laSalle

UNIVERSITAT RAMON LLULL

Escola Tècnica Superior d'Enginyeria La Salle

Treball Final de Màster

Màster Universitari en Enginyeria de Xarxes i Telecomunicacions

Conversor de DVB-PCF a Adobe Flash Lite

Alumne
Ramon Martín de Pozuelo Genís

Professor Ponent
Ricard Aquilué

ACTA DE L'EXAMEN
DEL TREBALL FI DE CARRERA

Reunit el Tribunal qualificador en el dia de la data, l'alumne

D. Ramon Martín de Pozuelo Genís

va exposar el seu Treball de Fi de Carrera, el qual va tractar sobre el tema següent:

Conversor de DVB-PCF a Adobe Flash Lite

Acabada l'exposició i contestades per part de l'alumne les objeccions formulades pels Srs. membres del tribunal, aquest valorà l'esmentat Treball amb la qualificació de

Barcelona,

VOCAL DEL TRIBUNAL VOCAL DEL TRIBUNAL

PRESIDENT DEL TRIBUNAL

Abstract

i3Media és un gran projecte d'investigació seleccionat pel programa *Cénit* del Ministeri d'Indústria com a projecte tecnològic tractor i dirigit per a fomentar la cooperació públic-privada en I+D+i. El seu desenvolupament es centra en els continguts intel·ligents, la personalització i els processos automatitzats, amb l'objectiu de promoure una millor producció, gestió, distribució i explotació dels continguts Media. Dins d'aquest marc, el desenvolupament principal d'aquest projecte és la implementació d'un conversor de DVB-PCF (Portable Content Format) a Adobe Flash Lite. Aquesta aplicació permet la transformació de descripcions genèriques de serveis interactius en aplicacions lleugeres i portables, adequades per a descodificadors de televisió digital i altres dispositius mòbils, i facilitant així la utilització de l'estàndard DVB-PCF per a la distribució i intercanvi d'aplicacions interactives.

Abstract

i3Media es un gran proyecto de investigación seleccionado por el programa *Cénit* del Ministerio de Industria como proyecto tecnológico tractor y dirigido para fomentar la cooperación público-privada en I+D+i. Su desarrollo se centra en los contenidos inteligentes, la personalización y los procesos automatizados, con el objetivo de promover una mejor producción, gestión, distribución y explotación de los contenidos Media. Dentro de este marco, el desarrollo principal de este proyecto es la implementación de un conversor de DVB-PCF (Portable Content Format) a Adobe Flash Lite. Esta aplicación permite la transformación de descripciones genéricas de servicios interactivos en aplicaciones ligeras y portables aptas para descodificadores de televisión digital y otros dispositivos móviles, facilitando así la utilización del estándar DVB-PCF para la distribución y el intercambio de aplicaciones interactivas.

Abstract

i3Media is a large research project selected by *Cénit* program from Spanish Ministry of Industry as a forerunner technologic project and headed to encourage the public-private R&D cooperation. Its development is focused in intelligent contents, personalization and automatic processes, with the objective of promote a better media content production, management, distribution and exploitation. In this framework, the fundamental development of this project is the implementation of a converter from DVB-PCF (Portable Content Format) descriptions to Adobe Flash Lite applications. This application permits the translation from interactive services' generic descriptions to light-weight and portable applications for digital TV decoders or handheld devices, easing the use of DVB-PCF standard for the distribution and interchange of interactivity applications.

Resum

Defensat per un ampli consorci d'empreses espanyoles que representen tota la cadena de valor del sector audiovisual, i3Media pretén estar a la primera línea de la investigació en continguts audiovisuals i generar noves tecnologies que ofereixin oportunitats de desenvolupament als entorns de les companyies participants en el projecte.

Amb l'objectiu de promoure una millor gestió dels continguts i poder competir en l'àmbit del mercat audiovisual, i3Media proposa l'ús de continguts intel·ligents, la personalització i els processos automatitzats. El resultat d'aplicar aquestes tecnologies, suposaria un impacte molt gran en les empreses del sector Media (producció, postproducció, emmagatzemament, indexació, distribució, etc), així com a altres sectors econòmics com l'oci, hostaleria, etc.

Aquest projecte es situa dins del marc de i3Media, tenint com a objectiu principal el disseny i la creació d'una aplicació dedicada a la gestió i explotació de continguts audiovisuals.

L'aplicació consisteix en un conversor de *DVB-PCF (Portable Content Format)*, un format de dades dissenyat pel grup *Digital Video Broadcasting* per a la descripció genèrica de serveis de televisió interactiva, a la plataforma Adobe Flash Lite. Mitjançant aquesta aplicació, a partir d'una descripció PCF (presentada en un fitxer XML), es poden obtenir de forma automàtica aplicacions Flash Lite, lleugeres, portables i suportades per un gran nombre de navegadors. A més, els baixos requeriments que presenten les aplicacions en aquest format en quant a gestió de memòria i cost computacional, fan que sigui un format final especialment apte i àmpliament suportat per descodificadors de televisió digital actuals i altres dispositius mòbils.

El desenvolupament d'aquesta aplicació s'ha realitzat dins del Departament de Tecnologies Media (DTM) associat a l'escola universitària d'Enginyeria La Salle. Personalment, voldria agrair el suport donat per a la realització del projecte dins d'aquest departament, especialment al cap del DTM, Gabriel Fernández, al cap del projecte, Francesc Enrich, al responsable científic, Francesc Pinyol, i als altres integrants del departament que han participat durant alguna o altra etapa del desenvolupament del conversor, David Cacenabes, David Bassols i Susagna Tapias.

Índex

1. INTRODUCCIÓ 1

2. PROJECTE I3MEDIA 5

2.1 INTRODUCCIÓ 5
2.2 CONVERSOR DVB-PCF A ADOBE FLASH LITE 6
2.2.1 Introducció 6

3. FONAMENTS TEÒRICS 7

3.1 XML 7
3.1.1 Introducció 7
3.1.2 Well-Formed Document 8
3.1.3 Valid Document 8
3.1.4 XML Schema 9
3.1.5 XML Schema Definition (XSD) 10
3.2 DVB-PCF 10
3.2.1 Introducció 10
3.2.2 Característiques 11
3.2.3 Arquitectura 12
3.2.4 Serveis interactius en DVB-PCF 12
3.2.5 Estructura d'una descripció PCF 14
3.2.6 Scoping Rules 16
3.2.7 Llenguatge d'accions 17
3.2.8 Avantatges 19
3.2.9 Inconvenients 19
3.2.10 Situació actual 19
3.3 ADOBE FLASH 20
3.3.1 Introducció 20
3.3.2 ActionScript 21
3.3.3 Adobe Flash Lite 26

4. ENTORN DE TREBALL 33

4.1 INTRODUCCIÓ 33
4.2 ECLIPSE 33
4.3 OPEN SOURCE FLASH 34
4.3.1 Introducció 34
4.3.2 MTASC 35
4.3.3 SWFMILL 43
4.3.4 AS Development Tools (ASDT) 46
4.3.5 ANT 49

5. IMPLEMENTACIÓ PRÀCTICA 55

5.1 DISSENY 55

5.1.1 Intèrpret i conversor *55*
5.1.2 Diagrama de paquets *56*
5.1.3 Diagrama de classes *59*
5.1.4 Eines de desenvolupament *64*
5.1.5 Preparació de l'entorn *64*
5.1.6 Implementació *65*
5.1.7 Utilització *72*
5.1.8 Resum del procés de conversió *73*

6. RESULTATS **75**

7. CONCLUSIONS I LÍNIES DE FUTUR **81**

8. BIBLIOGRAFIA **83**

Taula de Figures

Figura 1. Principals participants del consorci i3Media 5
Figura 2. Logo de l'estàndard DVB-PCF 10
Figura 3. Aplicacions descrites en PCF poden ser transcodificades a diferents plataformes de TVI (Font:[1]) 11
Figura 4. Exemple de Servei PCF 13
Figura 5. Arbre d'ítems d'una estructura PCF 14
Figura 6. Especificació en format XSD d'un ítem Service 15
Figura 7. Especificació en format XSD d'un ítem Scene 15
Figura 8. Especificació en format XSD d'un ítem Component 16
Figura 9. Especificació en format XSD d'un ítem Collection 16
Figura 10. Exemple d'arbre de jerarquia de components a DVB-PCF 17
Figura 11. Exemple d'ús de PCF Action Language 18
Figura 12. Exemple de drecera per a la navegació entre escenes 18
Figura 13. Logo d'Adobe Flash 20
Figura 14. Exemple de definició de variables en ActionScript 2.0 23
Figura 15. Exemple de creació d'un quadre de text "Hello World" en AS 2.0 25
Figura 16. Exemple de classe "Hello World en AS 2.0" 25
Figura 17. Adobe Flash Lite sobre múltiples dispositius de baixes capacitats 26
Figura 18. Versions de Flash Lite 27
Figura 19. Tendències d'utilització d'Adobe Flash Lite [25] 28
Figura 20. Taula comparativa de formats gràfics suportats 30
Figura 21. Taula comparativa de formats multimèdia suportats 30
Figura 22. Taula comparativa d'accessibilitat del dispositiu 31
Figura 23. Esquema de l'entorn de desenvolupament 33

Figura 24. Entorn de desenvolupament: Eclipse v.3.2.1 34
Figura 25. Comanda de compilació mitjançant MTASC 35
Figura 26. Comanda de compilació amb MTASC utilitzant classpath 36
Figura 27. Exemple "Hello World" d'ActionScript 2.0 utilitzant MTASC 38
Figura 28. Comanda de compilació de l'exemple 38
Figura 29. Exemple d'ús de l'argument "classpath" 39
Figura 30. Exemple d'ús de l'argument "pack" 39
Figura 31. Exemple d'ús de variables compilable amb MMC 40
Figura 32. Exemple d'ús de sobre-escriptura compilable amb MMC 40
Figura 33. Exemple d'ús correcte de variables locals per a compilar amb MTASC 40
Figura 34. Exemple de funció local compilable amb MMC 41
Figura 35. Exemple de funció local compilable amb MTASC 41
Figura 36. Exemple de la personalització de "trace" amb MTASC 41
Figura 37. Paràmetres extra afegits per MTASC en la personalització de traces 41
Figura 38. Exemple de definició d'array amb tipus definit 42
Figura 39. Definició de variable local marcant el tipus 42
Figura 40. Definició de variable local obviant el tipus 43
Figura 41. Exemple de fitxer XML per a la creació d'un SWF simple 44
Figura 42. Comanda de creació d'un SWF simple 44
Figura 43. Exemple de llibreria de recursos amb SWFmill 45
Figura 44. Comanda d'importació de fonts 45
Figura 45. Exemple d'importació de llibreria externa 45
Figura 46. Exemple complet d'ús de SWFmill 46
Figura 47. Edició de les preferències d'ASDT 47
Figura 48. Exemple de classe HelloWorld 48
Figura 49. Estructura de directoris d'una distribució ANT 49
Figura 50. Configuració de les variables d'entorn d'ANT a l'entorn Windows i OS/2 50
Figura 51. Configuració de les variables d'entorn d'ANT a l'entorn Unix (bash) 50
Figura 52. Configuració de les variables d'entorn d'ANT a l'entorn Unix (csh) 50
Figura 53. Comprovació de la configuració d'Ant 50
Figura 54. Comanda ant d'un arxiu XML alternatiu 51
Figura 55. Execució ANT d'una tasca concreta 51
Figura 56. Exemple de fitxer build.xml 51
Figura 57. Fitxer d'exemple d'ús d'ANT amb MTASC i SWFmill 53
Figura 58. DVB-PCF to Flash Lite Converter Packages 57
Figura 59. Diagrama UML del model de dades Actionscript 60
Figura 60. Classes principals del paquet Load Manager 63
Figura 61. Inicialització de la classe Converter 65

Figura 62. Inicialització de la classe PCFParser i ImageLibrary 65
Figura 63. Execució de la compil·lació amb Ant des de codi Java 66
Figura 64. Inicialització de l'estructura d'emmagatzemament d'informació AS 2.0 66
Figura 65. Parseig d'un element Service 68
Figura 66. Funció de càrrega d'una imatge 69
Figura 67. Fragment de la funció addOnEvent 70
Figura 68. Fragment de la funció setSceneVisibleOnly 71
Figura 69. Fitxer PCF2SWFConverterTest.java 72
Figura 70. Diagrama del procés de conversió 73
Figura 71. Fitxers de sortida creats pel conversor 74
Figura 72. Exemple de XML creat pel conversor per compilar amb SWFMill 75
Figura 73. Exemple de fitxer AS 2.0 amb la inicialització de l'aplicació 76
Figura 74. Exemple de fitxer AS 2.0 d'una escena 78
Figura 75. Fragment DVB-PCF amb la aplicació de prova de futbol interactiu 79
Figura 76. Imatge de l'aplicació de prova de futbol interactiu 79
Figura 77. Imatge de l'aplicació de prova amb doble menú 80

Acrònims

API: Interfície de programació d'aplicacions

AS: ActionScript

CDTI: Centre pel Desenvolupament Tècnic Industrial

DVB-PCF: Digital Video Broadcasting – Portable Content Format

IDE: Integrated Development Environment (Entorn de desenvolupament integrat)

iTV: Interactive Television

J2EE: Java Platform, Enterprise Edition

UML: Unified Modelling Language (Llenguatge de modelatge unificat)

URI: Uniform Resource Identifier

URL: Uniform Resource Locator

XML: Extensible Markup Language (Llenguatge de marques extensible)

XSD: XML Schema Definition (Definició de l'esquema XML)

1. INTRODUCCIÓ

L'increment actual de la televisió digital i les aplicacions interactives per a dispositius que no són ordinadors no té un estàndard de consens que guiï com s'ha de realitzar el desenvolupament de les aplicacions per a descodificadors de televisió, dispositius mòbils, etc.

El marc actual dels mitjans de distribució i plataformes on estan desenvolupades aquestes aplicacions fan difícil la realització d'una única aplicació que sigui compatible en múltiples plataformes. La diversitat de plataformes finals limiten la seva reutilització en altres plataformes degut a restriccions tant a nivell físic, operacional com comercial [1] [2].

Aquests fets ens porten a una situació on cadascú desenvolupa les aplicacions per a la seva plataforma específica, i on és molt difícil crear un mercat obert al voltant dels serveis interactius de televisió (*iTV services*).

El mercat dels serveis audiovisuals per televisió és un mercat que fins ara està centrat en resoldre les necessitats de plataformes en particular, tenint en compte només petits sectors de tot el mercat. Això resulta en un mercat vertical molt poc especialitzat i amb una capacitat de generar recursos molt més petita en relació a l'esforç necessari per la indústria de desenvolupament.

Observant aquest panorama, al 2006 emergeix de la mà de *Digital Video Broadcasting* (DVB) l'estàndard DVB-PCF (*Portable Content Format*) [3]. Aquest estàndard està centrat en la descripció de serveis interactius de forma que es descriu l'experiència visual que es pretén mostrar a l'usuari. Aquesta visió permet descripcions molt genèriques i totalment independents de la plataforma final en la qual és mostrada, i té com a objectiu proveir la indústria d'un format que permeti el lliure intercanvi d'aplicacions interactives entre les empreses agents (creadors, distribuïdors, emissors, etc.) i l'increment d'interoperabilitat entre les diferents eines de desenvolupament, plataformes finals i xarxes de radiodifusió d'aquests serveis.

D'aquesta forma, l'ús d'un estàndard comú per a la distribució de serveis iTV, permetria transformar aquest mercat vertical i poc especialitzat en un mercat horitzontal on un únic desenvolupament permetria el seu desplegament a un sector molt més ampli de mercat, ampliant el *target* i el benefici dels desenvolupadors d'aplicacions. A més, com a resultat d'això, l'indústria al voltant dels serveis iTV podria esdevenir molt més especialitzada i productiva.

De totes formes, aquest estàndard no ofereix una solució final per a tota la cadena de distribució, producció i emissió d'aquests continguts interactius. Utilitzant DVB-PCF, els creadors de continguts entreguen les aplicacions en format PCF als operadors de xarxa o *broadcasters,* els quals requereixen d'un element traductor final per a convertir la descripció PCF en una aplicació final apta per a la plataforma de radiodifusió que utilitzi. Així doncs, PCF s'ha d'entendre com un format només de *business-to-business* (B2B) i no en la oferta final a l'usuari, el qual no hauria de rebre directament aquest format en el seu descodificador de televisió digital.

DVB-PCF intenta d'aquesta obrir un mercat que fins ara ha estat completament diversificat i que ha estat difícil d'explotar eficientment. Però, és clar que aquest format necessita d'elements finals que adaptin aquestes descripcions genèriques a les diferents plataformes finals, i així poder oferir una solució òptima i homogeneïtzada d'extrem a extrem en el mercat dels serveis interactius de televisió digital.

En aquest punt, és on l'aparició d'Adobe Flash Lite [4] i el seu suport per part de gran part dels dispositius actuals de TV digital resulta rellevant. Adobe Flash Lite es presenta com un eina potent que facilita el camí per a la visió d'experiències més enriquidores a les televisions dels usuaris mitjançant aplicacions en Flash que poden funcionar sobre descodificadors, TV digitals, aparells mòbils i altres dispositius de baixes prestacions de memòria i cost computacional [5][6].

Tenint en compte aquests dos punts principals, l'objectiu principal que es presenta en aquest projecte és dur a terme una aplicació destinada a l'homogeneïtzació dels serveis interactius per a televisió digital, de forma que la seva distribució i el seu emmagatzemament sigui més còmode, eficient i ràpid.

A més, cal destacar també, que aquest projecte està realitzat sota el marc del projecte estatal i3Media d'R+D+i, enfocat en l'automatització de processos i la creació de contingut més intel·ligent per a la millora de la creació, producció, distribució i consum de contingut audiovisual.

Per tant, amb aquests objectius completament alineats, aquest projecte es centra en el desenvolupament d'un conversor de format DVB-PCF a Adobe Flash Lite. Aquesta aplicació, que intenta proporcionar facilitats en l'intercanvi de continguts als integrants del projecte i3Media, permetrà la transformació de descripcions genèriques de serveis interactius en aplicacions lleugeres i portables, adequades per a descodificadors de televisió digital i altres dispositius mòbils, i facilitant així la utilització de l'estàndard DVB-PCF per a la distribució i intercanvi d'aplicacions interactives.

A més, aquest projecte ha permès validar la viabilitat d'utilitzar el format DVB-PCF com a format de descripció d'aplicacions, ja que actualment hi ha molt poques referències d'implementacions reals que utilitzin aquest estàndard.

A més, es pretén que la implementació es faci d'una forma que pugui facilitar també la futura conversió de PCF a altres tipus de llenguatges útils en el sector, com MHP.

El segon capítol explicarà amb més profunditat la intenció i l'estat de l'art a nivell general del projecte i3Media, i després més en concret de la implementació d'aquest conversor, per entendre el marc en què s'ha realitzat la implementació d'aquestes aplicacions.

En el següent apartat es procedeix a explicar tots els fonaments teòrics del projecte i es dóna una visió global de les eines de desenvolupament i els entorns utilitzats. Aquests conceptes esdevenen els coneixements bàsics per a la comprensió i posterior creació del conversor.

Amb anterioritat a la implementació, es realitza un estudi de les possibles eines a utilitzar en la creació del conversor, on es detalla el perquè de l'elecció d'aquest format i les eines de desenvolupament utilitzades.

Un cop escollit l'entorn de treball i les tecnologies que s'empraran per a la realització de les aplicació, són presentats els dissenys i desenvolupaments dels diferents mòduls, i es dóna una completa revisió del seu desenvolupament i també del seu ús.

Per acabar es mostren els resultats obtinguts i s'extreuen conclusions de l'estat actual del projecte i les línies de futur existents, així com es contemplen possibles evolucions de l'aplicació a curt i llarg termini.

2. PROJECTE I3MEDIA

2.1 INTRODUCCIÓ

I3Media -Tecnologies per a la Creació i Gestió Automatitzada de Continguts Audiovisuals Intel·ligents- , és un projecte Cénit atorgat pel Ministeri d'Indústria a través del CDTI, i portat a terme per un consorci de 12 empreses, liderat per MEDIAPRO.

Figura 1. *Principals participants del consorci i3Media*

A més del consorci, i3Media compta amb la participació de 19 grups d'investigació adscrits a 10 organismes d'investigació: 2 centres tecnològics, 2 centres d'investigació, i 6 universitats, entre les quals es troba l'escola universitària d'enginyeria La Salle (Universitat Ramon Llull) i el Departament de Tecnologies Media, on s'ha realitzat aquest projecte.

El pressupost del projecte és de 30 milions d'euros i el calendari d'execució és de quatre anys, començant el 2007 i finalitzant el 2010. Els seus principals objectius són la generació de noves tecnologies que ofereixin oportunitats de desenvolupament a les companyies que formen el consorci, centrant-se en els continguts intel·ligents, la personalització i els processos automatitzats. Entre aquestes fites que es volen aconseguir, destaquen algunes especialment innovadores com el reconeixement i virtualització d'objectes, sistemes de so immersiu, o la aplicació de tecnologia emocional.

Com a principals resultats s'espera modificar l'actual model de negoci dels medis de comunicació, basat en la publicitat, subscripció i/o subvenció, permetent una comunicació més efectiva que permetrà als medis treure un major benefici dels seus continguts i als anunciants comunicar-se d'una forma més pròxima i controlada.

Es pretén aconseguir una evolució de la manera de produir i consumir televisió, oferint al consumidor continguts personalitzats en funció de les seves inquietuds i s'estructuraran en varies capes d'informació per les quals l'usuari podrà moure's de forma similar a com

ho fa per Internet. D'aquesta manera es podria trencar aquest flux de distribució massiva i unidireccional de continguts per a oferir una molt més bidireccional, personalitzada i sota demanda, implicant, principalment, canvis en els procediments de producció

Al final del primer any de desplegament es va celebrar una sessió de treball amb els grups que intervenen en i3Media i es van definir els objectius per als pròxims anys. Aquests es van orientar en la creació d'actius experimentals que permetran observar, escoltar, llegir, i en molts casos gaudir dels resultats.

2.2 CONVERSOR DVB-PCF A ADOBE FLASH LITE

2.2.1 Introducció

Actualment, hi ha una gran varietat d'eines de desenvolupament d'aplicacions interactives i cada plataforma de radiodifusió utilitza la que més s'adequa a les seves característiques. Això crea un buit que impedeix una transacció oberta dels serveis d'una forma estàndard entre els diferents agents. Tot i així, és imperatiu que les plataformes dels *broadcasters* no pateixin un canvi radical per la homogeneïtzació d'aquests processos, ja que això suposaria un desgast econòmic per aquestes empreses.

La utilització de DVB-PCF intenta pal·liar aquest buit sense necessitat de canvis de plataforma. D'aquesta manera, aconsegueix millorar els models de negoci ja existents entre desenvolupadors d'aplicacions i *broadcasters* i ofereix possibilitats de l'aparició de nous models sense un gran desemborsament per ambdues parts.

Però, la utilització de PCF requereix d'un element clau per a aconseguir que això sigui una solució extrem a extrem. És necessari la creació de traductors o adaptadors entre les descripcions DVB-PCF i les diferents plataformes de radiodifusió. Aquest treball estudia les funcions que han d'acomplir aquests elements i en crea un d'específic per adaptar-ho a Adobe Flash Lite 3.1. L'elecció d'aquest software ve donada (a més de restriccions del projecte i3Media) pel seu ampli suport en descodificadors actuals de televisió digital, així com l'eficiència i els baixos requeriments necessaris en les plataformes resultants per a córrer adequadament les aplicacions resultants.

En aquest context, donat un fitxer XML amb una descripció PCF d'entrada, el conversor resultant intenta interpretar-la i renderitzar-la en Flash Lite amb la màxima exactitud possible, perquè aquesta pugui ser visualitzada després en un navegador que suporti Flash Lite Player. Cal dir, però, que el conversor està dissenyat per la conversió en una única direcció, no podent realitzar el pas invers (crear una descripció PCF d'una aplicació Flash Lite ja creada). Es preveu que cada servei hauria de ser dissenyat i creat i intercanviat entre els agents intermedis mitjançant mitjançant DVB-PCF, aplicant el conversor únicament en les plataformes finals, just abans de la distribució cap als usuaris.

3. FONAMENTS TEÒRICS

3.1 XML

3.1.1 Introducció

Abans d'entrar en detall amb l'estàndard DVB-PCF, cal esmentar que aquest presenta unes descripcions de serveis interactius que venen presentades formatades en arxius XML. És per aquest motiu interessant fer un breu repàs d'XML abans d'explicar amb més detall en què consisteix el format de DVB.

XML, sigles angleses d'*Extensible Markup Language* (llenguatge de marques extensible), és un metallenguatge extensible d'etiquetes desenvolupat pel *World Wide Web Consortium* (W3C) [7]. Es basa en un sistema per a l'organització i etiquetat de documents, i permet definir la gramàtica de llenguatges específics, fet pel qual no es pot dir que sigui realment un llenguatge específic, sinó una forma de definir llenguatges.

El seu origen està en un llenguatge inventat per IBM durant els anys setanta i anomenat GML. Posteriorment, aquest va derivar en SGML [8], un llenguatge normalitzat per la ISO al 1.986, i definit a la norma ISO 8879. Al 1.996, i de la mà de Sun Microsystems, sorgeix el desenvolupament d'XML com a intenció de simplificar i adaptar SGML, que era molt efectiu, però en ocasions massa complex. El W3C també va contribuir activament en la creació i desenvolupament d'aquest estàndard, arribant l'any 1.998 a XML 1.0 [9], que acomplia les fites proposades pel grup de treball d'usabilitat en quant a compatibilitat amb SGML, llegibilitat, formalitat, facilitat d'autorització i facilitar el desenvolupament de processar software.

Després, amb el temps XML 1.0 ha anat patint petites variacions i ja es troba per la seva quarta edició. Per altra banda, es va crear un nou format, XML 1.1 [10], que conté caràcters que intenten fer XML més senzill d'usar en certs casos, principalment habilitant l'ús de caràcters de fi de línia i l'ús d'scripts i caràcters absents d'Unicode 2.0 [11]. Aquest estàndard està menys estès i només es recomana el seu ús per a qui necessiti característiques concretes.

A nivells de definició per a un correcte document XML, tenim:

- Well-formed Document.
- Valid Document.

3.1.2 Well-Formed Document

Un document well-formed és aquell que es cenyeix a totes les normes de sintaxis dels XML. Un document que no és well-formed no es considera com un XML. Les normes bàsiques que ha de complir un Well-Formed Document per assolir aquest nivell de validesa són:

- Els elements no buits han d'estar delimitats per una etiqueta d'inici i una final.
- Els elements buits poden ser indicats amb una etiqueta que es tanqui a si mateixa.
- Tots els valors d'atribut s'han de posar entre cometes, ja siguin simples o dobles.
- Les etiquetes poden ser niuades però no superposar-se. Cada element no pertanyent a l'arrel directament ha d'estar contingut en un altre element.
- El document ha de tenir en compte la declaració de la codificació de caràcters. Aquesta codificació es pot declarar externament o internament. Si no hi ha cap declaració s'assumeix una codificació Unicode.

També s'ha de tenir en compte que es distingeix entre majúscules i minúscules en els noms dels elements (*case-sensitive*) i es recomana que aquests estiguin posats per a una correcta comprensió sense necessitat de documentació addicional.

3.1.3 Valid Document

Un document *valid* es cenyeix, a més de a totes les regles de sintaxis dels XML, a algunes regles addicionals de sintaxis. Aquestes normes són definides per l'usuari o incloses en un XML schema.

La sintaxis general de cada tipus de llenguatge es rígida. Els documents s'han d'adaptar

a les normes generals de XML, assegurant-se de que tots els softwares que preparats per entendre XML puguin al menys llegir i entendre la informació que hi ha a dins seu.

L'schema el que fa és afegir unes certes limitacions a les regles de sintaxis. Restringeixen típicament els noms dels elements i els atributs, així com les jerarquies admissibles de la contenció d'aquests Les limitacions dels schema també poden incloure assignacions de tipus de dades que afecten a com la informació és processada. Un document XML que es conforma amb un schema particular, i és també well-formed, es diu que és vàlid.

Alguns dels schema més populars actualment són XML Schema Definition (XSD) [12] i DTD [13]. Als següents punts s'explica amb una mica més de detall els XML Schema, centrant-se especialment en XSD, ja que és el tipus de validador que es va utilitzar en les etapes inicials dels projecte per a definir les descripcions PCF vàlides.

3.1.4 XML Schema

3.1.4.1 Introducció

Generalment, un XML Schema és una descripció d'un tipus de document XML. Més enllà de les limitacions sintàctiques bàsiques imposades per XML, l'schema està expressat típicament en funció de les limitacions den l'estructura i el contingut de documents d'aquest tipus. En definitiva, són una sèrie de normes les quals un document XML ha de complir per tal de ser considerat un document *valid* segons aquest schema.

Un XML schema proporciona una vista del tipus de document a un nivel relativament alt d'abstracció. Hi ha llenguatges desenvolupats especialment per expressar XML schemas. El llenguatge Document Type Definition (DTD), que és natiu de la especificació de XML, és un llenguatge schema de capacitat relativa limitada però també té altres usos en XML a part de l'expressió d'schemas. Altres dos idiomes molt populars i més expressius d'schema d'XML són XML Schema (XSD) i RELAX NG.

El mecanisme per associar un document d'XML amb un schema varia segons el llenguatge usat. L'associació es pot aconseguir a través del mateix document XML o per alguns medis externs.

3.1.4.2 Validació

El procés de comprovació de si un document XML és conforma a un schema determinat s'anomena validació, que va a part del concepte sintàctic de well-formed dels documents XML. Tots els documents XML han de ser well-formed, però no es un requeriment que el document sigui vàlid a menys que l'analitzador sintàctic estigui validant, cas en el que el document també caldrà que estigui conformat amb el seu schema associat. Els validadors dels analitzadors sintàctics de DTD són els més comuns, però també es solen suportar els de W3C XML Schema o RELAX NG. Les validacions de XML Schema poden fer-se efectives amb analitzadors sintàctics especialitzats com JAXB o SAX.

Els documents es consideraran vàlids només si satisfan els requeriments de l'schema que tenen associat. Aquests requeriments típicament inclouen limitacions tals com:

- Els elements i atributs inclosos han de complir la seva estructura permesa.
- L'estructura està especificada per una expressió sintàctica regular.
- Interpretació concreta de les dades de caràcter.

Després de la validació basada en XML Schema, és possible expressar l'estructura d'un document XML i el seu contingut en termes del model de dades que estava implícit durant la validació. El model XML Schema de dades inclou: el vocabulari, en forma d'element i

nom d'atributs; el model de contingut, com a relacions i estructura; i el tipus de dades. Aquesta col·lecció d'informació s'anomena Post-Schema-Validation Infoset (PSVI).

Convertir un document XML en un objecte conegut de tipus de dades pot ser beneficiós en algunes parts del disseny de software, però en aquest sentit s'ha criticat que d'aquesta forma és limita la llibertat que donava XML. Els tipus de dades en un XML Schema estan classificats en dos tipus, els tipus de dades predeterminats i els definits per l'usuari. Hi ha 19 tipus de dades predefinits de forma primitiva i 25 derivats especificats. Representen tipus de dades comunes com poden ser cadenes, enters, decimals o la data.

3.1.5 XML Schema Definition (XSD)

Els XML Schema Definition (XSD), descrits per la W3C com els successors dels DTDs, donen molta més potència per a descriure llenguatges XML. Tenen en compte limitacions més detallades de l'estructura lògica de document XML i han de ser processats en un entorn més robust de validació. Un XSD defineix un tipus de document XML en termes de limitacions sobre quins elements i atributs poden aparèixer, la seva relació l'un amb l'altre, quins tipus de dades poden estar en ells i amb altres coses. Usen un format basat en XML que fa possible utilitzar XMLs ordinaris per ajudar a processar-los, tot i que les implementacions de XSD requereixen molt més que simplement l'habilitat de poder llegir XML.

Els fitxers XML Schema Definition usualment tenen l'extensió *.xsd*, tot i que encara no tenen un Internet Media Type únic registrat per XSD, així que generalment s'usa *application/xml* o *text/xml*, segons RFC 3023.

3.2 DVB-PCF

3.2.1 Introducció

El format PCF (*Portable Content Format*) [3] és un estàndard creat pel grup *Digital Video Broadcasting* amb l'objectiu de descriure serveis de televisió interactiva. Defineix un format de dades per a l'intercanvi de contingut interactiu entre empreses i permetre el desenvolupament a múltiples plataformes destí amb el mínim esforç de reedició. D'aquesta forma, s'intenta que hi hagi un increment en la interoperabilitat de les diferents eines d'autoria i xarxes de difusió.

Figura 2. *Logo de l'estàndard DVB-PCF*

PCF realitza una descripció del servei des del punt de vista de l'espectador, sense tenir en compte com s'ha implementat. Aquesta manera de plantejar l'estàndard permet la major portabilitat possible, ja que no es limita a elements molt concrets.

La descripció PCF no intenta ser una descripció usada per enviar-se a través de la xarxa de televisió actual, sinó més aviat com un sistema estàndard d'enviament de dades que siguin convertides posteriorment de forma específica per a cada plataforma.

Com a conseqüència d'això, alguns serveis interactius poden ser difícils d'emetre amb exactament la mateixa experiència per l'espectador, sense cap dependència de la plataforma. Malgrat aquests possibles problemes, generalment és possible oferir experiències molt similars al portar el PCF a diferents plataformes.

Amb PCF, DVB proveeix a la indústria de la televisió interactiva amb una eina que pot facilitar la coexistència i migració entre MHP [14], EBIF [15], Flash Lite i altres llenguatges, donant les eines per a l'intercanvi de descripcions de serveis a través de múltiples plataformes. Això resulta en un molt menor cost de desplegament dels serveis a través de múltiples plataformes i incrementant directament l'abast i la rellevància de la televisió interactiva.

3.2.2 Característiques

PCF és una descripció independent de la plataforma, més centrada en el *què* hauria de ser el què pot experimentar l'usuari final que en el *com* arribar-hi. La transformació a una plataforma específica es realitzaria mitjançant transcodificadors a cada plataforma específica. Aquestes transformacions utilitzen les característiques particulars de cada plataforma per a realitzar l'experiència descrita en PCF.

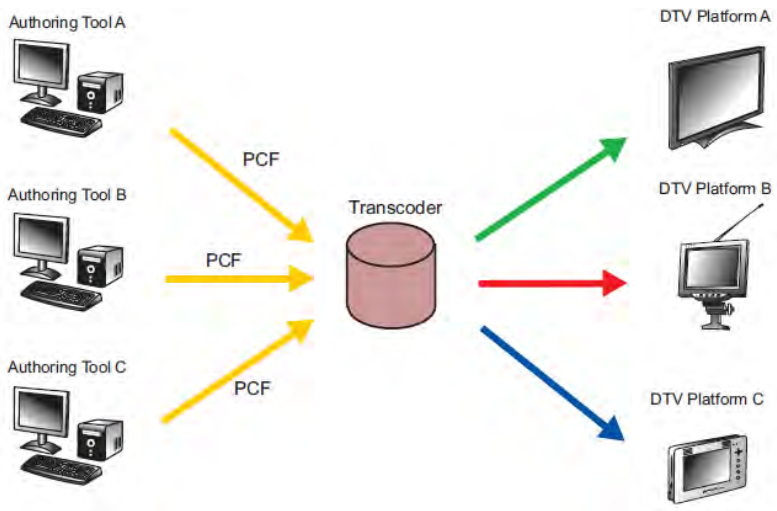


Figura 3. *Aplicacions descrites en PCF poden ser transcodificades a diferents plataformes de TVI (Font:[1])*

La descripció PCF no té perquè contenir tots els aspectes que s'han de descriure d'un servei en una única unitat física, com un fitxer. Per exemple, les descripcions poden ser distribuïdes arbitràriament mitjançant varis fitxers a Internet, usant referències representades mitjançant URIs.

L'estàndard està format per un model declaratiu d'alt nivell basat en estàndards ja existents com la sintaxi XML, els tipus MIME [16] i UML [17].

3.2.3 Arquitectura

En el format PCF descriu una arquitectura formada pels següents elements principals: tipus de dades, estructures de dades, identificació d'ítems i un model de referència. L'objectiu de definir l'arquitectura independentment d'altres aspectes de l'estàndard és la d'assegurar que totes les parts d'un servei PCF consten d'un marc de treball consistent i interoperable. Tot i així, es defineix una arquitectura únicament de format i no de sistema, deixant lliure l'entorn de desplegament d'un transcodificador PCF.

- Descriu completament els serveis interactius.
- Permet que aspectes de la descripció del servei puguin ser creats i modificats independentment uns dels altres.
- Té una sintaxi extensible per a suportar la creació de noves característiques.
- Expressa característiques a diferents nivells de granularitat.
- És pràctic de convertir a una màquina.
- És una descripció de servei no lligada a cap sistema, que parofita les característiques de tots els sistemes.
- Encapsula descripcions de capacitats addicionals.
- Suporta validació i verificació eficient del servei

PCF acompleix els requisits demanats de portabilitat entre diferents plataformes mitjançant un llenguatge de sintaxi declarativa d'alt nivell. Per permetre una descripció del servei el més eficient possible, i de forma que més convingui a l'autor, el format usa un mecanisme de referències. Això permet aparicionalment i flexibilitat, donant al transcodificador una gran llibertat de decisió, en quant a la manera de desplegar el servei de forma òptima a la plataforma de destí.

3.2.4 Serveis interactius en DVB-PCF

Les descripcions de servei PCF consisteixen en informació d'ítems, molts dels quals estan representats com components PCF. L'element arrel de la descripció és l'ítem *service*, el qual ha de contenir una o més ítems *scene*, i han d'especificar quina d'aquestes escenes

serà la primera que visualitzarà l'usuari. És important remarcar que en cap moment del cicle de vida del servei s'haurien de mostrar més d'una escena a la vegada i que totes les escenes que puguin ser renderitzades han d'estar dins de l'ítem *service* o en algun sub-contenidor d'aquest.

Els ítems *service* i *scene* deriven tots del mapa de tipus PCF, on cadascun d'aquests elements conté un grup d'altres ítems PCF, amb l'únic requisit que cada element tingui un nom únic i exclusiu. Així doncs, les descripcions PCF han de ser estructures jeràrquiques d'ítems PCF.

```
<Service name="hello_world">
     <URI name="firstScene" value="#hello_scene"/>
        <Scene name="hello_scene">
           <Component class="TextBox" name="hello_text">
             <Size name="size" value="100 40"/>
             <String name="content" value="Hello, World!"/>
           </Component>
           <OnEvent name="exit">
             <Trigger eventtype="KeyEvent">
             <UserKey name="key" value="VK_PREV"/>
             </Trigger>
             <ExitAction/>
            </OnEvent>
       </Scene>
</Service>
```

Figura 4. *Exemple de Servei PCF*

Aquest exemple il·lustra els components centrals d'una descripció de servei PCF:

- Components: Els blocs de dades amb què es construeix la descripció.
- Contingut: Administrat i presentat usant els components PCF.
- Comportament: El responsable que els events es generin en temps d'execució.

L'aparença visual de l'escena és definida com un conjunt de *components* que conté *scene*. En el cas de la figura anterior, es defineix un *TextBox* que s'utilitza per posar el missatge *Hello World*. PCF defineix una sèrie de components estàndard que poden ser inclosos per a definir el comportament i aparença de l'escena.

L'últim element que es pot veure a la figura, *OnEvent,* és un controlador d'event. Aquest tipus de components permeten afegir l'operació a efectuar segons accions que realitzi l'usuari. En aquest cas, l'event que conté és una única acció, *ExitAction*, que diria al servei que ha finalitzat.

3.2.5 Estructura d'una descripció PCF

Els ítems que formen l'estructura d'una descripció PCF d'un servei són *PCF container, component item, collection itel, scene item* i *service item.*

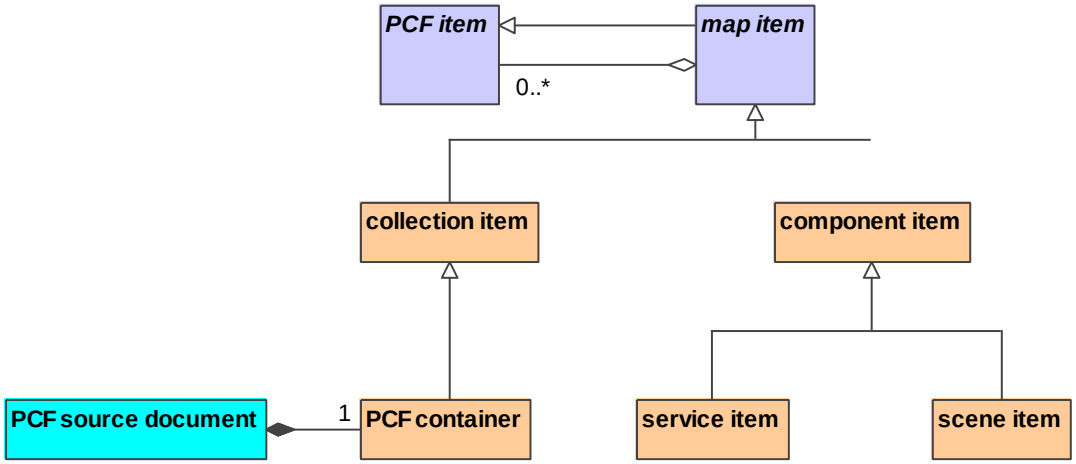


Figura 5. *Arbre d'ítems d'una estructura PCF*

3.2.5.1 PCF Container

Tota descripció PCF d'un servei ha de contenir un únic *PCF Container*, l'ítem de més alt nivell del document PCF i que proporciona el *root rendering context* (l'arrel del context de renderització del servei).

3.2.5.2 Service items

Representen la descripció completa d'un servei interactiu. També forma part dels *component item*, o ítems a visualitzar. Ha de contenir, directament o mitjançant referències tots els *scene items* (ítems d'escena) i tots els altres ítems PCF que siguin comuns a tots els *scene items*. Un *service item* no pot contenir cap altre *service item* dintre seu o a algun dels seus descendents.

La visió que ha de tenir un usuari en un moment determinat d'una sessió ha d'estar definida per una combinació de tots els component ítems (visualitzables) que estan al context de visualització del *service item* i tots els components que es troben a l'ítem d'escena activa.

Quan un servei és iniciat, el service item i tots els ítems associats (excepte els d'escena) hauran de ser activats. Tots aquests ítems mantindran el seu estat actiu durant tota la sessió del servei. Per contra, quan es surt del servei, tots els ítems descendents d'aquest *item service* hauran de ser desactivats.

```
<pcf:PCF xmlns:pcf="http://www.dvb.org/pcf/pcf">
      <pcf:Service name="interactive_Football">
            <pcf:Size value="640 480" name="referenceScreen" />
            <pcf:String value="1.0" name="pcfSpecVersion" />
            <pcf:URI value="" name="firstScene" />
            <pcf:Scene name="scene_Advert">
                  // Scene design
            </pcf:Scene>
      </pcf:Service>
</pcf:PCF>
```

Figura 6. *Especificació en format XSD d'un ítem Service*

3.2.5.3 Scene items

El propòsit d'aquests ítems és descriure una unitat espacial i temporalment coordinada de la descripció de l'experiència de l'espectador. També forma part del grup de *component items*, o ítems a visualitzar.

Només un *scene item* pot estar actiu al mateix moment i la navegació cap a una altra escena ha de causar la desactivació d'un *scene item* i l'activació d'un altre i tots els ítems que continguin dins dels seus contexts.

```
 <Scene name="Composers">
      <!-- Include referenced content within collection -->
      <Copy href="../MyContent"/>

      <TextBox name="Box1">
            <!-- Refer to included content item -->
            <String name="content" href="../Item1"/>
            <Size name="size" value="100 40"/>
      </TextBox>

 </Scene>
```

Figura 7. *Especificació en format XSD d'un ítem Scene*

3.2.5.4 Component items

Els *component items* proveeixen a la descripció d'una estructura d'instàncies a diferents tipus de components. Els sistemes PCF interpreten aquests components i renderitzar-los perquè l'usuari els visualitzi, fet pel que també poden ser coneguts com *renderable items*.

Aquests ítems poden estar actius o inactius i poden ser activats i desactivats per la descripció del servei actiu. Quan un està actiu, pot ser presentat i/o accessible a l'espectador, depenent de les regles definides a la descripció. Els *component items*, a part

de proveir una referència del context en que estan, també proveeixen d'un *rendering context* (context a visualitzar) per a tots els *PCF items* que conté.

```
<Component type="TextBox" name="hello_text">
      <String name="content" href="../mycontent/msg"/>
</Component>
```

Figura 8. *Especificació en format XSD d'un ítem Component*

3.2.5.5 Collection items

Collection items són grups d'ítems PCF, definides pels autors de serveis per a poder modular les descripcions i oferir una jerarquia de context que faciliti les referenciació. *Collection items* no han de ser ítems renderitzables, sinó usats simplement per a estructurar la informació i proveir d'una referència de context. Han d'estar presents al mapa de tipus de dades PCF.

```
<Collection name="news_style">
<Color name="textcolor" value="#DFDFDF"/>
      <String name="content">More news follows shortly ...</String>
      <Integer name="border-width" value="3"/>
</Collection>
```

Figura 9. *Especificació en format XSD d'un ítem Collection*

3.2.6 Scoping Rules

Les *PCF scoping rules* (regles d'abast) determinen la manera amb què uns components poden accedir i manipular les propietats que estan dins d'altres components en la jerarquia de components. Reflecteixen el rang de variables en temps d'execució i les propietats dels components. Les principals *scoping rules* definides són:

- Un *component item* hauria de poder accedir a les propietats d'ell mateix i a les de tots els *components items* descendents d'ell.
- A nivell de servei no s'hauria de tenir accés a les propietats de les escenes que en deriven, ni als components que pertanyen a aquestes. Però si té accés a totes les propietats dels components que no siguin *scene items* i descendents d'aquests.

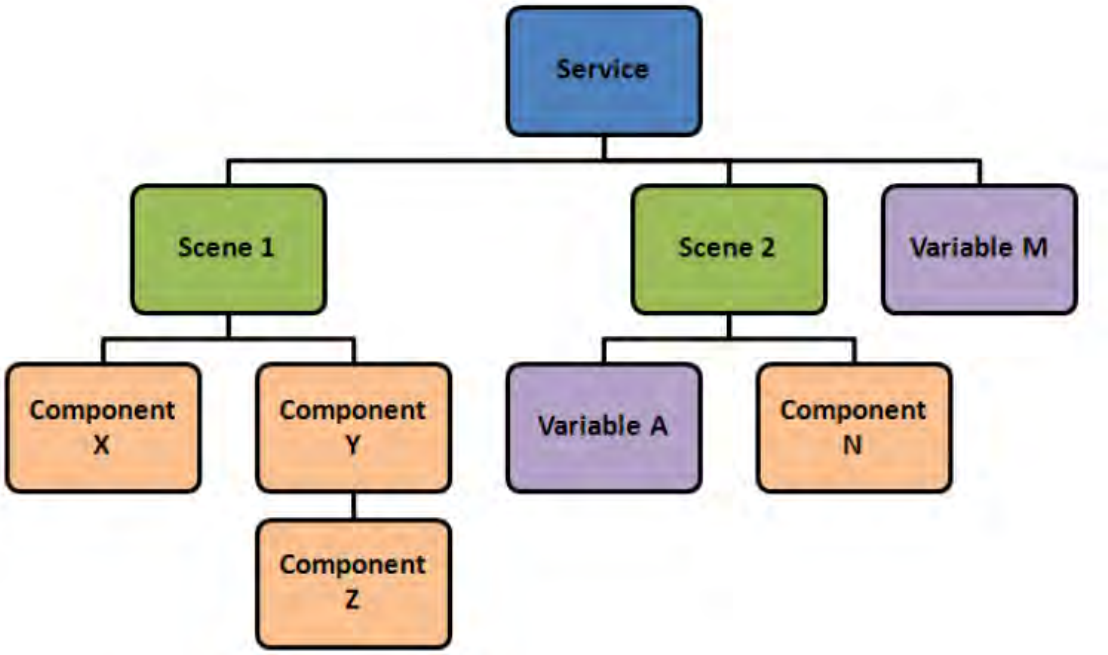


Figura 10. *Exemple d'arbre de jerarquia de components a DVB-PCF*

Per exemple, en la figura anterior, les regles estipularien que el component A, B i C serien visibles pel component S1, però no per *Service*. El component A no podria accedir a B o C, ni viceversa, però en canvi, B si podria accedir a les propietats de C, ja que és un component descendent d'ell.

3.2.7 Llenguatge d'accions

PCF inclou un llenguatge, *PCF Action Language*, que especifica les seqüències de les accions a executar quan es produeix un event. Aquestes accions poden ser produïdes de forma directa o indirectament per aquest event, com a resultat d'un estat de transició causat per aquest. Aquest llenguatge consisteix en accions, variables, condicions i controls, que poden ser combinats per formar *scripts* simples.

Les accions són essencialment comandes directes a components específics, incloent especialment els de servei i escena. A més de poder llegir i modificar propietats dels components, el llenguatge defineix accions per a realitzar transicions d'escena, fer operacions matemàtiques i lògiques simples, treballar amb cadenes de caràcters o executar comandes especifiques d'alguns components.

Les variables són utilitzades per a guardar i recuperar informació que és necessari mantenir durant l'execució d'una seqüència d'accions o entre accions successives. Alguns dels tipus possibles són: *boolean, color, component, currency, date, integer, position, size, string, time, timecode,* URI, etc.

Les condicions són utilitzades per a especificar que l'execució de la aplicació ha de seguir una seqüència depenent del compliment d'una determinada condició booleana, i permetent així especificar múltiples branques d'execució.

Els controls s'usen juntament amb les condicions per a aconseguir una major exactitud en l'especificació del flux d'accions que s'executaran. S'especifiquen comportaments de salt i bucles, com per exemple, definir que el salt cap a una escena de fi del joc quan el valor de l'enter que conté el nombre de vides del jugador arriba a zero. Tot i així, especificar a la perfecció algun tipus d'accions pot ser de difícil adaptació a algunes plataformes específiques, i *PCF Action Language* defineix dos nivells de funcionalitat

PCF també facilita un mecanisme perquè una acció pugui accedir a un component definit al component *Service* si és necessari, mitjançant l'ús de *parameter items*.

```
<Scene>
      <Collection name="variables">
            <IntegerVar name="counter">
                  <Integer name="value" value="1"/>
            </IntegerVar>
      </Collection>
      <OnEvent>
            <Trigger eventtype=”KeyEvent”>
                        <UserKey name=”key” value=” VK_ENTER”/>
            </Trigger>
            <ActionLanguage name="jumpCounter">
                  variables.counter.value += 10;
            </ActionLanguage>
      </OnEvent>
</Scene>
```

Figura 11. *Exemple d'ús de PCF Action Language*

I per facilitar algun tipus d'accions que s'utilitzen amb assiduïtat, el llenguatge també proporciona una sèrie de dreceres per a facilitar la generació d'aquestes accions.

```
<ActionLanguage name="jumpCounter">
      SceneNavigate(#../../../next_scene);
</ActionLanguage>
```

Figura 12. *Exemple de drecera per a la navegació entre escenes*

3.2.8 Avantatges

- Intercanvi simple de contingut interactiu, entre varis proveïdors i entre proveïdors i operadors.
- La habilitat que ofereix als proveïdors i operadors d'escollir quin és la eina òptima pel seu negoci en cada un dels punts de la cadena de producció de televisió interactiva.
- Desplegament del contingut interactiu més enllà dels mercats primaris.
- Reducció dels costos de producció per a contingut interactiu simulant nous models de negoci.

3.2.9 Inconvenients

- PCF és improbable que pugui ser usat per a totes les aplicacions en totes les plataformes.
- Limitacions fonamentals en una plataforma en particular poden fer difícil o impossible el suport d'un ampli ventall de serveis possibles a diferents plataformes.
- El suport de múltiples PCFs en una mateixa plataforma és possible, però requeriria un esforç extra d'implementació per part dels operadors.

3.2.10 Situació actual

La gran varietat actual de plataformes de televisió digital poden oferir una gran varietat de serveis interactius. Per un observador casual, això podria semblar el resultat d'un procés de producció ja desenvolupat i adult en la indústria. No obstant, la realitat és que la majoria dels serveis interactius segueixen desenvolupant-se de formes no portables i amb l'objectiu d'una única plataforma individual. El resultat d'això és un cost de producció alt i usualment limitat per un perfil de programació alt.

L'aparició de DVB-PCF no ha canviat de moment aquest fet com es pretenia amb la seva publicació a mitjans del 2006. La seva poca repercussió en la indústria en part es deguda al fet que no és una solució final extrem a extrem i a la manca d'intèrprets o conversors de descripcions PCF a les diferents plataformes emprades per la indústria.

A més de PCF, altres propostes han sorgit amb la idea d'homogeneïtzar una mica més el sector. The *Society of Motion Picture and Television Engineers* va publicar SMPTE 397M [18] l'abril del 2003, una especificació de format portable per a serveis de televisió interactius. Almenys, fins al coneixement de l'autor, hi ha un operador no europeu que hagi fet ús d'aquest estàndard per a l'enviament de contingut.

Tot i que actualment el grup de treball de DVB-PCF roman inactiu, cal destacar l'ús d'aquest estàndard per la *British Broacaster Corporation* (BBC), un dels *broadcasters* més importants i influents d'Europa. El grup *BBC TV Plantforms* ha desenvolupat un conversor de DVB-PCF a OpentTV [19], Liberate [20] i MHEG [21] i està en procés de proporcionar feedback al grup de treball de DVB-PCF.

Per contra, com a punt negatiu, no s'ha trobat encara cap ús de PCF en cap sistema de producció de continguts.

3.3 ADOBE FLASH

3.3.1 Introducció

Adobe Flash és una aplicació destinada a la producció i entrega de contingut interactiu sobre diferents sistemes i plataformes. Actualment està sent desenvolupat per Adobe Systems, tot i que va ser creat inicialment per un petit estudi nord-americà anomenat FutureWave Software i el seu nom original era FutureSplash Animator. En 1996 Macromedia va adquirir FutureWave, i va treure el programa d'animació vectorial conegut com Flash 1.0. Fins al 2005 va ser conegut com a Macromedia Flash, fins que Adobe Systems va adquirir Macromedia, canviant el nom del producte per Adobe Flash, i van ampliar el seu ventalls de productes en el mercat.

Figura 13. *Logo d'Adobe Flash*

El sistema disposa de gràfics vectorials i imatges ràster, so, codi de programació i flux de vídeo i àudio bidireccional. Els arxius Flash, que generalment la extensió d'arxiu SWF (ShockWave File), i poden ser reproduïts per reproductors independents de Flash o des d'una navegador mitjançant els plug-ins (dels quals disposen la gran majoria de navegadors actuals). Aquests arxius apareixen molt sovint en animacions de pàgines web i llocs web multimèdia, utilitzats en molts casos per a la publicitat i jocs interactius.

A les primeres versions, Macromedia va anar ampliant la plataforma Flash per realitzar més que animacions simples, convertint-lo en una eina de desenvolupament completa, principalment per a crear elements multimèdia i interactius per a Internet.

En sentit estricte, Flash és la plataforma de desenvolupament i Flash Player és el programa de màquina virtual sobre el qual s'executen els arxius generats amb Flash. Referent a aquest punt, amb el conversor realitzat en aquest projecte, es realitzen aplicacions per a

ser visualitzades amb Flash Player, i es fa ús de la programació definida amb Flash, però s'han utilitzat alternatives de software lliure per a generar-les, evitant l'ús del Flash IDE.

Adobe Flash Player és un reproductor d'arxius SWF que poden ser creats per Adobe Flash i Adobe Flex i altres eines de tercers i lliures. Aquests arxius es reprodueixen en un entorn determinat (en un sistema operatiu tenen el seu format d'aplicació del sistema, i en els navegadors són reproduïts pels plug-ins o objectes ActiveX en cas d'Internet Explorer). També és utilitzat per a oferir una millor qualitat d'experiència (QoE) en l'entrega de vídeo per Internet, fent que la càrrega d'aquest sigui més òptima i més ràpida.

3.3.2 ActionScript

3.3.2.1 Introducció

El llenguatge de programació base que utilitza la plataforma Flash s'anomena *ActionScript*. Aquest llenguatge va ser desenvolupat inicialment per *Macromedia*, tot i que conjuntament amb tota la plataforma, actualment és una marca registrada *d'Adobe*. No obstant, el llenguatge en sí és lliure, en el sentit que l'especificació que ofereix està lliure de càrrega i hi ha disponibles compiladors i una màquina virtual (anomenada *Mozilla Tamarin*) de codi lliure.

ActionScript va començar com un llenguatge de creació d'ordres (*scripting*) per a l'eina d'autoria *Macromedia* Flash. En les tres primeres versions de Flash la *interactivitat* que oferia eren molt limitades, amb la possibilitat de lligar comandes simples, anomenades accions, a un botó o un frame determinat. Així, el desenvolupador podia afegir accions com *play*, stop, *gotoAnbPlay* o *getURL*, que augmentaven la *interactivitat* de l'aplicació amb l'usuari i permetien controls de navegació.

Aquesta sèrie de comandes va esdevenir en un llenguatge *d'scripting* a la versió Flash 4, incloent variables, expressions, operadors, sistemes de bucles i condicions. El manual de Flash 4, però, seguia usant de cara a l'usuari el terme accions per descriure aquestes comandes, tot i oferir ja les possibilitats esmentades.

En un inici va ser dissenyat com un control d'animacions 2D vectorials simple. Inicialment centrat en l'animació. Les següents versions van anar afegint més i més funcionalitats per a la creació de jocs per a l'utilització en webs i aplicacions riques a Internet amb streaming de continguts audiovisuals. Avui en dia, ActionScript pot ser utilitzat també en l'ús d'algunes aplicacions de bases de dades i robòtica bàsica, com *Make Controller Kit.*

Addicionalment, les llibreries proporcionades poden ser usades juntament amb les capacitats de processament XML dels navegadors per a oferir contingut *rich media* (contingut audiovisual interactiu) a través d'Internet. L'ús d'aquesta tècnica és coneguda

com Asynchronous Flash and XML, en clara referència a AJAX (Asynchronous JavaScript and XML)

3.3.2.2 Versions

ActionScript 1.0

De l'any 1999 al 2003. Flash va desenvolupar el llenguatge d'accions que portaven les versions anteriors a Flash 4 i a partir de Flash 5 aquests va ser conegut ja com a ActionScript. Aquesta primera versió apareixia amb moltes influències de JavaScript i l'estàndard ECMA-262 [22], suportant aquest estàndard de model d'objectes i molts dels seus tipus de dades *core.* Les variables locals poden ser declarades mitjançant la declaració *var,* i es poden definir funcions per l'usuari amb pas de paràmetres i retorns de valors.

Tot i que la seva definició pot ser realitzada mitjançant el Flash IDE i anar-se afegint accions en les diferents llistes i caixes de diàleg de les accions, també pot ser realitzat directament sobre qualsevol editor de texts (un altre canvi principal respecte al llenguatge d'accions que aportava Flash en les versions anteriors). En la següents versions Flash 5 i MX (6) el llenguatge es va mantenir essencialment igual, amb l'aparició de canvis mínims com la introducció de l'operador d'igualtat estricte (===) a la versió MX. Dues característiques principals d'aquesta versió respecte a les posteriors és la seva definició poc estricta de variables que ofereixi més llibertat (però a la vegada menys exactitud) i la seva inherència basada en *prototype* [23]. Aquest tipus d'inherència és el mecanisme que utilitza ActionScript 1.0 per a reusar codi i realitzar programació orientada a objectes. ActionScript 1.0 utilitza un objecte especial que serveix de prototip per a una classe d'objectes. Totes les característiques comuns d'una classe són definides en aquest objecte prototip i cada instància de la classe definida amb aquest objecte està lligat a aquest objecte prototip.

ActionScript 2.0

Al 2003, corresponent amb la sortida de Flash MX 2004 (Flash Player 7) es produeix la primera gran revisió del llenguatge i se li dona el nom d'ActionSript 2.0. En resposta a algunes crítiques rebudes pels usuaris de millora d'aquest llenguatge per a realitzar aplicacions més complexes, ActionScript 2.0 porta una revisió del codi en temps de compilació i una sintaxi basada en classes, amb la possibilitat de realitzar *extends* les classes ja existents.

Així doncs, ActionScript 2.0 ofereix un sistema d'inherència basat en classes i interfícies, més usual de codis més complexos de programació orientada a objectes com Java o C++. No obstant, cal dir que aquest sistema d'inherència, és només un sistema posat per sobre del sistema d'inherència d'ActionScript 1.0 que es basava en prototips.

Amb ActionScript 2.0, els desenvolupadors poden afegir variables restriccions d'un tipus de variable determinat, per tal de trobar errors de no concordança de tipus a l'hora de compilar. És a dir, les variables, tot i seguir sent definides mitjançant la declaració *var*, poden ser restringides a un tipus de variable determinat per a facilitar la correcta definició i utilització d'aquestes variables.

```
var caixa_de_text: TextField;
var comptador: Number;
var text: String;
```

Figura 14. *Exemple de definició de variables en ActionScript 2.0*

Cal destacar, a més, que les versions de la plataforma *Adobe Flash Lite*, la versió de Flash per a dispositius amb menys prestacions (i en la que es basa el conversor realitzat en aquest projecte), utilitza aquest llenguatge des de la seva primera versió, i fins la versió *Flash Lite 4* no accepta ActionScript 3.0 (i amb algunes limitacions).

ActionScript 3.0

El llançament *d'Adobe Flex 2.0* i el *Flash Player 9* al juny del 2006 va suposar també el debut de la nova versió del llenguatge, ActionScript 3.0. Aquesta revisió va suposar una reestructuració important del llenguatge i utilitza una versió completament diferent de la màquina virtual. És per això que *Flash Player 9* conté en realitat dues màquines virtuals,, una AVM1 per a codi escrit en ActionScript 1.0 i 2,0 i una altra, AVM2, per a contingut escrit en ActionScript 3.0. A més, aquesta versió afegeix, amb limitacions, suport per a acceleració gràfica hardware (*DirectX*, *OpenGL*).

3.3.2.3 Sintaxi d'ActionScript 2.0

Aquesta secció es centrarà només en la segona versió d'aquest llenguatge, ja que serà la emprada com a base d'aquest conversor juntament amb Java.

Tipus de dades

ActionScript principalment consisteix en tipus de dades simples o fonamentals que són usades per a formar-ne d'altres. Aquests tipus de dades són similars als utilitzats a altres llenguatges com Java, per exemple. Les classes essencials d'ActionScript 2.0 són les següents:

- *String:* Una cadena de caràcters
- *Number:* Qualsevol valor numèric. A diferència d'altres llenguatges, AS 2.0 no fa distinció entre el tipus de valor numèric (p.e. enter, real, etc.), definint només un únic tipus de números.
- *Boolean:* Un binari simple que pot ser *true* o *false*.

- *Object:* Tipus de variable del qual hereten tots els altres tipus més complexos. Permet l'agrupació de mètodes, funcions, paràmetres i altres objectes.

Addicionalment, existeixen una sèrie de tipus de dades complexos ja especificats d'entrada.

- *MovieClip:* El clip de pel·lícula consisteix en una creació ActionScript que permet l'agrupació de forma senzilla de varis objectes visuals.
- *TextField:* Heretant del tipus MovieClip, aquest tipus de variable consisteix en un quadrat de text que pot ser estàtic o dinàmic i també pot permetre fer funcions d'entrada de text (no només visualització).
- *Button:* Consisteix en un butó simple amb quatre *frames* diferents (corresponent als estats de *Up*, *Over*, *Down* i *Hit*). També hereta de *MovieClip*.
- *Date:* Permet emmagatzemar i accedir a informació d'un moment de temps específic, guardant informació de data i hora.
- *Array:* Permet l'emmagatzemament d'un vector de dades no especificades (a AS 2.0 la variable *Array* pot contenir qualsevol tipus de variables o objectes)
- *XML:* Permet emmagatzemar la informació d'un arxiu XML i emmagatzemar-la amb un objecte per a la seva manipulació.
- *XMLNode:* Objecte que pot ser usat per a extreure la informació de l'objecte XML.
- *LoadVars:* Objecte que s'utilitza per a emmagatzemar i enviar comandes (*HTTP POST* i *HTTP GET*) a través de la xarxa.
- *Sound:* Objecte que emmagatzema la informació d'un arxiu de so i permet la seva manipulació i reproducció.
- *NetConnection:* Objecte emprat per a la connexió HTTP amb un altre dispositiu o pàgina web.
- *NetStream:* Objecte que serveix per a carregar un flux de dades rebut a través d'una connexió establerta amb *NetConnection.*
- *MovieClipLoader:* Objecte utilitzat per a la càrrega de MovieClips. En general és utilitzat per a realitzar pre-càrregues d'aquests, sobretot quan un MovieClip conté molts objectes i la seva càrrega en una web podria resultar lenta. L'utilització d'aquest objecte per a realitzar la càrrega permet que es faci la càrrega en memòria abans de la seva visualització i que la seva visualització completa després sigui automàtica, sense afectar negativament a la seva interactivitat.
- *EventListener:* Objecte que serveix a definir diferents tipus d'events i poder associar-los a un MovieClip determinat o a tot l'àmbit de l'aplicació.

3.3.2.4 Exemple de sintaxi

El següent codi (Figura 15), el qual funciona a qualsevol *Flash Player* (i compatibles), crea un quadrat de text. El primer dels paràmetres (*hola*) és l'identificador amb el qual es crearà el quadrat de text, i amb el qual es podran cridar les funcions i paràmetres d'aquests després.

El segon paràmetre és la profunditat, que està a zero. La profunditat marcarà la visibilitat de l'objecte en el cas que s'afegeixin més objectes que coincideixin visualment amb la posició del quadrat de text. El 0 és la profunditat més baixa, i qualsevol objecte definit amb una profunditat superior taparia aquest quadrat de text.

Els següents dos paràmetres indiquen que la posició on es crearà l'objecte és (0,0), tenint en compte que l'inici de coordenades a ActionScript és la cantonada superior esquerra de l'escenari de l'aplicació.

Per últim, els dos últims paràmetres, indiquen la grandària del quadrat de text, de 100 (amplada) per 100 (alçada) píxels.

Finalment, s'accedeix al paràmetre text del quadre de text, i és igualat a la cadena de caràcters Hello World, que serà el que el quadre de text mostrarà quan es mostri al reproductor:

```
createTextField("hola", 0, 0, 0, 100, 100);
hola.text = "Hello, world";
```

Figura 15. *Exemple de creació d'un quadre de text "Hello World" en AS 2.0*

Quan es vol afegir aquest codi en una classe AS 2.0, es pot crear una classe com la mostrada a la Figura 16, la qual hereta d'un MovieClip, que, en quant és carregat crea el quadrat de text tal com s'ha definit anteriorment. Aquesta codi aniria escrit en un fitxer de text anomenat HelloWorld.as, per ésser cridada des d'una altra classe d'execució principal.

```
class com.example.HelloWorld extends MovieClip
{
    public function HelloWorld() {}
    public function onLoad() :Void
    {
        var txtHello:TextField = this.createTextField("txtHello", 0,
0, 0, 100, 100);
        txtHello.text = "Hello, world";
    }
}
```

Figura 16. *Exemple de classe "Hello World en AS 2.0"*

3.3.3 Adobe Flash Lite

Flash Lite és una versió especial *d'Adobe Flash*, la qual va ésser creada per ser usada específicament en telèfons mòbils i altres dispositius electrònics portàtils però no de telefonia, com *Chumby* i *iRiver*, permetent als usuaris accedir a contingut multimèdia i aplicacions desenvolupades amb eines *d'Adobe Flash*, que fins al seu moment només estaven disponibles per a ordinadors. Així, *Flash Lite* no ha de ser considerat un sistema operatiu per a aquests dispositius, com pot ésser *Symbian OS, Windows Mobile OS, iOS*, o *Android*, sinó una tecnologia per a desenvolupar aplicacions multimèdia interactives sobre aquests sistemes operatius.

Figura 17. *Adobe Flash Lite sobre múltiples dispositius de baixes capacitats*

Igual que *Flash*, és una tecnologia implementada a nivell de client, a la capa d'interfície d'usuari. Disposa, igual que *Flash*, d'un reproductor determinat, *Flash Lite Player*, i una sèrie de funcions específiques per a millorar la integració amb aquest tipus de dispositius. Tant és així que és capaç de competir amb altres tecnologies que usualment actuen a nivells més baixos de dispositiu, com *Java ME* o *BREW* [24].

Les versions d'aquesta plataforma que, primer *Macromedia*, i després *Adobe,* ha publicat fins ara són:

- Macromedia Flash Lite 1.0 and 1.1
- Macromedia Flash Lite 2.0 (Desembre 2005)
- Adobe Flash Lite 2.1 (Desembre 2006)
- Adobe Flash Lite 3 (Febrer 2007)
- Adobe Flash Lite 3.1 (Febrer 2009)
- Adobe Flash Lite 4 (Maig 2010)

Flash Lite version	Year of release	Based on Flash Player version	ActionScript version
1.0	2003	4	1.0
1.1	2004	4/5	1.0
2.0	2006	7	2.0
2.1	2006	7	2.0
3.0	2007	8	2.0
3.1 Distributable player	2009	8	2.0

Figura 18. *Versions de Flash Lite*

Flash Lite 1.1 suporta ActionScript realitzat amb Flash 4. Flash Lite 2.0, està basat en Flash Player 7 i amb la versió ActionScript 2.0. Totes dues versions suporten World Wide Web Consortium estàndard SVG Tiny, que és un perfil per a mòbil de la recomanació SVG (Scalable Vector Graphics) que defineix aquest tipus d'imatges vectorials. Flash Lite permet l'addició d'àudio i altres elements interactius sense necessitat de JavaScript. També, a l'igual que Flash, Flash Lite por llegir i escriure contingut XML extern.

Flash Lite 3 està basat en Flash 8, i és la versió que va reduir realment la diferència entre les dues plataformes germanes, amb el suport a l'estàndard de vídeo H.264, a més dels còdecs de vídeo VP6 d'*O2* i *Sorenson*. També introdueix el suport als contenidors de vídeo FLV (*Flash Video*).

La recent aparició de Flash Lite 4, suposa un nou canvi, perquè ja suporta ActionScript 3.0, tot i que amb limitacions. A més, el contingut pot ser reproduït directament amb un plugin al navegador del dispositiu, i no com un reproductor autònom.

No obstant, l'aparició d'aquest producte després de l'inici d'aquest projecte no ha fet possible l'adaptació encara del conversor a aquesta versió, amb l'inconvenient, a més, que només és suportable pels dispositius més novedosos. Els dispositius finals amb els quals es va realitzar l'especificació de requeriments només suportaven AS 2.0 i Flash Lite 3.1, i les característiques de la primera versió d'aquest conversor que es veuran en les posteriors seccions d'aquest document estan pensades segons aquests requeriments.

La tendència d'implantació al mercat de Flash Lite, tal com es veu a la Figura 19, ha anat en augment fins l'any 2009, i el nombre de dispositius que disposen d'aquesta plataforma ha augmentat en gran mesura sobretot arrel de l'aparició de les versions 2.1 i 3, que disposen de noves funcionalitats de reproducció de vídeo.

	2008	2009
Global Total		
Flash Lite 1.0	12,049,000	10,143,000
Flash Lite 1.1	270,263,000	163,098,000
Flash Lite 2.0	250,634,000	281,552,000
Flash Lite 2.1	145,875,000	314,648,000
Flash Lite 3.0+	40,687,000	284,225,000
Total	**719,508,000**	**1,053,666,000**

Figura 19. *Tendències d'utilització d'Adobe Flash Lite [25]*

Aquest va ésser els factors principals que va suposar l'elecció d'aquesta tecnologia. Tot i així, el disseny plantejat per al conversor faria possible una ràpida migració cap a una altra tecnologia per a crear aplicacions interactives, tant sigui *Flash Lite 4*, *Flash 10*, com una altra de programació orientada a objectes.

3.3.3.1 Limitacions respecte Flash

Una de les principals preocupacions que es van tenir en l'elecció de *Flash Lite* va ser la desconeixença de quines limitacions podia presentar respecte a la versió *Flash*. Adjunt als annexos d'aquest document hi ha una taula comparativa de les característiques d'una i altra plataforma. A més, en aquesta secció es comentaran algunes altres de les limitacions que suposa l'ús d'aquesta plataforma.

A causa de les limitacions de recursos de hardware disponibles en mòbils i altres dispositius portàtils s'han de seguir algunes de les següents indicacions per a assegurar un correcte comportament de l'aplicació *Flash Lite*:

- Evitar operacions que requereixin un ús intensiu de CPU. Això significa principalment l'ús reduït de gràfics vectorials, transparències i efectes *alpha*, animacions complexes, ja que tot aquest material requereix molt procés de càlcul que consumeix molts recursos en les plataformes mòbils. A més, formes vectorials poden patir petits defectes visuals en els mòbils, ja que la majoria dels dispositius no disposen de hardware que suporti càlculs en coma flotant i el dibuix de gràfics vectorials.
- Reemplaçament de totes les formes vectorials per *bitmaps* (mapes de bits) pot incrementar la mida del fitxer i l'ús de la memòria del sistema. A més, pot presentar problemes d'escalabilitat si es realitza per a ser mostrat en diferents resolucions i dispositius. La memòria disponible per a contingut *Flash Lite* és un altre recurs limitat, ja que hi ha d'haver un compromís entre la memòria utilitzada i la càrrega de CPU. D'altra banda, la mida de la pantalla en mòbils fa que no es requereixin mapa de bits, ni vídeos de grans resolucions, i això afavoreix l'estalvi de requeriments en quant a memòria i consum d'energia.

- Mantenir una connexió a la xarxa activa consumeix molta energia i pot repercutir també en un cost monetari per a l'usuari; per tant, les aplicacions han d'intentar minimitzar el seu accés a la xarxa. Si l'aplicació necessita transmetre dades a la xarxa, s'ha d'intentar minimitzar la mida d'aquestes dades i el temps que es tarda en transmetre-les, mantenint la connexió oberta el menys temps possible. D'aquesta manera, és preferible l'ús d'enviament per ràfegues en lloc d'un enviament continu. D'aquesta manera, l'estalvi energètic serà major, fet molt important per a dispositius mòbils.
- Captura de localització mitjançant GPS sol tenir també un cost energètic molt alt, i el seu ús també s'ha de minimitzar per tal d'allargar la vida de la bateria.

3.3.3.2 Avantatges

A continuació es mostraran una sèrie d'avantatges de l'ús de *Flash Lite* respecte altres alternatives per a dispositius mòbils:

- Ràpid desenvolupament gràcies a entorns de desenvolupament que permeten una realització iterativa de creació de prototips de software i una prova fàcil de les funcionalitats.
- No depèn de APIs específiques del dispositiu, i resulta en processos de portabilitat molt senzills o inexistents, a diferència de *Java ME*.
- Els gràfics utilitzats en la seva majoria són vectorials (tot i el suport també a mapa de bits), fet que permet fàcilment l'escalatge, rotació i altres transformacions sense perdre qualitat.
- Possibilitat d'empaquetar més d'una animació i gràfics dins del mateix fitxer. Això fa que el procés de distribució sigui molt més senzill
- La possibilitat de convertir contingut Flash per a web (i escriptori) en contingut per a mòbil amb un esforç mínim.

3.3.3.3 Inconvenients

Respecte els inconvenients que presenta la plataforma respecte a altres sistemes d'animacions interactives per a mòbil, es podrien destacar els següents:

- Al 2008 encara hi havia escassament 361 tipus de mòbils que suportessin *Flash Lite*. Això suposa una audiència limitada en comparació a altres tecnologies com *Java ME*, *Symbian*, etc. però, com s'ha vist a la Figura 19, aquesta és una tendència que ha anat canviat els últims anys.
- Relativament pobre rendiment gràfic en alguns dispositius
- Pobre tractament de so respecte a altres programes

3.3.3.4 Alternatives

Una de les alternatives pel desenvolupament d'aplicacions interactives sobre dispositius mòbils i altres de baixes prestacions és treballar amb Java ME. Aquesta plataforma dóna un accés a molt més baix nivell, però sol ser bastant depenent del dispositiu, limitant molt la programació multiplataforma. La quantitat de dispositius que suporten Java ME és molt major, a causa d'un major temps en el mercat. Tot i així, l'augment de dispositius que suporten Adobe Flash Lite ha anat augmentant en els últims anys i ofereix certs avantatges respecte Java ME, sobretot a nivell gràfic.

Flash Lite 3 suporta un gran nombre de estàndards de gràfics. No obstant, no ofereix suport a gràfics 3D (veure Figura 20)

Graphics		
	Java ME MSA	Flash Lite 3
GIF		X
JPEG	X	X
PNG	X	X
SVG	X	X
3D	X	
FLA		X

Figura 20. *Taula comparativa de formats gràfics suportats*

Les opcions que ofereix en quant a reproducció multimèdia també són molt majors, suportant alguns dels estàndards de codificació més utilitzats i que ofereixen més qualitat (Figura 21).

Multimedia		
	Java ME MSA	Flash Lite 3
FLV		X
3gpp	X	X
on2		X
Sorensen		X
MP3 playback	X	X
streaming video	X	X
streaming audio	X	X

Figura 21. *Taula comparativa de formats multimèdia suportats*

Per altra banda, les possibilitats que ofereix Java ME a baix nivell és molt major (Figura 22).

Local Device Accessibility

	Java ME MSA	Flash Lite 3
persistent data	X	X
file I/O	X	
calendar access	X	
contact list access	X	
photo capture	X	
recording video	X	
recording audio	X	
microphone access	X	
GPS access	X	
SIM card access	X	
accelerometer access		
phone call initiation	X	X

Figura 22. *Taula comparativa d'accessibilitat del dispositiu*

A part de Java, l'aparició d'HTML5 en programació web i l'aparició d'Android en l'àmbit de dispositius mòbils, ha augmentat la competència a la plataforma Flash a tots els nivells.

El debat que s'ha obert en la discussió entre Flash i HTML5 ha passat de ser tècnic a quasi polític, degut als grans interessos econòmics que hi ha al darrera. De totes formes, deixant de banda aquests altres aspectes no funcionals, HTML5 ofereix la possibilitat de realitzar webs que no requereixen de *plugins* addicionals (a part del suport del navegador a aquest format), i amb grans possibilitats gràfiques, de vídeo i d'interactivitat.

A més, permet fer un ús molt més transparent, sobretot per a produccions a petita escala, utilitzant HTML5 i H.264 per a reproduir continguts de vídeo directament al navegador sense la necessitat de cap reproductor especial per a mòbils o cap *caixa negra* a dins del navegador.

Però, de la mateixa manera, un dels aspectes que pot interessar més als distribuïdors i que HTML5 no ofereix, és la possibilitat d'ús de DRMs o algun tipus d'emmascarament de l'aplicació o el vídeo al client. En aquest punt, Flash surt clar vencedor.

Les tendències fan presagiar que aquesta batalla de poder entre aquestes dues plataformes no tindrà un clar vencedor, almenys durant els propers anys.

En quant a dispositius mòbils, l'aparició del sistema operatiu *Android* no hauria de resultar un rival directe per *Flash Lite*, ja que la versió *d'Android 2.1* suporta *Flash Lite* i en alguns dispositius fins i tot *Flash 10* en els seus navegadors. Com s'ha comentat anteriorment, Flash Lite és una aplicació que està per sobre del sistema operatiu, però l'oferta d'APIs que ofereix Google per a la lliure programació d'aplicacions sobre Android limita el mercat Flash Lite únicament a web.

A més, Google no ha tancat les seves portes a Flash, però d'altra banda dóna un clar suport a l'estàndard HTML5 en els seus navegadors i ha obert al públic el format de codificació VP8, amb la intenció de convertir-lo en l'estàndard *de facto* per a la reproducció de vídeo a través de la xarxa, un dels usos més habituals de la plataforma *Flash*.

4. ENTORN DE TREBALL

4.1 INTRODUCCIÓ

En aquest apartat es detallarà l'entorn de programació, compilació i desplegament utilitzat, per a ubicar l'usuari en la forma en què s'ha portat a terme la part pràctica del projecte. Els elements que s'explicaran són Eclipse com a IDE de programació, juntament amb els *plug-ins* ASDT (*ActionScript Development Tool*), per a la programació d'*ActionScript*, i ANT per a la compilació dels projectes. A més, també s'explicarà els mètodes d'instal·lació i d'ús dels compiladors de codi lliure d'*ActionScript* que s'han emprat (SWFMill i MTASC).

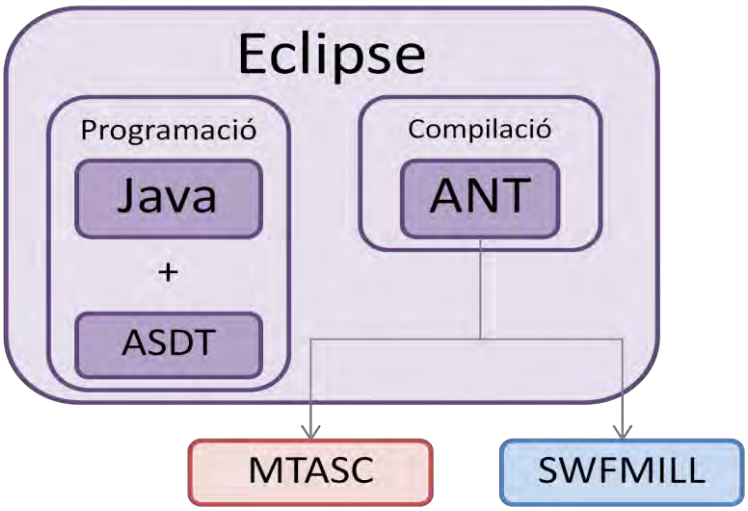


Figura 23. *Esquema de l'entorn de desenvolupament*

4.2 ECLIPSE

Els desenvolupament d'aquest projecte s'ha realitzat integrament amb Eclipse [26], un entorn de desenvolupament de codi obert i multiplataforma. L'elecció d'aquest entorn de treball s'ha realitzat principalment pel coneixement previ que es tenia en el seu ús per a programació en Java i per la possibilitat que ofereix de programació d'ActionScript 2.0 de forma oberta i sense necessitat de la llicència per el IDE (Entorn de Desenvolupament Integrat) oficial d'Adobe.

Un altre factor important en l'elecció era la integració d'Eclipse amb l'eina d'automatització de processos Ant [27], fet que resultava vital per a l'execució de varies tecnologies (Java i AS 2.0) a la vegada i l'automatització de tot el procés de conversió amb la mínima interacció possible amb l'usuari.

A més, cal destacar que a diferència de l'IDE oficial d'Adobe, Eclipse té la possibilitat d'integrar plug-ins de control de versions (Subclipse [28]) que proporcionen un treball en equip concurrent més eficient i d'un control més exhaustiu del desenvolupament realitzat.

La versió utilitzada per al desenvolupament del conversor ha estat la versió Eclipse v.3.4.1 i amb l'IDE per a desenvolupament de projectes Java EE, ASDT (ActionScript Development Tool) i la versió 1.5 de JDK (*Java Development Kit*).

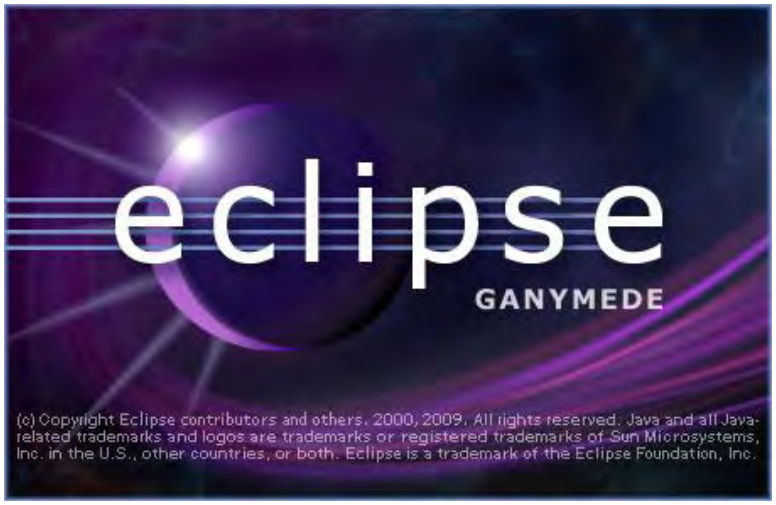


Figura 24. *Entorn de desenvolupament: Eclipse v.3.2.1*

4.3 OPEN SOURCE FLASH

4.3.1 Introducció

Per a realitzar aplicacions *Flash* o *Flash Lite*, a part del programa de desenvolupament proporcionat per *Adobe* (*Flash IDE*), existeixen altres opcions.

Existeix una comunitat de Flash de Codi Obert (*Open Source Flash* [29] molt dinàmica, amb projectes nous que apareixen regularment. Aquesta comunitat va sorgir principalment a partir de l'aparició del compilador MTASC [30].

Aquest és un compilador lliure d'Action Script 2.0 que ja ha estat provat en la compilació de projectes per a *Flash Lite* 2 i 3 [31]. Partint d'aquest compilador, es va anar creant aquesta comunitat de desenvolupament de Flash sense utilitzar o utilitzant mínimament el *Flash IDE*. A la seva pàgina oficial [32], explica com configurar una estructura d'un projecte flash mitjançant el conjunt d'eines alternatives presentades per la comunitat.

En la revisió d'aquestes eines, s'ha trobat destacable l'ús comú tant del MTASC com a compilador d'AS 2.0, juntament amb un altre compilador SWFMILL [33], ja sigui utilitzant *Eclipse* amb el *plug-in* ASDT (*ActionScript Development Tool* [34]) per desenvolupament d'AS 2.0 o d'un entorn anomenat *FlashDevelop* [35].

4.3.2 MTASC

4.3.2.1 Introducció

MTASC (*Motion-Twin ActionScript 2 Compiler*) és un compilador d'ActionScript 2.0 escrit en llenguatge de programació l'*Objective Caml* (OCaml [36]) que va ser desenvolupat per l'empresa francesa Motion-Twin. És software lliure i pot ser utilitzat de manera sola o amb altres eines com *SWFmill* per produir arxius SWF, que contenen contingut multimèdia interactiu reproduïble amb *Flash Player*.

A més, MTASC es mostra com un compilador molt més ràpid que el compilador d'AcionScript propi d'*Adobe Flash* [37].

MTASC va ser construït a partir de la optimització de compiladors *OCaml*, els quals proporcionen una millora substancial de la velocitat de compilació respecte al Compilador de *Macromedia* (MMC) que proporciona l'*Adobe Flash IDE* (abans *Macromedia Flash IDE*). A més, MTASC corregeix alguns problemes de seguretat que ocorren utilitzant MMC [37].

El compilador de MTASC és més estricte que MMC i pot detectar més errors que MMC; a causa d'aquest fet, hi ha algunes diferències entre MMC i MTASC, com la incompatibilitat amb les variables d'àmbit local i definicions locals. Aquestes diferències són revisades amb més detall a la secció 4.3.2.4.

Com a principal punt feble, cal destacar que MTASC no dóna, ni donarà suport a ActionScript 3.0, ja que els desenvolupadors en donen suport a través del que consideren el seu successor, *haXe* [38].

4.3.2.2 Instal·lació

A continuació es detallen els passos del procés d'instal·lació a través de línia de comandes.

Primer cal descarregar el paquet de MTASC corresponent sistema on anirà i instal·lar/descomprimir a algun lloc en el seu ordinador. Llavors s'ha d'afegir la ruta de l'executable que s'ha instal·lat a la variable d'entorn PATH [39]. Per exemple si ha instal·lat a *c:\program files\mtasc* caldria afegir aquesta ruta al PATH. D'aquesta manera la comanda mtasc es fa accessible des de qualsevol ubicació.

4.3.2.3 Utilització

Després de realitzar els passos de la secció anterior, ja podem utilitzar la comanda per a compilar els nostres arxius de la següent manera:

```
mtasc.exe <arxius_a_compilar.as> ... -swf <arxiu_de_sortida.swf>
```

Figura 25. *Comanda de compilació mitjançant MTASC*

Si es vol estalviar-se la utilització de la línia de comandes, una opció senzilla i recomanable és posar la línia definida a la Figura 25 en un arxiu de text i guardar aquest arxiu de text amb extensió .bat (`compila.bat`, per exemple)

En tot cas, MTASC és un compilador per línia de comandes que pot ser fàcilment integrat a dins de molts editors de textos i entorns de desenvolupament perquè realitzi la compilació d'arxius .as (ActionScript).

El compilador permet la compilació de múltiples arxius .as amb la línia de comandes especificada anteriorment. Tot i així, si s'especifica la compilació d'una classe principal (per exemple, `mtasc Main.as`), i aquesta és depenent d'altres classes AS, no cal incloure-les en la comanda (sempre i quan estiguin al mateix directori que la classe principal). Si l'execució de la comanda dóna algun error degut a que no ha trobat alguna classe, sempre es pot afegir l'etiqueta *–cp* abans dels arxius a compilar, per tal d'afegir la variable ClassPath (el directori on el programa ha d'anar a buscar els arxius a compilar .as).

```
mtasc.exe –cp <ruta_directori_arxius> <arxius_a_compilar.as... -swf
arxiu_de_sortida.swf
```

Figura 26. *Comanda de compilació amb MTASC utilitzant classpath*

MTASC agafa l'arxiu especificat per l'etiqueta *–swf*. Després compila tots els arxius AS especificats i actualitza l'arxiu .swf, reemplaçant les classes que conté amb les noves classes que s'han compilat.

Arguments principals a utilitzar:

- *-swf <arxiu>* : Amb la comanda –swf també es pot especificar un fitxer d'entrada SWF amb recursos.
- *-out* <arxiu> : Fitxer SWF de sortida.
- *-cp <ruta>* : Per a afegir una ruta (o rutes, separades per ';') on trobar els arxius font AS.
- *-main* : afegint aquesta opció es crida automàticament la funció estàtica *main* un cop totes les classes han estat registrades.
- *-header <amplada:alçada:imatges_per_segon:color_de_fons>* : Amb aquesta funció no carrega un SWF, sinó en crea un de nou amb la informació de capçalera proporcionada amb aquest argument i les classes proporcionades. L'amplada i l'alçada s'ha de proporcionar en píxels, i el color de fons ha de ser introduït mitjançant 6 dígits hexadecimals.

- *-mx*: per a la utilització de classes MX (un paquet de classes precompilades d'Adobe)

Altres arguments per usuaris més avançats:

- *-version <número_de_versió>* : Especifica la versió de l'arxiu SWF de sortida. Les versions compatibles són des de Flash 6 fins al 8. No permet especificar versions de Flash Lite, però per les característiques de Flash Lite 3, la compilació que es necessita és Flash 8, el qual suporta pràcticament totes les característiques de Flash Lite 3.
- *-v* : Activa el mode verbós (*verbose*), el qual pinta per pantalla informació addicional del procés de compilació, que ajuda al procés de *debug*.
- *-msvc* : Canvia l'estil del format d'errors que es mostra, mostrant-los com si s'utilitzés Microsoft Visual Studio. Per defecte es mostren en l'estil que es mostra amb Java.
- *-strict* : Amb aquesta opció, la compilació es torna més estricta, requerint que el tipus de totes les variables estigui explícitament definit.
- *-exclude <fitxer_llista>* : Aquesta comanda exclou la generació de codi del llistat de classes especificat en un fitxer de text *<fitxer_llista>.* Aquest fitxer ha de contenir una llista de fitxers on es troba una classe (amb el seu PATH relatiu o absolut) per línia.
- *-trace <funció_de_traces>* : especifica una funció de traces determinada per a una millor depuració del programa. La opció *no* deshabilita les traces per pantalla.
- *-keep* : Aquesta opció ofereix la possibilitat de mantenir les funcions compilades amb MMC que estan dins del SWF destí. L'utilització d'aquesta comanda només és recomanables en casos concrets on s'hagi d'utilitzar una classe determinada que no és suportada amb MTASC, ja que pot causar que en un SWF hi hagi la mateixa classe dos cops, compilada amb MMC i amb MTASC, i que provoqui incongruències en la seva execució.
- *-frame f* : Per exportar classes AS 2 en un frame determinat del SWF.
- *-pack <ruta_directori>* : compilar totes les classes que conté el paquet especificat. Cal observar, però, que aquesta comanda no és recursiva.
- *-group* : Comanda que ajunta totes les classes compilades en un mateix clip de vídeo. Això suposa una disminució de la mida del SWF resultant i pot causar problemes amb els arguments *–mx* i *–keep*.

- -wimp : Afegeix avisos en el moment de compilació si s'han realitzat imports de fitxers que després no s'han utilitzat.
- -infer : Afegeix el tipus d'inferencia per a variables locals ja inicialitzades (explicat amb més detall a l'apartat 4.3.2.4)

Exemple d'ús

En aquest exemple es mostrarà un com realitzar una aplicació SWF des de zero, sense l'ús només amb la utilització de codi, de contingut extra SWF.

```
class Tutorial {
      static var app : Tutorial;
      function Tutorial() {
// crea un camp de text anomenat 'tf' de mida 800x600 a la posició 0,0
            _root.createTextField("tf",0,0,0,800,600);
// Escriu el text que portarà el camp de text
            _root.tf.text = "Hello world !";
      }
// punt d'entrada
      static function main(mc) {
            app = new Tutorial();
      }
}
```

Figura 27. *Exemple "Hello World" d'ActionScript 2.0 utilitzant MTASC*

Per compilar aquest exemple, l'únic que cal fer és cridar el compilador MTASC amb els següents arguments:

```
Mtasc –swf tutorial.swf –main –header 800:600:20 Tutorial.as
```

Figura 28. *Comanda de compilació de l'exemple*

El primer dels arguments `–swf tutorial.swf` s'encarrega de definir quin serà el fitxer de sortida; `–main` indica a MTASC que executi el punt d'entrada (funció *main*) un cop hagi inicialitzat totes les classes; amb el tercer paràmetre, `-header`, s'especifica que el fitxer SWF de sortida ha de tenir una mida de 800x600 píxels i un temps de refresc de 20 *fps* (imatges per segon); i per últim s'introdueix el fitxer a compilar `Tutorial.as`. Si s'han de compilar diverses classes es poden llistar una a una a la línia de comandes.

És important anotar que no cal introduir la ruta sencera d'aquelles classes que estiguin a la carpeta de *classpath* si compilem usant la comanda *–cp.* Per a les rutes que contenen espais, recordar que s'han de posar entre cometes dobles. Per exemple, si es té un conjunt de classes a *C:\Arxius de Programa\Flash\Classes*, i es vol compilar les classes *MyButton.as* i *MyBall.as* que tenim en aquest directori, la comanda introduïda seria la següent:

```
mtasc -cp "C:\Arxius de Programa\Flash\Classes" MyButton.as MyBall.as
```

Figura 29. *Exemple d'ús de l'argument "classpath"*

Si el que es vol és compilar un paquet de classes complet, es pot utilitzar l'atribut *–pack*:

```
Mtasc –cp "C:\Arxius de Programa\Flash\Classes" –pack i3Media
```

Figura 30. *Exemple d'ús de l'argument "pack"*

D'aquesta manera compilaríem tot el paquet de classes i3Media que es troba en el *classpath* especificat.

4.3.2.4 Comparació amb el Flash IDE

MTASC intenta corregir molts dels problemes de seguretat del compilador que s'ofereix amb el Flash IDE. És un compilador molt més estricte que pot detectar més errors i ajudar al programador a escriure programes més depurats.

Degut a això, és possible que un usuari acostumat a compilar amb el Flash IDE pugui obtenir alguns errors a causa d'aquestes diferències. No obstant, segons els propis creadors, els objectius principals de MTASC és que els usuaris puguin fer el pas del MMC a MTASC de forma ràpida i senzilla, i que aquestes diferències siguin ràpides d'incloure al codi i facin que el codi final sigui més segur. A més, destaquen la compatibilitat amb MMC com un dels altres punts a favor, sense obligar als usuaris d'MTASC a passar d'un sistema tancat com és el *Flash IDE* a un altre també tancat.

Velocitat

MTASC està basat en una de les millors tecnologies de compiladors disponibles (el llenguatge de programació OCaml) que fa que millori la velocitat proporcionada per MMC. És capaç de compilar 100 classes en menys de 5 segons.

Errors

El codi és més concís i més robust. Al ser obert és també és molt més fàcil trobar on s'ha produït l'error i el feedback proporcionat per la comunitat de codi obert ajuda a fixar-ne els errors ràpidament.

Cost

MTASC és gratuït i de codi obert. Això significa que es pot utilitzar de manera gratuïta, modificar-ne el codi si és necessari, i aconseguir les modificacions i correccions d'errors de forma gratuïta a cada nova versió.

Àmbit de les variables locals

L'ús general de les variables a Flash és locals de la funció i no de bloc. La màquina virtual de flash careix d'una definició de l'àmbit de les variables clara que provoca que exemples com els següents, que no compilarien en la majoria de llenguatges, funcionen:

```
function f() {
      if( true )
            var x = 1;
      return x + 1; // what about the above is false ?
}
```

Figura 31. *Exemple d'ús de variables compilable amb MMC*

Aquest exemple mostra com el compilador MMC permet l'ús d'una variable fora del rang del bloc perquè el rang d'ús de la variable està marcat per la màquina virtual de Flash.

```
function f() {
      var x = 1;
      if( true ) {
            var x = "hello";
            // ...
      }
      trace(x+1);
}
```

Figura 32. *Exemple d'ús de sobre-escriptura compilable amb MMC*

En canvi, l'exemple de la Figura 32 pintaria per pantalla *hello*, però no pintaria *2.*

Per tal de corregir aquestes indeterminacions que podrien provocar errors de compilació i sobretot moltes dificultats de depuració, MTASC prohibeix la sobre-escriptura de variables locals, el qual significa que no es pot redefinir una variable local amb el mateix nom, ni usar tampoc variables locals fora de l'àmbit normal d'altres llenguatges. La figura següent mostra un exemple que si seria compatible amb MTASC:

```
function f() {
      {
            var x : Number = 1; // ... use x as Number
      }
      // you cannot use x here
      {
            var x : String = "hello";  // ... use x as a String
      }
      // you cannot use x here
}
```

Figura 33. *Exemple d'ús correcte de variables locals per a compilar amb MTASC*

Definició de funcions locals

MMC permet la definició de funcions locals (Figura 34). MTASC només permet funcions locals anònimes, les quals poden ser guardades a una variable local (Figura 35). Això permet distingir la definició de les funcions locals de la definició dels mètodes.

```
function f(x) {
      function g(y) {
            return x+y;
      }
}
```

Figura 34. *Exemple de funció local compilable amb MMC*

```
function f(x) {
      var g = function(y) {
            return x+y;
      }
}
```

Figura 35. *Exemple de funció local compilable amb MTASC*

Utilitats de traces

MMC només proporciona la funció `trace` que permet treure per consola el text introduït, però només sota la consola del propi IDE. Això ha provocat que molta gent usi les seves pròpies funcions de traces. MTASC utilitza la funció `trace`, però la fa modificable en temps de compilació. Per exemple, si es realitzés la següent classe:

```
class Test {
    static function main(mc) {
        trace("hello world");}
}
```

Figura 36. *Exemple de la personalització de "trace" amb MTASC*

Al compilar utilitzant la comanda `mtasc -trace MyClass.myTrace Test.as (...)`, el compilador canviarà totes les crides a `trace` per crides a la classe `MyClass.myTrace` (la funció de traces personal) i afegirà més paràmetres de depuració, com si s'escrivís:

```
class Test {
    static function main(mc) {
        MyClass.myTrace("hello world","Test::main","Test.as",4);
    }
}
```

Figura 37. *Paràmetres extra afegits per MTASC en la personalització de traces*

Aquests tres paràmetres afegits pel compilador darrera del text de la traça són:

- El nom de la classe i el mètode en la qual s'ha realitzat la traça.
- El nom del fitxer en el que es realitza.
- La línia del fitxer en la que hi ha la traça

D'aquesta manera és molt més fàcil realitzar les funcions de traça pròpies i la depuració dels programes. L'ús d'aquests paràmetres implícits afegits per MTASC no limita l'ús d'altres paràmetres abans d'aquests. Aquests tres paràmetres sempre aniran després de tots els paràmetres marcats per l'usuari a la seva pròpia funció de traces.

A més, si es vol realitzar una versió de *release* i fer que no apareguin les crides a les funcions de traces, l'argument `-trace no` a la comanda de compilació permet treure-les d'una forma senzilla i ràpida.

Arrays de tipus definit

Una de les característiques d'ActionScript 2.0 que sorprèn als programadors acostumats a treballar amb objectes és que la definició d'arrays no va associada a cap tipus de variable en concret. MTASC introdueix una extensió en la sintaxi de definició d'arrays per a crear arrays que permetin només l'ús del tipus definit sense perdre compatibilitat amb ActionScript.

```
var x : /*String*/ Array;
```

Figura 38. *Exemple de definició d'array amb tipus definit*

D'aquesta manera, com a l'exemple de la Figura 38, l'array només permetrà l'emmagatzemament de cadenes de caràcters. Cal recalcar que no s'han de deixar espais entre el tipus de variable que es defineix i els símbols '/*' i '*/'.

També és important destacar que aquesta definició només afegeix comprovacions en temps de compilació i no provoca cap canvi en temps d'execució.

Variables locals amb l'argument `-infer`

Amb l'utilització de l'argument `-infer` a línia de comandes, quan s'usa una variable local amb un valor inicial, el tipus de la variable és automàticament associat per el del tipus del valor que se li està donant. D'aquesta manera, les definicions de Figura 39 i Figura 40 són totalment equivalents.

```
var x : String = "hello";
var x : Number = Math.cos(Math.PI);
var x : MyClass = new MyClass();
```

Figura 39. *Definició de variable local marcant el tipus*

```
var x = "hello";
var x = Math.cos(Math.PI);
var x = new MyClass();
```

Figura 40. *Definició de variable local obviant el tipus*

D'aquesta manera, l'ús de l'argument `-strict`, si és juntament amb `-infer`, no provoca errors de compilació per no definir la classe de la variable.

Altres canvis:

- Amb MTASC, les funcions que no tenen cap declaració de retorn de variable automàticament retornen `Void`.
- No suporta la inicialització de variables directament dins del cos de la classe, a no ser que sigui una constant o una expressió estàtica.
- L'ús d'MTASC no permet l'accés a les propietats del *Movieclip root.* Per exemple, si s'intenta accedir a la propietat *_parent* en una classe que no conté aquesta propietat, s'obtindrà un error.
- L'ús de la funció d'avaluació `eval("... ")` no funciona.

4.3.3 SWFMILL

4.3.3.1 Introducció

SWFmill [33], és un compilador escrit per Daniel Fischer i que permet l'exportació de SWF incloent-hi recursos com imatges, fonts o altres components. Són varis els programes que permeten l'exportació en SWF, SWFmill intenta emplenar el buit que provoca la no utilització de *Flash IDE*, preparant arxius SWF que ja estan preparats per afegir-hi codi compilat. És per tant una combinació perfecta amb MTASC, ja que permet afegir la compilació de codi AS realitzada amb MTASC al fitxer SWF prèviament compilat amb SWFmill.

Està publicat sota la llicència GPL i és capaç de convertir un arxiu SWF en un arxiu XML (escrit en un llenguatge anomenat SWFML) i viceversa. Això permet importar recursos dels següents formats i emmagatzemar-los en una llibreria:

- SWFs
- SWCs
- Fonts *TrueType*
- Imatges JPEG
- Imatges PNGs (24 and 32 bit, incloent-hi alpha)

A més, permet realitzar les següents accions:

- Crear *Movieclips* amb múltiples marcs que continguin altres *Movieclips.*
- Col·locar els recursos visuals importats en un lloc determinat de l'escenari i identificar-los per tal que puguin ser accessibles posteriorment.
- Proporcionar un control de quins recursos s'inclouen i a dins de quin marc s'han ubicat.

4.3.3.2 Instal·lació

Per a realitzar la instal·lació de SWFmill només cal descarregar l'aplicació que s'ofereix a la pròpia pàgina web del programa [33] i descomprimir-lo per poder ser usar des de línia de comandes. Si es vol utilitzar la crida al programa des de qualsevol ubicació, es pot afegir la ubicació de l'executable a la variable d'entorn PATH seguint els passos descrits a la instal·lació de MTASC.

4.3.3.3 Utilització

Creació d'un arxiu SWF basic

Per a la creació d'un SWF simple, només cal crear un arxiu XML com el de la Figura 41:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<movie width="320" height="240" framerate="12">
  <background color="#ffffff"/>
  <frame/>
</movie>
```

Figura 41. *Exemple de fitxer XML per a la creació d'un SWF simple*

Si guardem aquest fitxer amb el nom *prova.xml* i volem crear un arxiu de sortida *prova.swf,* la comanda a realitzar és la següent:

```
swfmill simple foo.xml bar.swf
```

Figura 42. *Comanda de creació d'un SWF simple*

Aquest procés crearia un SWF buit amb una mida de 320x240 píxels, de color de fons blanc i que té un temps de refresc de 12 imatges per segon.

Creació de la llibreria

Per a crear la llibreria de recursos que el fitxer SWF contindrà, es crea un fitxer XML com el mostrat a la Figura 43:

```
<frame>
  <library>
    <clip id="foo" import="library/foo.jpg"/>
  </library>
</frame>
```

Figura 43. *Exemple de llibreria de recursos amb SWFmill*

En aquest fitxer d'exemple es crea un directori *library*, on es posaran els recursos que es volen importar. En aquest cas s'importa una imatge que es troba al directori *library/foo.jpg*, i se li dóna un identificador de recurs (*foo*), per referenciar-lo des del codi ActionScript, per exemple mitjançant l'ús de comandes com *MovieClipattachMovie().*

Tant per a la importació d'imatges (JPEG o PNG), com de clips de vídeo (SWF) s'utilitza l'etiqueta *<clip>* tal com a la Figura 43.

Importació de fonts

En el cas que es vulguin utilitzar fonts especials que no estan instal·lades al sistema, SWFmill permet importar fonts a la llibreria, per poder-se utilitzar posteriorment en el nostre clip de vídeo de forma transparent. La comanda d'importació (Figura 44) és lleugerament diferent a l'anterior, i ve marcada per l'etiqueta *<font>*, on s'especifica el identificador que se li vol donar a la font dins de l'àmbit de la màquina virtual de Flash, la ruta de l'arxiu de font a importar, i els caràcters que es volen utilitzar.

```
<font id="vera" import="library/vera.ttf" glyphs="0123456789"/>
```

Figura 44. *Comanda d'importació de fonts*

Ús de llibreries externes

Si no és necessari crear una llibreria de recursos pròpia i es vol utilitzar una llibreria externa, amb SWFmill és possible importar-la mitjançant l'etiqueta *<import> :*

```
<import file="library/library.swf" url="http://foo.com/library.swf"/>
```

Figura 45. *Exemple d'importació de llibreria externa*

Exemple

A continuació, a la Figura 46, es mostra un exemple complet d'arxiu XML per a la creació d'una llibreria amb SWFmill. Al fitxer es pot observar l'especificació de la mida, el color de fons i la freqüència de refresc del *movieclip* base. Després s'importen una sèrie de recursos (imatges i clips de pel·lícula) i els caràcters numèrics de la font *Vera.* I finalment, l'import d'una llibreria externa *foobar.swf* i un nou frame on utilitza recursos que ha importat amb aquesta llibreria.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<movie width="320" height="240" framerate="12">
  <background color="#ffffff"/>
  <!-- first frame -->
  <frame>
    <!-- add some assets to the library -->
    <library>
      <clip id="picture" import="library/picture.jpg"/>
      <clip id="bitmap" import="library/bitmap.png"/>
      <clip id="clip" import="library/clip.swf"/>
    </library>
    <!-- import the numerical characters of vera.ttf -->
    <font id="vera" import="library/vera.ttf" glyphs="0123456789"/>
    <!-- import a shared library -->
    <import file="library/library.swf"
url="http://foo.com/library.swf"/>
  </frame>
  <!-- frame "myFrame" -->
  <frame name="myFrame">
    <library>
      <clip id="anotherClip" import="library/foobar.swf"/>
    </library>
  </frame>
</movie>
```

Figura 46. *Exemple complet d'ús de SWFmill*

4.3.4 AS Development Tools (ASDT)

4.3.4.1 Introducció

ASDT és un *plug-in* d'*Eclipse* de codi obert que serveix per desenvolupar programes per a la plataforma Flash.

És un editor que proveeix d'una integració completa amb *Eclipse* i els seus *plug-ins*, i permet una millor programació concurrent, depuració del codi, administració del contingut, revisió de la sintaxi i opcions d'autocompletar. També la integració amb MTASC per a realitzar la compilació i opcions de traces.

4.3.4.2 Instal·lació

Per a instal·lar el *plug-in* ASDT a l'entorn *Eclipse* s'han de seguir els següents passos:

1. Iniciar *Eclipse*.
2. Escollir al panell superior la opció: *Help > Software Updates > Find and Install*.
3. Seleccionar *Search for new features to install* i prémer *Next*.

4. Prémer *New Remote Site*.
5. Introduir *ASDT* a l'opció *Name*, *http://aseclipseplugin.sourceforge.nte/updates* a l'opció URL i prémer *OK*.
6. Marcar l'opció *ASDT* del llistat resultant, i prémer *Finish*.

Cal remarcar que per a la instal·lació del *plug-in* és necessari tenir connexió a Internet. Després de seguir aquests passos, *Eclipse* començarà a descarregar el *plug-in* i realitzar la seva instal·lació. Un cop acabat es demanarà que es reinicïi *Eclipse* per a completar la instal·lació.

4.3.4.3 Utilització

Preparació de les preferències del *plug-in*

Abans de començar a escriure el programa, és important definir les preferències del *plug-in*. Per això, cal seguir els següents passos:

1. Escollir l'opció del panell superior: *Windows > Preferences.*
2. A la pestanya *ActionScript 2 > Compiler*, escollir *MTASC*, a l'esquerra.
3. S'obrirà una finestra d'exploració. Buscar i seleccionar l'executable *mtasc.exe*, allà on estigui instal·lat (p.ex. *C:\Arxius de Programa\MTASC\mtasc.exe*).
4. Seleccionar a l'esquerra l'opció *Core Path* i buscar la ruta on estan guardades les classes intrínseques de *Flash*.
5. Seleccionar l'opció *Flashout* a l'esquerra i repetir el mateix procés.
6. Prémer OK per a aplicar els canvis.

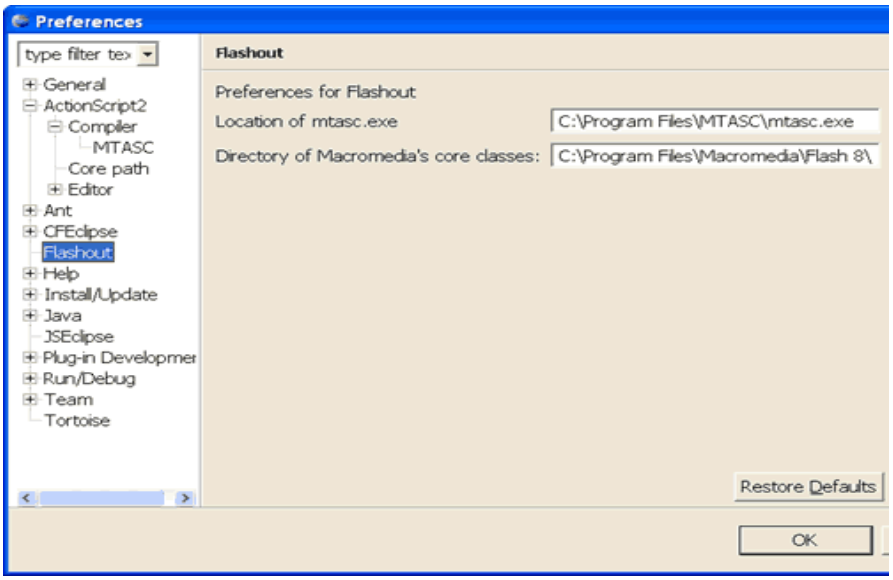


Figura 47. *Edició de les preferències d'ASDT*

Creació d'una aplicació d'exemple

En aquesta secció es realitzarà una aplicació *Hello World* d'exemple amb *ASDT*, que ajudaran a mostrar com generar un SWF sense utilitzar el *Flash IDE*.

Primer de tot, s'obre l'aplicació *Eclipse* i s'escull l'opció *File > New > New ActionScript 2 Project*. En el cas que aquesta opció no estigui disponible, és necessari canviar de perspectiva, amb l'opció: *Window > Open Perspective > Other* i seleccionar *ActionScript 2*.

S'introdueix un nom pel projecte, en aquest cas *FAME_tutorial*, i es selecciona *Finish*. En aquest punt, un directori anomenat *core* hauria d'aparèixer en la carpeta del projecte. Aquest directori és un directori virtual a on es troben les classes Flash (la ruta real és la introduïda a l'apartat anterior, en configurar les preferències)

Mitjançant la vista de navegació *d'Eclipse* es crea una nova carpeta a dins del projecte amb nom *deploy,* una amb nom *com* dins de la carpeta *src*. Dins de la carpeta com es crea una nova classe (botó dret del ratolí i *New > New ActionScript 2 Class*) amb el nom *HelloWorld.as.* S'introdueix el següent codi a aquesta classe:

```
class com.HelloWorld{
  private var tfOutput:TextField;
  private var mcScope:MovieClip;

  function HelloWorld(scope:MovieClip){
    mcScope = scope;
  }
  public static function main():Void{
    var hello:HelloWorld = new HelloWorld(_root);
    hello.sayHello();
  }
  public function sayHello():Void{
    trace("sayHello");
    trace("Stage.width="+Stage.width+";Stage.height="+Stage.height);
    mcScope.createTextField("tfOutput", 1, 50, 50, 250, 250);
    mcScope.tfOutput.html = true;
    mcScope.tfOutput.multiline = true;
    mcScope.tfOutput.wordWrap = true;
    mcScope.tfOutput.border = true;
    mcScope.tfOutput.htmlText = "<font face='Courier'
size='10px'>Hello World!</font>";
  }
}
```

Figura 48. *Exemple de classe HelloWorld*

Aquest exemple crea un camp de text HTML en temps d'execució i hi escriu el text *Hello World.*

4.3.5 ANT

Apache ANT [27] és una eina d'automatització de processos utilitzada per a la realització de tasques mecàniques i repetitives. És usada generalment durant les fases de compilació, construcció i desplegament del software.

És similar a la realització d'un *Make* [40], però desenvolupat en llenguatge Java i requereix de la plataforma Java. És usada per molts desenvolupadors Java i és inclosa de sèrie amb les últimes versions d'*Eclipse IDE,* a més de funcionar a la perfecció amb MTASC. Una altra diferència important respecte *Make* és la utilització d'arxius XML per a definir les tasques, per defecte emmagatzemades en un arxiu *build.xml.* Aquests fets donen l'avantatge de no ser depenent del sistema operatiu, resultant una gran eina multiplataforma.

Va ser creat per James Duncan Davidson quan realitzava la transformació d'un projecte de *Sun Microsystems* a codi obert (que després acabaria resultant en *Tomcat*). Davidson necessitava una eina que pogués realitzar el mateix que realitzava Make, però sota una plataforma no determinada (al voler-ho passar a codi obert). A partir d'aquest punt, l'eina va anar adoptant noves funcionalitats i actualment és un estàndard de l'entorn Java.

4.3.5.1 Instal·lació

Per a utilitzar ANT només és necessari disposar d'una distribució binaria d'ANT i tenir instal·lat la versió 1.4 del *Java Development Kit* (JDK). No confondre, però, Ant amb una extensió de JDK i instal·lar-la al directori *lib/ext* d'aquesta, ja que això podria portar problemes de seguretat. La instal·lació d'ANT s'ha de realitzar com qualsevol altre aplicació.

Qualsevol distribució binaria d'ANT hauria de tenir la següent estructura de directoris:

```
ant
  +--- bin  // conté l'executor d'scripts
  |
  +--- lib  // conté els .jars (llibreries)i altres dependències
  |
  +--- docs // documentació
  |      +--- ant2    // curta descripció dels requeriments d'ant2
  |      |
  |      +--- images  // varis logos per la documentació en HTML
  |      |
  |      +--- manual  // Manual d'ús d'ANT
  +--- etc
```

Figura 49. *Estructura de directoris d'una distribució ANT*

4.3.5.2 Configuració

Abans d'utilitzar Ant, s'han de configurar varies de variables d'entorn. S'ha d'afegir el directori *bin* a la variable *PATH* (per poder executar ANT des de qualsevol directori); afegir la variable *ANT_HOME* amb el directori on s'hagi instal·lat Ant (per a trobar l'executable llançador dels *scripts*); i revisar que la variable *JAVA_HOME* estigui definida amb el directori on està instal·lat el JDK (per resoldre les dependències d'Ant amb el JDK de Java). Per a realitzar aquest procés es pot seguir els mateixos passos explicats a la configuració d'MTASC per a la configuració de variables d'entorn o es poden utilitzar les següents comandes (Figura 50, Figura 51 i Figura 52) segons el sistema operatiu:

Windows and OS/2 (Suposant que s'ha instal·lat ANT a *c:\ant*)

```
set ANT_HOME=c:\ant
set JAVA_HOME=c:\jdk1.2.2
set PATH=%PATH%;%ANT_HOME%\bin
```

Figura 50. *Configuració de les variables d'entorn d'ANT a l'entorn Windows i OS/2*

Unix (bash) (Suposant que s'ha instal·lat ANT a */usr/local/ant*)

```
export ANT_HOME=/usr/local/ant
export JAVA_HOME=/usr/local/jdk-1.2.2
export PATH=${PATH}:${ANT_HOME}/bin
```

Figura 51. *Configuració de les variables d'entorn d'ANT a l'entorn Unix (bash)*

Unix (csh)

```
setenv ANT_HOME /usr/local/ant
setenv JAVA_HOME /usr/local/jdk-1.2.2
set path=( $path $ANT_HOME/bin )
```

Figura 52. *Configuració de les variables d'entorn d'ANT a l'entorn Unix (csh)*

Per a comprovar que la instal·lació i la configuració s'han realitzat correctament, es pot escriure per `ant` línia de comandes, i el resultat hauria de ser el següent:

```
Buildfile: build.xml does not exist!
Build failed
```

Figura 53. *Comprovació de la configuració d'Ant*

Aquest missatge vol dir que la configuració és correcte i que ant s'està executant correctament, però que no troba el fitxer `build.xml que hauria d'executar.`

4.3.5.3 Utilització

Execució d'ANT

Per executar ANT només és necessari escriure *ant* per línia de comandes. Això executarà per defecte el fitxer *build.xml* que estigui al directori on s'ha executat. Si es volgués utilizar un fitxer XML diferent d'aquest, es pot utilitzar la comanda:

```
ant -buildfile <fitxer_alternatiu>.xml
```

Figura 54. *Comanda ant d'un arxiu XML alternatiu*

També es poden realitzar tasques concretes del fitxer, en lloc de la tasca que està definida per defecte al fitxer XML. Per exemple, si només es vol realitzar una tasca que s'anomena compila, es pot escriure:

```
ant compila
```

Figura 55. *Execució ANT d'una tasca concreta*

En aquest cas s'ignorarà la tasca per defecte de l'arxiu *build.xml* i només s'executarà l'objectiu especificat.

Estructura d'un fitxer *build.xml*

Els fitxers de tasques d'ANT estan escrits en XML. A la Figura 56 es mostra un exemple d'aquest tipus de fitxers:

```
<project name="MyProject" default="compile" basedir=".">
   <description>
       simple example build file
   </description>
 <property name="src" location="src"/>
 <property name="build" location="build"/>
<target name="init">
   <tstamp/>
   <mkdir dir="${build}"/>
 </target>
 <target name="compile" depends="init" description="compile the source
" >
   <javac srcdir="${src}" destdir="${build}"/>
 </target>
<target name="clean" description="clean up" >
   <delete dir="${build}"/>
</target>
</project>
```

Figura 56. *Exemple de fitxer build.xml*

Cada fitxer conté un projecte (*project*) i com a mínim un objectiu (*target*). A la vegada, cada objectiu pot contenir varies tasques (*task*), que són fragments de codi a executar. Un projecte a més pot contenir diverses propietats (*property*), que consten d'un nom i un valor, i són usats com a constants.

A l'inici de l'exemple es poden observar els següents atributs de l'etiqueta <*project*>:

- `Name = "MyProject"` , que identifica el nom del projecte.
- `default = "compile"` , que especifica l'objectiu que s'ha d'executar per defecte si no s'ha especificat cap en concret.
- `Basedir = "."` , directori base sobre el qual es treballarà. En aquest cas s'especifica el directori actual.

Després s'especifiquen un llistat de propietats on s'especifica la localització dels atributs *src* i *build*, que s'utilitzaran posteriorment en la definició dels objectius.

A continuació apareixen els objectius. En aquest cas apareix l'objectiu *init*, on s'especifiquen les tasques *tstamp* i *mkdir*, les quals s'encarreguen de mostrar la data i crear el directori descrit per la propietat *build* respectivament.

El següent objectiu que apareix és *compile*, que conté l'atribut *depends.* Aquest atribut és especialment interessant, doncs especifica les dependències amb altres objectius del fitxer. En cas de posar com a objectiu per defecte *compile*, si aquest depèn d'altres objectius, ANT executarà primer totes les dependències abans d'executar *compile*. El cos de l'objectiu està format per la tasca *javac*, que s'encarrega de compilar en Java la font especificada amb l'atribut *srcdir* i extreu els *.class* al directori *destdir*.

I per últim hi ha l'objectiu *clean*, que conté la tasca *delete*, encarregada d'esborrar el directori temporal *build*. No hi ha cap dels altres objectiu que depengui d'aquest objectiu, i la capçalera del projecte tampoc l'especifica com a objectiu per defecte, fet pel qual s'ha d'executar de forma explícita en cas de voler realitzar aquesta tasca.

Tasques específiques per ActionScript

La llista de tasques [41] que venen per defecte amb la instal·lació d'ANT és molt extensa. Tot i així, per a integrar ANT amb MTASC i SWFmill, existeixen tasques específiques que ofereixen un ús molt més natural i fàcil d'aquestes eines amb ANT [42]. Algunes d'aquestes tasques són:

- <*mtasc*> per la compilació de codi usant el compilador MTASC
- <*swfmill*> per processar, amb SWFmill, SWFs basats en XML
- <*swf*> per a crear arxius SWFs amb una llibreria de recursos i compilar les classes dins seu.

Algunes de les opcions més destacables de la tasca *mtasc* són :

- *main* : Aquest atribut igual a true indica que la opció –main està activada.
- *Version* : Indica la versió de *Flash* en la qual es vol compilar.
- *Src* : Les classes font que es volen compilar
- *Classpath : La ubicació de les llibreries i classes que són dependències de la font.*
- *Swf* : Fitxer de sortida SWF que es vol actualitzar

No obstant, l'ús d'aquestes tasques específiques no és obligatori, i són substituïbles mitjançant la tasca <exec> que ve per defecte amb la distribució d'ANT, i permet l'execució de fitxers executables, però l'ús d'aquest mètode pot portar algun problema de portabilitat.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="SWFCompiler" basedir="." default="build">
      <taskdef name="mtasc" classname="org.as2lib.ant.Mtasc" />
      <property name="mtasc" location="mtasc/mtasc" />
      <property name="swfmill" location="swfmill/swfmill" />
      <property name="lib.dir" location="lib" />
      <property name="src.dir" location="src" />
      <target name="build" depends="swfmill, mtasc"
description="Compile actionscript"/>
      <target name="swfmill">
            <exec executable="${swfmill}">
                  <arg value="-V"/>
                  <arg value="simple"/>
                  <arg value="${output.dir}/${library.name}.xml"/>
                <arg value="${output.dir}/${application.name}.swf"/>
            </exec>
      </target>
      <target name="mtasc">
            <mtasc mtasc="${mtasc}"
            version="8"
            main="true"
            src="${output.dir}/${application.name}.as"
            classpath="${src.dir} ; ${lib.dir} ; ${output.dir} ;"
            swf="${output.dir}/${application.name}.swf"/>
      </target>
</project>
```

Figura 57. *Fitxer d'exemple d'ús d'ANT amb MTASC i SWFmill*

A la Figura 57, per exemple, es pot veure els diferents usos possibles, amb la utilització de la tasca *mtasc*, però l'ús de la tasca *exec* per a l'execució d'SWFmill.

4.3.5.4 Portabilitat

Un dels primers avantatges que va aportar l'ús d'ANT va ser solucionar els problemes de portabilitat que oferia *Make*. En un arxiu *Makefile* les accions necessàries per a crear un objectiu s'especifiquen com a ordres de l'intèrpret de comandes, que són específics per a cada plataforma. Ant resol aquest problema proveint d'un gran nombre de tasques per ell mateix, que garanteixen l'execució independentment de la plataforma. Per exemple, l'ús de la comanda *<delete dir="classes/>* és traduïda per ANT a *rm –rf classes* en un entorn Unix, i *rd /s classes* en un entorn Windows.

4.3.5.5 Limitacions

El principal problema d'ANT ve donat pel fet d'utilitzar fitxers XML, que ofereix problemes d'escalabilitat en projecte molt grans, resultant en fitxers XML grans i complexos.

La majoria d'eines més antigues d'ANT (com per exemple, *exec*, *java* i *javac*) tenen males configuracions per defecte, i valors d'opcions no coherents amb altres tasques que s'han creat més recentment.

Una altra deficiència a causa de la compatibilitat enrere és que quan s'expandeixen les propietats en una cadena o un element de text, les propietats no definides no són plantejades com un error, sinó que es deixen com una referència sense expandir (p.ex: *{$unassigned.property}*).

A més, ANT no és un llenguatge per a un flux de treball general. No hauria de ser usat d'aquesta forma, ja que té una administració de control d'errors limitat, i no té persistència d'estat, amb la qual cosa no pot ser usat per a realitzar una construcció de varis dies fiable.

5. IMPLEMENTACIÓ PRÀCTICA

5.1 DISSENY

5.1.1 Intèrpret i conversor

El desenvolupament del conversor es va realitzar en dues fases. En la primera d'elles es va escriure completament en ActionScript (AS) 2.0 (llenguatge base de Flash Lite 3) un intèrpret de descripcions DVB-PCF. Com a resultat, l'intèrpret podia carregar de forma dinàmica descripcions i renderitzar-les en Flash Lite. En aquesta fase permetia la càrrega dels elements visuals bàsics d'un servei interactiu (escenes simples amb imatges i quadres de text) i una navegació simple basada en menús senzills.

Per a la creació d'aquest intèrpret es va crear un model de dades en AS 2.0 que representés els elements bàsics de PCF i una sèrie de classes que en permetés la seva càrrega. El seu funcionament feia necessari que l'intèrpret anés acompanyat sempre de l'arxiu XML amb la descripció PCF per tal de veure l'aplicació interactiva definida. Aquest, és un fet interessant perquè permet canviar dinàmicament la descripció PCF que acompanya l'intèrpret i l'aplicació es recarrega amb aquesta nova descripció.

No obstant, per interessant que pugui resultar el fet de modificar dinàmicament l'aplicació només canviant un fitxer XML, això podria generar alguns problemes de seguretat en alguns entorns, per la llibertat que es pot donar a l'usuari de canviar dinàmicament les aplicacions.

Tenint en compte l'actual model econòmic de serveis iTV, no és interessant per als proveïdors de serveis donar la possibilitat als usuaris de modificar aplicacions per si mateixos. Això suposaria com proveir del codi font de les seves aplicacions als clients i el permís de modificar-les sense restriccions. En algun altre àmbit podria ser interessant donar l'oportunitat de crear noves característiques d'un servei només modificant l'arxiu XML, però això suposaria la fi del mercat de serveis iTV tal com està concebut fins ara.

És per això que en una segona fase, es va dissenyar un conversor complet, que permetés la creació de serveis iTV autocontinguts, sense la necessitat de proporcionar també la descripció en XML. Això ofereix més flexibilitat en alguns aspectes, ja que com a proveïdor de serveis, permet proporcionar aplicacions autònomes als usuaris.

Com a resultat del desenvolupament d'aquest disseny, s'obté un nou conversor que només necessita la descripció i els recursos audiovisuals en temps de compilació, moment en que crea un arxiu Adobe Flash SWF, amb l'aplicació autocontinguda, preparada per ser enviada als usuaris finals i reproduïda als seus descodificadors de TV digital.

5.1.2 Diagrama de paquets

Aquesta secció detalla l'estructura del conversor i com va ser dissenyat en la segona fase esmentada a la secció anterior. Primer de tot, s'estableixen els diferents mòduls que formarien part del conversor, tenint en compte la possibilitat d'un canvi de tecnologia, especialment en aquells aspectes on la tecnologia emprada és més específica d'una plataforma, com el cas de Flash Lite. És per això que el disseny utilitzat intenta crear una divisió de mòduls de tal forma que un conjunt d'aquests es mantindria pràcticament igual en el cas de realitzar el conversor per a una altra plataforma final.

Els paquets definits són els següents (també mostrats a la Figura 58): El diagrama de paquets representa tots els paquets emprats en la implementació del conversor i les relacions entre ells.

- **DVB-PCF ActionScript 2.0 Data Model** està format per un conjunt de classes *ActionScript* 2.0 que representen els objectes que poden estar definits en una descripció PCF.
- **LoadManager** és el paquet que conté totes les classes relacionades amb la càrrega i visualització de la informació de la descripció PCF, incloent aspectes com la navegació i la càrrega d'imatges i altres recursos.
- **XMLParser** inclou totes les classes relacionades amb la interpretació de la descripció PCF i fa ús dels dos paquets anteriors per a crear un (o diversos) arxius *ActionScript* (AS) que continguin la definició i la càrrega dels objectes especificats a la descripció.
- **ImageLibrary** conté les classes encarregades de la importació de les imatges i altre material audiovisuals per a crear una llibreria de recursos que pugui ser utilitzat pel paquet *LoadManager*.
- **ConverterManager** és el mòdul amb les classes encarregades de controlar totes les fases de la conversió i mitjançant els *outputs* creats des dels paquets *XMLParser* i *ImageLibrary*, crear l'arxiu final SWF (Shockwave Flash), que és autoexecutable i autocontingut.

Els dos primers paquets estan escrits completament en llenguatge ActionScript 2.0 i els tres següents en Java.

Per intentar aconseguir la manera d'adaptar que el servei final *renderitzat* fos el més exacte possible a la descripció del servei, el primer pas de l'etapa de disseny va ser un extens estudi de tots els elements que estaven definits a l'estàndard DVB-PCF, quins eren els més adients per a una primera versió del conversor, i com podien ser mapejats en paràmetres de Flash.

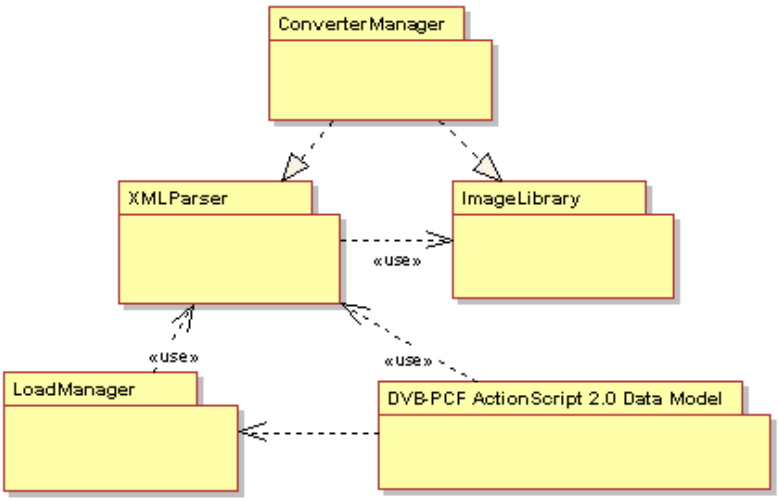


Figura 58. *DVB-PCF to Flash Lite Converter Packages*

L'estàndard PCF bàsicament ha de proporcionar eines que permetin als autors de serveis interactius descriure l'experiència que tindrà l'usuari. És per això que es defineixen un gran nombre d'elements fonamentalment visuals. Tal com s'ha revisat en l'apartat 3.3, Adobe Flash Lite 3 utilitza AS 2.0, fet pel qual és una plataforma limitada en comparació al Flash Player suportat per dispositius més potents. Per aquest motiu es va realitzar un estudi previ de la viabilitat de les característiques bàsiques definides per l'estàndard PCF mitjançant la combinació de components AS 2.0.

Com a resultat d'aquest estudi, s'ha creat un model de dades i un sistema de càrrega que reprodueix gran quantitat dels elements que defineix DVB-PCF. Aquest paquet defineix els elements PCF (*Service, Scene, Background, Image, TextBox, Menu*, etc.) i proveint els paràmetres que poden formar part d'aquests ítems, segons la seva definició a l'estàndard. La possibilitat d'inclusió de tots els paràmetres i d'un mapeig a AS 2.0 adequat dóna com a resultat una representació més fidedigna de l'element.

No obstant, la intenció d'aquest paquet és ser únicament contenidors que mantinguin la informació dels elements PCF definits, i per tant no incorporen els mètodes de visualització d'aquests elements i les seves característiques en Flash.

Un cop definit aquest model de dades es van definir dos paquets que s'encarreguen d'agafar tots els paràmetres guardats en cada element i definir una càrrega de l'element el més exacta possible respecte la descripció. Aquests paquets s'encarreguen (1) de parsejar la descripció PCF, carregar tota la informació de la descripció en temps de compilació i crear-ne arxius AS 2.0 amb l'equivalent en aquest llenguatge; i (2) carregar tots els recursos audiovisuals definits a la descripció i crear-ne una llibreria. Els arxius AS 2.0 creats, la llibreria de recursos, juntament amb el model de dades i les funcions de

renderització de LoadManager, són els elements que permeten la compilació final de l'aplicació.

5.1.2.1 DVB-PCF Data Model

Les descripcions PCF estan formades per estructures jeràrquiques amb ítems d'informació, la majoria d'ells representats com a PCF Components i cada ell amb un identificador únic. Tots deriven de l'ítem *service* que és l'element arrel i sota el qual es poden definir múltiples escenes. És important emfatitzar el fet que el servei no deu mostrar més d'una escena simultàniament en el seu cicle de vida i que totes les escenes que poden ser visualitzades han d'estar dins de *service* o algun sub-contenidor d'aquest.

Tal com s'ha mostrat prèviament a la secció 3.2.4, tots els aspectes d'una descripció PCF bàsica poden ser resumits en components, contingut i comportament. Per a suportar aquests tres tipus d'informació i proveir d'una experiència real d'un servei interactiu de televisió, es van definir un conjunt mínim d'elements PCF. Aquest conjunt inclou components visuals (*Scene, Image, Textbox, Rectangle*, etc.), components d'informació bàsica (*Size, Position, Color*, etc.), elements de comportament (*OnEvent, SceneNavigate, Target*) i altres que involucren aspectes visuals però que també estan directament relacionats amb el control del cicle de vida del servei (*Service, Scene, Menu, Button*).

5.1.2.2 Load Manager

La manera amb que són renderitzats tots els objectes en una aplicació interactiva és independent de com s'han creat. Això implica que es necessitaran funcions de càrrega específiques per a cada objecte. El paquet *LoadManager* conté un conjunt de classes, amb les funcions de càrrega de cada un dels elements visuals o de comportament definits al model de dades (els elements d'informació bàsica no requereixen d'una funció *Load* més enllà de la càrrega de la informació de la que consten).

Quan la complexitat dels elements DVB-PCF inclosos augmenta, és necessari una major combinació d'objectes ActionScript per a realitzar una representació més acurada de l'element. Aquest paquet, juntament amb el model de dades ja va ser creat en la primera fase de l'intèrpret PCF esmentat a la secció 5.1.1.

5.1.2.3 Parser

El mòdul de parseig de la descripció és un paquet de classes implementades completament en Java. Aquest s'encarrega de llegir i interpretar l'arxiu XML amb la descripció PCF i crear un arxiu AS 2.0 que realitza crides simples a les classes del model de dades i la seva funció de càrrega específica de *LoadManager*. Aquesta separació és la que permet el canvi dels mòduls d'AS 2.0 per altres d'una altra tecnologia i la possibilitat de reaprofitar els mòduls de Java per a la realització d'un altre conversor.

5.1.3 Diagrama de classes

5.1.3.1 Classes del model de dades ActionScript 2.0

En aquesta secció es detallen els diagrames que representen l'estructura de totes les classes necessàries per a reproduir els elements d'una descripció PCF de forma genèrica. En aquests també es mostra quines són les relacions entre les classes.

PCFItem

Aquesta classe representa un ítem PCF genèric. L'estructura d'aquesta classe conté tota la informació (nom, context i referència) relacionada amb l'element *PCFItem* de l'estàndard DVB-PCF. És la classe mare, de la qual hereten gran part dels altres elements.

PCF

Aquesta classe representa l'element *PCF* d'una descripció DVB-PCF. *PCF*, com a element arrel de la descripció, conté l'element Service i defineix el *NameSpace* [43] o àmbit en que està definit l'arxiu XML de la descripció.

Service

La classe Service representa un objecte *PCF Service.* Aquesta hereta de l'objecte *PCFItem* i l'estructura de la classe conté, entre d'altres paràmetres, el nom del servei, un atribut URI que indica l'escena inicial del servei i la mida (resolució) de la pantalla de referència.

Scene

Aquesta classe representa un objecte *PCF Scene.* Hereta de la classe *PCFItem* i pot contenir un objecte *Background* i un objecte *OnEvent*. Tot i així, és un objecte merament organitzador, ja que farà de contenidor d'altres objectes, simplificant l'administració (activació, descativació, ocultació i aparició) de tots els objectes que en formin part.

Position

Aquesta classe representa un objecte *PCF Position.* Aquest objecte defineix la posició en els eixos vertical i horitzontal d'un component visual, tenint en compte com a origen de coordenades l'extrem superior esquerre de la pantalla.

Size

Aquesta classe representa un objecte *PCF Size*. Defineix els atributs d'alçada i amplada d'un objecte visual.

PCFColor

Aquesta classe representa un objecte *PCF Color*. Conté una cadena de caràcters iniciada pel símbol '#', amb els sis següents caràcters definint els valors hexadecimals del color en RGB i els dos següents el valor d'*alpha* (transparència). En el cas de la transparència 0 significa no visible, i "FF" significa completament opac.

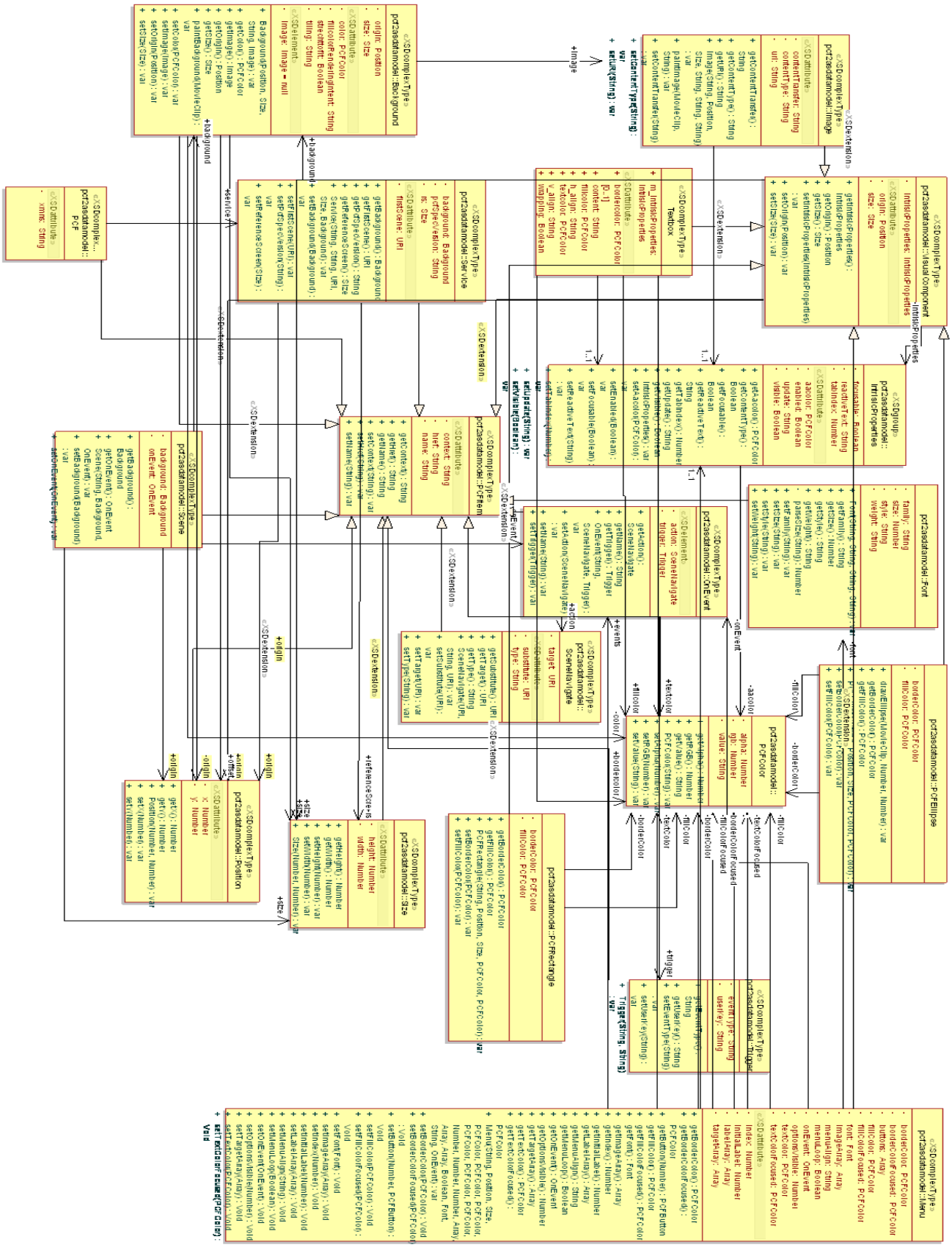


Figura 59. *Diagrama UML del model de dades Actionscript*

URI

Aquesta classe representa un objecte *PCF URI*. Mitjançant una cadena de caràcters, defineix la ubicació d'un recurs o una escena.

VisualComponent

Aquesta classe representa la classe mare de qualsevol objecte visual PCF. Serveix de base als objectes *Image, Menu, TextBox, Button*, etc. Conté els atributs vector *Position* and *Size* per especificar la mida i la posició origen de l'objecte, a més d'un objecte *Intrinsic_Properties-Visible-Components*.

Intrinsic_Properties-Visible-Components

Classe que engloba les principals propietats comuns que tenen la majoria dels objectes visibles. Instanciada únicament per *VisualComponent*.

Image

Classe que representa una imatge, definida per l'objecte PCF *Image.* Hereta els atributs de *VisualComponent* i afegeix un atribut *string* amb el tipus d'imatge i una URI amb la ubicació de la imatge.

Textbox

Aquesta classe representa un quadre de text, definit per un objecte PCF *Textbox*. Hereta els atributs de *VisualComponent* i afegeix un atribut *content* que és una cadena de caràcters amb el text que mostrarà, objectes Color que especifiquen el color de línea, de fons i de la lletra, el tipus de font amb què es representarà la lletra i l'alineació del text dins del quadre.

PCFButton

Aquesta classe representa un botó, definit per un objecte PCF *Button*. Hereta els atributs de *VisualComponent* i afegeix un atribut *label* que és una cadena de caràcters amb el text que mostrarà, objectes Color que especifiquen el color de línea, de fons i de la lletra, el tipus de font amb què es representarà la lletra i l'alineació del text dins del botó. Aquest objecte també ofereix la possibilitat de mostrar una imatge de fons en comptes d'un color pla.

PCFRectangle

Classe que representa un rectangle, definit per un objecte PCF *Rectangle*. Hereta els atributs de *VisualComponent* i afegeix els atributs de tipus Color, *borderColor* i *FillColor,* que especifiquen el color de Color de línea i de fons del rectangle.

PCFEllipse

Classe que representa una el·lipse, definit per un objecte PCF *Ellipse*. Hereta els atributs de *VisualComponent* i afegeix els atributs de tipus Color, *borderColor* i *FillColor,* que especifiquen el color de Color de línea i de fons de l'el·lipse.

Background

Classe que representa el fons de tot un servei o d'una escena, i ve definit per un objecte PCF *Background*. Hereta els atributs de *PCFItem* i afegeix els atributs *origin* (*Position*) i *size* i *FillColor*, a més d'un objecte Image per si es volgués una imatge de fons en lloc d'un color pla.

OnEvent

Classe que representa un event, definit per un objecte PCF *OnEvent*. Hereta els atributs de *PCFItem* i a més conté atributs *action* (de tipus *SceneNavigate*) i *trigger* (*Trigger*).

SceneNavigate

SceneNavigate és la classe que especifica quina acció prendre quan un event ha estat activat. El nom d'aquesta classe ve donat perquè en la versió actual del conversor les accions a prendre es basen únicament en canvis d'escena, i és molt possible que en següents versions *SceneNavigate* sigui part d'un objecte entremig de tipus *Action.* Aquesta classe aporta com a atribut destacable un objecte URI que especifica l'escena que s'ha de visualitzar quan es produeixi l'event.

Trigger

Trigger és la classe que especifica quan és activat un event. Aquesta classe aporta com a atribut destacable un objecte URI que especifica l'escena que s'ha de visualitzar quan es produeixi l'event.

Menu

Aquesta classe representa un menú simple com un conjunt de botons, tal com està definit per un objecte PCF *Menu*. Hereta els atributs de *VisualComponent* i afegeix un atribut *labelArray* (array d'*strings* amb el text que mostraran els botons), *targetArray* (array d'objectes *SceneNavigate* indicant a quina escena cal anar quan es premi cada botó del menú), *imageArray* (array amb les imatges de fons de cada un dels botons), els objectes Color que especifiquen el color de línea, de fons i de la lletra quan els botons estan en menú i quan estan seleccionats, el tipus de font amb què es representarà la lletra i l'alineació del text dins del botons. A més, també afegeix un booleà per especificar si la disposició del menú és vertical o horitzontal i un altre per especificar si el menú és cíclic o no.

5.1.3.2 Classes de LoadManager

LoadManager

La classe *LoadManager* conté tots els mètodes necessaris per a la càrrega de qualsevol objecte visual, generalment creant un objecte *MovieClip* d'AS 2.0 que mostri en Flash tot el que l'objecte *VisualComponent* defineix en DVB-PCF

ControlFunctions

Classe que conté les funcions de control de la visibilitat dels components i la navegació entre escenes. A més, també s'encarrega de tot el control d'events que poden afectar al servei global a una escena o a un botó en concret.

Constants

La classe Constants conté la definició de tots els valors de constants que són utilitzats en l'aplicació. A més, en aquesta classe també són mapejats els valors de les tecles del control remot.

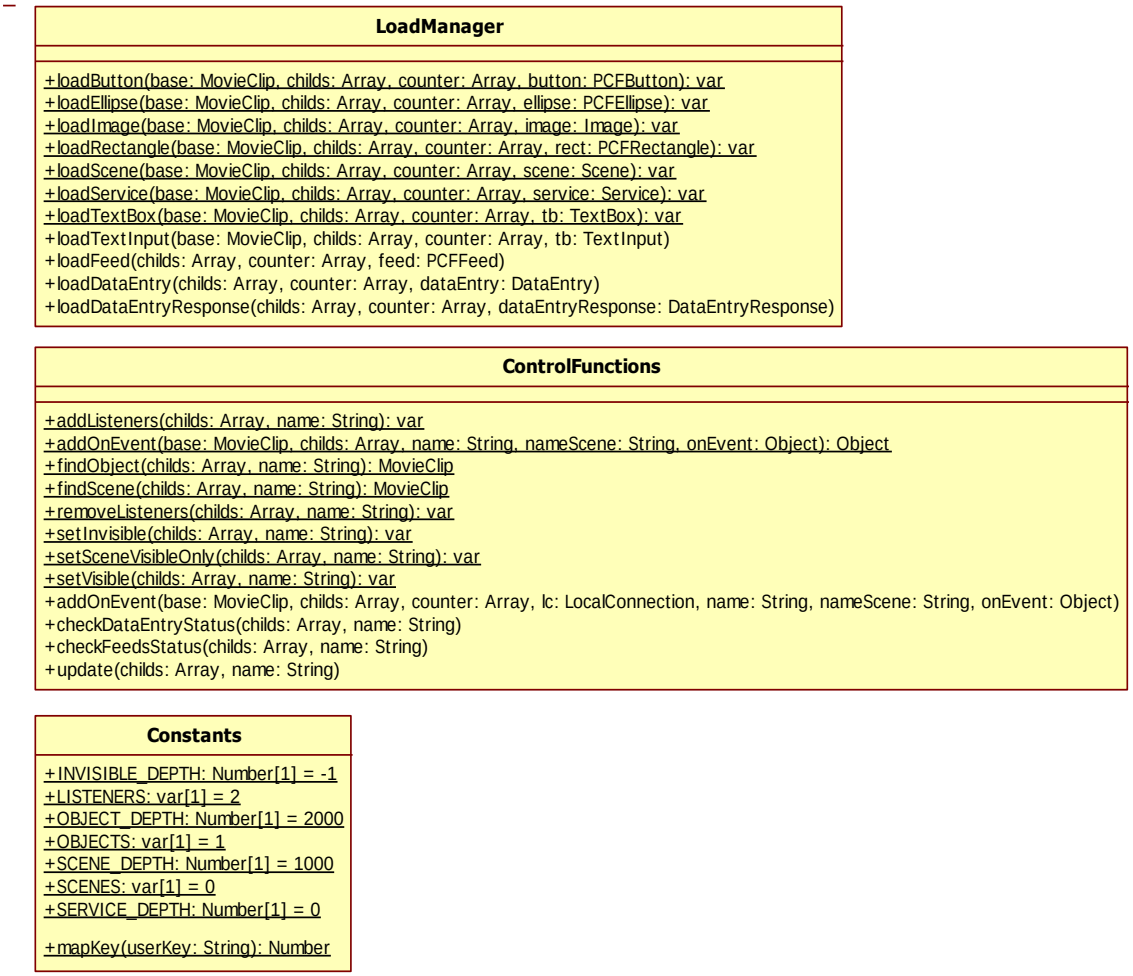


Figura 60. *Classes principals del paquet Load Manager*

5.1.3.3 XMLParser

Aquest paquet de classes conté una única classe *PCFParser* encarregada de llegir el fitxer XML amb la descripció PCF i crear arxius .as amb les crides als mòduls de model de dades i *LoadManager*.

5.1.3.4 ImageLibrary

Aquest paquet conté principalment una classe anomenada *ImageLibraryCreator* que s'encarrega de cercar a la descripció PCF tots els recursos audiovisuals (fins al moment d'aquesta memòria imatges, sons, vídeos i altres arxius Flash SWF) que estan definits, i crear un arxiu XML que farà les funcions de la llibreria de recursos del Flash IDE.

El paquet també conté les classes encarregades d'agafar aquests recursos i copiar-los en la màquina de compilació si és necessari.

5.1.3.5 ConverterManager

ConverterManager està format per una classe principal *Convert* que s'encarrega de gestionar tot els processos associats a la conversió, mitjançant la crida a les classes dels altres mòduls, des del parseig de la descripció, la creació dels arxius AS i la obtenció dels recursos, fins a la compilació de l'arxiu final SWF.

5.1.4 Eines de desenvolupament

El desenvolupament d'aquest conversor està realitzat completament per a ser multiplataforma i amb eines de codi obert. La seva implementació ha estat realitzada amb *l'Eclipse IDE*, amb el *plug*-in *ASDT* (*ActionScript Development Tool*). Per a la compilació s'ha utilitzat els compiladors de Flash de codi lliure *Mtasc* i *SWFMill* i l'eina *Ant* d'*Apache*.

Ha estat provat (i desenvolupat en algun moment) sobre els sistemes Windows (versió XP) i Linux (distribució *Mandriva 2009*). Les aplicacions resultants del conversor han estat provades utilitzant el *Flash Lite 3.1 Player* i *Flash 10 Player* en varis entorns i dispositius (varis ordinadors *Pentium Core 2 Duo* i *Pentium IV* amb una velocitat de processador de 2.2 GHz i 1GB de memòria RAM o varis tipus de *set-top box*).

5.1.5 Preparació de l'entorn

Per a la preparació de l'entorn es requereix la instal·lació d'Eclipse i també de les diferents eines de desenvolupament Flash de forma lliure (*ASDT, MTASC, SWFMill* i *ANT*), tal com s'especifica a la secció 4.3. Addicionalment a aquests passos, per a la integració entre *Eclipse*, *Ant* i l'ús de tasques específiques de compilació d'*ActionScript* amb *Ant*, s'ha d'afegir al *classpath* dels diferents projectes la llibreria *as2ant.jar* [44]. D'aquesta manera,

en els arxius de compil·lació amb Ant (*build.xml*) es podran utilitzar crides a tasques pròpies d'aquest entorn com *mtasc* o *swfmill.*

5.1.6 Implementació

En aquesta secció es repassaran alguns dels aspectes més rellevants de la implementació, així com s'explicaran en detall algunes de les classes principals del conversor.

5.1.6.1 Control de la conversió

El procés de conversió està dividit en varies fases que venen controlades per la classe *Converter*. Primer de tot, la classe és inicialitzada amb una sèrie de rutes que indiquen la ubicació de la descripció PCF i on volem generar el fitxer de sortida SWF.

```
public Converter(String _inputPath, String _inputFile, String
_AS2SWFPath, String _outputPath, String _outputFile){
      inputPath=_inputPath;
      inputFile=_inputFile;
      AS2SWFPath=_AS2SWFPath;
      outputPath=_outputPath;
      outputFile=_outputFile;
```

Figura 61. *Inicialització de la classe Converter*

Després, es crea una classe PCFParser que será l'encarregada de parsejar la descripció PCF i crear els arxius .as, i una clase ImageLibrary que s'encarregarà també de llegir la descripció PCF, buscar totes les referències a recursos externs i preparar l'arxiu XML que tractarà el compilador SWFMill.

```
PCFParser parser;
parser = new PCFParser();
parser.ParsePCF2AS(inputPath+inputFile,outputPath,outputFile);
ImageLibrary.createLibrary(inputPath,inputFile,outputPath,outputFile);
```

Figura 62. *Inicialització de la classe PCFParser i ImageLibrary*

I per últim, un cop ja s'han creat els arxius .as i la llibreria XML, per a realizar la compilació es crida el fitxer Ant mitjançant el següent codi:

```
      private void runAnt(String AS2SWFPath, String outputPath, String
outputFile){
      File buildFile = new File(AS2SWFPath+"build.xml");
      Project p = new Project();
      p.setUserProperty("ant.file", buildFile.getAbsolutePath());
      p.setUserProperty("application.name", outputFile);
      p.setUserProperty("library.name", outputFile+"_library");
      p.setUserProperty("output.dir", outputPath);
```

```
	p.init();
	System.out.println("After set properties and init");
	ProjectHelper helper = ProjectHelper.getProjectHelper();
	p.addReference("ant.projectHelper", helper);
	helper.parse(p, buildFile);
	try{
		p.executeTarget("swfmill");
		p.executeTarget("mtasc");
	}catch(BuildException e){
		e.printStackTrace();
	}
}
```

Figura 63. *Execució de la compil·lació amb Ant des de codi Java*

5.1.6.2 Parseig de la descripció DVB-PCF

La classe PCFParser d'encarrega de realitzar el parseig de la descripció PCF i crear arxius AS 2.0 que representin els elements que conté la descripció. Per a això, el primer que fa és crear una capçalera per al fitxer principal que sempre serà igual, i que servirà per iniciar l'estructura de classes que emmagatzemaran la informació de l'aplicació resultant (Figura 64).

```
String header="import com.pcf2asdatamodel.*;\n"+
              "import com.loadmanager.*;\n\n"+

             "class Application {\n"+

             "\tpublic static function main(base:MovieClip) : Void\n" +
             "\t{\n" +
                   "\t\tvar scenes:Array = new Array();\n" +
                   "\t\tvar objects:Array = new Array();\n" +
                   "\t\tvar cScene:Number=0;\n" +
                   "\t\tvar cObjects:Number=0;\n" +
                   "\t\tvar cListeners:Number=0;\n" +
                   "\t\tvar selected:Number;\n" +
                   "\t\tvar counter:Array = new Array();\n" +
                   "\t\tvar listeners:Array = new Array();\n" +
                   "\t\tvar childs:Array = new Array(scenes,objects,
listeners);\n" +
                   "\t\tcounter[Constants.SCENES]=cScene;\n" +
                   "\t\tcounter[Constants.OBJECTS]=cObjects;\n" +
                   "\t\tcounter[Constants.LISTENERS]=cListeners;\n\n" +
          output.write(header);
```

Figura 64. *Inicialització de l'estructura d'emmagatzemament d'informació AS 2.0*

Amb aquest codi, el què es fa és inicialitzar una sèrie d'arrays (escenes, objectes, *listeners*), que a la vegada seran emmagatzemats en un array anomenat *childs*. Així, *childs* és un array que contindrà la informació de context que pot ser necessària durant el cicle de vida de l'aplicació. A *Scenes* es guardaran els *MovieClips* que representen les escenes de l'aplicació; a *Objects* es guardaran tots els objectes visuals que pot contenir una escena (imatges, botons, menús, etc.); i a *Listeners* es guardara tota la informació relativa als events, tant de botons, menús, com events relacionats directament a una escena o a tot el servei.

Un cop feta aquesta inicialització, la lectura de la descripció continua llegint cada un dels elements de forma seqüencial, i escrivint a l'arxiu *ActionScript* el codi corresponent a la creació de l'objecte AS relacionat amb aquell element PCF. Després, es va carregant tota la informació de l'element a l'objecte creat, i finalment es crida la funció de càrrega associada a aquell element. Per exemple, per l'objecte Service es realitza segons el codi mostrat a la Figura 65.

```
Node serviceNode = input.getElementsByTagName("pcf:Service").item(0);
      Element serviceElmnt = (Element) serviceNode;
      output.write("\t\tvar service:Service=new
                  Service(null,null,null,null,null);\n");
//Service pcfSpecVersion
      Node child = null;
      int k;
      if((child = serviceElmnt.getElementsByTagName("pcf:String")
.item(0))!=null){
            if(child.getParentNode().getNodeName()=="pcf:Service"
                  if("pcfSpecVersion".equals
(child.getAttributes().getNamedItem("name").getNodeValue())){
                        String name="\""+child.getAttributes().
                           getNamedItem("value").getNodeValue()+"\"";
                        output.write("\t\tservice.
                           setPcfSpecVersion("+name+");\n\n");
                  }
            }
      }
//Service serviceFirstScene
      if((child = serviceElmnt.getElementsByTagName("pcf:URI").
item(0))!=null){
            if(child.getParentNode().getNodeName()=="pcf:Service"){
                  String name="service_firstScene";
                  parseURI(name,output,child);
                  output.write("\t\tservice.
                     setFirstScene("+name+");\n\n");
            }
      }
```

```
//Service serviceReferenceScreen
      if((child = serviceElmnt.getElementsByTagName("pcf:Size").
item(0))!=null){
            if(child.getParentNode().getNodeName()=="pcf:Service"){
                  String name="service_referenceScreen";
                  parseSize(name,output,child);
                  output.write("\t\tservice.
                     setReferenceScreen("+name+");\n\n");
            }
      }
//Service background
      if((child = serviceElmnt.getElementsByTagName("pcf:Background").
item(0))!=null){
            if(child.getParentNode().getNodeName()=="pcf:Service"){
                  String name="service_background";
                  parseBackground(name,output,child);
                  output.write("\t\tservice.
                      Background("+name+");\n\n");
            }
      }

output.write("\t\tLoadManager.loadService(base,childs,counter,service)
;\n\n");
```

Figura 65. *Parseig d'un element Service*

Aquest procés es va realitzant per a cada un dels elements que van apareixent en la descripció, seguint per les escenes (*Scene*), i posteriorment tots els objectes que apareguin a dins d'aquestes escenes. Cal matissar, però, que per a evitar la creació de fitxers AS molt grans, es crea un fitxer per a cada nova escena, i al moment de compilar el fitxer principal crida aquests altres fitxers per mitjà de crides estàtiques a aquests fitxers.

5.1.6.3 Càrrega de components

La càrrega de components es fa per mitjà de la classe *LoadManager, escrita íntegrament en AS 2.0*. Aquesta classe té una funció per a cada element PCF, des d'un servei, fins a un botó o una imatge. La crida a aquestes funcions segueix generalment la mateixa estructura, passant-li els paràmetres base, que correspon al *MovieClip* a on es vol carregar aquest element. Per exemple, si estem carregant un botó a l'escena 2, base serà el *MovieClip* que representa l'escena 2 (remarcar que és el *MovieClip* que representa l'escena, i no la classe *Scene* del model de dades d'AS 2.0). Després se li passa la classe *childs*, ja que després de la càrrega, s'emmagatzemarà aquest nou objecte a l'array. Se li passa l'array *counter*, que s'encarrega de comptabilitzar quants elements (escenes,

objectes i *listeners*) tenim a l'array *childs* i facilitar-ne l'accés. I per últim se li passa l'element que es vol fer el *load*, ja inicialitzat amb tota la informació agafada de la descripció.

Després, la manera de càrrega és adaptada a cada element, intentant sempre que la visualització final sigui el més fidedigne possible al què s'havia descrit. A continuació es mostra un exemple d'una d'aquestes funcions:

```
public static function loadImage(base:MovieClip,childs:Array,
            counter:Array,image:Image){
      var globalX=image.getOrigin().getX()-base._x;
      var globalY=image.getOrigin().getY()-base._y;
      childs[Constants.OBJECTS][counter[Constants.OBJECTS]]=
         base.attachMovie(image.getURI(),image.getName(),
         Constants.OBJECT_DEPTH+counter[Constants.OBJECTS],
         {_x:globalX,_y:globalY,_width:image.getSize().getWidth(),
         _height:image.getSize().getHeight()});
      childs[Constants.OBJECTS][counter[Constants.OBJECTS]].
         addProperty("type");
      childs[Constants.OBJECTS][counter[Constants.OBJECTS]].
         type="image";
      counter[Constants.OBJECTS]++;
}
```

Figura 66. *Funció de càrrega d'una imatge*

5.1.6.4 Control d'events

El control dels events ve marcat per la rigidesa del desenvolupament. Haver de definir tot el cicle de vida de l'aplicació llegint seqüencialment un arxiu de text és el procés que ha resultat més costós. De forma similar a la càrrega d'objectes i escenes que realitza la classe LoadManager, la classe ControlFunctions és la encarregada de carregar un event, associar-lo a un objecte Listener, i definir el seu àmbit d'actuació (en tot el servei o en quina escena) i de quin tipus es tracta (event de teclat associat a una escena, a un botó, menú, o altres tipus d'events no de teclat, com per exemple, temporitzadors). A la figura següent es pot veure com es realitza la càrrega d'un event simple d'escena.

```
public static function addOnEvent(base:MovieClip,childs:Array,
  counter:Array,lc:LocalConnection,name:String,
  nameScene:String,onEvent:Object){
//function that creates a listener related to a scene or a button
//It recieves name=nameScene if it's a scene listener
//and name=button.getName if it belongs to a button
      if(onEvent.getTrigger().getEventType()=="KeyEvent"){
            var keyListener:Object = new Object();
            keyListener.addProperty("name");
```

```
            keyListener.name = name;
            keyListener.addProperty("sceneName");
            keyListener.sceneName=nameScene;
            keyListener.addProperty("type");
            keyListener.type="scene";
            keyListener.addProperty("active");
            keyListener.active=false;
            keyListener.addProperty("userKey");
            keyListener.userKey=Constants.mapKey
              (onEvent.getTrigger().getUserKey());
            keyListener.addProperty("target");
            keyListener.target=onEvent.getAction().
               getTarget().getHref().substr(3,onEvent.getAction()
               .getTarget().getHref().length);
//and Actions different from SceneNavigate
            keyListener.onKeyDown = function() {
                  if(Key.getCode()==keyListener.userKey){
                        ControlFunctions.setSceneVisibleOnly(childs,
                           keyListener.target);
                  }
            };
...
```

Figura 67. *Fragment de la funció addOnEvent*

5.1.6.5 Navegació

La tasca més difícil que va suposar de generar un iTV de servei mitjançant una descripció XML va ser la d'emmagatzemar tota la informació necessària per a la interactivitat. En una aplicació estàndard de Flash, és molt fàcil de definir un botó o un menú que fa una tasca específica. En el cas, al conversor, s'han de definir aquestes accions i controlar d'una manera tan genèrica com sigui possible. Com l'aplicació es crea de forma seqüencial, els controls de l'aplicació són creats sobre la marxa, i associats als elements. Com un botó pot fer coses diferents en funció del punt del cicle de vida del servei, es requereix de la gestió i la creació d'un context per a l'aplicació i seguiment d'ella.

Així doncs, aquestes dades de context que no estan especificades en el model de dades (perquè no són paràmetres especificats en la descripció), s'emmagatzemen en forma de propietats. Aquestes propietats són afegides als objectes *Listener,* que són els encarregats de captar els events i definir quina funció es fa en funció de l'event captat i el context en què es troba en aquell moment.

Destacar que l'especificació DVB-PCF defineix amb precisió com especificar els events i les accions resultants, el qual resulta molt útil a l'hora de definir la navegació entre escenes.

D'aquesta manera, el què s'ha realitzat és carregar totes les escenes a l'inici de l'aplicació i però deixar només visible l'escena marcada com *FirstScene* pel servei. Posteriorment, i mitjançant els events amb la càrrega de l'escena, s'anirà controlant la navegació entre les diferents escenes. En quant succeeixi un event que faci canviar d'escena, els events de l'escena actual es deshabilitaran i es posarà a invisible l'escena. Al mateix temps, es posarà visible l'escena indicada per l'event i s'habilitaran tots els events que estiguin associats a aquesta escena. Tot això es realitza per mitjà de la funció *setSceneVisibleOnly*, de la classe *ControlFunctions*.

```
public static function setSceneVisibleOnly(childs:Array,name:String){
//set visible only a scene and all of its contents
      var selected=findScene(childs,name);
      //set selected scene visible
      if(selected!=null){
            selected._visible=true;
      }

      for (i=0;i<childs[Constants.SCENES].length;i++) {
            //set invisible the rest of them
            if(childs[Constants.SCENES][i]!=selected){
                  childs[Constants.SCENES][i]._visible=false;
            }
      }
      removeOtherListeners(childs,name);
      //remove all listeners not related to the scene
      addSceneListeners(childs,name);
      //add scene listeners
...
```

Figura 68. *Fragment de la funció setSceneVisibleOnly*

No obstant, hi ha event implícits que són més indeterminats, com per exemple, la navegació entre els objectes dins d'una pròpia escena. Aquestes accions implícites no definides en la descripció han estat desenvolupades d'una manera estàndard per a totes els aplicacions. En aquesta primera versió del conversor, s'implementa una navegació senzilla entre tots els objectes que poden ser seleccionats (*focusable*) i que es troben dins una mateixa escena. Per a aquesta navegació s'utilitzen les tecles de fletxes del dispositiu o el control remot. Per defecte, per moure's entre els objectes l'usuari utilitzarà les tecles dreta i esquerra, i només en cas que l'escena contingui menús horitzontals, aquest control canvia, movent-se cap dreta i esquerra en els menús i, a amunt i avall per moure's entre els objectes de l'escena.

5.1.7 Utilització

5.1.7.1 Contingut de la *release*

La versió final d'ús (no de desenvolupament) del conversor ve proporcionada mitjançant un arxiu .zip comprimit amb el següent contingut:

- *ASCompiler*: carpeta que conté el compilador d'*ActionScript* per passar els fitxers *.as* d'*ActionScript* a un fitxer amb la versió final de l'aplicació en *.swf*. També conté el model de dades *ActionScript 2.0 PCF* i el paquet de classes *LoadManager*.
- *Input*: carpeta que conté exemples de les descripcions de *DVB-PCF* i les imatges associades a aquestes descripcions (les imatges s'especifiquen en l'exmeple de descripció DVB-PCF (*pcf.xml*), on es fa referència al recurs amb un *path* relatiu respecte l'arxiu XML (per exemple, */images/ <nom_arxiu_imatge>*).
- *PCF2SWFConverterTest*: Un projecte d'Eclipse que conté:
  - *PCF2SWFConverter.jar*: Llibreria *Java* que conté els paquets *XMLParser*, *ImageLibrary* i *ConverterManager* per a executar la conversió.
  - *PCF2SWFConverterTest.java*: Arxiu de prova, que inicialitza el conversor amb les rutes de fitxers d'entrada / sortida i la ubicació de *ASCompiler*, i executa la conversió.

5.1.7.2 Instal·lació

Per instal·lar el conversor, cal copiar la carpeta *ASCompiler* en un ordinador. A continuació, importar el projecte *PCF2SWFConverterTest* a l'àrea de treball (*Workspace*) d'Eclipse i executar *PCF2SWFConveterTest.java* (com a aplicació Java). Aquest fitxer (mostrat a la) és només un exemple de com realitzar una crida al conversor.

```
import com.PCFASConverter.Converter.Converter;
public class PCF2SWFConverterTest {
      public static void main(String [] args){
                  String inputPath="D:/PCF2SWFConverter/input/";
                  String inputFile="provaTot.xml";
                  String AS2SWFPath="D:/PCF2SWFConverter/ASCompiler/";
                  String outputPath="D:/PCF2SWFConverter/output/";
                  String outputFile="pcf";
                  Converter converter=new Converter(inputPath,
inputFile, AS2SWFPath, outputPath, outputFile);
                  converter.createLibrary();
                  converter.convert();
      }
}
```

Figura 69. *Fitxer PCF2SWFConverterTest.java*

També és possible instanciar i cridar el conversor des d'altres projectes, simplement afegint la llibreria *PCFSWFConverter.jar* al *classpath* del projecte i fer la crida

Per a inicialitzar la classe *Converter* (classe principal que controla la conversió), cal definir les rutes per trobar en la descripció de DVB-PCF és a dir, el nom de la descripció i la ruta. També especificar el nom de l'arxiu SWF de sortida i la ruta de sortida on es vol crear. És important, a més, establir la ruta de la carpeta *ASCompiler*, per tal de conèixer la ubicació del model de dades AS 2.0 i el compilador per transformar els arxius *ActionScript* a l'aplicació autocontinguda SWF.

5.1.8 Resum del procés de conversió

La Figura 70 mostra els diferents passos que es segueixen en el procés de conversió, en el qual, el paquet *Imagelibrary* extreu un fitxer XML que després processarà el compilador SWFmill, i el *XMLParser* una sèrie de fitxers *ActionScript* amb el codi associat als diferents elements PCF. Aquests fitxers contindran definicions d'objectes definits al model de dades i càrregues d'aquests objectes que estan implementades al paquet *LoadManager*. Després, en una segona etapa, el *ConverterManager* s'encarrega de cridar els compiladors MTASC i SWFmill perquè compilin els arxius creats, i extreguin l'arxiu .swf final.

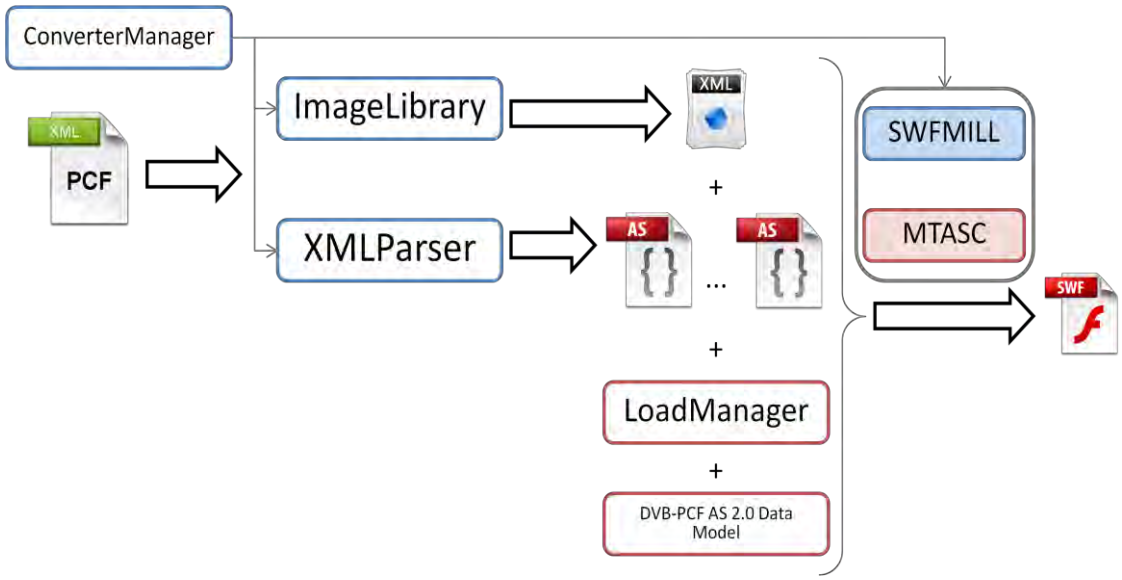


Figura 70. *Diagrama del procés de conversió*

Un cop executat el conversor, s'haurien d'haver creat a la carpeta de sortida (Figura 71) els arxius ActionScript (un principal i un altre per a cada escena del servei especificat), un arxiu XML per a l'importació de les imatges, i l'aplicació SWF final.

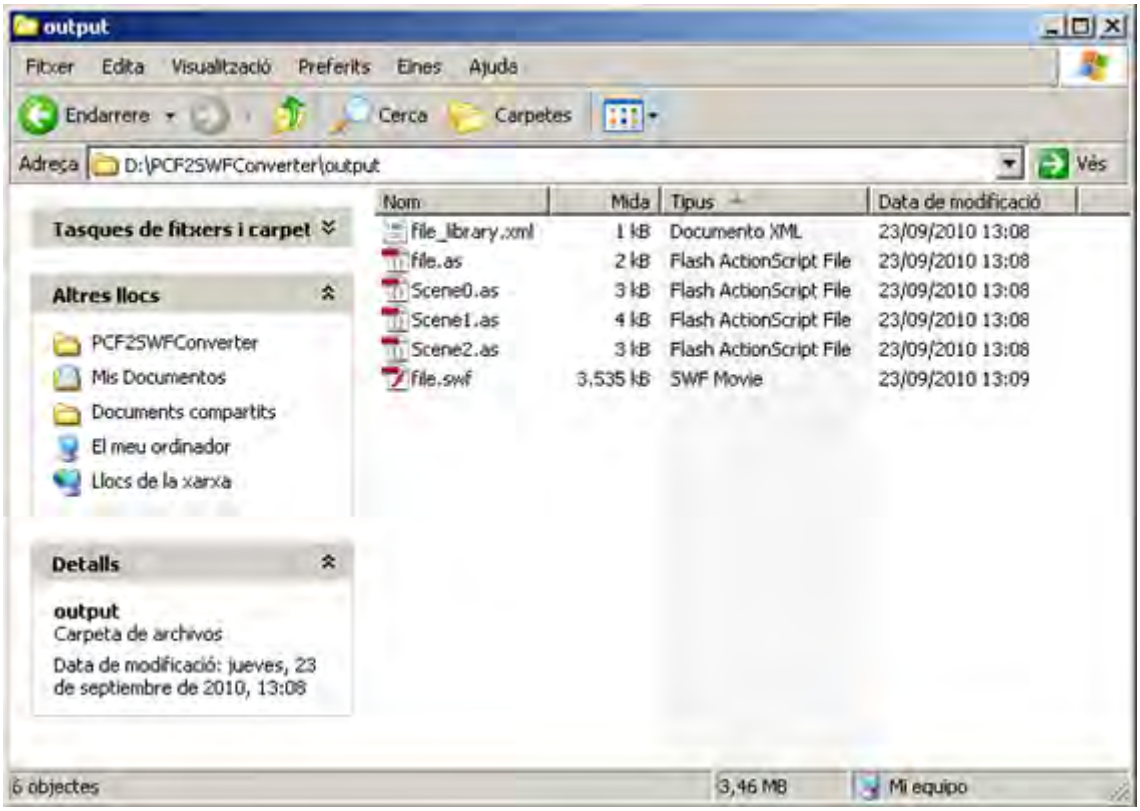


Figura 71. *Fitxers de sortida creats pel conversor*

6. RESULTATS

Tal com s'ha explicat a l'anterior apartat, el procés de conversió es realitza en varies fases. Primer la part Java del conversor s'encarrega de tractar la descripció DVB-PCF i extreure el fitxer XML que servirà de base al compilador SWFMill per crear un SWF amb tots els recursos necessaris. A L'exemple de la Figura 72, es pot veure un exemple d'aquest fitxer resultant. En aquest cas, la descripció tenia com a pantalla de referència una resolució de 1280x720 píxels, i contenia tres referències a imatges (*Nature_Green.png, DSC6587.png* i *Rose.png*) i a un clip de vídeo. Com es pot veure, a les referències a les imatges es fa un *import* directament de la imatge, mentre que al vídeo el què es fa és una importació d'un contenidor de vídeo *Flash*, on s'inclourà després el vídeo en moment d'execució.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<movie width="1280" height="720" framerate="30">)
      <background color="#ffffff"/>
      <frame>
            <library>
                  <clip id="D:/PCF2SWFConverter/input/Nature_Green.png"
import="D:/PCF2SWFConverter/input/Nature_Green.png"/>
                  <clip id="D:/PCF2SWFConverter/input/DSC6587.png"
import="D:/PCF2SWFConverter/input/DSC6587.png"/>
                  <clip id="D:/PCF2SWFConverter/input/Rose.png"
import="D:/PCF2SWFConverter/input/Rose.png"/>
                  <clip id="VideoContainer0"
import="resources/H264_AS2_video_test.swf"/>
            </library>
      </frame>
</movie>
```

Figura 72. *Exemple de XML creat pel conversor per compilar amb SWFMill*

A més d'aquest XML, el *parser* s'encarrega de seguir llegint la descripció PCF i crear els fitxers de codi *ActionScript 2.0*.. A continuació, a la Figura 73 es pot veure el fitxer principal, i que conté la funció *main.* En aquest fitxer es pot veure primer de tot com s'ha inclòs la capçalera on s'inicialitzen tots els arrays de *MovieClips* i *Listeners* que són necessaris durant el cicle de vida de l'aplicació. Després es procedeix a la creació del servei i la càrrega de les diferents escenes. Per últim, després de la càrrega de totes les escenes, es crida la funció *setSceneVisibleOnly*, perquè oculti totes les escenes que no són l'escena inicial.

Per qüestions de mida del fitxer resultant, el codi *ActionScript* queda dividit en les diferents escenes, i al fitxer principal només es fan crides a aquestes classes que s'acaben de crear.

```
import com.pcf2asdatamodel.*;
import com.loadmanager.*;

class Application {
      public static function main(base:MovieClip) : Void
      {
            var scenes:Array = new Array();
            var objects:Array = new Array();
            var cScene:Number=0;
            var cObjects:Number=0;
            var cListeners:Number=0;
            var cVideos:Number=0;
            var selected:Number;
            var counter:Array = new Array();
            var listeners:Array = new Array();
            var videos:Array = new Array();
            var childs:Array = new Array(scenes,objects,listeners,
videos);
            counter[Constants.SCENES]=cScene;
            counter[Constants.OBJECTS]=cObjects;
            counter[Constants.LISTENERS]=cListeners;
            counter[Constants.FEEDS]=cFeeds;
            counter[Constants.DATAENTRIES]=cDataentries;
            counter[Constants.VIDEOS]= cVideos;

            var service:Service=new Service(null,null,null,null,null);
            service.setPcfSpecVersion("");
            var service_firstScene:URI=new URI(null,null,null);
            service_firstScene.setValue("#~/prueba4");
            service.setFirstScene(service_firstScene);
            var service_referenceScreen:Size=new Size();
            service_referenceScreen.setWidth(1280);
            service_referenceScreen.setHeight(720);
            service.setReferenceScreen(service_referenceScreen);

            LoadManager.loadService(base,childs,counter,service);

            Scene0.loadScene(base,childs,counter);
            Scene1.loadScene(base,childs,counter);
            Scene2.loadScene(base,childs,counter);
            var
firstScene:String=service.getFirstScene().getValue().substr(3,service.
getFirstScene().getValue().length);
            ControlFunctions.setSceneVisibleOnly(childs,firstScene);
      }
}
```

Figura 73. *Exemple de fitxer AS 2.0 amb la inicialització de l'aplicació*

Les classes *Scene<N>* són les classes que crea el *parser* juntament amb el fitxer *.as* principal. En crea una per cada escena que especifica la descripció on N és igual al número d'escena començant per zero (seguint l'ordre seqüencial d'escenes trobades a la descripció).

Com a exemple d'una d'aquestes classes, la Figura 74, mostra una escena que conté un *Background*, un event de teclat per a navegar a la següent escena i una imatge. El procés de creació d'aquests components és sempre molt similar. Quan el *parser* troba un nou element a la descripció DVB-PCF, crea el component buit. Després, a mesura que va trobant atributs d'aquest element, va introduint els paràmetres en l'objecte creat. Així, és important tenir en compte que aquestes classes *ActionScript* creades no tenen sentit sense el model de dades creat amb anterioritat. Igualment, quan el *parser* troba el final de l'element, afegeix al fitxer *ActionScript* la crida a la funció de càrrega corresponent (p.ex. *LoadManager.loadImage*, *LoadManager.loadScene*).

```
import com.pcf2asdatamodel.*;
import com.loadmanager.*;

class Scene0{
      public static function
loadScene(base:MovieClip,childs:Array,counter:Array,lc:LocalConnection
) : Void{
            var scene0:Scene = new Scene("prueba4",null,null);

            var scene0_background:Background = new
Background(null,null,null,null);
            var scene0_background_size:Size=new Size();
            scene0_background_size.setWidth(1280);
            scene0_background_size.setHeight(720);
            scene0_background.setSize(scene0_background_size);
            var scene0_background_origin:Position=new Position();
            scene0_background_origin.setX(0);
            scene0_background_origin.setY(0);
            scene0_background.setOrigin(scene0_background_origin);
            var scene0_background_color:PCFColor=new
PCFColor("#0000FFFF");
            scene0_background.setColor(scene0_background_color);
            scene0.setBackground(scene0_background);

            var scene0_onEvent0:OnEvent = new OnEvent(null,null);
            scene0_onEvent0.setName("rightEvent4a");
            var scene0_onEvent0_action:SceneNavigate = new
SceneNavigate(null,null,null);
            var scene0_onEvent0_trigger:Trigger = new
Trigger(null,null);
```

```
            scene0_onEvent0_trigger.setEventType("KeyPressed");
            scene0_onEvent0_trigger.keyType("VK_RIGHT");
            scene0_onEvent0.setTrigger(scene0_onEvent0_trigger);
            var scene0_onEvent0_target:URI=new URI(null,null,null);
            scene0_onEvent0_target.setHref("#~/prueba4b");
            scene0_onEvent0_target.setContext("derived");
            scene0_onEvent0_action.setTarget(scene0_onEvent0_target)
            scene0_onEvent0.setAction(scene0_onEvent0_action);
            scene0.setOnEvent(scene0_onEvent0);

            ControlFunctions.addOnEvent(base,childs,counter,
scene0_onEvent0.getName(),scene0.getName(),scene0.getOnEvent());

            LoadManager.loadScene(base,childs,counter,scene0,10000);

            var scene0_image0:PCFSwf= new
PCFSwf(null,null,null,null,null,null,null,null);
            scene0_image0.setName("umbrella");

            var scene0_image0_size:Size=new Size();
            scene0_image0_size.setWidth(550);
            scene0_image0_size.setHeight(550);
            scene0_image0.setSize(scene0_image0_size);

            var scene0_image0_origin:Position=new Position();
            scene0_image0_origin.setX(0);
            scene0_image0_origin.setY(0);
            scene0_image0.setOrigin(scene0_image0_origin);

            scene0_image0.setContentType("image/png");
            scene0_image0.setURI("/images/umbrella.png");
            LoadManager.loadImage(ControlFunctions.findScene(childs,
scene0.getName()),childs,counter,scene0_image0,scene0.getName());
      }
}
```

Figura 74. *Exemple de fitxer AS 2.0 d'una escena*

Com a exemple de prova de conversió d'un servei interactiu, la Figura 75 mostra un fragment de descripció DVB-PCF d'una aplicació de futbol ideada per a ser enviada durant l'emissió d'un partit futbol. Aquesta aplicació mostra els diferents escenaris on l'usuari pot navegar amb el vostre control remot les tecles de fletxa esquerra i dreta.

```
<?xml version="1.0"?>
<pcf:PCF xmlns:pcf="http://www.dvb.org/pcf/pcf">
   <pcf:Service name="football_app">
         <pcf:Size name="referenceScreen" value="640 576" />
         <pcf:String name="pcfSpecVersion" value="1.0" />
         <pcf:URI name="firstScene" value="#~/League_Table" />
         <pcf:Scene name="League_Table">
               <pcf:Image name="InfoPanel">
                     <pcf:Position name="origin" value="62 110" />
                     <pcf:Size name="size" value="258 342" />
                     <pcf:ImageData name="content">
                        <pcf:ExternalBody content-type="image/png" uri="images/Table_Panel.png" />
                     </pcf:ImageData>
               </pcf:Image>
               <pcf:Image name="MainBG">
                     <pcf:Position name="origin" value="0 452" />
                     <pcf:Size name="size" value="640 124" />
                     <pcf:ImageData name="content">
                        <pcf:ExternalBody content-type="image/png" uri="images/Table_MainBG.png" />
                     </pcf:ImageData>
               </pcf:Image>
               <pcf:Menu name="MenuH">
                     <pcf:Position name="origin" value="100 458" />
                     <pcf:Size name="size" value="286 36" />
                     <pcf:String name="menuAlign" value="horizontal" />
                     <pcf:FontFamily value="Verdana" />
                     <pcf:FontSize value="12pts" />
                     <pcf:String name="font-style" value="regular" />
                     <pcf:String name="font-weight" value="normal" />
                     <pcf:Integer name="optionsVisible" value="3" />
                     <pcf:OnEvent name="selectEvent">
                        <pcf:Trigger eventtype="KeyEvent">
                              <pcf:UserKey name="key" value="VK_ENTER" />
                        </pcf:Trigger>
                        <pcf:SceneNavigate>
                              <pcf:URI context="derived" href="targetArray[index]" name="target" />
                        </pcf:SceneNavigate>
                     </pcf:OnEvent>
            ...
```

Figura 75. *Fragment DVB-PCF amb la aplicació de prova de futbol interactiu*

Figura 76. *Imatge de l'aplicació de prova de futbol interactiu*

Al passar aquesta aplicació pel conversor, s'obté una aplicació com la mostrada a la Figura 76. En aquesta figura es mostra una escena d'aquesta aplicació on l'usuari pot veure la classificació de la lliga, i un menú horitzontal que permet el control de la navegació entre escenes (classificació, resultats de la jornada i resum del partit que s'està veient). A més, en aquesta escena hi ha tres objectes diferents seleccionables (aquest menú horitzontal, un botó per anar al menú principal i un botó de sortida) que l'usuari pot enfocar amb les tecles de fletxa amunt i avall.

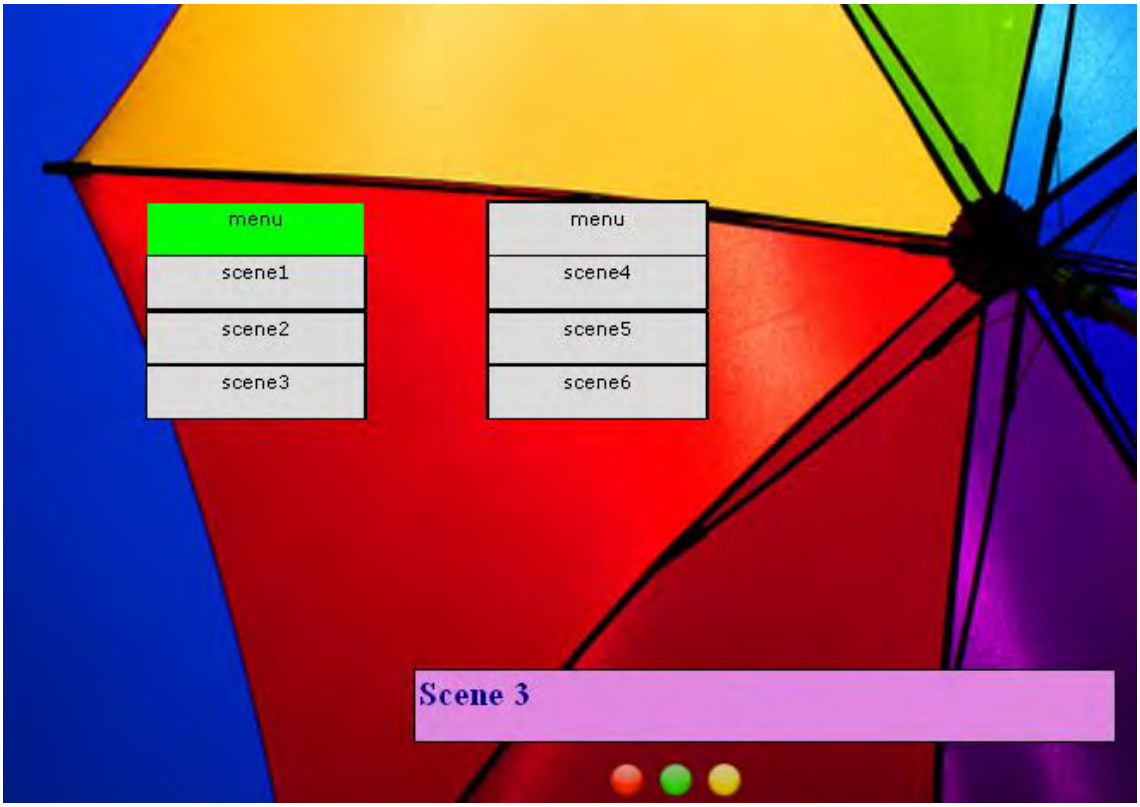


Figura 77. *Imatge de l'aplicació de prova amb doble menú*

En cas que l'escena contingui menús verticals, s'observa com el control canvia, movent-se cap amunt i cap avall en els menús i, a l'esquerra i dreta per moure's entre els objectes de l'escena. Aquesta característica també es mostra a la Figura 77, on es presenta una imatge d'una aplicació amb una escena del menú doble. L'usuari pot moure's horitzontalment per a desplaçar-se d'un menú a un altre, i verticalment dins d'ells. Aquest mètode de control de la navegació en les aplicacions prestats per aquesta primera versió del conversor limita l'ús de menús horitzontals i verticals en la mateixa escena, i requeriria d'un altre mètode que millorés aquestes característiques en properes versions.

7. CONCLUSIONS I LÍNIES DE FUTUR

En aquest treball, es presenta un estudi del format DVB-PCF i el seu ús en sistemes de producció de serveis interactius de televisió. Després es presenta un conversor de descripcions DVB-PCF a *Adobe Flash Lite*, permetent una solució extrem a extrem de distribució i desplegament de serveis *iTV*, sota el sistema proposat per l'estàndard DVB-PCF.

L'aplicació creada demostra el potencial del format DVB-PCF i mostra com és possible transformar descripcions DVB-PCF en serveis interactius reals, *renderitzats* amb la tecnologia d'*Adobe Flash Lite*. En aquest sentit, és important remarcar la simplicitat de la creació d'aplicacions utilitzant aquest sistema, el qual només requereix de descripcions molt senzilles de crear i llegir, que són emmagatzemades en fitxers XML.

Els resultats d'aquest conversor demostren els avantatges que pot aportar un mercat de serveis interactius de televisió centrat en el format DVB-PCF. L'ús d'un estàndard comú de creació i distribució de serveis per a totes les plataformes, incentivaria el mercat de les aplicacions interactives per a la televisió.

La possibilitat de crear aplicacions multiplataforma fa que el desenvolupament d'aquestes pugui ser molt més genèric i ampli, sense la necessitat de desenvolupadors interns a l'empresa i amb coneixements de la plataforma de distribució per a la creació d'aplicacions. A més, la seva senzillesa alleuja aquest procés de creació, mostrant-se a l'abast de dissenyadors i altres persones amb molts pocs coneixements tècnics.

D'altra banda, s'aconseguiria una forma de distribució molt transparent entre empreses, ja que es podrien fer intercanvi d'aplicacions sense donar a conèixer la plataforma o tecnologia utilitzada per cada una d'elles. Així doncs, tot i un ús comú d'aquesta tecnologia, això no suposa una obertura de les empreses, sinó que també afavoreix l'emmascarament de la tecnologia emprada per cada empresa i mantenir possibles secrets comercials.

A més, un altre dels grans avantatges de cara als desenvolupadors és la possibilitat de reutilització integral (o només una part) de les descripcions. Per a generar ràpidament nous serveis, és tant fàcil com copiar i enganxar els components XML que ens interessin amb un simple editor de text.

Sobre l'abast d'aquest conversor, els elements DVB-PCF suportats en aquesta primera versió, com es mostra en la secció de resultats, són suficients per generar una experiència interactiva de gran qualitat per a l'usuari, mostrant molts tipus diferents d'objectes visuals.

També dóna suport a alguns elements d'events definits en l'especificació DVB-PCF, oferint a l'usuari una navegació completa i intuïtiva entre els diferents objectes i escenes del servei.

No obstant això, aquesta versió del conversor encara no és compatible amb les descripcions més avançades que permeten objectes gràfics molt complexos o serveis interactius més avançats o connexions pel canal de retorn. Aquestes característiques estan encara en desenvolupament i s'espera que en properes versions.

En quant a propers passos a seguir en el seu desenvolupament, a més de la inclusió de nous elements, està la transformació del model de dades i el sistema de càrrega de components d'*ActionScript 2.0* a *ActionScript 3.0* i altres tecnologies (p.ex. HTML5), per a obtenir nous elements finals que adaptin les descripcions DVB-PCF a altres plataformes.

Adobe Flash Lite, tot i ser pensat per a funcionar sota baixes prestacions, com a mòbils o dispositius de llar digital, resulta ser una eina relativament limitada en comparació amb un *Adobe Flash 10* o *HTML5*. Aquest tipus de dispositius cada dia tenen més potència i són capaços de suportar tecnologies més capdavanteres. És per això que aquest conversor és només una prova del què es pot arribar a aconseguir amb una bon processat de descripcions DVB-PCF, i ha de servir d'exemple per a crear-ne de nous basats en una estructura que no hauria de variar en gran mesura a la plantejada per aquest.

8. BIBLIOGRAFIA

[1] R. Bradbury, R. Cartwright, J. Hunter, S. Perrott, and J. Rosser, *Portable Content Format: a standard for describing an interactive digital television service*. BBC Research & Development, 2006.
[2] P. Bugnot, *Commission of european communities; COMMISSION STAFF WORKING PAPER on the interoperability of digital interactive television services pursuant to Communication COM(2004)*. 2004.
[3] *Digital Video Broadcasting; Portable Content Format (PCF) specification 1.0*. 2006.
[4] K. Rush and S. I. Inc, "GETTING STARTED WITH ADOBE FLASH® LITE™," *A technical whitepaper, Adobe Inc. Publications*, 2007.
[5] Adobe, *Adobe and Broadcom Bring the Adobe Flash Platform to TVs*. 2009.
[6] BBC, *Adobe Flash secures set-top deal*. 2009.
[7] "World Wide Web Consortium (W3C)." [Online]. Available: http://www.w3.org/. [Accessed: 31-Aug-2010].
[8] E. Raymond, "DocBook Demystification HOWTO," *Seite zuletzt \überpr\üft am*, vol. 15, 2005.
[9] F. Yergeau, T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler, "Extensible markup language (xml) 1.0," *W3C Recommendation*, 2004.
[10] F. Yergeau, J. Cowan, T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler, "Extensible markup language (XML) 1.1," *W3C Recommendation*, vol. 4, p. 220, 2004.
[11] W. W. Do, G. Archives, F. Guides, and S. an Event, "The Unicode Standard, Version 2.0."
[12] H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn, and others, "XML schema part 1: Structures," *W3C, work-in-progress, current as of Apr*, pp. 1–20000407, 2000.
[13] J. Wusteman, "Document Type Definition (DTD)."
[14] S. Morris and A. Smith-Chaigneau, *Interactive TV Standards: A Guide to MHP, OCAP and Java TV*. Elsevier, 2005.
[15] R. M. Arlein, R. D. Gaglianello, D. Liu, and L. Spergel, "Developing applications for EBIF and tru2way< sup>™</sup>," in *Broadband Multimedia Systems and Broadcasting (BMSB), 2010 IEEE International Symposium on*, pp. 1–6, 2010.
[16] N. Borenstein, N. Freed, and O. N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of...," *RFC 1521, BELLCORE, INNOSOFT*, 1993.
[17] M. Fowler and K. Scott, *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2000.
[18] SMPTE, "SMPTE 397M-2003 Television: "Declarative Data Essence - Transitional"," Apr-2003.
[19] F. de Lange, "The Philips-Open TV® Product Family Architecture for Interactive Set-Top Boxes," *Software Product-Family Engineering*, pp. 114–118, 2002.
[20] S. Bryant and R. Drewry, "The liberate technologies TV navigator for DTV: a thin web-centric client for digital television," in *Interactive Television (Ref. No. 1999/200), IEE Colloquium on*, pp. 7/1-7/8, 1999.
[21] K. Hofrichter, "MHEG 5—Standardized presentation objects for the Set Top Unit environment," *Interactive Distributed Multimedia Systems and Services*, pp. 33–44, 1996.
[22] E. L. Specification, "Standard ecma-262," *ECMA Standardizing Information and Communication Systems*, vol. 3, 1999.

[23] J. Tapper, J. Talbot, and R. Haffner, *Object-oriented programming with ActionScript 2.0*. New Riders Pub, 2004.
[24] R. Rischpater, *Software Development for the QUALCOMM BREW Platform*. Apress, 2003.
[25] E. Eldrom, S. Janousek, and T. Joos, *AdvancED Flash on Devices*. Apress and friends of ED books, 2009.
[26] E. Eclipse Foundation, *Eclipse*. 2005.
[27] N. Serrano and I. Ciordia, "Ant: automating the process of building applications," *IEEE software*, vol. 21, no. 6, pp. 89–91, 2004.
[28] Collabnet, "subclipse.tigris.org," 2005. [Online]. Available: http://subclipse.tigris.org/. [Accessed: 01-Sep-2010].
[29] A. Balkan, "Open Source Flash." [Online]. Available: http://osflash.org/. [Accessed: 06-Sep-2010].
[30] Motion-Twin, "Motion-Twin ActionScript Compiler." [Online]. Available: http://www.mtasc.org/. [Accessed: 06-Sep-2010].
[31] Dave Williamson, "Using MTASC to compile Flash Lite (FSCommand2) on OSX." [Online]. Available: http://blog.bittube.com/2008/11/17/using-mtasc-to-compile-flash-lite-fscommand2-on-osx/. [Accessed: 06-Sep-2010].
[32] Balkman, "How to structure and set up a Flash project without using the Flash IDE Open Source Flash." [Online]. Available: http://osflash.org/projectsetup. [Accessed: 06-Sep-2010].
[33] "swfmill." [Online]. Available: http://swfmill.org/. [Accessed: 06-Sep-2010].
[34] Carlos Rovira, ".:: ASDT - Actionscript Development Tool ::.." [Online]. Available: http://aseclipseplugin.sourceforge.net/wordpress/. [Accessed: 06-Sep-2010].
[35] Flash Develop Community, "Main Page - FlashDevelop." [Online]. Available: http://www.flashdevelop.org/wikidocs/index.php?title=Main_Page. [Accessed: 06-Sep-2010].
[36] INRIA, "The Caml language: Home." [Online]. Available: http://caml.inria.fr/. [Accessed: 06-Sep-2010].
[37] C. Allen, A. Balkan, W. Arnold, J. Grden, and N. Cannasse, *The Essential Guide to Open Source Flash Development*. Apress, 2008.
[38] haXe Community, "haXe." [Online]. Available: http://haxe.org/. [Accessed: 06-Sep-2010].
[39] "Apuntes de DOS: Variables de entorno." [Online]. Available: http://www.ignside.net/man/dos/env.php. [Accessed: 06-Sep-2010].
[40] R. Mecklenburg, *Managing Projects with GNU Make (Nutshell Handbooks)*. O'Reilly Media, Inc., 2004.
[41] "Apache Ant User Manual." [Online]. Available: http://ant.apache.org/manual/index.html. [Accessed: 13-Sep-2010].
[42] "Apache Ant Open Source Flash." [Online]. Available: http://osflash.org/ant. [Accessed: 13-Sep-2010].
[43] T. Bray, D. Hollander, A. Layman, and R. Tobin, "Namespaces in XML 1.0," *W3C Recommendation*, vol. 16, pp. 2009–12, 2006.
[44] "as2lib Files on SourceForge.net." [Online]. Available: http://sourceforge.net/projects/as2lib/files/as2ant/. [Accessed: 27-Sep-2010].

Annexos

Annex 1. Taula Comparativa Flash – Flash Lite

Adobe

Feature Comparison
Adobe Flash® Lite™, Flash® Player and Flash® SDK at a Glance

	Flash Lite 1.1	Flash Lite 2.1	Flash SDK 7	Flash Lite 3.1	Flash Player 8	Flash Player 9
Sound Support						
MIDI	Device dependent	Device dependent		Device dependent		
PCM and ADPCM	✓	✓	✓	✓	✓	✓
MP3	✓	✓	✓	✓	✓	✓
8 Channel					✓	✓
32 Channel					✓	✓
Image & Video Support						
PNG	During authoring	During authoring or device dependent [1]	During authoring	During authoring or device dependent [1]	✓	✓
JPEG	✓	✓	✓	✓	✓ (progressive JPEG)	✓ (progressive JPEG)
GIF	During authoring	During authoring or device dependent	During authoring	During authoring or device dependent	✓	✓
Animated GIF	During authoring	During authoring	During authoring	During authoring	During authoring	During authoring
MPEG-4 and other video formats		Device dependent	During authoring	Device dependent	During authoring	During authoring
Flash Video (FLV)			✓ (On2 & Sorenson)	✓ (On2, Sorenson, and support for H.264)	✓ (On2 & Sorenson)	✓ (On2, Sorenson, & H.264)

Adobe

8 bit alpha channel video					✓	✓
Import/Encode video			✓		✓	✓
Multimedia Support						
Dynamic loading of multimedia files		✓ [2]	JPEG / MP3 only	✓ [2]	JPEG / MP3 / GIF / PNG / Progressive JPEG	JPEG / MP3 / GIF / PNG / Progressive JPEG
Text Support						
Character set	Latin-1 and Shift-JIS	UTF-8	UTF-8	UTF-8	UTF-8	UTF-8
Complex languages (Thai, Arabic, Hebrew, etc.)		✓		✓		
Dynamic text	✓	✓	✓	✓	✓	✓
Device-specific vector fonts		✓	✓	✓	✓	✓
Improved small text readability		✓	✓	✓	✓	✓
Text measurement		✓	✓	✓	✓	✓
Text wrap		✓	✓	✓	✓	✓
Inline Text Input		✓		✓	✓	
Predictive Text Support		✓		✓		
Rich text styles			✓		✓	✓
FlashType					✓	✓
Improved text layout					✓ (full justified, kerning, character spacing)	✓ (full justified, kerning, character spacing)

Emoticons	✓	✓		✓		
Emoticons in predefined color				✓		
Interactivity						
Keyboard events	Device dependent	Device dependent	✓	Device dependent	✓	✓
Key-based navigation	✓	✓	✓	✓	✓	✓
Mouse/Stylus events	Device dependent	Device dependent	✓	Device dependent	✓	✓
Mouse wheel support			✓		✓	✓
Programming Features						
Flash version supported	Flash 4 or earlier	Flash 7 or earlier	Flash 7 or earlier	Flash 8 or earlier	Flash 8 or earlier	Flash 9 or earlier
ActionScript Version	FlashScript (Flash 4 or earlier)	ActionScript 1.0, 2.0 (Flash 7 or earlier)	ActionScript 1.0,2.0 (Flash 7 or earlier)	ActionScript 1.0, 2.0 (Flash 8 or earlier)	ActionScript 1.0,2.0 (Flash 8 or earlier)	ActionScript 1.0, 2.0, 3.0 (Flash 9 or earlier)
Dynamic loading of SWF data	✓	✓	✓	✓	✓	✓
XML parsing		✓	✓	✓	✓	✓
String/Array/XML-to-native-objects conversion		✓	✓	✓	✓	✓
ActionScript strict mode		✓	✓	✓	✓	✓
Set/Clear interval		✓	✓	✓	✓	✓
Shape-drawing API		✓	✓	✓	✓	✓
Ability to store data		✓	✓	✓	✓	✓
Bitmap effects & filters					✓	✓

Bitmap caching					✓	✓
BitmapData API					✓	✓
Blend modes					✓	✓
9-Slice scaling					✓	✓
Stroke enhancements (endcaps & joins)					✓	✓
Linear & radial gradiant enhancements					✓	✓
External API for browser scripting				✓	✓	✓
File Upload/Download					✓	✓
IME API enhancements					✓	✓
Other Features						
Generic browser interface	✓	✓	ActiveX or Netscape	✓	ActiveX or Netscape	ActiveX or Netscape
Dynamic memory handling	✓	✓	✓	✓	✓	✓
Device-specific capabilities	✓	✓		✓		
Meta data support				✓		
Background Transparency		✓ (set in host application)		✓ (set in host application)	✓	✓
Forward lock				✓ (set in host application)		
Printing			✓		✓	✓

Object model (for components)		✓	✓	✓	✓	✓
Improved event model		✓	✓	✓	✓	✓
XMLSockets		✓		✓	✓	
Flash Media Server connectivity (RTMP Streaming)			✓	✓	✓	✓
Flash Media Server connectivity (Remote Shared Object)			✓		✓	✓
Scriptable masks			✓	✓	✓	✓
SWF file compression		✓	✓	✓	✓	✓
Accessibility			✓		✓	✓
Dynamic discovery of device features		✓	✓	✓	✓	✓
ActionScript exception handling		✓	✓	✓	✓	✓
Improved ActionScript performance				+15-20% over FL 2.1	✓	✓
Improved rendering performance				+15-20% over FL 2.1	✓	✓
Web services and SOAP API		✓	✓	✓	✓	✓
New preloader API		✓	✓	✓	✓	✓
Progressive download			✓	✓	✓	✓
Basic version check and update mechanism					✓	✓
Express install					✓	✓
Enhanced local file security				✓	✓	✓

SVG-T 1.1	✓	✓				
Flash Lite Features						
Access to device-specific features (volume, backlight, vibrate, and so on)		✓		✓		
Launch native applications	✓	✓		✓		
Reduced runtime memory consumption	✓	✓		✓		
Graceful handling of out-of-memory condition.	✓	✓	✓	✓		
Interruptible/Re-entrant player	✓	✓		✓		
Runaway script limit	✓	✓	✓	✓	✓	✓
ActionScript slicing	✓	✓		✓		
System Requirements						
Player size (core player DLL)	275K	450K	1.0MB (Win/WinCE)	374K	1.4MB (Win)	~2.0 MB (Win)
CPU characteristics	32-bit data bus, 200Mhz ARM9	32-Bit data bus, 200Mhz ARM9	32-bit data bus, 300Mhz ARM9	32-Bit data bus, 200Mhz ARM9	Desktop Hardware	Desktop Hardware
Minimum RAM requirements	64K	128K	4.5MB	128K	Desktop Hardware	Desktop Hardware
Recommended RAM		2 MB	32MB	2 MB for stand alone content, more for video (varies by file size, duration)	32 MB	32 MB
Content size-to-heap ratio	1:10 [2]	1:15 [2]	1:30	1:15 [3] [4]		

Footnotes	Platform/Browser Support						
	Reference platforms	Symbian	Symbian, BREW	Windows XP (Standalone, ActiveX), WinCE (activeX), Linux (Standalone, Netscape plug-in), PocketPC (ActiveX)	Symbian, Windows XP, Linux		
	Browsers supported			IE, Firefox	IE, Opera	IE, Netscape, Firefox, Mozilla, AOL, Opera	IE, Netscape, Firefox, Mozilla, AOL, Opera

(1) Transparency supported
(2) JPEG & MP3; other formats dependent on device-specific codecs
(3) Estimated worst-case memory consumption: for example, for playback of 100k SWF file, the recommended memory configuration is 1.5 MB
(4) Memory usage when playing back video is managed by active buffering in Flash Lite 3.0
(5) UI/DataServices requirements

Annex 2. Short Paper acceptat a EUROiTV 2010, Tampere, Finlàndia

DVB-PCF to Flash Lite Transcoder

Ramon Martín de Pozuelo
GTM - Grup de Recerca en Tecnologies Mèdia
LA SALLE - UNIVERSITAT RAMON LLULL
Quatre Camins 2, 08022
Barcelona, Catalonia, Spain
ramonmdp@salle.url.edu

Francesc Enrich
GTM - Grup de Recerca en Tecnologies Mèdia
LA SALLE - UNIVERSITAT RAMON LLULL
Quatre Camins 2, 08022
Barcelona, Catalonia, Spain
frane@salle.url.edu

Gabriel Fernández
GTM - Grup de Recerca en Tecnologies Mèdia
LA SALLE - UNIVERSITAT RAMON LLULL
Quatre Camins 2, 08022
Barcelona, Catalonia, Spain
gabrielf@salle.url.edu

ABSTRACT

The heterogeneity of tools in interactive television applications development makes more difficult a general deployment of them. In order to solve that, DVB-PCF (Digital Video Broadcasting-Portable Content Format) is a standard that aims to consolidate a standard way to define these applications, allowing an easier interchange of them, regardless of their platform target.

However, in order to support this standard, it will be necessary to create converters that could interpret its descriptions and create the platform-specific applications. In this context, this paper introduces the design, implementation and results of a DVB-PCF to Adobe Flash Lite transcoder, which intends to render a DVB-PCF description, as accurate as possible, in an Adobe Flash Lite 3.1 application.

Categories and Subject Descriptors

J.m [**Computer Applications**]: Miscellaneous;
I.7 [**Document and Text Processing**]: Standards

General Terms

Algorithms, Design, Standardization, Languages.

Keywords

DVB-PCF, Flash-Lite, Interactive TV, transcoder, parser, converter.

1. INTRODUCTION

Currently, the increasing market of digital television and interactive applications for non-PC devices does not have a consensus standard that guides how to develop this kind of applications for set-top boxes, hand-held devices, etc. The variety of mediums and platforms where those applications are deployed complicates the development of a proposition to be delivered to multiple target platforms, due to physical, operational and commercial constraints [4]. This fact brings us to a situation where all the developers make their applications for a specific target platform, and it is hard to create a broad market around interactive digital television (iTV) services.

However, looking at this panorama, in 2006 emerges DVB-PCF (Portable Content Format) [1]. The PCF is a standard focused in the description of interactive services by describing the visual experience to be shown to the user. It provides the industry with a platform-independent format for the business-to-business interchange of interactive content, and consequently a means for increasing the interoperability of authoring tools, head-end systems and broadcast networks.

Using PCF, content developers will deliver PCF content to network operators or broadcasters, who will then run it through a translator to create something appropriate for their network. It is not intended to be transmitted to final users' set-top boxes or integrated digital TVs. It enhances interoperability by enabling content providers to author their content once and run it on multiple API platforms, covering 80% of interactive television applications [5]. This opens a market that has been completely diversified and hard to exploit efficiently, but PCF is not an end-to-end system that could solve this immediately. Indeed, although it becomes the followed standard to define most of iTV services, other technologies are needed in order to adapt PCF content to every platform and finish the distribution chain.

The recently supported for many Digital Home devices [2] [6] [3], Adobe Flash Lite presents a powerful tool that paves the way for rich Flash-based entertainment experiences on televisions that offer viewers new options for accessing web content on their TVs, phones, etc.

Taking these two facts into account, DVB-PCF to Flash Lite Transcoder wants to fill the gap between the two technologies and bring a solution that can interpret a PCF description and render an application that could be visualized in any device that supports Flash Lite Player.

2. TRANSCODER FRAMEWORK

2.1 Development process

The development of this transcoder has been performed in two phases. In the first one, a DVB-PCF interpreter was written in ActionScript (AS) 2.0. As a result of this, a Flash Lite application that dynamically renders a DVB-PCF description was obtained. It can render a service, scenes and basic elements as images or textboxes, and a simple navigation between scenes using basic menus. This resultant application should always be accompanied by an XML file with the DVB-PCF description, and it will render it when loaded. It is a relevant fact, because it permits to dynamically change the application by modifying the description and reloading the application. For this purpose, data model of basic DVB-PCF elements was written in AS 2.0, defining a set of basic PCF elements. However, as interesting as it could be to modify dynamically the rendered application, it sometimes falls back to security issues and would not be useful in some environments.

In a second phase, a complete transcoder was made, which permits the creation of iTV services without the necessity of the description, once the application had been created. This offers more flexibility in some aspects, as the service provider does not need to send multiple files to its users: just a standalone application is supplied. It also solves the security problem posed by the fact of delivering an application that could be modified by changing the associated XML description file.

Taking into account the current economic model of iTV, it will not be interesting for service providers to give their users the possibility of modifying the application by themselves. It would be like providing all the source code to their clients, allowing them to modify it without restrictions. It would be interesting to give users the chance to create new features by simply changing an XML file, but it could be the end of iTV market as we know it.

As a result of this development, the new transcoder only needs the description and images of the service in compilation time, when it creates an Adobe Flash SWF file, a self-contained application that could not be modified, and a easier way to deliver the application to users.

2.2 Package structure

This section details the structure of the final transcoder mentioned above, and how it was designed. First of all, different modules were established, foreseeing the possibility to swap some of them to other technologies, especially those that are more platform-specific, like the data model.

The packages finally defined were the following (also shown in Figure 1):

- **DVB-PCF ActionScript Data Model** package includes all classes related to AS objects that can form a PCF description.
- **LoadManager** package contains all classes related to load PCF information into memory including image loading issues.

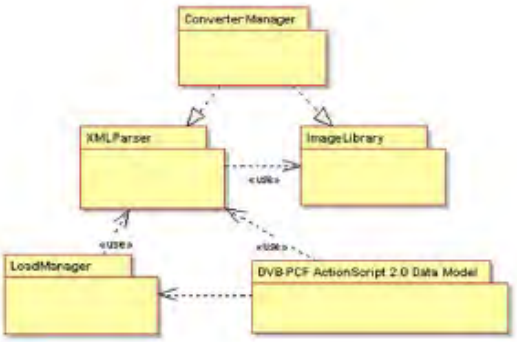


Figure 1: DVB-PCF to Flash Lite transcoder packages

- **XMLParser** package includes all classes related to PCF description interpretation, and use LoadManager and Data Model package to create an AS file that includes all the elements defined in PCF description.
- **ImageLibrary** package classes control image imports and create a library with all the images specified in the PCF description file.
- **ConverterManager** package includes all classes related to complete the conversion and make up the SWF file, managing the outputs from XMLParser and ImageLibrary.

First two packages were written completely in ActionScript 2.0, whereas the following three were written in Java.

Trying to provide a way to adapt the description of the service as exact as possible to the final application rendered, the first step in design stage was an extensive study of all the elements defined in DVB-PCF standard and how they could be mapped into Flash parameters.

PCF should basically provide means to allow application authors to describe the intended experience, so most of its elements are fundamentally visual. Flash Lite 3 uses AS 2.0, and it is a limited platform in comparison to Flash Player used in higher power devices, but it still has enough potential to make it feasible to recreate all PCF basic features by combining AS 2.0 components.

Resulting from that study, a package of AS classes describing most DVB-PCF elements is created. This package defines PCF elements (Service, Scene, Background, Image, Menu, etc.), providing its required parameters to each item, so that it could be represented as exact as possible to what is described. They also provide getter and setter functions for each parameter to easily interact with them, and a basic constructor that only initializes all the parameters of the class.

These classes do not provide methods to render all these features in Flash; their intention is only to work as containers, maintaining the PCF element information.

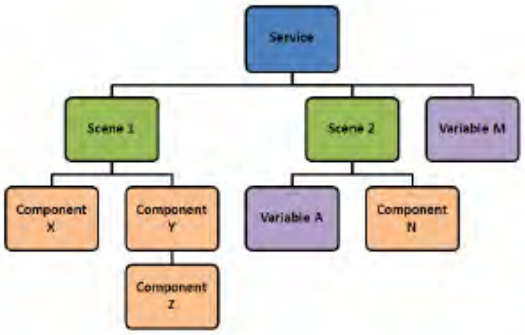


Figure 2: Hierarchical tree sample of DVB-PCF Components

Once this data model had been accomplished, and it made possible to storage every feature of PCF elements, a render package was needed. This package will be the responsible of taking all the parameters stored in every element defined and render it as exact as possible.

Two more packages were defined in order to control the load of all the information in compilation time: (1) a set of classes to load the images included in DVB-PCF description, and (2) a parser that reads from DVB-PCF description and creates AS 2.0 files. These resulting files, in addition to `Data Model` and `LoadManager`, are used to compile the final application.

2.2.1 *DVB-PCF Data Model*

PCF service descriptions are formed by hierarchical structures of information items, most of them represented as PCF components, and each one with a unique name identifier. Figure 2 shows an example tree of its typical structure.

The root element of a description is `service` item, which has to contain one or more `scene` items, and define which of these scenes will be the first to be shown to the user. An important fact to emphasize is that the service should never show more than one scene simultaneously in its life-cycle. All the scenes that could be rendered have to be inside `service` item or some of its sub-containers.

As shown in [4], main aspects of a basic PCF service description could be summarized in:

- **Components :** data blocks that build the description.
- **Content :** managed and presented using PCF Components.
- **Behaviour :** responsible of the generation and control of events generated in execution time.

To support these aspects, the data model of the transcoder has to contain a minimum set of DVB-PCF elements to create a real iTV service experience. This set includes visual components (`Scene, Images, Textbox, Rectangle, etc`), content components (`Size, Position, Color, URI, etc`), behaviour components (`OnEvent, SceneNavigate, Target`) and others that involve visual aspects, but are also closely related to the control of the service (`Service, Scene, Menu`)

2.2.2 *Load Manager*

The way that all objects are rendered in the application is independent from how they are created, so it implies specific load functions for every object. `LoadManager` package contains a set of classes, almost one for every PCF element supported in the transcoder. Each of these classes obtains all the features information of the element and creates the visual equivalent object in Flash. When the complexity of DVB-PCF elements included increases, a bigger combination of ActionScript objects is usually required, in order to be more accurate. But the chosen method has been effective up to now. This package, jointly with DVB-PCF Data Model, was exactly the same in the first version of the interpreter, mentioned above.

2.2.3 *Description parser*

This module is completely implemented in Java. It makes XML reading easier, permitting to recognize every PCF element and creating an AS file. Based on previous `Data Model` and `LoadManager`, these files contain only simple calls to these modules, first initialising its elements and then calling its specific loading function. This separation is what permits to change AS 2.0 modules by other technology ones, reusing Java modules to create another transcoder.

2.3 Scene navigation

The most difficult task of generating a iTV service using an XML description is to store all the interactivity information. In a standard Flash application, it is very easy to define a button or a menu that makes a specific task. In the case of the transcoder, it has to define these actions and handle them in a way as generic as possible, because the application controls are created on the fly, and associated to elements, as the application is created sequentially. It requires managing and creating a context for the application, and monitor it, as a button could do different things depending on the life-cycle point of the service.

In addition, DVB-PCF specification has precisely defined the way of how to define events, and it was very helpful in the definition of the navigation between scenes, but there are implicit events that are more undetermined, like the navigation within a scene. However, this transcoder implements a simple navigation between all focusable objects that are in a scene using *arrow keys* of the device or remote control.

2.4 Development tools

The development of this transcoder is completely multiplatform and open source. It has been implemented using Java and Open Flash tools, as Mtasc and Swfmill compilers, over Eclipse IDE framework. It offers a flexibility extra point, so transcoder can work (and has been tested) over different operating systems like Windows, Linux, etc.

The Flash Lite applications resulted from this transcoder has been tested using Flash Lite 3.1 Player in different devices (a Pentium IV 2.2GHz PC and different types of set-

Figure 3: Football Interactive Application Sample

Figure 4: Multiple Menu Interactive Application Sample

top boxes), with good processing performance in all of them. In its current version, transcoder can efficiently render most of the visual effects presented in DVB-PCF. In order to test these elements and to check that every feature is depicted correctly, some interactive services were designed.

3. RESULTS

As an example of interactive test service, a football interactive application is devised in order to be sent during a football match broadcasting. This application shows different scenes where user can navigate using their remote control's left and right arrow keys. Figure 3 shows a scene of this application that contains the league's classification table, and a horizontal menu that permits the scene navigation control. In addition, in this scene there are three different focusable objects (horizontal menu, a button to go to main menu and an exit button) that user can focus using up and down arrow keys. In case that the scene contains vertical menus, the control will swap, moving up and down within the menus, and left and right to move between objects.

This feature is also shown in Figure 4, where an application with a double menu scene is presented. User can move from one menu to another horizontally and vertically within them. This method of handling navigation in the applications rendered by the transcoder, limits the use of horizontal and vertical menus in the same scene, so it still needs to be improved in next versions.

4. CONCLUSIONS

In this paper, a solution for a transcoding application from DVB-PCF to Flash Lite is presented, allowing finalizing the iTV service production system proposed by DVB-PCF standard. This application has proved the potential of DVB-PCF format and it has presented how is possible to render DVB-PCF interactive services descriptions using Flash Lite, a widely used and on its way up format.

In this sense, it is important to emphasize the simplicity of creating applications using descriptive XML files, and transforming them into real applications. It brings forward the possibility to create interactive services even to people that do not have any technical knowledge. This method also permits to create applications regardless of the final target, and reuse their descriptions integrally (or only part of them) to easily generate new services.

DVB-PCF elements supported by the first version of the transcoder, as shown in the results section, are enough to generate a powerful interactive experience to user, showing many different types of visual objects. It also provides support to some event elements defined in DVB-PCF specification that offers to the user a complete and intuitive navigation among different objects and scenes of the service. However, this version still does not support more advanced descriptions that allow videos, other more complex graphical objects or return channel connections. These features are still in development and they are expected in next versions.

5. REFERENCES

[1] Digital video broadcasting; portable content format (pcf) specification 1.0. ETSI TS 102 523 V1.1.1, 2006.

[2] Adobe. Adobe and broadcom bring the adobe flash platform to tvs. 2009 International CES, press release, January 2009.

[3] BBC. Adobe flash secures set-top deal. http://news.bbc.co.uk/2/hi/8008070.stm, April 2009.

[4] R. Bradbury, R. Cartwright, J. Hunter, S. Perrott, and J. Rosser. Portable content format: a standard for describing an interactive digital television service. R&d white paper whp 134, BBC Research & Development, 2006.

[5] P. Bugnot. Commission of european communities; commission staff working paper on the interoperability of digital interactive television services pursuant to communication com(2004), 2004.

[6] E. Eldrom, S. Janousek, and T. Joos. *AdvanceED Flash on Devices*. Apress and friends of ED books, 2009.

Annex 3. Pòster presentat a EUROiTV 2010, Tampere, Finlàndia

DVB-PCF to Flash Lite Transcoder

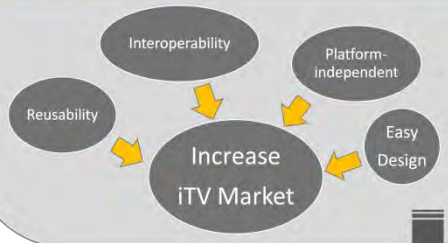
R. Martín de Pozuelo, F. Enrich, G. Fernandez
GTM - Grup de Recerca en Tecnologies Mèdia
ENGINYERIA I ARQUITECTURA LA SALLE. UNIVERSITAT RAMON LLULL

Heterogeneity of Interactive TV (iTV) services and platforms

- Increasing market of digital iTV applications for non-PC devices without a consensus standard
- Developers make their applications for a specific target platform → Reusability difficulties

DVB PCF Portable Content Format

- DVB-PCF focuses in the description of interactive services by describing the visual experience
- Platform-independent format for the business-to-business interchange of interactive content
- Interoperability of authoring tools, head-end systems and broadcast networks
- Not an end-to-end system that could solve this immediately → Necessity of translators



ADOBE® FLASH® LITE™

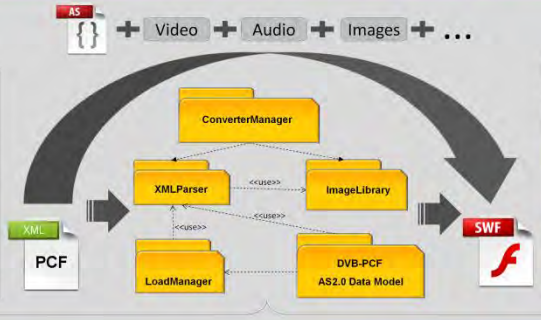
- New options for accessing web content on Digital Television, smart-phones, hand-helds, etc. with low processing requirements
- Powerful tool that paves the way for dynamic applications and rich entertainment experiences
- Flash Lite 3 supports widely used video codecs H.264 , On2 VP6 andSorenson
- Multiplatform support → Flash Lite 3 works across multiple mobile and consumer electronics devices (LG, Motorola, Nokia, NTT DoCoMo, Samsung, Sony Ericsson, etc)

	2008	2009
Global Total		
Flash Lite 1.0	12,049,000	10,143,000
Flash Lite 1.1	270,263,000	163,098,000
Flash Lite 2.0	250,634,000	281,552,000
Flash Lite 2.1	145,875,000	314,648,000
Flash Lite 3.0+	40,687,000	284,225,000
Total	**719,508,000**	**1,053,666,000**

DVB-PCF to Flash Lite 3.1 Transcoder = End-to-end Solution

- Transcoder development for multi-platform environments using a combination of open source tools (Eclipse + Java + Open Source Flash)
- Automatic render of iTV Flash-based services using a DVB-PCF description (XML format)
- Support for a complete set of DVB-PCF components → Video, audio, images, menus, buttons, textboxes, scene navigation, feeds, etc.
- Simplification of iTV applications development: Permits to focus on the application design development without technical knowledge

DVB-PCF template



iTV application

- **ConverterManager** *package includes all classes related to complete the conversion and make up the SWF file, managing the outputs from XMLParser and ImageLibrary*
- **XMLParser** *package includes all classes related to PCF description interpretation, and use* **LoadManager** *and Data Model package to create an AS file that includes all the elements defined in PCF description*
- **ImageLibrary** *package classes control image imports and create a library with all the images specified in the PCF description file*
- **DVB-PCF ActionScript Data Model** *package includes all classes related to AS objects that can form a PCF description*
- **LoadManager** *package contains all classes related to load PCF information into memory including image loading issues*