen totes les opcions que tenim i com les ha mostrat el sistema operatiu. Si l'usuari selecciona la primera opció *Movie's Feed*, a la pantalla de l'iPhone apareix la imatge de la dreta amb les diferents opcions a seleccionar.

Per accedir a aquesta configuració des del codi de l'aplicació tan sols cal escriure les següents línies:

Codi 4.1 - **Adquirint l'enllaç a l'origen de dades dels paràmetres de configuració**

```
NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
currentStrUrl = [defaults objectForKey:kSettingMoviesFeedKey];
```

En la variable `currentStrUrl` tenim l'enllaç seleccionat en els paràmetres de configuració. La variable `kSettingMoviesFeedKey` conté el text "feeds" que és l'identificador definit per a la variable que conté l'origen de dades (es pot veure a la figura 4.5).

4.6.2 Inici de l'aplicació

En el moment en que l'aplicació s'inicia, aquesta es connecta a l'origen de dades seleccionat i es descarrega el fitxer .XML amb la llista de pel·lícules. Aquesta llista és un fitxer en format XML que conté els enllaços als fitxers d'índex de cadascun dels vídeos, els seus títols i les imatges respectives (si en tenen).

Per a poder traduir aquesta llista de pel·lícules a entitats que puguin ser gestionades per l'aplicació s'ha definit un conjunt d'elements que representen els diferents vídeos i que contenen tota la informació d'aquests. La informació que contenen aquests elements va des de l'enllaç al fitxer d'índex fins a tota la informació d'aquest: segments, duració, versió, codificació, etc. Aquesta llibreria de classes conté totes les dades necessàries per a mostrar-les a l'usuari i reproduir el vídeo seguint el protocol HTTP Live Streaming d'una manera didàctica, tal i com s'espera.

4.6.2.1 Model de dades

El model de dades que gestionarà l'aplicació es defineix a la figura 4.7.

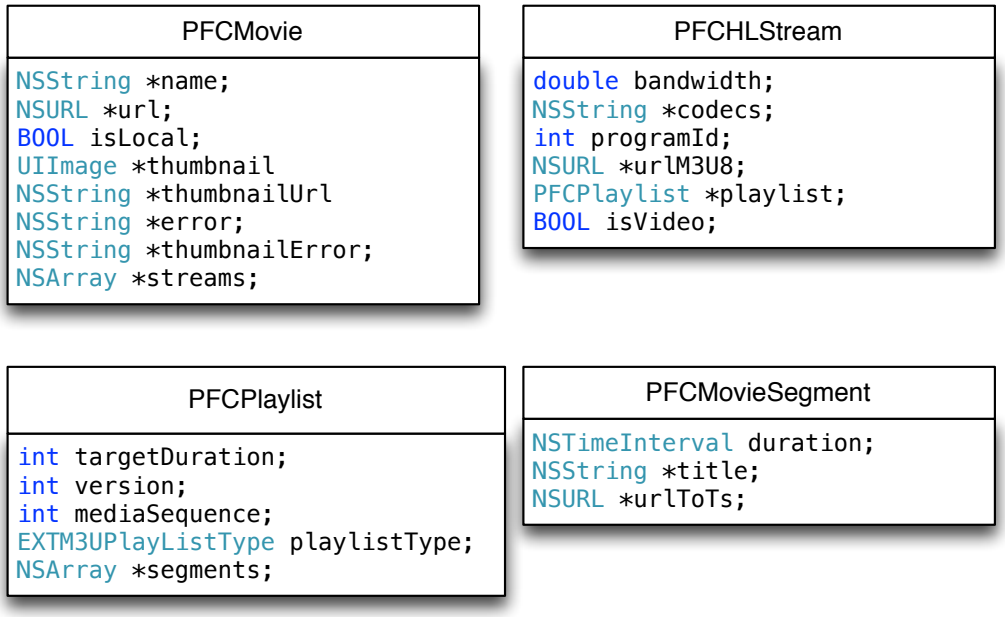

Fig. 4.7 - **Diagrama de classes per a la gestió de vídeos**

Amb aquest diagrama de classes es gestionen les dades de l'aplicació.

L'entitat **PFCMovie** és la classe principal; aquesta conté tota la informació disponible d'una pel·lícula i d'ella pengen les altres classes. Conté el nom o títol d'aquesta (*name*), l'enllaç al fitxer d'índex mare on es defineixen les diferents qualitats (*url*), la imatge de la pel·lícula (*thumbnail*), l'enllaç a aquesta imatge (*thumbnailUrl*), un text que indica si hi ha hagut cap error (*error*), un altre text que indica si hi ha hagut cap error carregant la imatge (*thumbnailError*) i, per últim, una llista d'entitats de tipus **PFCHLStream** (*streams*). Una entitat *PFCMovie* conté tants *PFCHLStreams* com qualitats hi ha definides en el fitxer d'índex.

Si tornem al capítol 2, la definició de les diferents qualitats en el fitxer d'índex es descriuen en la paraula clau EXT-X-STREAM-INF; i aquesta paraula clau té diferents paràmetres; aquests són les dades que gestiona l'entitat **PFCHLStream**. L'ample de banda del *stream* (*bandwidth*), la codificació del vídeo i l'àudio d'aquest (*codecs*), l'identificador del contingut (*programId*), l'enllaç al fitxer .m3u8 amb la definició dels segments (*urlM3U8*), una classe que contindrà les

dades de la llista de reproducció de l'enllaç anterior (*playlist*) i un indicador de si el contingut és un vídeo o només àudio (*isVideo*).

L'entitat **PFCPlaylist** conté les dades del fitxer d´índex d'una de les qualitats; la duració màxima d'un segment (*targetDuration*), la versió del protocol en la que s'ha creat el fitxer d'índex (*version*), el número de seqüència del primer segment (*mediaSequence*), el tipus de llista de reproducció, VOD o EVENT, (*playlistType*) i la llista de segments que conformen la pel·lícula (*segments*).

Per últim, l'entitat **PFCSegment**[7] conté les dades de cada segment: la duració (*duration*), el títol (*title*) i l'enllaç al fitxer .TS d'aquest (*urlToTs*).

4.6.2.2 L'obtenció de les dades

Un cop descarregat el fitxer XML, el llegeix un programa anomenat "parser" que transforma les dades del fitxer en una llista d'elements del tipus *PFCMovie*. Cada cop que es llegeix una pel·lícula es mostra per pantalla fins a generar la llista completa (Fig. 4.8). I, per poder generar una experiència d'usuari més rica, on els temps d'espera per la càrrega i l'obtenció de les dades no bloquegin l'aplicació, tots aquests processos es realitzen d'una manera asíncrona. Cada cop que una pel·lícula es crea i mostra comença un procés asíncron per obtenir la imatge i les dades d'aquesta.

[7] Totes les classes creades en aquest projecte comencen amb el prefix PFC (Projecte Fi de Carrera) per poder diferenciar-les de les classes pròpies del sistema o d'altres components. A més a més, si es re-utilitzen en algun altre projecte, queda identificada la seva procedència.

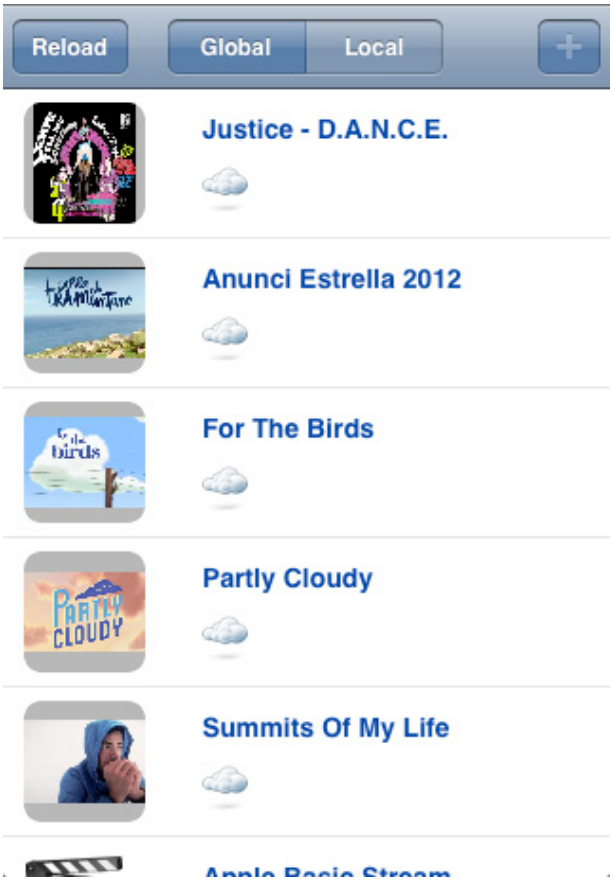

Fig. 4.8 - **Llista de pel·lícules disponibles**

En aquesta captura podem veure la llista de pel·lícules. Aquestes són les dades que conté el fitxer .xml descarregat:

Fitxer movies.xml - **Fitxer XML amb la llista de pel·lícules**

```
<?xml version="1.0" encoding="UTF-8"?>
<movies>
    <movie id="1">
        <title>Justice - D.A.N.C.E.</title>
        <m3u8>http://napibook.home/~Jordi/hls/JusticeDance/justiceDance.m3u8</m3u8>
            <thumbnail>http://napibook.home/~Jordi/hls/JusticeDance/justiceDANCE.png</
thumbnail>
    </movie>
    <movie id="2">
        <title>Anunci Estrella 2012</title>
        <m3u8>http://napibook.home/~Jordi/hls/Estrella2012/estrella2012.m3u8</m3u8>
```

```
            <thumbnail>http://napibook.home/~Jordi/hls/Estrella2012/estrella2012.jpg</
thumbnail>
    </movie>
    <movie id="3">
        <title>For The Birds</title>
        <m3u8>http://napibook.home/~Jordi/hls/ForTheBirds/forTheBirds.m3u8</m3u8>
              <thumbnail>http://napibook.home/~Jordi/hls/ForTheBirds/forTheBirds.jpg</
thumbnail>
    </movie>
    <movie id="4">
        <title>Partly Cloudy</title>
        <m3u8>http://napibook.home/~Jordi/hls/PartlyCloudy/partlyCloudy.m3u8</m3u8>
            <thumbnail>http://napibook.home/~Jordi/hls/PartlyCloudy/partlyCloudy.jpg</
thumbnail>
    </movie>
    <movie id="5">
        <title>Summits Of My Life</title>
        <m3u8>http://napibook.home/~Jordi/hls/SummitsOfMyLife/summitsOfMyLife</m3u8>
                  <thumbnail>http://napibook.home/~Jordi/hls/SummitsOfMyLife/
summitsOfMyLife.jpg</thumbnail>
    </movie>
    <movie id="6">
        <title>Apple Basic Stream</title>
                <m3u8>https://devimages.apple.com.edgekey.net/resources/http-streaming/
examples/bipbop_4x3/bipbop_4x3_variant.m3u8</m3u8>
    </movie>
    <movie id="7">
        <title>Apple Advanced Stream</title>
                <m3u8>https://devimages.apple.com.edgekey.net/resources/http-streaming/
examples/bipbop_16x9/bipbop_16x9_variant.m3u8</m3u8>
    </movie>
</movies>
```

Tot aquest procés de descàrrega de fitxers (tant els fitxers .m3u8 com les imatges de les pel·lícules) i de generació de les dades es pot veure en el diagrama de seqüència de la Fig. 4.9.

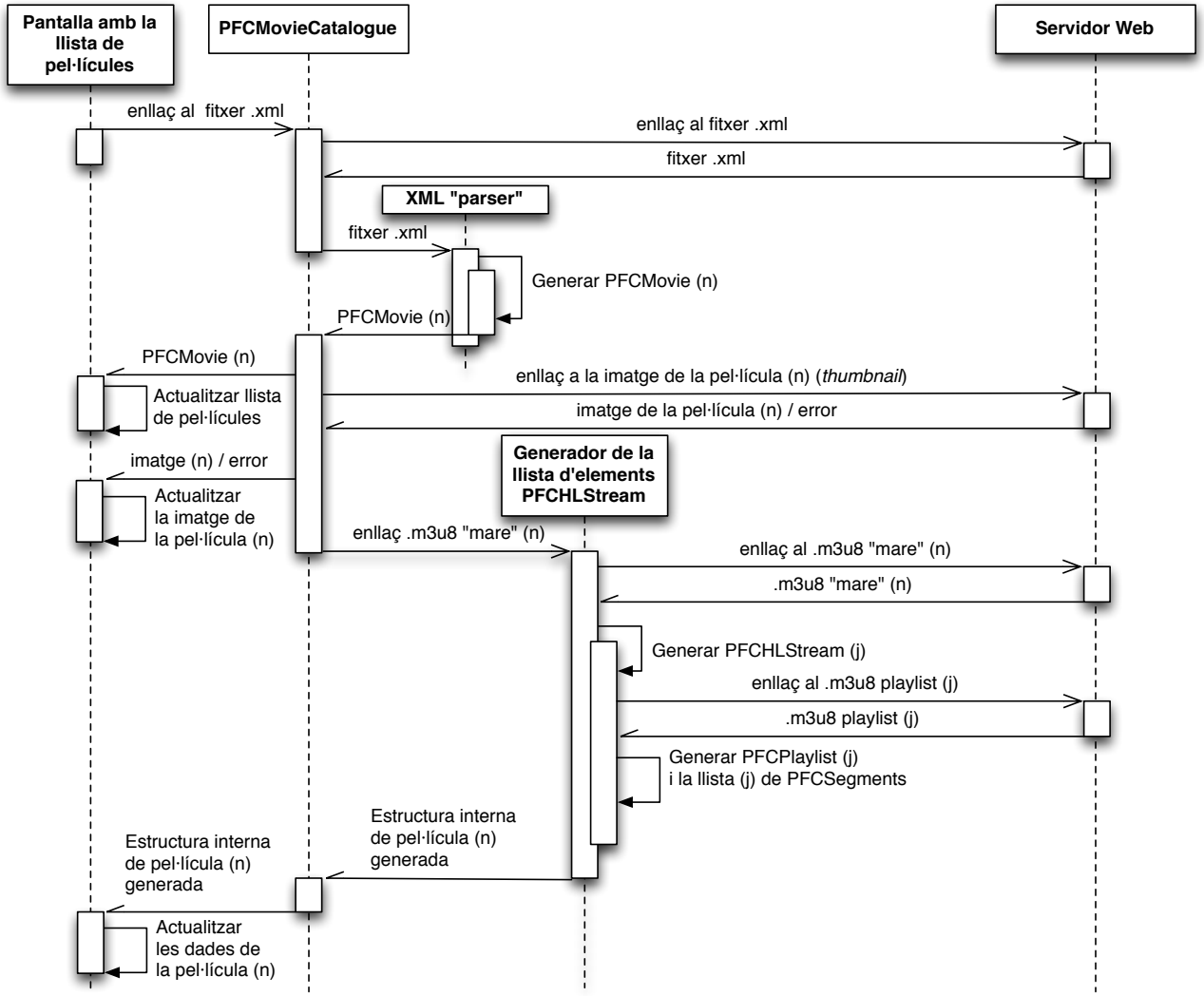

Fig 4.9 - **Procés asíncron de descàrrega i obtenció de totes les dades**

Quan es mostra una pel·lícula en la llista, un procés asíncron descarrega la imatge d'aquesta (si existeix) ,i un cop descarregada, actualitza la llista per la pantalla. Al mateix temps, sense esperar que la imatge es descarregui, es procedeix a descarregar l'arxiu .m3u8 de la pel·lícula i a generar l'estructura d'entitats (classes) amb les dades d'aquesta. Es descarreguen les llistes de reproducció internes per a cada qualitat i es creen les entitats *PFCHLStream*, *PFCPlaylist* i *PFCSegment* necessàries per a cadascun dels *streams*. Durant el temps que tarda a realitzar-se aquest procés, si l'usuari intenta veure les dades d'una pel·lícula l'aplicació l'informa que les dades s'estan carregant, però, un cop generades les dades del vídeo, l'usuari ja pot entrar a veure-les i descobrir com estan generats els diferents fitxers d'índex per aquest vídeo sense haver d'esperar a que es generin tots els altres vídeos (Fig. 4.10).

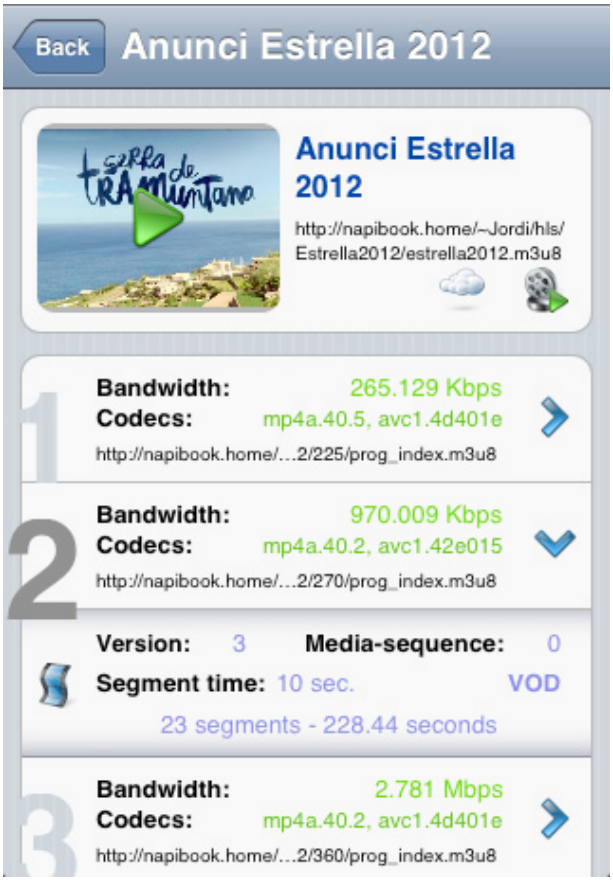

Fig. 4.10 - **Detall d'una pel·lícula**

Com podem veure en aquesta figura que mostra el detall d'una pel·lícula, l'estructura creada amb els fitxers d'índex del protocol HTTP Live Streaming queda definida gràficament. En aquest cas tenim (tot i que no s'acaba de veure en la imatge) 5 qualitats.

Si analitzem el fitxer d'índex mare d'aquest vídeo podem comprovar que així és, i veure com les dades mostrades per pantalla són les d'aquest fitxer.

Fitxer estrella2012.m3u8 - **Fitxer d'índex "mare"**

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=265129,CODECS="mp4a.40.5, avc1.4d401e"
225/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=970009,CODECS="mp4a.40.2, avc1.42e015"
270/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=2781119,CODECS="mp4a.40.2, avc1.4d401e"
360/prog_index.m3u8
```

```
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=4047837,CODECS="mp4a.40.2, avc1.4d401f"
540/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=7258815,CODECS="mp4a.40.2, avc1.64001f"
720/prog_index.m3u8
```

En la pantalla de detall d'un vídeo (Fig. 4.10) veiem una primera part amb les dades del fitxer .XML: el títol i la imatge. Després es mostren les dades del fitxer d'índex mare, tantes línies com qualitats (*streams*) tingui definides. Podem veure que es mostren dos dels valors definits en l'etiqueta EXT-X-STREAM-INF: l'ample de banda i la codificació del vídeo i l'àudio. També es mostra l'enllaç al fitxer d'índex de cada *stream*.

Si l'usuari selecciona una de les qualitats, l'aplicació mostra les dades del arxiu .m3u8 on estan definits el segments d'aquesta i d'altres paràmetres. És mostren valors importants com: la versió, el valor de l'etiqueta EXT-MEDIA-SEQUENCE, la duració màxima d'un segment, el tipus de llista (VOD o EVENT), el número de segments i la seva duració total real - calculada sumant tots els valors de les etiquetes EXT-INF dels segments. Podem veure que aquestes dades són les del fitxer d'índex:

Fitxer prog_index.m3u8 - **Fitxer d'índex del segon *stream* de la pel·lícula**

```
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:3
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-PLAYLIST-TYPE:VOD
#EXTINF:10,
fileSequence0.ts
#EXTINF:10,
fileSequence1.ts
...
#EXTINF:10,
fileSequence21.ts
#EXTINF:8.44,
fileSequence22.ts
#EXT-X-ENDLIST
```

En la pantalla de detall d'una pel·lícula l'usuari pot realitzar l'acció principal per la que està desenvolupada aquesta aplicació: reproduir la pel·lícula fent servir el reproductor HLS didàctic. Només cal que l'usuari pressioni sobre la imatge de la pel·lícula per obrir el reproductor propi i començar el procés definit pel protocol HLS.

4.6.2.3 Fitxers d'índex estàtics

Un dels requisits de l'aplicació és la reproducció de vídeo sota demanda. Amb això queda definit el tipus de llista de reproducció que podem reproduir: EXT-X-PLAYLIST-TYPE:VOD. Queden fora de l'abast de l'aplicació el *streaming* en viu, on la llista de reproducció pot ser actualitzada per servidor, la paraula clau EXT-X-MEDIA-SEQUENCE pot ser diferent de zero i el fitxer d'índex pot no acabar amb EXT-X-ENDLIST. Llavors aquestes dades sempre mostraran el mateix valor per a totes les qualitats i pel·lícules vàlides per l'aplicació. Els casos de reproducció en viu no funcionen en aquesta versió de l'aplicació i tampoc és capaç de reproduir només àudio.

Donat que està preparada tan sols per fitxers d'índex de vídeo sota demanda, amb la descàrrega dels diferents fitxers l'aplicació ja disposa de la informació amb tots els segments de cada qualitat i pot mostrar-la al usuari.

4.6.3 - Reproducció didàctica de l'HLS

L'altre part més important de l'aplicació és la reproducció de les pel·lícules mitjançant el protocol HTTP Live Streaming d'una manera didàctica per l'usuari. L'objectiu és aconseguir reproduir la pel·lícula utilitzant el protocol i mostrar tots els passos seqüencials de manera que l'usuari pugui visualitzar-los i entendre el funcionament del protocol i veure, sobretot, la qualitat adaptativa d'aquest.

Al enfocar-se tan sols en vídeo sota demanda, l'aplicació es centra en el caràcter adaptatiu del protocol, que és el que demanen els requeriments.

Com s'ha explicat al capítol 2, el procés de selecció dels segments de diferents qualitats no entra dins de la definició del protocol i, tot i que Apple dóna uns passos a seguir en la seva documentació, és el programador l'encarregat de decidir l'algoritme que s'ha de seguir.

Com que l'objectiu és que el reproductor d'HLS de l'aplicació mostri el funcionament del protocol, l'usuari ha de ser capaç de veure alhora, el vídeo, l'actual velocitat de la xarxa (velocitat de descàrrega del segment), el segment que s'està descarregant i el que s'està reproduint. Cada segment mostra de quina qualitat o *stream* és i quin número de segment és, amb aquesta informació l'usuari pot veure la relació amb la velocitat de descàrrega i el següent segment descarregat: la capacitat d'adaptació al medi del protocol.

A més a més, per a no haver de trobar un medi amb un ample de banda variable (haver de fer servir l'aplicació movent-se pel carrer i per zones de diferent cobertura no és gaire pràctic a efectes didàctics), l'aplicació facilita a l'usuari un mitjà per simular canvis en l'ample de banda de la xarxa (és un requeriment de l'aplicació).

4.6.3.1 Procés de selecció del següent segment a descarregar

Seguint els consells de la documentació d'Apple, el procés de selecció del següent segment a descarregar es basa només en la velocitat de descàrrega de l'últim segment. El primer segment a descarregar és el primer segment del *stream* de qualitat inferior. Havent descarregat aquest primer segment, l'aplicació té una mesura de l'ample de banda de la xarxa i el següent segment a descarregar es tria a partir d'aquesta mesura. Els diagrama del procés es pot veure a la figura 4.11.

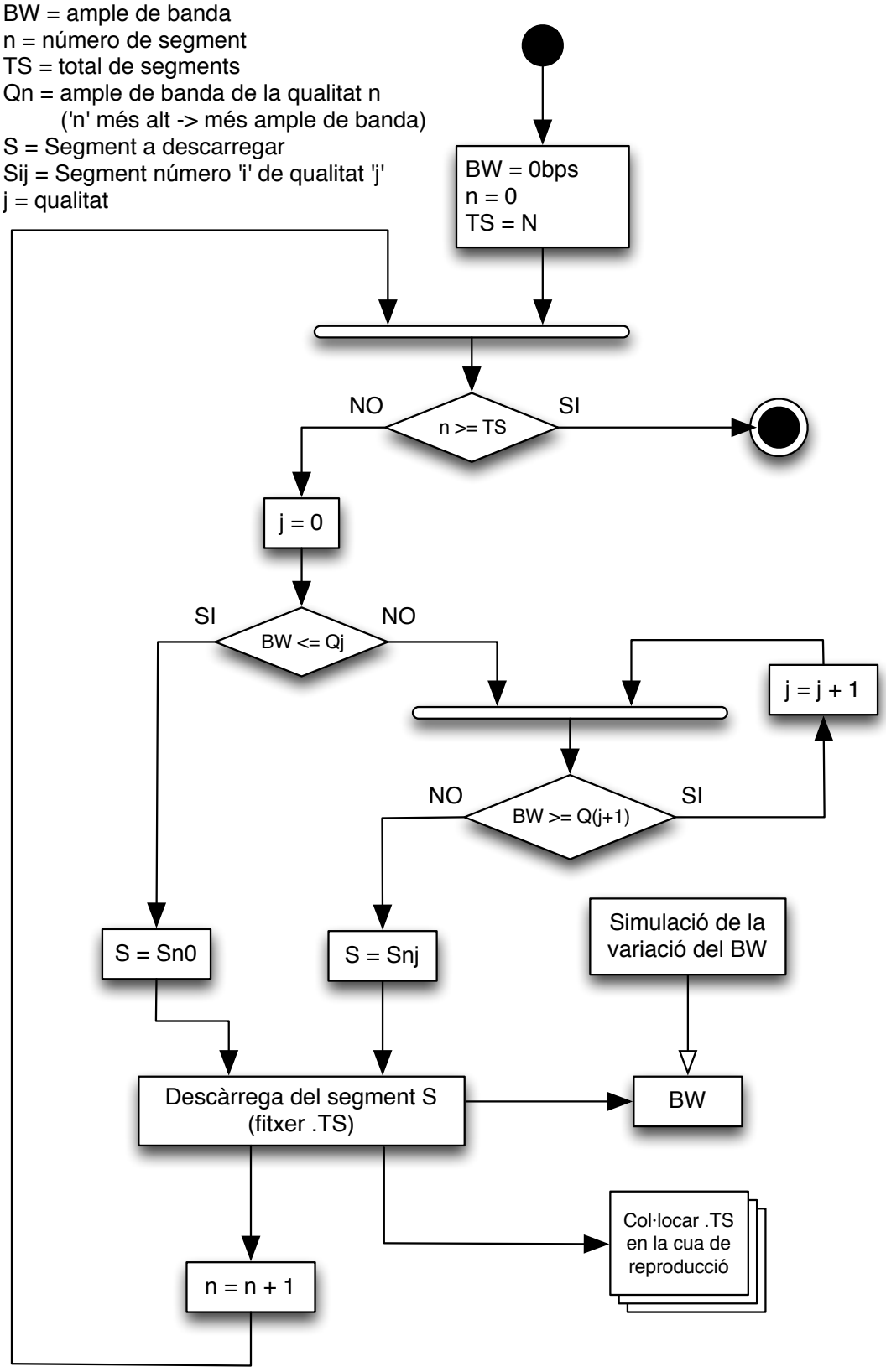

Fig. 4.11 - **Diagrama de descàrrega de segments**

El reproductor tan sols va reproduint els segments de vídeo descarregats i posats en la "cua de reproducció". Aquests són fitxers .TS i es reprodueixen en l'ordre que s'han posat en la cua. En el moment que no en queden més el reproductor es para fins a trobar-ne un de nou.

4.6.3.2 Traces de les operacions

A mesura que el procés de descàrrega de segments es va executant i que els segments descarregats es van reproduint, l'aplicació va guardant una traça amb la informació de les diverses operacions i decisions preses.

Un cop acabat el vídeo o quan l'usuari torna a la pantalla d'informació de la pel·lícula, la traça es guarda i l'usuari té l'opció de veure-la: un nou botó apareix a la part superior de la pantalla, al costat de la imatge i sota del títol (Fig. 4.12).

Fig. 4.12 - **Botó per veure la traça d'una reproducció**

Un cop l'usuari obre la pantalla es troba amb la traça generada tal i com es veu a la figura 4.13.

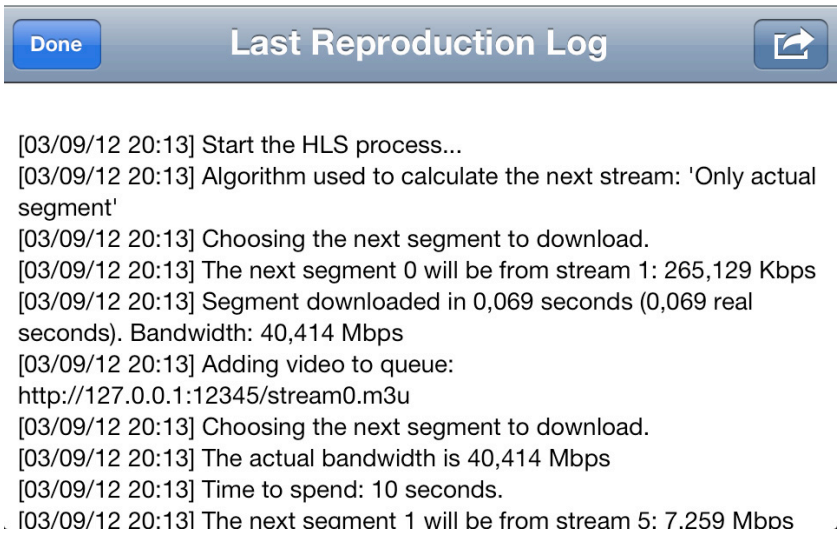

Fig. 4.13 - **Pantalla de visualització de la traça de reproducció**

En l'annex A3 es pot veure la traça sencera d'una reproducció. Aquesta traça intenta indicar clarament les accions dutes a terme; el temps que s'ha trigat per descarregar un segment, quin ample de banda ha resultat, el temps que té l'aplicació per descarregar el següent segment i, a partir d'aquests valors, decidir la qualitat (*stream*) del següent segment.

4.6.3.3 Disseny de la interfície visual del reproductor

Un dels requisits és que el reproductor d'HLS ha de ser didàctic; l'usuari ha de ser capaç de veure i entendre el procés de descàrrega adaptativa del protocol fent servir aquest reproductor. Per poder aconseguir aquest repte s'ha de dissenyar una interfície on és puguin veure tots els passos. S'ha de poder veure, a part de la reproducció dels segments: quin segment s'està reproduint i de quina qualitat és, el segment (i la seva qualitat) que s'està descarregant i a quina velocitat ho fa i, per últim, quins segments tenim ja descarregats i quins reproduïts.

La Figura 4.14 mostra la interfície per al seguiment dels segments: descàrrega, reproducció i qualitat.

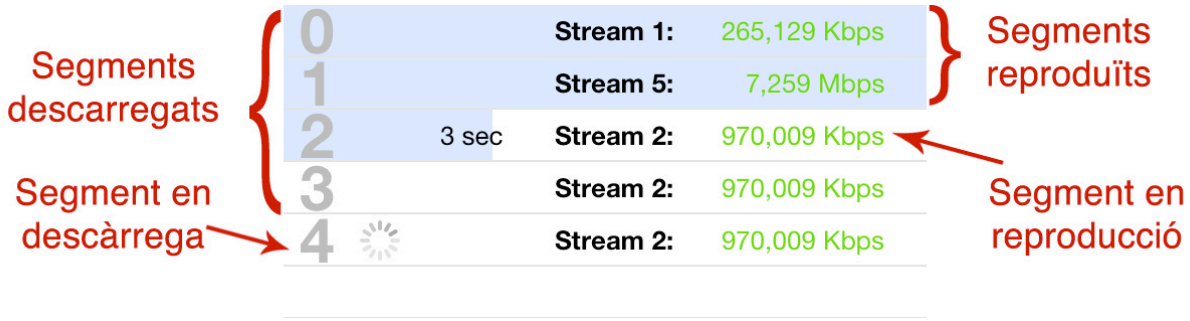

Fig. 4.14 - **Seguiment dels segments**

Cada línia és un segment. A l'esquerra es veu el número de segment i al mig la qualitat escollida i el seu ample de banda. Els segments en color blau són els que ja s'han reproduït, els blancs encara s'han de reproduir però ja s'han descarregat i l'últim, que té un petit cercle donant voltes, és el que s'està descarregant en aquell moment.

L'usuari també ha se ser capaç de veure la velocitat de descàrrega del fitxer que s'està adquirint, ja que la mitjana d'aquesta és la que decidirà la qualitat del següent segment. Al costat del controls de reproducció surt la velocitat de descàrrega (Fig. 4.15).

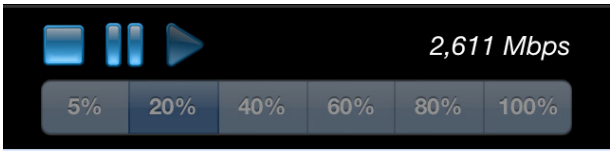

Fig. 4.15 - **La velocitat de descàrrega és veu en temps real**

Per últim falta la simulació de la variació de l'ample de banda de la xarxa. Si l'usuari es troba connectat a una xarxa mòbil i es va movent, molt possiblement la velocitat d'aquesta varii amb el temps i la cobertura faci que l'ample de banda disponible en cada moment sigui diferent. En aquest entorn el protocol HLS demostraria fàcilment el seu component adaptatiu, canviant contínuament la qualitat dels segments descarregats per a adaptar-se a les variacions de la xarxa. Si l'usuari es troba fixe en un mateix lloc o connectat a una xarxa Wifi, és difícil que l'ample de banda de la xarxa varii, per tant el protocol sempre descarregarà els segments de la mateixa qualitat i no es podria veure el caràcter adaptatiu del protocol (que és el que es pretén amb aquesta aplicació).

Per solucionar aquest problema s'ha afegit a la interfície de reproducció una barra amb un simulador d'ample de banda (Fig. 4.16). Aquest simulador té 6 opcions disponibles, que indiquen el percentatge de l'ample de banda actual que l'usuari vol que simuli l'aplicació.

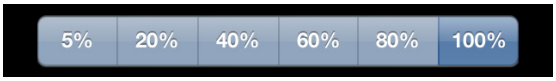

Fig. 4.16 - **Simulador d'ample de banda**

Si l'ample de banda real de la xarxa és de 1 Mbps i l'usuari selecciona l'opció de 20%, l'ample de banda simulat serà de 200 Kbps. L'aplicació, un cop descarregat el segment, aplicarà un temps d'espera proporcional al temps en que s'hauria descarregat el segment amb l'ample de banda simulat. Durant la simulació d'aquest temps, el control esmentat es desactiva i apareix l'ample de banda simulat en l'indicador de velocitat (Fig. 4.15).

Amb aquests tres controls podem generar una interfície per a la reproducció didàctica dels vídeos mitjançant el protocol HTTP Live Streaming on queda clar el procés que segueix el protocol i la capacitat adaptativa d'aquest.

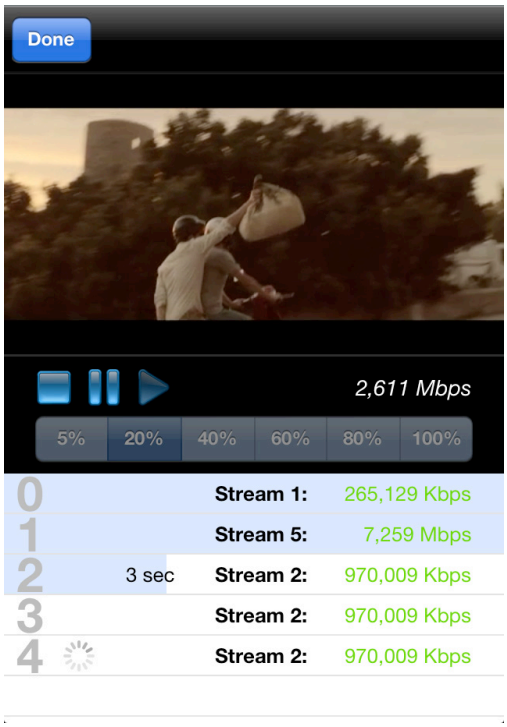

Fig. 4.17 - **Pantalla de reproducció didàctica amb tots els elements**

Tots els requisits de l'aplicació han estat implementats: es pot escollir l'origen de dades, es llegeixen i modelen les llistes de reproducció, és mostra la informació d'aquestes a l'usuari, es reprodueixen de forma didàctica els vídeos, es veu, clarament, el caràcter adaptatiu del protocol, es poden simular canvis en l'ample de banda de la xarxa i es genera una traça de les accions preses pel reproductor on es segueixen els passos del protocol HTTP Live Streaming. Tot i això, s'han desenvolupat algunes noves característiques en l'aplicació que queden fora dels requisits inicials. El següent punt es centra en elles.

4.6.4 Característiques afegides

En el moment de fer servir l'aplicació s'han anat trobant algunes mancances, algunes d'elles s'han convertit en noves opcions en l'aplicació. Hi ha cinc noves característiques implementades en l'aplicació que no estaven contemplades en els requisits inicials: l'opció d'afegir enllaços a fitxers d'índex (pel·lícules) que es desin en local, l'opció de fer servir el reproductor intern del *framework* de Cocoa, la capacitat de generar una gràfica on es vegi el caràcter adaptatiu del protocol després de la reproducció d'un vídeo, l'opció d'enviar un correu electrònic amb la traça capturada o amb la gràfica i, per últim, l'opció d'escollir entre dos o tres algoritmes alhora d'escollir el proper segment a descarregar.

4.6.4.1 Enllaços locals a fitxers d'índex

Com s'ha vist al començament d'aquest capítol, en els arxius XML d'origen de dades cada vídeo consta de tres elements: un títol, l'enllaç al fitxer d'índex d'aquest i una imatge que és opcional. L'aplicació dóna a l'usuari l'opció d'escollir entre un conjunt d'orígens de dades o d'afegir el seu propi origen de dades (enllaç al XML amb la llisa de vídeos), però no hi ha cap manera d'afegir un vídeo individualment (un enllaç a un .m3u8).

Aquesta nova característica supleix aquesta carència i dóna l'opció d'afegir nous vídeos que es guardaran de forma local en el dispositiu. Per a la correcta gestió dels vídeos, la pantalla principal amb la llista de vídeos es divideix en dos vistes; una amb els vídeos que venen de l'origen de dades escollit i l'altre amb els vídeos afegits localment. Per intercanviar entre aquestes dues vistes l'usuari ha de fer servir els botons que apareixen en la part superior de la pantalla amb les etiquetes "Global" i "Local", tal i com es veu en la figura 4.18. A més, un cop

seleccionem la llista de pel·lícules "Local", podem afegir-ne més amb el botó "+", o ordenarles i eliminar-les amb el botó "Edit".

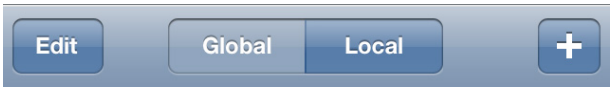

Fig. 4.18 - **Opcions de selecció de llista de pel·lícules i d'afegir o editar les "Locals"**

Quan s'afegeix una nova pel·lícula s'obre un formulari on l'usuari ha d'introduir el títol i l'enllaç al fitxer d'índex d'aquesta (Fig. 4.19).

Fig. 4.19 - **Formulari per afegir noves pel·lícules localment**

Un cop afegida la pel·lícula, aquesta apareix en la llista "Local" i, encara que es tanqui l'aplicació, es desa per a la propera vegada que s'executi l'aplicació. Aquestes pel·lícules es desen cada cop que es creen. Per desar-les es tradueixen les dades de la pel·lícula en una llista de parells de valors (propietat o atribut - valor d'aquest) en un fitxer local.

4.6.4.2 Reproducció amb el reproductor del framework Cocoa

La principal característica del reproductor didàctic és també una de les seves grans limitacions. Per la seva pròpia definició, aquest està totalment enfocat a visualitzar el procés de descàrrega i adaptatiu del protocol, però no és una bona opció per a veure la pel·lícula. Un dels errors que genera és que hi ha un gran salt del vídeo entre segment i segment. Per a aquesta primera versió del reproductor no és important, ja que aquest està enfocat en el protocol, no en la reproducció. L'altre petit inconvenient, és que el vídeo es veu massa petit.

Per aquestes raons s'ha decidit donar l'opció a l'usuari de reproduir el vídeo d'una manera més eficient i còmoda. El protocol HLS és suportat pels dispositius d'Apple des de la versió 3 de l'iOS, i el *framework* de Cocoa proporciona un reproductor que és capaç de reproduir vídeos en HLS des del seu fitxer d'índex, realitzant tot el procés del protocol de forma interna i òptima.

Des de la vista de detall d'una pel·lícula l'usuari troba un petit botó (Fig. 4.20) que obre el reproductor intern del sistema per a reproduir la pel·lícula.

Fig. 4.20 - **L'usuari pot reproduir el vídeo amb el reproductor intern de l'iOS.**

Aquest botó reprodueix el vídeo a pantalla completa fent servir el reproductor intern del framework (Fig. 4.21).

Fig. 4.21 - **Reproductor intern del *framework* a pantalla completa**

A més a més, també genera una traça de reproducció. Aquesta traça no és tan detallada com la del reproductor propi, però també és pot veure com va canviant la qualitat dels segments a mesura que avança la reproducció.

Codi 4.2 - **Traça generada pel reproductor intern del sistema**

```
URL: http://napibook.home/~Jordi/hls/Estrella2012/225/prog_index.m3u8
Bitrate: 277,337 Kbps (3,277 Mbps)
Segments downloaded: 1
Watched 5.018630 seconds from 10.000000 seconds downloaded.
--------------------------------
URL: http://napibook.home/~Jordi/hls/Estrella2012/270/prog_index.m3u8
Bitrate: 989,932 Kbps (12,009 Mbps)
Segments downloaded: 6
Watched 12.243261 seconds from 60.000000 seconds downloaded.
```

Al seguir la traça generada pel reproductor del *framework* es veu que durant els 5 primers segons s'ha reproduït part del primer segment de la qualitat més baixa (de 277,337 Kbps), la velocitat de la xarxa ha estat de 3,277 Mbps (Wifi), i que tan sols s'havia descarregat un segment. En la segona part de la traça ja s'han reproduït 12,24 segons, s'han descarregat 6 segments (aquests són de la qualitat de 989,932 Kbps) i la velocitat de la xarxa ha estat de

10,009 Mbps. En veritat aquests 12,24 segons són els reproduïts dels segments de la qualitat de 989,943 Kbps. Com veiem la traça no informa de quan o per què les accions han estat preses, però es segueix veient el caràcter adaptatiu del protocol.

4.6.4.3 Generació de gràfiques

Amb la traça generada pel reproductor propi i la interfície visual d'aquest, ja es veu clarament com el protocol va adaptant la qualitat dels segments a l'ample de banda disponible. Però, al cap i a la fi, hi ha tres paràmetres marcats en el temps que defineixen clarament aquest comportament: la velocitat de descàrrega, la qualitat del segment que s'està descarregant i la qualitat del segment que s'està reproduint. L'objectiu d'aquesta nova característica es plasmar aquests tres paràmetres en una gràfica durant el temps i veure, d'una manera visual, el que la traça ens indica.

Aquests paràmetres que hem indicat es van desant durant la reproducció i, un cop s'ha acabat la pel·lícula o l'usuari para la reproducció i torna a la pantalla de detall del vídeo, un nou botó apareix juntament amb el de visualització de la traça (Fig. 4.22).

Fig. 4.22 - **L'usuari pot veure una gràfica un cop acabada la reproducció**

Quan l'usuari escull aquesta opció, l'aplicació genera una gràfica a pantalla completa on és veu l'evolució dels paràmetres esmentats (Fig. 4.23).

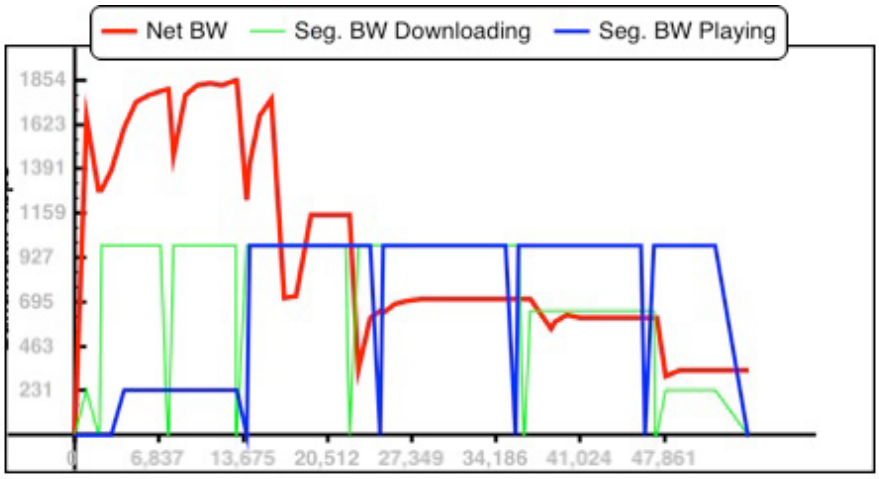

Fig. 4.23 - **Gràfica generada per l'aplicació**

En aquesta gràfica hi ha tres paràmetres; el vermell indica la velocitat de la xarxa, el verd el segment que s'està descarregant i el blau el segment que s'està reproduint. El l'eix horitzontal hi ha el temps (en segons) i en el vertical s'indiquen els quilo-bits per segon. En el cas dels segments, l'eix vertical indica la qualitat del segment en concret.

Si s'analitza la informació de la figura 4.23 podem veure que la velocitat de la xarxa, és a dir l'ample de banda disponible, ha anat disminuint amb el temps. El primer segment que s'ha descarregat ha estat el de qualitat més petita possible, tot i que l'ample de banda era elevat: això és degut a que fins que el sistema no es descarrega un segment aquest no coneix l'ample de banda disponible i, per recomanació d'Apple, el primer segment sempre és el de qualitat menor. Poc més d'un segon més tard d'haver-se descarregat aquest es reprodueix el segment. Durant la reproducció d'aquest primer segment es descarreguen dos més d'una qualitat una mica per sobre dels 927 kbps. Quan comença a reproduir-se el segon segment, el tercer també es comença a descarregar però triga una mica més que els dos anteriors ja que l'ample de banda disponible comença a baixar durant la descàrrega. Tot i que en acabar la descàrrega d'aquest segment l'ample de banda de la xarxa és menor que la qualitat del segment, la mitjana d'aquest durant el temps de descàrrega és més elevada i el següent segment a descarregar és de la mateixa qualitat i, com que l'ample de banda ha disminuït, el temps de descàrrega encara és més elevat. El següent segment a descarregar ja és d'una qualitat menor (no arriba a 695 Kbps).

En aquesta gràfica es pot veure clarament el caràcter adaptatiu del protocol i els paràmetres que fan decidir el següent segment a descarrega. Aquestes gràfiques són una eina molt útil per entendre el protocol.

4.6.4.4 Compartir els resultats per correu electrònic

Per a poder compartir i desar els resultats obtinguts de les reproduccions tant a la interfície de visualització de la traça com en la de la gràfica, s'ha afegit un botó a la barra superior per a compartir el resultat (Fig. 4.24).

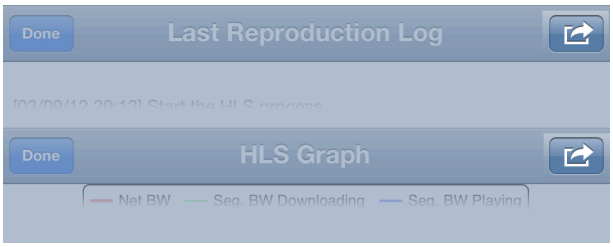

Fig. 4.24 - **L'opció per compartir per correu electrònic es troba en la barra superior**

En les dues interfícies hi ha l'opció de compartir el resultat per correu electrònic i, en el cas de la gràfica, també es pot desar a la llibreria d'imatges del propi telèfon (Fig. 4.25).

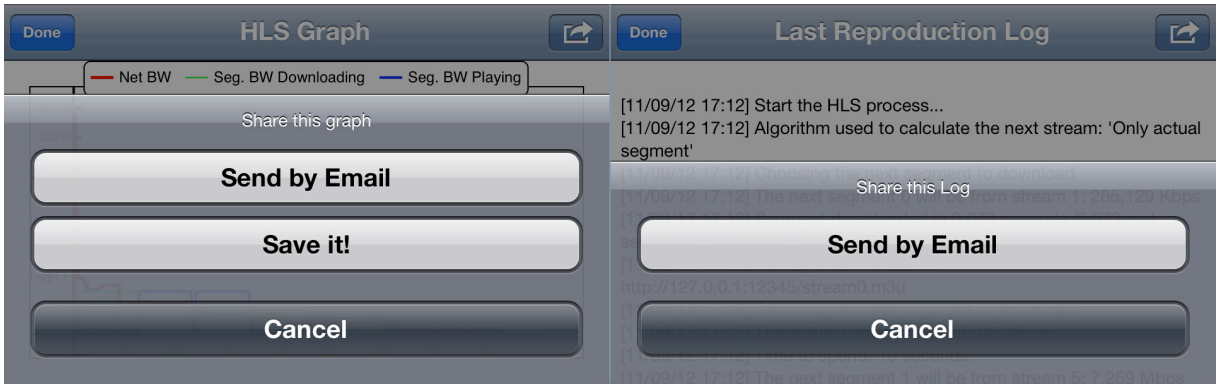

Fig. 4.25 - **Menú de l'opció de compartir per correu o desar a la llibreria d'imatges**

4.6.4.5 Diversos algoritmes en el moment d'escollir el següent segment

Tal i com s'explica al capítol 2, el protocol d'HTTP Live Streaming no defineix el procés de decisió del següent segment a descarregar, tan sols limita l'ordre d'aquests. En un sistema amb més

d'una qualitat on aquesta decisió repercuteix en la capacitat adaptativa del protocol, és molt important que aquesta no es prengui a la lleugera.

Apple recomana partir d'un sistema senzill en el moment d'escollir el següent segment, però també anima als desenvolupadors a definir uns sistemes robustos i eficients en aquests aspecte. En el punt 4.6.3.1 es defineix el sistema per defecte que fa servir l'aplicació per escollir el següent segment. La qualitat d'aquest depèn exclusivament de l'ample de banda detectat en l'últim segment descarregat. El que s'ha buscat amb aquesta millora és que l'usuari pugui escollir entre més d'un algoritme per calcular la qualitat del següent segment.

L'opció per escollir l'algoritme desitjat es troba amb les opcions de configuració de l'aplicació (punt 4.6.1). La figura 4.26 mostra l'opció de configuració i els algoritmes implementats.

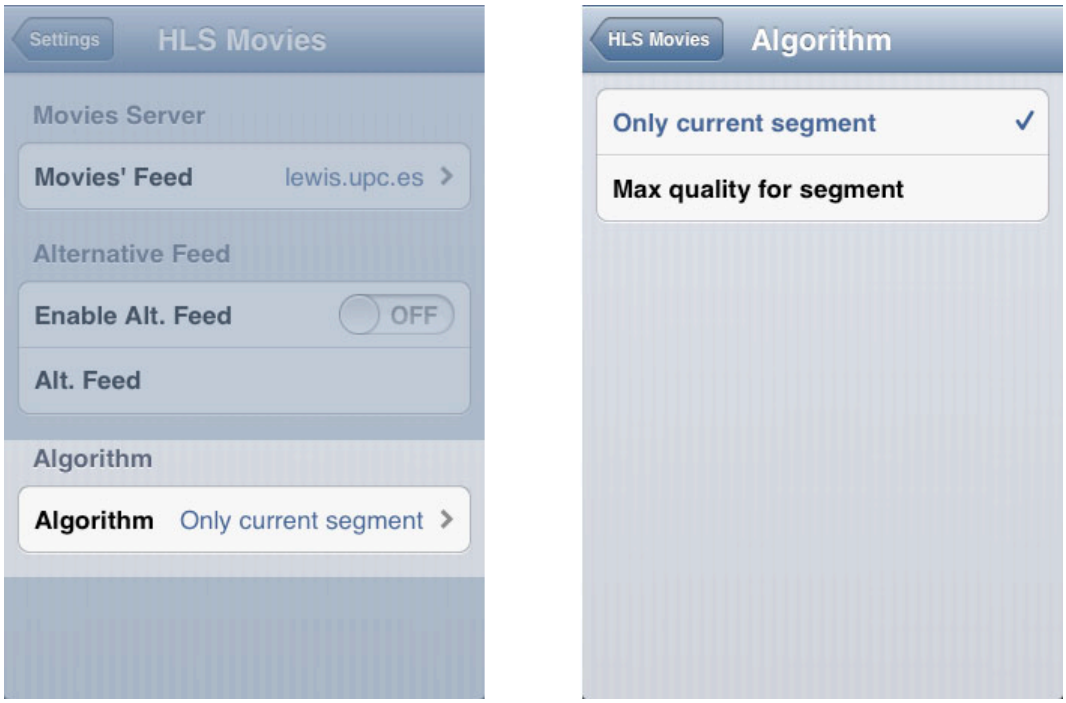

Fig. 4.26 - **Configuració de l'algoritme per a la selecció de la qualitat del proper segment**

Per defecte, l'aplicació fa servir l'algoritme anomenat "Only Current Segment" que, com el seu nom indica, tan sols té en compte el segment actual per calcular la qualitat del proper segment. Això vol dir que la velocitat mitjana de baixada de l'últim segment més un temps de marge i el

temps de reproducció d'un sol segment (10 segons per defecte) són els únics paràmetres que s'han de tenir en compte per a la selecció de la qualitat del proper segment a descarregar.

La segona opció, anomenada "Max quality for segment", intenta descarregar el segment de més alta qualitat possible. Per a fer-ho fa servir la velocitat mitjana de descàrrega de l'últim segment com a l'opció anterior, però a més de tenir en compte el temps de reproducció d'un segment més un temps de marge, també calcula el temps real que té per descarregar-se el segment, és a dir, tot el temps que té disponible. Aquest temps és la suma del temps que li falta de reproducció al segment que actualment s'està reproduint, més el temps de tots els segments que falten per reproduir, més els temps de marge per al canvi de segment que introdueix el reproductor. Això vol dir que, en comptes de tenir tan sols un temps de 10 segons, té 10 vegades el nombre de segments que falten per reproduir més els segons que falten del segment que s'està reproduint.
Les gràfiques de les figures 4.27 i 4.28 mostren la diferència entre els dos algoritmes esmentats.

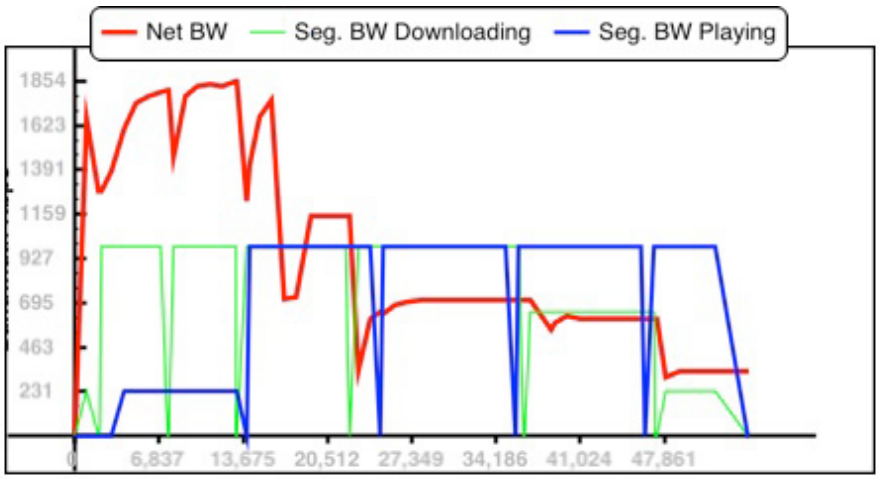

Fig. 4.27 - **Gràfica generada amb l'algoritme "Only Current Segment"**

En aquesta gràfica es pot veure com durant la descàrrega del cinquè segment, l'ample de banda ha disminuït i, tot i que tan sols s'havien reproduït tres segments, el sistema ha decidit descarregar el segment de qualitat inferior. Això ha passat per què no ha tingut en compte tot el temps que tenia disponible (dos segments per reproduir són 20 segons), sinó que ha tingut en compte tan sols el temps d'un segment (10 segons).

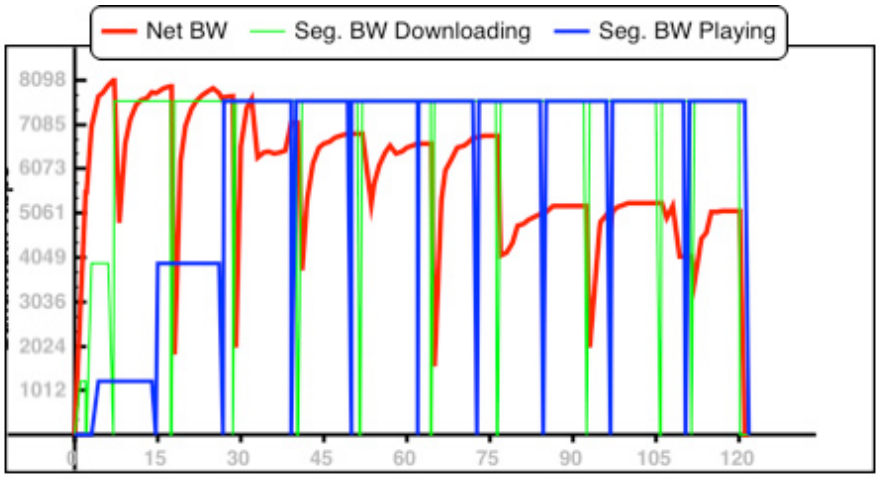

Fig. 4.28 - **Gràfica generada amb l'algoritme de "Max quality for segment"**

En aquesta altra gràfica, l'ample de banda ha anat disminuint però no la qualitat dels segments escollits per descarregar, donat que el sistema ha fet servir tot el temps disponible per calcular la qualitat dels segments a descarregar. Així no tenia només 10 segons, sinó que disposava de 10 segons multiplicats pels segments que encara no s'havien reproduït (però si descarregat) més els temps que quedava de reproducció del segment reproduint-se en aquell moment.

4.7 Solucions tecnològiques

Gran part del temps del projecte s'ha dedicat a la immersió en el món de la programació per a iOS amb Objective-C i per això s'entrarà en els següents punts en el detall de la programació de l'aplicació. No es tracta de descriure les més de 4.500 línies de codi del projecte, sinó de detallar aquelles parts del projecte que han suposat un repte, un problema o que, per algun altre motiu, mereixen ésser documentades aquí.

4.7.1 Descàrrega d'arxius

Una de les parts més importants del programa és la descàrrega d'arxius. Hi ha moltes ocasions on és necessari descarregar arxius: al començament i durant la reproducció d'un vídeo. Al iniciar-se l'aplicació, aquesta descarrega l'arxiu XML, totes i cadascuna de les llistes de reproducció o fitxers d'índex i també descarrega les imatges. Durant la reproducció, l'aplicació descàrrega de cadascun dels arxius .TS.

En el moment de desenvolupar la descàrrega de diferents arxius s'ha dissenyat un sistema capaç de suportar múltiples descàrregues asíncrones (per a la primera part de la descàrrega de la llista de pel·lícules i les seves dades) i que, a més a més, es pugui saber la velocitat a la qual s'està fent la descàrrega per a poder realitzar el reproductor.

Per poder assolir aquests objectius s'ha dissenyat un sistema de classes i protocols com el que es veu en la figura 4.29.

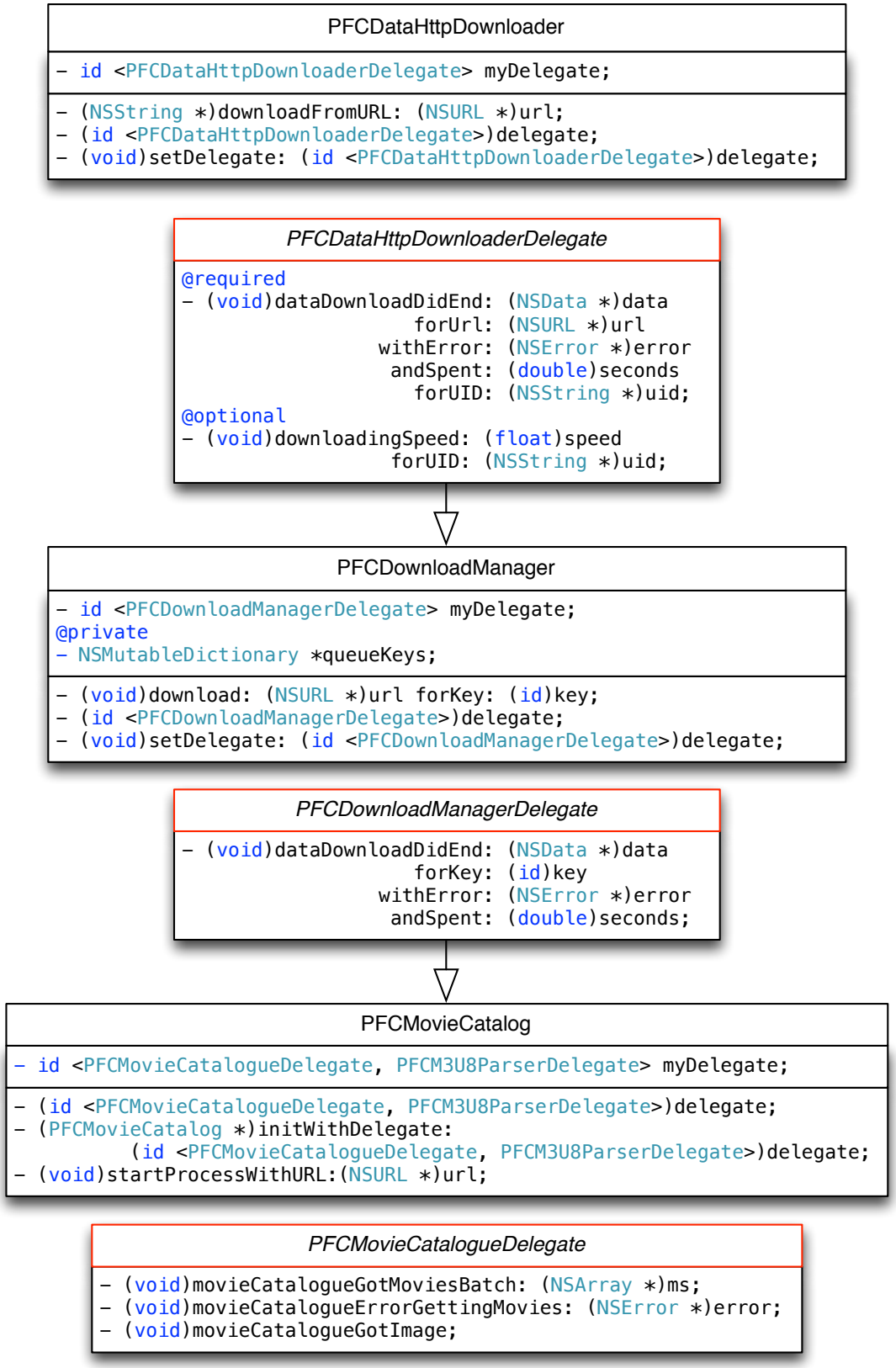

Fig. 4.29 - **Diagrama de classes per a la descàrrega de fitxers.**

Algunes d'aquestes classes tenen moltes més propietats i mètodes, que s'ometen d'aquest diagrama per poder mostrar d'una manera més clara el seu funcionament bàsic.

4.7.1.1 Descàrrega única

La classe **PFCDataHttpDownloader**, que a partir d'ara s'anomenarà "descarregador", és una classe destinada a descarregar un fitxer de manera asíncrona. Per a assolir-ho es defineix un protocol, **PFCDataHttpDownloaderDelegate**. Aquest protocol ha de ser implementat per la classe que faci de delegat del *descarregador* i serveixen per avisar que la descàrrega ha finalitzat (`dataDownloadDidEnd:forUrl:withError:andSpent:forUID:`) o per anar avisant sobre la velocitat amb la que té lloc la descàrrega (`downloadingSpeed:forUID:`).

Aquest descarregador només potde descarregar una classe al mateix temps. És asíncron per que un cop s'envia el missatge `downloadFromURL:`, la descàrrega comença i el codi pot seguir. El mètode del protocol implementat avisarà quan la descàrrega acabi.

Codi 4.3 - **Definició del mètode per descarregar un fitxer**

```
@interface PFCDataHttpDownloader() <NSURLConnectionDelegate>
...
@end


@implementation PFCDataHttpDownloader

...

-(NSString *)downloadFromURL:(NSURL *)url
{
    //get UID (this method generates a unique identifier string)
    UID = [self generateUidString];
    //Set the requested url to the interbal var
    requestedURL = url;

    NSLog(@"Start downloading from '%@'", requestedURL);

    if (!requestedURL)
    {
        //If the url passed asparameter is null indicate the error
        ...
        return UID;
    }

    //If the download link was downloaded in meantime return the file to the user
    if ([cache objectForKey:requestedURL] != nil) {

        NSLog(@"Download is in cache: %@", requestedURL.absoluteString);
```

```
        //return the data after a while because first the UID created must be returned;
        [self performSelector:@selector(returnCacheData) withObject:nil afterDelay:100];

    } else {
        // Use NSURLConnection to asynchronously download the data. This means the main
        // thread will not
        // be blocked - the application will remain responsive to the user.
        //
        // IMPORTANT! The main thread of the application should never be blocked!
        // Also, avoid synchronous network access on any thread.
        //
        NSURLRequest *urlRequest = [NSURLRequest requestWithURL:requestedURL];
        //get startTime
        downloadStarted = [NSDate date];

        //start request
        self.connection = [[NSURLConnection alloc] initWithRequest:urlRequest
                                                          delegate:self];

        // Test the validity of the connection object. The most likely reason for the
        // connection object
        // to be nil is a malformed URL, which is a programmatic error easily detected
        // during development.
        NSAssert(self.connection != nil, @"Failure to create URL connection.");

        // Start the status bar network activity indicator. We'll turn it off when the
        // connection finishes or experiences an error.
        [UIApplication sharedApplication].networkActivityIndicatorVisible = YES;
    }

    return UID;
}

- (void)returnCacheData
{
    [self.delegate dataDownloadDidEnd:[cache objectForKey:requestedURL]
                               forUrl:requestedURL
                            withError:nil
                            andSpent:0
                               forUID:UID];
}

...
```

Al codi 4.3 podem veure un dels mètodes principals del descarregador. Aquest és el mètode principal per a la descàrrega d'un fitxer. El primer que podem veure és que la clase implementa un protocol anomenat *NSURLConnectionDelegate*, això és degut a què, després de comprobar que la URL és correcte i assignar un identificador únic a la descàrrega, fa servir una instància del *NSURLConnection* per a realitzar la descàrrega. Aquesta classe és del framework de Cocoa i fa servir el protocol esmentat per a retornar les dades de la petició asíncronament. El descarregador és el delegat d'aquesta classe, així que és aquest qui ha d'implementar el protocol *NSURLConnectionDelegate.*

```
@interface PFCDataHttpDownloader() <NSURLConnectionDelegate>
```

Al codi 4.2 es pot veure la implementació de dos dels mètodes del protocol *NSURLConnectionDelegate* que s'han implementat.

Codi 4.2 - **Implementació dels mètodes del protocol *NSURLConnectionDelegate***

```
#pragma mark -
#pragma mark NSURLConnection delegate methods

- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data
{
     // Add bytes got to the data buffer
    [requestedData appendData:data];

    if ([self.delegate respondsToSelector:@selector(downloadingSpeed:forUID:)])
        // Call the delegate method to indicate the current download speed
        [self.delegate downloadingSpeed:
         (requestedData.length*8/[[NSDate date] timeIntervalSinceDate: downloadStarted])
                                 forUID:UID];

}

- (void)connectionDidFinishLoading:(NSURLConnection *)connection
{
    self.connection = nil;
    [UIApplication sharedApplication].networkActivityIndicatorVisible = NO;
    [cache setObject:requestedData forKey:[requestedURL copy]];

    // Call the delegate method to indicate that the data has been downloaded
    [self.delegate dataDownloadDidEnd:requestedData
                               forUrl:requestedURL
                            withError:nil
                            andSpent:[[NSDate date] timeIntervalSinceDate:
                                                              downloadStarted]
                               forUID:UID];
    self.connection = nil;
}
```

Com es pot veure, el primer mètode, anomenat `connection:didReceiveData:`, es crida cada cop que la connexió rep dades, aquestes s'afegeixen al *buffer* que hi ha definit (un cop acabada la descàrrega aquestes dades conformaran el fitxer que s'ha demanat) i es crida al mètode `downloadingSpeed:timeIntervalSinceDate:` del protocol *PFCDataHttpDownloaderDelegate* per indicar l'actual velocitat de descàrrega. Aquest mètode es opcional, així que pot ser que el delegat no l'implementi, per aquest motiu es fa primer la validació amb la següent crida:

```
if ([self.delegate respondsToSelector:@selector(downloadingSpeed:forUID:)]) ...
```

Un cop acabada la descàrrega es crida el mètode `connectionDidFinishLoading:`, en aquest s'executa el mètode `dataDownloadDidEnd:` del protocol *PFCDataHttpDownloaderDelegate* per

indicar que la descarrega s'ha acabat i retornar les dades descarregades, o l'error si hi ha hagut cap, i el temps emprat en la descàrrega.

És molt important poder saber el temps i la velocitat, ja que aquesta classe és la que es fa servir en el reproductor per descarregar els diferents segments. Per realitzar el procés de selecció dels segments a descarregar, i aconseguir un comportament adaptatiu, es necessita saber la velocitat de descàrrega i el temps de descàrrega de l'últim segment.

Aquesta classe és útil si tan sols és vol descarregar un fitxer, però si es volen descarregar una serie d'enllaços, s'han de crear tantes instàncies del *descarregador* com siguin necessàries i enviar el missatge `downloadFromURL:` a cadascuna d'elles. És possible que existeixi més d'un *descarregador* executant-se alhora i descarregant fitxers, però tots comparteixen el mateix delegat; tots aquests *descarregadors* criden el mateix mètode de protocol en acabar i el delegat ha de saber identificar de quina de les descàrregues que ha executat es tracta. Per a fer-ho, el mètode `downloadFromURL:` retorna un identificador únic de descàrrega al delegat. Quan una descàrrega acaba i el *descarregador* crida el mètode del delegat li passa també l'identificador de descàrrega, d'aquesta manera el delegat pot saber de quina descàrrega es tracta si n'està gestionant més d'una.

4.7.1.2 Descàrregues múltiples

Per realitzar aquest descàrrega asíncrona de múltiples fitxers s'ha definit la classe **PFCDownloadManager**, que anomenarem "gestor de descàrregues". Aquest gestor rep el missatge `download:forKey:` per a començar una descàrrega, i se li pot enviar aquest missatge per a tantes descàrregues com sigui necessari. Com que aquest gestor fa servir el descarregador, aquest implementa el protocol *PFCDataHttpDownloaderDelegate*.

```
@interface PFCDownloadManager() <PFCDataHttpDownloaderDelegate>
```

Aquest gestor s'ha definit per a executar-se al començament de l'aplicació. Un cop descarregat el XML es van creant les diverses instàncies de *PFCMovie* i es van descarregant totes les imatges dels vídeos a mesura que aquests es creen de forma asíncrona. Un cop s'ha descarregat cada imatge s'actualitza la fila en que apareix, tal i com es mostra en la figura 4.30.

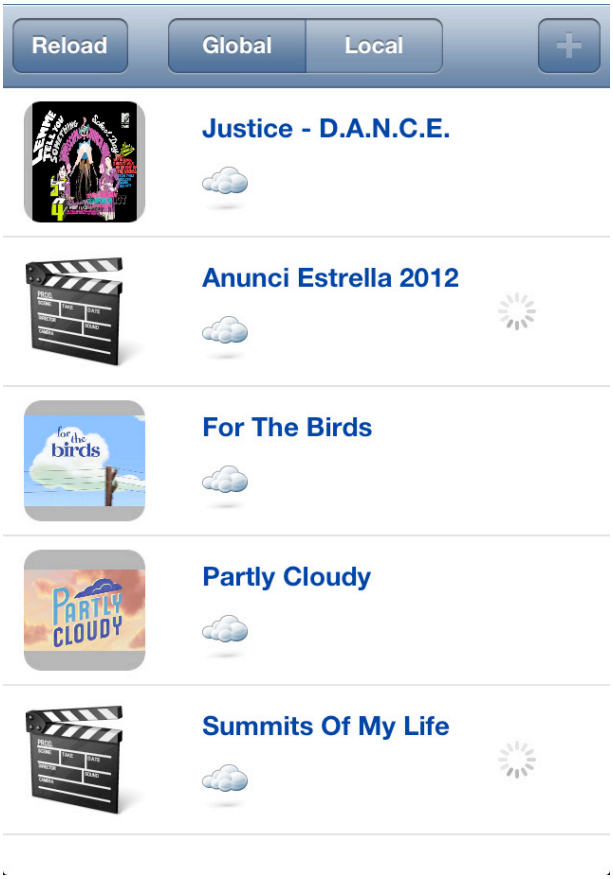

Fig. 4.30 - **Les películes i les seves imatges es decarreguen asíncronament**

La classe *PFCDownloadManager* també realitza totes les descàrregues asíncronament i ho fa per mitjà d'un nou protocol anomenat **PFCDownloadManagerDelegate**. Aquest protocol, a diferència del seu homòleg per al *downloader*, només consta d'un mètode (Codi 4.3)

Codi 4.5 - **Definició del protocol *PFCDownloadManagerDelegate***

```
@protocol PFCDownloadManagerDelegate <NSObject>

- (void)dataDownloadDidEnd: (NSData *)data
                    forKey: (id)key
                 withError: (NSError *)error
                  andSpent: (double)seconds;

@end
```

La classe que fa les funcions de delegat del gestor de descàrregues i que ha d'implementar aquest protocol tan sols ha d'implementar un únic mètode. Com podem veure, aquest mètode,

a part del fitxer descarregat 'data', de la informació de l'error si n'hi ha 'error' i del temps en el que s'ha realitzat 'seconds', rep un paràmetre anomenat 'key'. Si mirem el missatge enviat per a començar una descàrrega veurem que aquest, a part del enllaç al fitxer a descarregar, també conté un paràmetre anomenat 'key':

```
- (void)download: (NSURL *)url forKey: (id)key;
```

Degut a l'elevat nombre de descàrregues, cal identificar cadascuna d'elles. Aquest paràmetre és l'identificador i pot ser qualsevol objecte. Així que podem identificar directament la descàrrega, per exemple, amb la instància de la classe *PFCMovie* a la que pertany.

4.7.1.3 Catàleg de pel·lícules

Per separar totalment el model de dades, la gestió i adquisició d'aquestes de la interfície d'usuari (la llista de pel·lícules), s'ha dissenyat una entitat que gestiona aquestes tasques: **PFCMovieCatalogue** (que anomenarem "Catàleg"). Aquesta entitat s'encarrega d'orquestrar el procès d'adquisició i generació de les entitats *PFCMovie.*

Aquesta classe, *PFCMovieCatalogue*, conté una instància del gestor de descàrregues per obtenir els diferents fitxers d'índex i les imatges de les pel·lícules, i una instància d'una altre classe que s'encarrega de generar els diferents elements que conformen la totalitat de la classe PFCMovie (Fig. 4.7); els streams (PFCHLSStream), les llistes de reproducció (PFCPlaylist) i els segments (PFCSegment). També conté la classe que fa de "parser" del fitxer XML amb la llista de pel·lícules.

Com es tracta d'un entorn asíncron es defineix un protocol per a comunicar-se amb el delegat (en aquest cas, el delegat és la instància de la classe que gestiona la interfície gràfica). Aquest protocol s'anomena **PFCMovieCatalogueDelegate** i defineix tres mètodes que s'han d'implementar en el delegat:

Codi 4.6 - **Definició del protocol *PFCMovieCatalogueDelegate***

```
@protocol PFCMovieCatalogueDelegate <NSObject>

- (void)movieCatalogueGotMoviesBatch: (NSArray *)ms;
- (void)movieCatalogueErrorGettingMovies: (NSError *)error;
- (void)movieCatalogueGotImage;

@end
```

El primer mètode `movieCatalogueGotMoviesBatch:` retorna les pel·lícules un cop han estat generades des del XML. Les retorna en blocs de 4 perquè es vagi refrescant la interfície d'una forma continuada amb les noves pel·lícules. Aquestes pel·lícules no tenen ni la imatge ni l'estructura interna, però tenen les dades bàsiques per mostrar-les per pantalla. El *catàleg* segueix processant aquestes pel·lícules; descarregant les imatges i generant l'estructura interna a través de la informació dels fitxers d'índex.

La interfície, un cop rep les dades bàsiques de les pel·lícules, les mostra per pantalla sense imatge i amb el símbol de càrrega (Fig. 4.31).

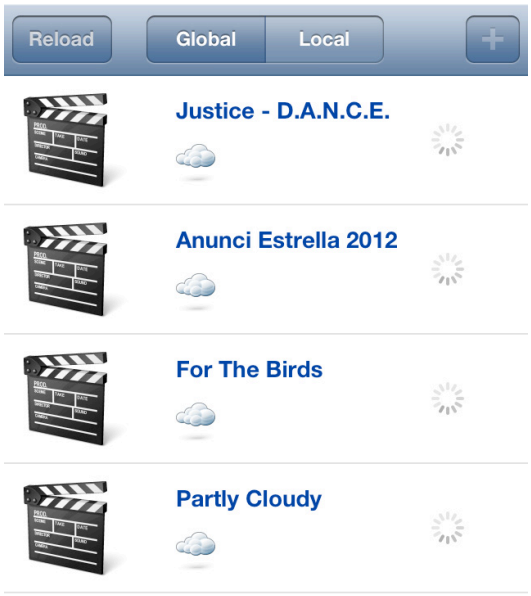

Fig. 4.31 - **Carregant les primeres pel·lícules, aquestes no tenen ni imatge ni estructura**

A mesura que el *catàleg* va processant les imatges, avisa al seu delegat que les imatges estan carregades. Si hi ha cap error en una pel·lícula també ho indica al seu delegat. Cada cop que el delegat rep un d'aquests mètodes (missatges) refresca la llista per a que els canvis es puguin visualitzar. En la figura 4.32 es poden veure tres pel·lícules, la primera no té ni imatge ni estructura interna; la segona ja ha rebut la imatge i la tercera ja està llesta per a la seva visualització.

Fig. 4.32 - **Actualització asíncrona dels nivells de generació de les pel·lícules**

Si l'usuari entra a una pel·lícula per a veure els seus detalls, per veure l'estructura dels fitxers d'índex i la seva informació, però aquests encara no han estat generats, el sistema mostra un missatge informantiu (figura 4.33).

Fig. 4.33 - **L'estructura interna de la pel·lícula encara no s'ha generat**

4.7.1.4 Generador de l'estructura interna d'una pel·lícula

Una de les peculiaritats de la definició de la classe *PFCMovieCatalogue* està en la definició del delegat. Fins ara hem vist que els delegats han d'implementar el protocol pertinent. En aquest cas, el delegat de la classe catàleg na d'implementar dos protocols:

```
id <PFCMovieCatalogueDelegate, PFCM3U8ParserDelegate> myDelegate;
```

El primer protocol és el definit per gestionar la comunicació amb els mètodes de la classe *PFCMovieCatalogue.* Però, i el segon? El segon protocol el defineix la classe que genera l'estructura interna de les entitats *PFCMovie* a partir de la llista de reproducció mare, **PFCM3U8Parser**. Aquesta classe es comunica asíncronament amb el seu delegat a través del protocol **PFCM3U8ParserDelegate**.

> **Nota:** l'estructura de la classe PFCM3U8Parser i el seu funcionament queda fora de l'objectiu d'aquest punt: explicar el component asíncron i de descàrrega de l'aplicació.

El catàleg defineix el seu delegat amb els dos protocols, perquè al fer servir la classe *PFCM3U8Parser* i indicar-li el delegat, li passa directament el seu. Així és la classe *PFCM3U8Parser* la que indica directament a la interfície que ja s'ha generat l'estructura interna de la classe *PFCMovie* per mitjà del mètode de protocol `m3u8ParserMovieEnded:`.

Codi 4.7 - **Definició del protocol PFCM3U8Parser**

```
@protocol @protocol PFCM3U8ParserDelegate <NSObject>
- (void)m3u8ParserMovieEnded: (PFCMovie *)movie;

@end
```

La classe *PFCM3U8Parser* descarrega el fitxer .m3u8 i llegeix línia a línia el seu contingut creant les entitats *PFCHLSStream* segons la definició de les paraules clau d'aquests, que dóna el protocol HTTP Live Streaming. Després descarrega els fitxers d'índex corresponents de cada qualitat i torna a llegir-los línia a línia creant l'entitat *PFCPlaylist* i la seva llista de segments *PFCSegment*. Finalment desa aquesta llista d'objectes *PFCHLSStream* en la *PFCMovie* i ho indica al seu delegat.

La classe de gestió de la interfície gràfica que crea la instància de la classe *PFCMovieCatalogue* només ha d'inicialitzar el *catàleg* amb el delegat pertinent (que en aquest cas és la mateixa classe de gestió de la interfície) i executar el mètode `startProcessWithURL` (passant-li el enllaç al fitxer XML) per iniciar el procés. Automàticament i de forma asíncrona es descarregarà i generarà tota la llista de pel·lícules donant forma i contingut a l'aplicació.

4.7.2 Reproductor dels segments

Durant la programació del sistema de reproducció didàctic fent servir el protocol HLS ha sorgit una de les dificultats més grans del projecte. A continuació es planteja el problema i la solució aplicada.

El reproductor didàctic implementat a l'aplicació fa servir un component del framework de vídeo de Cocoa anomenat **AVQueuePlayer**. Aquest component permet reproduir qualsevol format de vídeo conegut (que el sistema és capaç de reproduir) i, a més, anar afegint elements a la reproducció perquè que els reprodueixi l'un darrera l'altre.

El sistema segueix els protocols anteriorment explicats per descarregar els arxius .TS del segment que toqui, un cop descarregats es desen en el mòbil i es posen a la llista de reproducció del reproductor *AVQueuePlayer.*

La sorpresa ha aparegut quan al intentar reproduir el fitxer .TS descarregat, el reproductor retornava un error; el format no era reconegut i no era capaç de reproduir-lo. Era ben estrany, doncs el sistema i el reproductor eren capaços de reproduir el vídeo si se'ls hi passava directament la llista de reproducció .m3u8; el reproductor procedia a implementar internament el protocol HLS i el vídeo es reproduïa sense cap mena de problema. Però el que es buscava era que reproduís cada segment per separat; és feina de la nostra aplicació dur a terme el protocol. Resulta, però, que si al reproductor se li passa l'enllaç al fitxer .TS tal i com apareix en el fitxer d'índex, aquest si que era capaç de reproduir-lo sense cap mena de problema.

Resulta que en tractar-se d'un format de vídeo encapsulat en una trama de transport, MPEG-2 TS, el reproductor tan sols és capaç de reproduir-lo si aquest fitxer ve dins d'una trama HTTP. Després d'una llarga recerca per Internet i de veure que el problema era comú a tots els

desenvolupadors que intentaven reproduir aquesta mena de fitxers (amb les eines del framework de Cocoa), es van trobar dues possibles solucions.

La primera solució és endinsar-se en el fantàstic món de la codificació de vídeo i codificar un reproductor que fos capaç de reproduir arxius .TS. Aquesta solució queda totalment fora del àmbit del projecte i comportaria una gran inversió en temps i recursos dels quals no es disposa. Així que va quedar descartada.

La segona solució és aconseguir passar-li l'arxiu .TS dins d'una connexió HTTP. Per aconseguir-ho s'ha hagut d'integrar un servidor HTTP en l'aplicació. Un cop descarregats els fitxers .TS aquests es desen en un carpeta dedicada que serà la carpeta arrel que es configura en el servidor HTTP intern. Un cop els fitxers són desats en aquesta carpeta queden publicats en enllaços del tipus *http://127.0.0.1:12345/fileSequence0.ts*. Un cop reproduïts s'esborren per a no ocupar lloc al disc.

Per a la implementació del servidor HTTP intern s'ha utilitzat un component de tercers. S'ha fet servir un servidor HTTP per a iOS creat per Robbie Hanson anomenat CocoaHTTPServer. Tot i que aquest servidor es bastant complet, les línies de codi per fer-lo funcionar han estat molt poques i la seva integració a l'aplicació no ha estat gaire complicada.

A l'inici de l'aplicació s'inicialitza el servidor en una carpeta i port determinats i aquest ja està llest per servir fitxers.

Codi 4.8 - **Inicialització del servidor HTTP**

```
//start the server
server = [[HTTPServer alloc] init];
//set server port
[server setPort:kPORT];
//set server foler
[server setDocumentRoot:[PFCCommonHTTPServer getFolderPath]];
// Start the server (and check for problems)

NSError *error;

if([server start:&error])
{
     NSLog(@"Started HTTP Server on port %hu", [server listeningPort]);
    [PFCCommonHTTPServer setServerRunning:YES];
     return YES;
}
```

```
else
{
     NSLog(@"Error starting HTTP Server: %@", error);
    [PFCCommonHTTPServer setServerRunning:NO];
    return NO;
}
```

A partir d'aquest moment es pot accedir a tots els fitxers que es desin dins del directori indicat mitjançant una cadena HTTP.

Així doncs, tornant al reproductor d'HLS implementat en l'aplicació, un cop decidit quin segment (de quina qualitat) s'ha de descarregar, es descarrega i es desa dins del directori del servidor web. Un cop desat, es posa a la cua del reproductor *AVQueuePlayer*, però en comptes de posar directament la ruta del arxiu, es posa la ruta web a aquest: *http://127.0.0.1:12345/fileSequence0.ts*. Com el fitxer .TS ve dins d'una trama HTTP el reproductor el pot reproduir sense cap problema; tot i ser un enllaç web, el fitxer està en el mateix telèfon i no hi ha pèrdua de temps en la descàrrega d'aquest.

4.7.3 Disseny de la interfície gràfica

Per al disseny de la interfície gràfica i de la navegació entre diferents pantalles a través de les accions de l'usuari s'ha fet servir una de les novetats que incorpora el programa XCode a partir de la versió 5 del iOS.

Fins a la versió 4 del iOS, el mètode de disseny d'interfícies es basava en l'edició d'arxius .xib en l'editor gràfic. Cada pantalla tenia el seu .xib corresponent i s'havia d'editar per separat, tal i com s'ha vist en el capítol anterior en el punt 3.3.2. A partir del iOS 5 s'ha incorporat una nova manera de dissenyar les diferents interfícies i els vincles de navegació entre elles: els anomenats *storyboards*.

Els *storyboards* són una eina del programa XCode per agilitzar el procés de disseny d'interfícies d'usuari. Es tracta de dissenyar totes les interfícies, o escenes (pantalles), en una mateixa vista, i enllaçar-les entre elles amb el que en diuen *segue.* Aquests enllaços uneixen una vista, ja sigui un botó, una taula o la vista completa, amb un *UIViewController*. D'aquesta manera s'indica que

l'acció realitzada sobre la vista origen (si és un botó, serà el fet d'accionar-lo; si és una taula, l'acció de selecció d'una fila, etc.) fa que es carregui el controlador destí.

En aquesta gran pantalla de disseny, l'*storyboard*, es treballa sobre cada escena com si d'un .xib individual es tractés. Es defineixen les diferents vistes que formen la pantalla, amb els seus atributs i identificadors, i tot allò que calgui definir en un arxiu .xib. La diferència és que per a cadascuna de les escenes s'ha de definir quina de les interfícies, que hereten de *UIViewController,* és la que gestiona cada vista.

Un dels avantatges dels *storyboards* és que amb un sol cop d'ull és pot veure tota l'aplicació i les pantalles que la formen i com es navega per elles. És a dir que l'*storyboard* de l'aplicació permet veure tota l'estructura.

A l'*storyboard* de l'aplicació es pot veure com es parteix d'un control de navegació que carrega primerament la interfície amb la llista de pel·lícules. Es veu com d'aquesta pantalla es pot accedir a dues més: una per afegir una nova pel·lícula i l'altre que mostra els detalls d'una pel·lícula en concret. Des d'aquesta última es pot anar a tres altres pantalles o escenes, que és com ho anomenen els d'Apple: el reproductor didàctic d'HLS, el visor de la traça i el visor de la gràfica. Aquest *storyboard* es pot veure en la figura 4.34.

La reproducció d'una pel·lícula fent servir el reproductor intern en el framework no té una escena pròpia, ja que està integrada dins del disseny de la pantalla amb els detall de la pel·lícula.

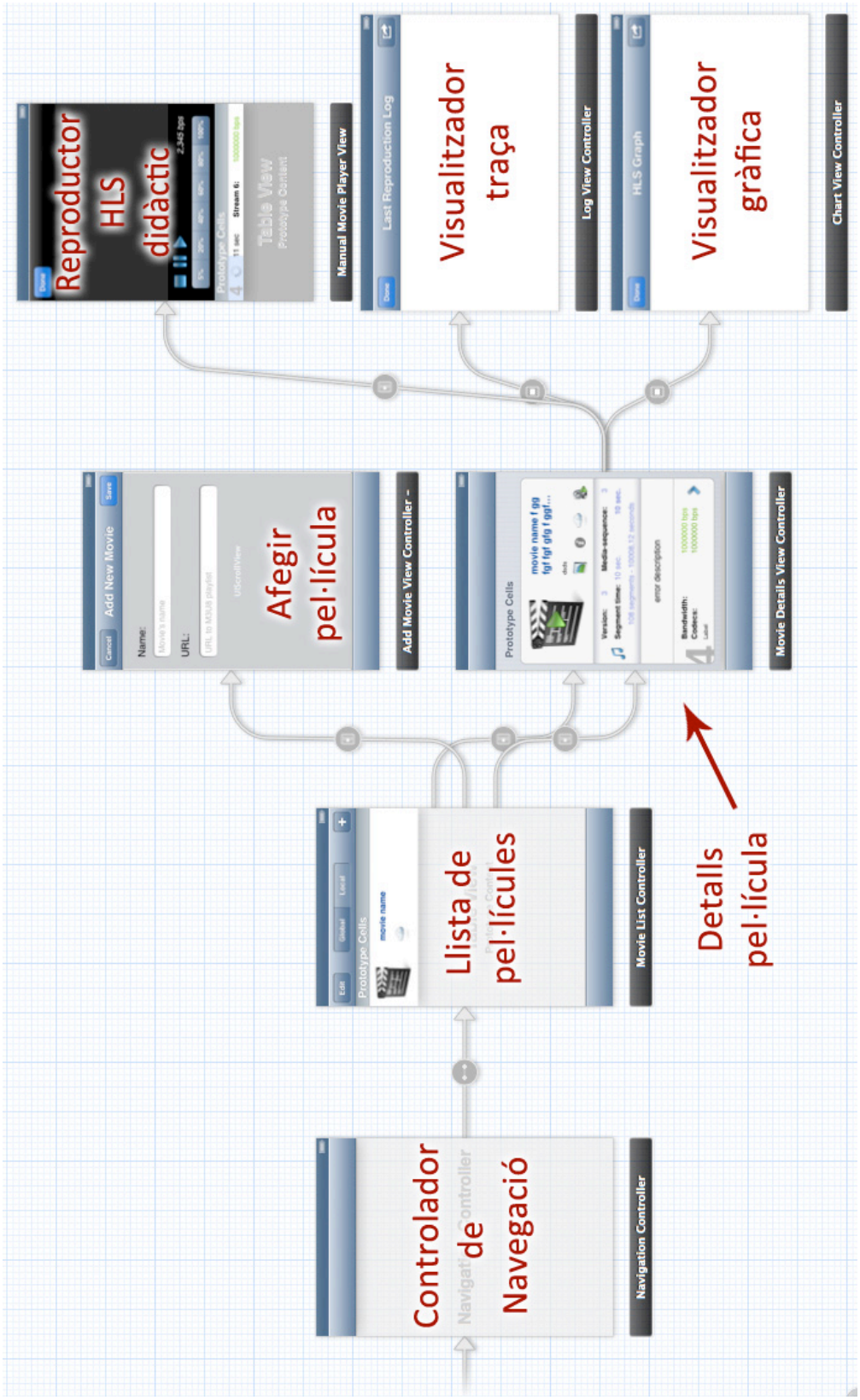

Fig 4.34 - **Storyboard l'aplicació**

4.7 Millores en l'aplicació

Tot i que l'aplicació funciona i assoleix tots els requeriments, fins i tot amb algunes característiques afegides, hi ha molts aspectes que s'haurien de millorar i ampliar per a poder donar-la per acabada.

En l'àmbit de funcionament d'aquesta hi ha encara alguns errors coneguts en aquesta versió que s'haurien d'arreglar: hi ha alguns casos que es poden donar que no estan contemplats i que es traduirien en un error en l'aplicació i un mal funcionament.

Aquests són els errors coneguts.

- Si la càrrega de les pel·lícules falla o el servidor no respon, l'única manera de tornar a intentar-ho és tancar del tot l'aplicació i tornar-la a executar.
- De vegades reproductor d'HLS dóna error després d'algunes reproduccions i s'ha de tancar l'aplicació del tot i re-iniciar-la. De vegades passa molt aviat, en la segona reproducció, d'altres funciona bé durant molta estona.
- La gràfica es carrega de tal manera que no es veuen les etiquetes dels eixos, s'ha de fer zoom amb els dits per poder veure-les.
- De vegades el símbol de descàrrega de dades que apareix al costat del nom de l'operadora i del senyal del tipus de xarxa (Wifi, 3G, GPRS, ...) es queda funcionant tot i que, en teoria, no s'està descarregant res.
- Durant la reproducció hi ha un espai de temps entre segment i segment en el que el reproductor es queda parat i pensant. Està carregant el següent vídeo de la cua. S'hauria de muntar un sistema per eliminar aquesta pausa.

A part del fet d'arreglar els errors hi ha algunes millores que s'haurien d'introduir per tal de millorar la capacitat didàctica per poder ensenyar tot el funcionament complet del protocol HTTP Live Streaming. Ara mateix, l'aplicació només compleix els requisits mínims i és capaç de reproduir *streams* de vídeo sota demanda i res més. No s'ha implementat ni la part de seguretat i xifrat, ni tampoc està preparada per reproduir àudio sota demanda.

Aquestes són algunes de les millores que s'haurien d'afegir en properes versions per tal de cobrir totes les parts de les primeres versions del protocol HLS:

- Suport per a àudio sota demanda
- Suport per a àudio i vídeo en temps real (fitxers d'índex dinàmics)
- Suport de la part de seguretat i xifrat del protocol.
- Habilitar les accions de l'usuari per moure's a través de tota la línia temporal del vídeo.

Un cop incorporades aquestes millores, l'aplicació seria molt bona al meu parer per a la seva funció. Seria una aplicació que ensenyaria didàcticament totes les característiques esmentades en la documentació i tot el funcionament del protocol HTTP Live Streaming.

- Capítol 5 -

CONCLUSIONS I OBSERVACIONS PERSONALS

5 Conclusions i observacions personals

La gran expansió de dispositius portàtils connectats a xarxes de comunicacions de telefonia mòbil ha fet que la necessitat de protocols d'un sistema d'*streaming* adaptatiu sigui més que evident. Al mercat existeixen diverses solucions que cobreixen aquesta necessitat.

La raó per la qual s'ha escollit el protocol HTTP Live Streaming d'entre els diversos sistemes diferents que podem trobar ha estat per la simplicitat i facilitat d'aquest protocol com a exemple per ensenyar el concepte d'*streaming* adaptatiu dins del marc docent. En el primer capítol s'ha vist com l'arquitectura bàsica dels sistemes d'*streaming* adaptatiu no varia gaire entre les diverses solucions, però mentre que l'HLS, tot i ser d'Apple, no està lligat a cap *software* propi, les altres opcions sí que ho estan: un altre factor important per escollir l'HLS com a eina docent idònia en aquest camp.

A continuació s'exposen algunes reflexions i observacions sobre els dos aspectes principals que es tracten en aquest projecte: el protocol HLS i el desenvolupament per al sistema operatiu iOS.

5.1 Conclusions i observacions sobre el protocol HTTP Live Streaming

Un dels principals avantatges d'aquest protocol és la facilitat de programació per disposar d'un servei d'*streaming* de continguts. Com a experiència personal, i tot i que ha estat el primer cop que he treballat amb sistemes d'*streaming* de vídeo, tant els conceptes com la metodologia emprada han resultat molt senzills d'entendre i de dur a terme.

A nivell professional i d'implantació, el fet de què el facin servir els dispositius d'Apple, que sigui compatible amb l'etiqueta <video> d'HTML5, que no calgui cap *software* propietari per crear la part del servidor i que només es necessitin servidors web per posar el vídeos a disposició de l'usuari, fa que la seva projecció i implantació no comporti gaires dificultats per a sistemes i serveis de nova creació. Crec que el valor afegit de les poques despeses que comporta la implementació d'un sistema d'*streaming* en HLS farà que molts dels nous serveis optin per fer servir aquest protocol.

En el cas dels serveis més estesos i assentats al mercat, com per exemple YouTube, malgrat que poc a poc migren cap a tecnologies compatibles amb l'HTML5 i sistemes adaptatius, trobo que és molt difícil que es decideixin per fer servir l'HTTP Live Streaming. Gran part del principals serveis d'*streaming* de vídeo han fet servir fins ara les solucions d'Adobe en els seus sistemes i, tot i que ara migren a solucions adaptatives, segueixen fent servir serveis d'Adobe. Tots els seus continguts i recursos ja estan codificats i emmagatzemats en formats compatibles amb aquests sistemes. Tot i que l'adopció de l'HLS suposaria unes despeses inferiors a llarg termini, hi ha dos factors importants en contra de l'adopció d'aquest protocol. El primer factor és degut a l'elevat cost de personal, recursos i temps que suposaria la migració i transformació de tots els continguts a aquest nou format i el desenvolupament de les noves interfícies (pàgines web, aplicacions, etc.) necessàries. El segon factor ve implícit en el primer, totes aquestes tasques d'adaptació dels recursos i nous desenvolupaments comporten la introducció d'errors en el sistema i moltes possibilitats de pèrdua de dades o destrucció d'aquestes. Aquestes despeses i aquests riscos són factors que s'han de tenir en compte.

El fet que una gran empresa com Apple sigui la responsable d'aquest protocol dóna un valor afegit i una seguretat en el moment d'escollir un protocol. Com s'ha vist al capítol 2, tot i que el protocol encara és un *Internet Draft*, ja està implementat en molts sistemes. Moltes empreses importants fan que els seus productes siguin compatibles amb el protocol. A més, per finalitzar, Apple segueix afegint-hi millores per enriquir l'experiència de l'usuari i les capacitats de l'HLS.

5.2 Desenvolupament per l'iOS

Abans de realitzar aquest projecte mai havia desenvolupat res per iOS ni programat en Objective-C, així que, per començar, m'hi vaig trobar amb una dificultat afegida al projecte: aprendre a desenvolupar aplicacions fent servir el framework de Cocoa i les regles de l'iOS. La meva experiència en programació ha fet que aquest procés d'aprenentatge hagi estat bastant ràpid. Addicionalment a l'interés per treballar en un sistema nou i amb dispositius tant atractius i moderns com l'iPhone, que m'ha ajudat a que el camí fos més fàcil, l'entorn de desenvolupament i les metodologies emprades per treballar amb el sistema operatiu iOS, que marquen els patrons d'Apple, m'han ajudat a sentir-me ràpidament còmode en aquest sistema.
Porto cinc anys treballant com a professional amb la plataforma .NET de Microsoft i desenvolupant programes de gestió d'escriptori per a Windows. L'entorn de treball que faig

servir és el Visual Studio, el qual porta anys creixent i evolucionant. Tot i això, l'Xcode m'ha sorprès.

He treballat molt a gust amb aquest programa; les facilitats i les característiques que aporta per facilitar les tasques més comunes fan que la tasca de programació sigui més eficient. El programa no està a l'alçada del Visual Studio (l'únic amb el qual el puc comparar) en termes de característiques, però si parlem d'un entorn còmode per treballar, l'XCode és, al meu parer, millor.

En la programació i el desenvolupament de característiques i accions per a una aplicació, el programador es troba de vegades amb un problema afegit quan es treballa en iOS: les limitacions que Apple ha introduït en el sistema i les limitacions per a poder publicar l'aplicació a la botiga d'aplicacions d'Apple. Cada aplicació s'executa en un entorn tan tancat que no permet l'accés als recursos del sistema ni a d'altres aplicacions si no hi ha un servei d'Apple enfocat a aquesta finalitat (càmera, galeria de fotografies, etc.). Naturalment, la seguretat ha estat sempre un aspecte vital en tots els productes d'Apple, però també limita molt les opcions del programador en el moment de solucionar problemes o desenvolupar característiques.

ANNEXES

A1 Enllaços i bibliografia

En aquest annex es troba la llista d'enllaços a diferents documents necessaris per a la total comprensió del document. En els diversos capítols d'aquesta memòria hi ha referències a aquests enllaços. També es poden trobar els llibres que han estat necessaris per a la realització del projecte.

A1.1 Enllaços

1. El document oficial sobre el protocol HTTP Live Streaming. Aquest enllaç fa referència a la vuitena versió del document de descripció del protocol. Es tracta, com s'ha descrit en el segon capítol, d'un *Internet-Draft*, i s'ha fet servir per la realització de tot el projecte.
   http://tools.ietf.org/html/draft-pantos-http-live-streaming-08

2. Apple posa a l'abast del programador una serie de documentació i recursos sobre el protocol HTTP Live Streaming. És en aquest enllaç on es poden trobar exemples i directius de com treballar amb aquest protocol.
   https://developer.apple.com/resources/http-streaming/

3. A Cookbook for Using View-Controller User the Model-Interface Paradigm in Smalltalk-80. Escrit per Gleen E. Krasner i Stephen T. Pope.
   http://www.ics.uci.edu/~redmiles/ics227-SQ04/papers/KrasnerPope88.pdf

4. Hi ha un programador que ha desenvolupat les eines de servidor per a preparar els continguts per a realitzar HTTP LIve Streaming en entorns Linux. En aquest enllaç es pot veure l'article en concret i els enllaços als recursos necessaris.
   http://blog.kyri0s.org/post/271121944/deploying-apples-http-live-streaming-in-a-gnu-linux

5. En el següent enllaç es pot veure un article sobre el *streaming* adaptatiu en Internet.
   http://www.javierrodriguez.com.es/blog/2011/10/24/streaming-adaptativo-en-internet-2/

6. Una de les parts més complicades de programar ha estat el reproductor didàctic. Per a poder utilitzar els recursos del *framework* de Cocoa per a la reproducció de vídeo s'ha hagut de seguir el següent enllaç.
   https://developer.apple.com/library/ios/#documentation/AudioVideo/Conceptual/AVFoundationPG/

7. La solució al problema de la reproducció d'arxius .TS directament amb els components del *framework* de Cocoa s'ha trpbat en el següent enllaç:
   http://stackoverflow.com/questions/5642248/can-avfoundation-be-coerced-into-playing-a-local-ts-file/10869827#10869827

8. El servidor fet servir en l'aplicació ha estat aconseguit des del següent enllaç:
   https://github.com/robbiehanson/CocoaHTTPServer

9. Per a la realització de les gràfiques s'ha fet servir el següent projecte:
   http://core-plot.googlecode.com/hg/documentation/html/iOS/index.html

10. Una gran quantitat de solucions als problemes que s'han anat trobant al desenvolupar l'aplicació s'han trobat en la següent adreça:
    http://stackoverflow.com

A1.2 Bibliografia

La bilbiografia utilitzada en el desenvolupament d'aquest projecte es basa, únicament, en tres llibres orientats a la programació amb Objective-C i per a l'iPhone.

1. Programming in Objective-C (4th Edition) - Kochan, Stephen G
2. Beginning iOS 5 Development: Exploring the iOS SDK - Dave Mark, Jack Nutting i Jeff LaMarche
3. The iOS 5 Developer's Cookbook: Core Concepts and Essential Recipes for iOS Programmers (3rd Edition) (Developer's Library) - Erica Sadun

A2 Taula de reproductors i servidors compatibles

Les següents taula mostra els clients i el servidors compatibles amb el protocol HTTP Live Streaming. Les dades han estat aconseguides del article que es troba en el aquest enllaç:

http://en.wikipedia.org/wiki/HTTP_Live_Streaming

A2.1 Clients

Producte	Tipus	Versió	Empresa
HLS Streaming Player and SDK for Flash	VOD and Live	2.0	Onlinelib.de
HLS Streaming Player and SDK for Flash	VOD and Live	1.0	Yospace
HLS Stream Player SDK per Windows 8 & Windows Phone 8 Apps	VOD and Live	1.0	3ivx
Android	VOD and Live	3.0 Honeycomb	Google
VLC media player	VOD and Live	2.0[10]	VideoLAN
iOS (iPhone, iPod Touch)	VOD and Live	3.0+	Apple
iOS (iPad)	VOD and Live	3.2+	Apple
QuickTime Player	VOD and Live	10+	Apple
ffplay	VOD and Live		FFmpeg
MythTV	VOD and Live	0.26	MythTV
nangu.TV on Motorola STB	VOD and Live	2.0	nangu.TV
Roku Digital Video Player	?	Roku OS / SDK 2.6	Roku
DicePlayer (Android 2.2+)	?	Diceplayer 1.0+	INISOFT
MX Player (Android)	?	MX Player 1.6	
WebOS (HP Touchpad)	?	3.0.5	HP
LifeVibes Quickplayer (Android 2.2-4.0)	?	cinexplayer & other	NXP software
Helix Client SDK (Android 2.2+)	VOD and Live	1.x	RealNetworks
XBMC Media Center (cross-platform media player)	VOD and Live	12.0 Alpha 5 and later	XBMC Foundation

A2.2 Servidors

Producte	Tecnologia	Versió	Empresa	Notes
VLC		1.02		
Unicorn Once		SaaS	Unicorn Media	
IIS Media Services		4.0[3]	Microsoft	
Adobe Flash Media Server		4.05	Adobe	
MythTV		0.25	MythTV	
MACNETIX VOD-Server		3.0	MACNETIX	
Anevia ViaMotion Servers	Transcapsulation: from one input, several outputs (HLS, MS Smooth Streaming, ADS Flash, MPEG DASH)		Anevia	
nangu.TV Streamers	on-the-fly adaptation: content is stored once enabling several outputs (HLS, MS Smooth Streaming, ADS Flash, MPEG DASH)		nangu.TV	
TVersity Media Server		1.09	TVersity	Pro Edition only
Helix Universal Server		14.0+	RealNetworks	High performance
Wowza Media Server			Wowza Media Systems	
Flixwagon Platform Video Server			Flixwagon	
StreamCoder Live Encoder	Realtime video encoder (inputs : DVB/IP stream or video signal). Supports multi-bitrates and multi-languages		Ektacom	
VODOBOX Live Server	Apple HTTP Live Streaming (Compliant with iOS and Android)		Vodobox	

Producte	Tecnologia	Versió	Empresa	Notes
Apache HTTP Server			Apache Software Foundation	
Unreal Media Server		9.05	Unreal Streaming Technologies	Latency of live streams can be as low as 2.5 seconds over the Internet
lighttpd		1.4.x		

A3 Traça del reproductor didàctic

La següent traça ha estat generada per l'aplicació desenvolupada en aquest projecte. La reproducció s'ha realitzat amb el servidor en la mateixa xarxa Wifi, per aquest motiu les velocitats de descàrrega són elevades. La variació de l'ample de banda ha estat fent servir el simulador que inclou l'aplicació. Per últim, l'algoritme utilitzat ha estat el que té només en compte l'últim segment, "Only Current Segment"

En aquesta traça es pot veure la capacitat adaptativa del protocol i els passos que el codi realitza per escollir el següent segment a descarregar.

```
[03/09/12 19:52] Start the HLS process...
[03/09/12 19:52] Algorithm used to calculate the next stream: 'Only current segment'
[03/09/12 19:52] Choosing the next segment to download.
[03/09/12 19:52] The next segment 0 will be from stream 1: 265,129 Kbps
[03/09/12 19:52] Segment downloaded in 0,088 seconds (0,088 real seconds). Bandwidth: 31,385 Mbps
[03/09/12 19:52] Adding video to queue: http://127.0.0.1:12345/stream0.m3u
[03/09/12 19:52] Choosing the next segment to download.
[03/09/12 19:52] The current bandwidth is 31,385 Mbps
[03/09/12 19:52] Time to spend: 10 seconds.
[03/09/12 19:52] The next segment 1 will be from stream 5: 7,259 Mbps
[03/09/12 19:52] Segment downloaded in 21,722 seconds (4,549 real seconds). Bandwidth: 3,264 Mbps
[03/09/12 19:52] Finish reproducing segment 0
[03/09/12 19:52] Adding video to queue: http://127.0.0.1:12345/stream1.m3u
[03/09/12 19:52] Choosing the next segment to download.
[03/09/12 19:52] The current bandwidth is 3,264 Mbps
[03/09/12 19:52] Time to spend: 10 seconds.
[03/09/12 19:52] The next segment 2 will be from stream 3: 2,781 Mbps
[03/09/12 19:52] Segment downloaded in 9,018 seconds (1,799 real seconds). Bandwidth: 2,727 Mbps
[03/09/12 19:52] Adding video to queue: http://127.0.0.1:12345/stream2.m3u
[03/09/12 19:52] Choosing the next segment to download.
[03/09/12 19:52] The current bandwidth is 2,727 Mbps
[03/09/12 19:52] Time to spend: 10 seconds.
[03/09/12 19:52] The next segment 3 will be from stream 2: 970,009 Kbps
```

```
[03/09/12 19:52] Segment downloaded in 3,677 seconds (0,724 real seconds). Bandwidth: 2,686 Mbps
[03/09/12 19:53] Finish reproducing segment 1
[03/09/12 19:53] Adding video to queue: http://127.0.0.1:12345/stream3.m3u
[03/09/12 19:53] Choosing the next segment to download.
[03/09/12 19:53] The current bandwidth is 2,686 Mbps
[03/09/12 19:53] Time to spend: 10 seconds.
[03/09/12 19:53] The next segment 4 will be from stream 2: 970,009 Kbps
[03/09/12 19:53] Segment downloaded in 2,256 seconds (0,771 real seconds). Bandwidth: 4,209 Mbps
[03/09/12 19:53] Adding video to queue: http://127.0.0.1:12345/stream4.m3u
[03/09/12 19:53] Choosing the next segment to download.
[03/09/12 19:53] The current bandwidth is 4,209 Mbps
[03/09/12 19:53] Time to spend: 10 seconds.
[03/09/12 19:53] The next segment 5 will be from stream 4: 4,048 Mbps
[03/09/12 19:53] Segment downloaded in 5,71 seconds (2,557 real seconds). Bandwidth: 6,601 Mbps
[03/09/12 19:53] Finish reproducing segment 2
[03/09/12 19:53] Adding video to queue: http://127.0.0.1:12345/stream5.m3u
[03/09/12 19:53] Choosing the next segment to download.
[03/09/12 19:53] The current bandwidth is 6,601 Mbps
[03/09/12 19:53] Time to spend: 10 seconds.
[03/09/12 19:53] The next segment 6 will be from stream 4: 4,048 Mbps
[03/09/12 19:53] Segment downloaded in 6,511 seconds (2,685 real seconds). Bandwidth: 5,81 Mbps
[03/09/12 19:53] HLS Process stopped by the user
```