Instituto de Computación
Facultad de Ingeniería
Universidad de la República

Distribución de video en vivo para el Plan Ceibal utilizando la plataforma GoalBit

Autores:
Ernesto Dufrechou
Alberto Almeida

Tutor:
Dr. Pablo Rodríguez Bocca

Resumen

El Plan CEIBAL es un proyecto llevado a cabo por el gobierno del Uruguay con el objetivo de contribuir a la equidad, la democratización del conocimiento y la mejora educativa [20]. Consiste, entre otras cosas, en la entrega de un computador personal de bajo costo (llamado XO) a cada estudiante y maestro de escuela primaria pública, haciendo posible para la gran mayoría de los niños el acceso a este tipo de tecnologías.

Desde los comienzos del Plan Ceibal se ha realizado un importante esfuerzo con el objetivo de crear aplicaciones para las nuevas PC, que contribuyan al entretenimiento y a la educación de los niños uruguayos. En este sentido, una de las carencias de estas PC consiste en la ausencia de un reproductor multimedia adecuado a las capacidades de las mismas y la incapacidad de producir y consumir *streaming* de video de una forma acorde a la realidad del Plan Ceibal. Esta realidad incluye aspectos tecnológicos, como por ejemplo conectividad y capacidad de procesamiento de las PCs, y aspectos sociales considerando las características del usuario final y la misión del Plan. Por ejemplo, deben preverse y evitarse usos indebidos del sistema.

Para alcanzar los objetivos anteriores se eligió utilizar GoalBit, la primera red de distribución de video *peer to peer* gratuita y de código abierto. Este proyecto consistió en adaptar GoalBit a la realidad del Plan Ceibal obteniendo un producto que permita a los niños del Plan generar y consumir *streaming* de video y reproducir archivos multimedia de forma segura.

Se diseñó una interfaz de usuario Web como *frontend* del GoalBit Media Player para las XO extendiendo un Web Plugin del reproductor multimedia VLC para el navegador Mozilla Firefox.

Para poder generar un *streaming* de video a través de la Web, se diseñó e implementó la funcionalidad llamada Web Professional Broadcasting, incorporándola a los dos subsistemas principales de GoalBit, el GoalBit Media Player y a la GoalBit Suite.

También se diseñó e implementó un Portal Web en donde los niños pueden buscar contenidos, reproducirlos y transmitirse a ellos mismos utilizando la cámara Web de la XO.

Con respecto a evitar el uso indebido del sistema y garantizar la seguridad de los usuarios, se implementó un sistema de firmas digitales que evita ataques de imitación (impersonation attack) y la inyección de contenido indebido en los canales de video. También se incluye un sistema que permite denunciar y dar de baja canales con contenido indebido desde un Panel de Control para administradores.

Finalmente, se empaqueta el cliente de la solución en una "Actividad" (así son llamadas las aplicaciones en el contexto del Plan Ceibal) que puede instalarse y ejecutarse en una XO de forma muy sencilla. Dicha actividad consiste en un navegador web diseñado para el entorno gráfico de las XO (Sugar) modificado para utilizar GoalBit Media Player junto con el Portal Web de forma automática y transparente al usuario.

Índice

Parte I
Introducción

Las mayores posibilidades tecnológicas con las que cuenta la infraestructura de Internet hoy en día y la masificación de su uso han provocado que la Red se convierta en otro medio de entretenimiento. Gracias a esto, los últimos años han mostrado un crecimiento sustancial del tráfico de Internet dedicado a la distribución de contenido multimedia. Según [25], en junio de 2010 el 40 % del tráfico total de Internet correspondía a este tipo de contenido, y se espera que para el 2014 esta cifra alcance el 57 %. En particular, la transmisión de video en vivo está cobrando cada vez más importancia.

En respuesta a este fenómeno, la industria ha dedicado un gran esfuerzo en generar soluciones eficientes para la distribución de video en vivo de alta calidad. De esta manera han surgido los sistemas de distribución de video en vivo entre pares. Dentro de este contexto se ha desarrollado GoalBit, la primer red P2P para la distribución de video en vivo, gratuita y de código abierto [51].

GoalBit consiste en una plataforma completa para la generación y distribución de video de alta calidad. Dicha plataforma está basada en el funcionamiento de BitTorrent[27], y está formada en principio por un *tracker* y varias clases de clientes o *peers* quienes intercambian la información de video entre sí.

El Plan Ceibal es un proyecto educativo del gobierno uruguayo que consiste parcialmente en entregar a cada alumno de la escuela primaria pública un PC de bajo costo llamado XO. Como contribución a dicho plan se propuso adaptar la plataforma GoalBit para poder ser utilizada en el contexto del mismo.

Actualmente el Plan Ceibal no cuenta con una solución para poder generar y compartir *streaming* de video. El navegador que está instalado por defecto en las XO permite visualizar *streaming* de video almacenado en páginas del estilo de www.youtube.com. Sin embargo, la capacidad de cómputo requerida por Web Plugins como Flash Player 10 es más que la que el procesador y memoria de la XO pueden ofrecer, provocando que el video se entrecorte y sea dificultoso de entender.

Existen trabajos previos que han intentado dar una solución a este problema. El Mundial de Fútbol (Sudáfrica 2010) dejó latente la necesidad de una solución para el problema del consumo de *streaming* desde las XO del Plan Ceibal. En respuesta a lo anterior, la organización de voluntarios CeibalJAM desarrolló JA-Media, un reproductor multimedia completo, adaptado para la XO, que puede actuar como cliente de diversos protocolos de *streaming* de audio y video. Sin embargo, a pesar de que sí resuelve la falta de un reproductor de medios en la XO, no termina de resolver el problema del consumo de *streaming*. Un escenario que podría esperarse dentro de un salón de clase en la escuela primaria es el de varios o todos niños del salón intentando consumir el *streaming* que les ha asignado la maestra del grupo. Actualmente, tal como se da en el caso de JA-Media, lo anterior significa que cada niño intentará descargar completamente el *streaming* para poder visualizarlo, ya que no existe ninguna estrategia eficiente

para la distribución del contenido de video. Al estar todas las computadoras de una escuela detrás de un *Proxy*, la salida hacia internet es única y tiene un ancho de banda de a lo sumo 10Mbps de bajada, que debe repartirse entre todos los niños que lo utilizan. Esto hace imposible una buena calidad de experiencia al consumir el *streaming* en el caso anterior.

Contrario al enfoque clásico cliente-servidor, GoalBit utiliza un mecanismo de distribución de video *peer to peer* en el cual los pares intercambian piezas del video entre sí. En este enfoque, los clientes descargan el video desde muchas fuentes simultáneas, en particular, descargan de aquellos otros clientes interesados en visualizar el mismo video. En el escenario del salón de clase en el cual varios niños intentan ver el mismo video, la mayor parte del contenido se distribuyen internamente en la red local. En un caso ideal, el video se descargaría una única vez siendo re-distribuido dentro del salón.

Por lo expresado anteriormente, resulta de particular interés evaluar el desempeño de GoalBit en el contexto del Plan Ceibal. Para lograr este objetivo es necesario adaptarlo para cumplir con los requerimientos que impone la realidad del Plan.

Tal como lo fue en el caso de JAMedia, es necesario construir una interfaz de usuario para GoalBit que funcione en el entorno gráfico de las PC del Plan Ceibal. Esta interfaz debe seguir los principios básicos del desarrollo de interfaces para las XO, algunos de los cuales son expresados en [48]. En particular, la interacción con el usuario debe ser sencilla y fácil de aprender, con un *look and feel* adecuado a la franja etaria del usuario objetivo. La mejor manera de brindar esta interfaz es a través del desarrollo de una "Actividad" (véase Sección 8) (así se llaman a las aplicaciones dentro del programa OLPC[6]) instalable de forma en la que el niño está acostumbrado.

Además de proveer una interfaz para la XO, se desea proporcionar una plataforma completa para la producción, la publicación y el consumo de *streaming* para las XO.

Se está marcando cada vez más un cambio de paradigma en cuanto a cierto tipo de aplicaciones, en el cual se tiende a abandonar la interfaz de escritorio y proporcionar una interfaz Web para acceder al servicio. Las aplicaciones multimedia y sobre todo las que trabajan con *streaming* de video son un claro ejemplo. Basta ver como se han popularizado portales del estilo de Youtube [10] o Veetle [14], los cuales brindan acceso a contenido, proporcionan un reproductor multimedia y en algunos casos, por ejemplo el de Veetle, permiten incluso realizar broadcast de un contenido. Este tipo de portales han resultado ser interfaces idóneas para los mencionados sistemas en la medida de que vuelven más accesible el producto y proporcionan una comunicación natural con los servidores que publican o administran el contenido. Es por esto que se construyó un Portal Web que reúne todas estas funcionalidades haciéndolas disponibles a través de Internet para el Plan Ceibal.

La interfaz de usuario, entonces, es una interfaz Web compuesta por un Web Plugin para Firefox, que es compatible con la actividad Navegar de las XO del Plan Ceibal y actúa como *frontend* del GoalBit Media Player, y un portal Web en donde los usuarios podrán acceder a las funcionalidades del sistema. La

solución es presentada al usuario como una Actividad instalable en el entorno Sugar que incluye una versión del navegador modificada para utilizar el portal y el GoalBit Media Player de forma automática. La Actividad puede ser instalada de la forma acostumbrada dentro del Plan Ceibal.

Al ser accesible desde Internet, la solución debe proporcionar mecanismos para evitar su uso indebido. El primero consiste en la identificación de los usuarios del sistema y en no permitir el uso anónimo del mismo. Luego debe proveerse un medio que permita confiar en la autenticidad del usuario del que se descargan piezas de video. En particular, debe garantizarse que todas las piezas de video consumidas por un cliente hayan sido originadas por el usuario "titular" del canal, aunque por la naturaleza del sistema P2P puedan haber sido descargadas de otros usuarios. De esta forma todos los contenidos que se consumen en el sistema están asociados a un usuario. Es deseable también que dichos usuarios estén asociados a una persona física u organización dentro del Plan Ceibal, lo cual también se tuvo en cuenta.

Para lograr dicha confianza se implementó un sistema de firmas digitales utilizando el algoritmo DSA [45], en el cual todas las piezas de video viajan firmadas con la clave privada de quien las genera. El sistema se integra totalmente a GoalBit e incluye la generación y distribución de claves, el firmado de piezas y la verificación de las mismas de forma totalmente transparente al usuario.

Todas las funcionalidades anteriores se engloban dentro de una nueva forma de utilizar GoalBit que surge a partir de este proyecto: el "Broadcast Web Profesional" (o Web Professional Broadcasting). Esta funcionalidad permite hacer uso de la GoalBit Suite (un subsistema de GoalBit) al realizar un broadcast desde el navegador Web. La GoalBit Suite mantiene un control centralizado sobre los flujos de video que se realizan a través del portal. Por ejemplo se pueden dar de alta canales, borrarlos, saber si un canal está activo, si es público o es privado, controlar el acceso seguro al mismo, etc.

El resto del documento se estructura de la siguiente manera:

En la parte siguiente se exponen conceptos generales que se manejan durante el desarrollo del documento y se analizan trabajos relacionados con el área de este proyecto. Luego, en la parte III se detalla la solución que se ha descrito brevemente en esta introducción. En un primer capítulo de esta parte se describe el ambiente de desarrollo del proyecto y luego se encuentran capítulos correspondientes a los distintos aspectos que forman la solución global, a saber, el GoalBit Media Player (edición Web Plugin), la Actividad GoalBit, el Portal GoalBit Ceibal, una descripción de la funcionalidad llamada Web Professional Broadcasting, el capítulo de Seguridad, en el cual se describe el desarrollo del sistema de firmas digitales, y finalmente se presentan las pruebas realizadas al producto obtenido y la evaluación de las mismas. Por último, la parte IV corresponde a las conclusiones obtenidas.

Parte II
Conceptos Generales

1. Multimedia Streaming sobre Internet

En los últimos años han habido mejoras en la infraestructura de Internet, como por ejemplo enlaces residenciales de alta velocidad, redes de distribución de contenido y nuevos protocolos de Internet orientados a brindar calidad de servicio [38]. A su vez, el costo del hardware de almacenamiento se abarata y la capacidad de dicho hardware aumenta a una tasa significativa, incluso más alta que la de la capacidad de procesamiento y memoria, permitiendo almacenar grandes cantidades de información, en particular audio y video de alta calidad. Estos factores facilitan en gran medida la distribución de contenido audiovisual a través de Internet, lo cual ha generado una creciente demanda de dicho contenido, en particular de transmisiones en vivo, juntando dos tecnologías ya de por sí muy populares: la televisión y la Web. Se han popularizado entonces las aplicaciones multimedia sobre Internet y en particular las de "live streaming" o transmisión de audio y video en tiempo real. Estas aplicaciones son grandes consumidoras de ancho de banda y deben intentar mantener cierta calidad de servicio para que su uso sea satisfactorio, por lo cual son necesarios métodos eficientes de distribución de video que se enfoquen en el ahorro de ancho de banda y en la recuperación eficiente de errores y retardos generados por la red "best-effort" sobre la cual deben trabajar.

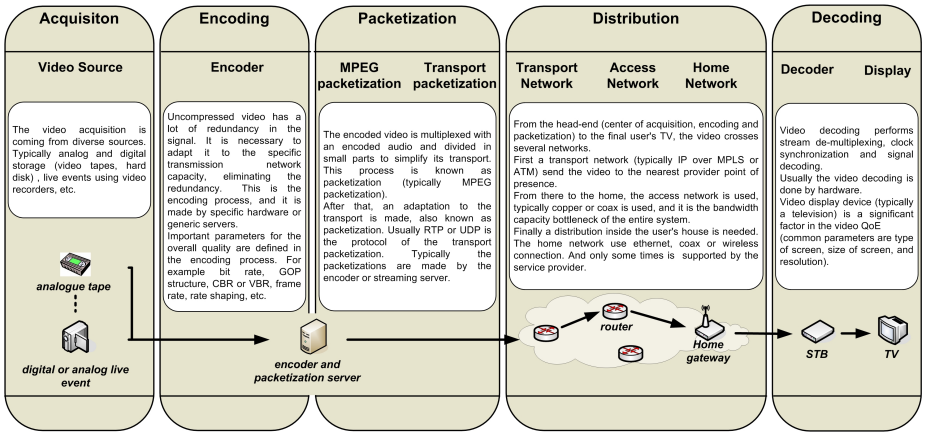

Figura 1: Proceso de transmisión de audio/video

El proceso de transmisión de video implica distintas etapas, y deben tomarse medidas en cada una de estas para que el servicio sea eficiente y para atender los distintos tipos de necesidades relacionadas por ejemplo con la calidad demandada por el cliente, el medio de transmisión, etc [23]. Puede verse un esquema

de dichas etapas en la Figura 1. Las mismas son:

- Adquisición del contenido
- Codificación / Paquetización
- Distribución
- Decodificación

A continuación describiremos brevemente cada etapa:

Adquisición Es realizada por ejemplo por una cámara, micrófono o capturadora, e incluye posiblemente un medio de almacenamiento el cual puede ser analógico o digital. En esta etapa se realiza el muestreo y la cuantificación de los datos. El muestreo consiste en tomar muestras periódicas de una señal analógica mientras que la cuantificación es básicamente la conversión de los valores discretos de amplitud obtenidos durante el muestreo a un código binario acotado, o sea su representación digital.

Compresión / Codificación y Paquetización Frecuentemente la información es capturada a una alta tasa de muestreo, teniendo en cuenta el teorema de Nyquist [60], en el cual se establece que para que una señal analógica acotada en frecuencia pueda ser reconstruida completamente a partir de una serie de muestras, éstas deben ser tomadas a más del doble de la frecuencia máxima alcanzada por la señal. Por ejemplo, el rango de audición humana está entre los 20 Hz y los 20 KHz. Para satisfacer este rango una señal de audio debería ser muestreada a por lo menos 40 KHz. Por esta y otras razones los CD de audio utilizan una tasa de muestreo de 44.1 KHz. Si cada dato muestreado se representa luego de la etapa de cuantificación por 16 bits y se utilizan 2 canales de audio (como en el caso del stereo) una canción de 3 minutos ocuparía unos 30 Mb. Aplicando la misma lógica y agregando información de video al audio capturado obtenemos grandes cantidades de información.

Debido a la disponibilidad de medios de almacenamiento, en muchos escenarios es válido mantener esta información almacenada en un servidor por ejemplo, pudiendo ser descargada completamente y luego reproducida por algún reproductor multimedia. Sin embargo, para escenarios en donde la información es reproducida en tiempo real se requeriría un altísimo ancho de banda. En el ejemplo del CD de audio sería necesario un ancho de banda de por lo menos 1411,2 Kbps dedicado a la descarga y reproducción del audio. Si se le agregara video donde cada muestra es una imagen posiblemente de varios Kb el ancho de banda necesario sería muchísimo mayor. Por esta razón no es aceptable transmitir video en tiempo real sin antes pasar por una etapa de compresión del mismo. Por otro lado debe tomarse en cuenta que comprimir excesivamente un video puede significar en un deterioro ostensible de su calidad, generando artefactos molestos e incluso distractores en el mismo.

En la compresión de video se intenta reducir la cantidad de información necesaria para representar con más o menos fidelidad la información original.

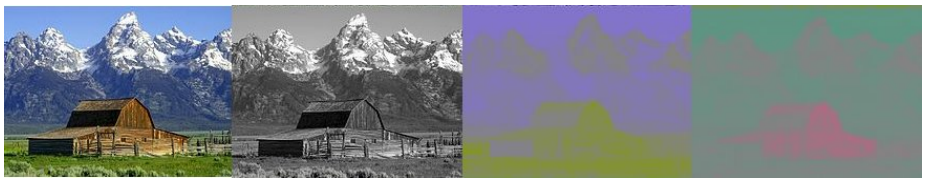

Figura 2: Una imagen a color y los elementos Y, CB y CR de la misma. La imagen Y es escencialmente una copia en escala de grises de la original.[62]

Frecuentemente se toman en cuenta factores relacionados con la percepción para eliminar información que no es sensible para el ser humano y así representar el video con menos cantidad de datos sin que éste aparente una pérdida significativa de calidad. Por ejemplo, entre las técnicas más comunes para comprimir video se encuentran el entrelazado, que consiste en enviar por cada frame sólo las líneas pares o impares intercaladamente, de forma de transmitir la mitad de la información sin que el ojo humano detecte ningún cambio. Otras técnicas tienen que ver con la representación de la información de color. Mientras que una representación común del color de un pixel es la RGB (valor de rojo, verde y azul) ya que está directamente relacionada con el funcionamiento del hardware de display (tubo de rayos catódicos por ejemplo), esta representación posee redundancia mutua entre los tres valores y no es adecuada al modelo perceptual del ojo humano. La visión humana es generada a partir del reflejo de luz en la retina, en la cual existen dos tipos de células foto-receptoras: conos y bastones. Los bastones detectan cambios en la intensidad de luz (brillo y oscuridad) mientras que los conos se especializan en el color . Por esta razón es utilizada una representación del color más adecuada a nuestro modelo perceptual (YCrBr) que separa la información de color de la de luminancia. Al separar esta información es posible eliminar parte de la información de color transmitida de una forma adecuada a nuestro sistema perceptual. Por ejemplo, el ojo humano es más sensible a cambios en la luminosidad por lo cual esta información puede ser enviada a mayor ancho de banda que la información de tinte. Usualmente se envía la información de luminancia de todos los píxeles de determinado frame (Y) pero solo se envía la información de color de algunos píxeles. También son utilizadas técnicas de compensación de movimiento, las cuales consisten en representar un frame mediante transformaciones aplicadas sobre un frame anterior o incluso uno futuro que ya ha sido almacenado.

Una vez realizada la compresión de los datos el video es repartido en piezas o paquetes antes de ser transmitido. Este proceso es conocido como Paquetización o muxing, y consiste en generar paquetes con datos de audio, video y meta-data que permite identificar los fragmentos de video y audio, transportar subtítulos dentro del *streaming*, e incluso transmitir varios canales de audio o video en un mismo *streaming* (por ejemplo dos canales de audio con idiomas distintos). La paquetización implica también el encapsulamiento de los paquetes de video en paquetes de algún protocolo de transporte (al trabajar con redes IP como en

nuestro caso) como por ejemplo paquetes TCP, UDP o RTP.

Distribución La mayoría de los sistemas de transmisión de video varían muy poco en lo que se refiere a las primeras tres etapas. Pese a diferencias entre distintos encoders y muxers utilizados, en general se utilizan implementaciones del estándar MPEG, y los procedimientos para codificar video son similares entre estos sistemas. La etapa de distribución del video, por otro lado, es la etapa en donde se encuentran más diferencias. Esta etapa depende del medio en el que se distribuya el video (vía satélite, por cable, distribución terrestre o mediante una red IP) y sus características.

Un caso particular de distribución de video en una red IP es el caso que interesa en este proyecto: la distribución de video a través de Internet. El diseño mismo de Internet impone ciertas dificultades a la hora de transmitir video. Internet puede definirse como una red "best effort" de conmutación de paquetes. La capa de red (o capa 3 de acuerdo al modelo OSI) proporciona un servicio de enrutamiento de paquetes que no garantiza la confiabilidad de los datos (los paquetes pueden perderse, alterarse, llegar duplicados, retrasarse, etc) ni brinda seguridad de ningún tipo. Para contar con características como confiabilidad, seguridad o calidad de servicio deben implementarse protocolos en las capas superiores que atiendan estos problemas.

Las aplicaciones multimedia en tiempo real, a su vez, tienen características particulares también. Son sensibles a los retardos, o sea, se experimenta una mala calidad de servicio cuando el reproductor multimedia intenta reproducir información que todavía no ha llegado. Por otro lado son tolerantes ante las pérdidas, ya que la pérdida de un paquete puede causar un efecto que incluso pase desapercibido durante la reproducción. Es por esto que UDP parece un protocolo adecuado para manejar este tipo de información. Al utilizar UDP, la aplicación debe implementar los servicios de detección/corrección de errores, manejo de retardos y pérdidas, etc. Debe tenerse en cuenta que UDP transmite a una tasa constante sin considerar el grado de congestión de la red. TCP por otro lado proporciona confiabilidad en la recepción de los datos, pero basa su funcionamiento en retransmisiones ante la pérdida de paquetes, lo cual en una aplicación de video en vivo podría detener el video mientras es detectada una falla y retransmitida cierta información. También transmite a una tasa fluctuante debido a que estima el grado de congestión de la red a partir de los paquetes perdidos. Aunque TCP y HTTP son menos eficientes para este tipo de aplicaciones, son elegidos en la práctica dado que son más seguros y "atraviesan mejor" los firewalls. También es posible utilizar protocolos de transporte que funcionan sobre UDP y lo extienden de forma de agregar información útil para la aplicación de video. Estos protocolos usualmente consisten en agregar un encabezado con cierta información con la que UDP no cuenta (por ejemplo números de secuencia y timestamps) a los datos de video antes de empaquetarlos en un datagrama UDP. Pese a que por sus características son clasificados como protocolos de transporte, son implementados a nivel de aplicación. Un ejemplo popular de estos protocolos es RTP acompañado por su protocolo de control

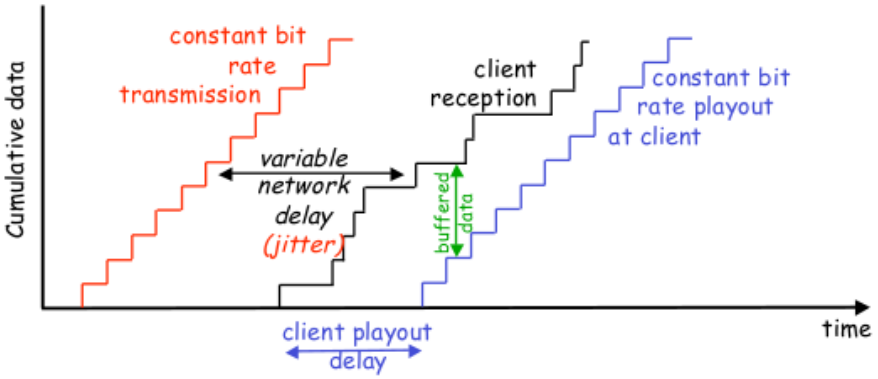

Figura 3: JITTER. Sea cual sea el protocolo utilizado la aplicación debe proporcionar una forma de lidiar con la variación del retardo con que se reciben los paquetes respecto a cuando se envían (jitter). En general todas las aplicaciones de este estilo utilizan *buffers* y retardan el comienzo de la reproducción unos segundos para evitar que el jitter produzca cortes en la reproducción del video.

RTCP.

Sea cual sea el protocolo utilizado la aplicación debe proporcionar una forma de lidiar con la variación del retardo con que se reciben los paquetes respecto a cuando se envían (jitter). En general todas las aplicaciones de este estilo utilizan *buffers* y retardan el comienzo de la reproducción unos segundos para evitar que el jitter produzca cortes en la reproducción del video. Lo anterior puede apreciarse en el esquema de la Figura 3.

Para disminuir los retardos y las pérdidas de paquetes ocasionadas por rutas largas (con demasiados saltos), y además proporcionar balance de carga a los servidores de *streaming,* existen las Content Distribution Networks (CDN). Éstas son redes de servidores distribuidos alrededor del Mundo los cuales replican los contenidos de determinado proveedor en distintas zonas geográficas a los efectos de acercar lo más posible el contenido al usuario final. Las CDN (ver Figura 4) crean un mapa con distancias entre ISP's y los nodos de la CDN. Cuando llega una petición de contenido, se determina a qué ISP corresponde la petición y un servidor DNS determina, usando el mapa, qué servidor de la CDN es el más cercano al ISP devolviendo la dirección de dicho servidor.

Otra limitante impuesta por Internet a las aplicaciones de *streaming* es la carencia de infraestructura especial para realizar broadcast o multicast. Estos servicios consisten en transmitir el mismo flujo de video a todos o algunos de los miembros de un grupo evitando la replicación de los paquetes. En general estos servicios deben ser implementados por los routers y los host de una red IP mediante protocolos como IGMP y no son masivamente difundidos. La alternativa más comunmente aplicada es el unicast, en donde cada cliente consume el *streaming* desde un servidor y éste duplica los paquetes para todos los clientes

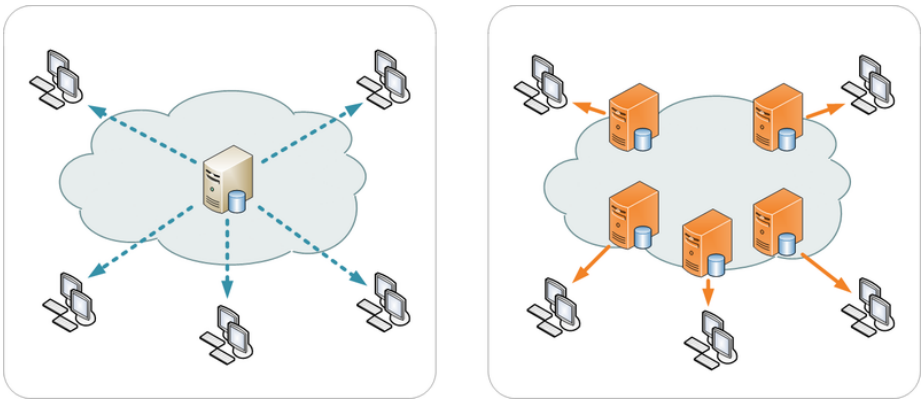

Figura 4: Las CDN crean un mapa con distancias entre ISP's y los nodos de la CDN. Cuando llega una petición de contenido, se determina a qué ISP corresponde la petición y un servidor DNS determina, usando el mapa, qué servidor de la CDN es el más cercano al ISP devolviendo la dirección de dicho servidor.

que consuman el mismo *streaming*. Esto trae consigo un desperdicio de ancho de banda y una inyección mayor de tráfico en la red que provoca congestión en los routers y retardos en la llegada de los paquetes. Provoca también la aparición de un cuello de botella en el servidor de *streaming* y en los gateways detrás de los cuales hay varios host consumiendo determinado flujo de video. Frente a este escenario, ha ido ganando protagonismo la alternativa de la distribución de video P2P, la cual nos interesa particularmente en este trabajo. La alternativa consiste en utilizar los recursos ociosos de los nodos de una red (por ejemplo capacidad de procesamiento de los nodos y ancho de banda) para realizar la distribución del contenido. En esta forma de distribución cada nodo de la red o "par" (peer) intercambia de forma "justa" piezas de video con el fin de conseguir las piezas que le faltan para poder reproducirlo. A su vez cada peer contribuye con los demás brindándole las piezas que ya ha conseguido a otros usuarios que las necesiten. Mediante este sistema cada host de una red local recibiría parte del video, lo cual no congestiona el enlace hacia afuera, y luego el video se distribuye dentro de la red local. Por otro lado el sistema P2P mejora su rendimiento a medida que crece la cantidad de pares colaborando dentro de él, lo cual en principio lo hace adecuado a un escenario en donde un número grande de pares desean consumir determinado *streaming* de video. Por tanto las redes P2P implementan el multicast en capa de aplicación no requiriendo de CDNs (Content Distribution Network) para la distribución, pero logrando el mismo objetivo: Distribuir el contenido desde un punto cercano al consumidor.

Decodificación Solucionada la distribución, la última etapa consiste en la decodificación y reproducción de los datos recibidos. En este proceso participa el demuxer, artefacto dual al muxer, y el decoder, contra-parte del encoder. Estos artefactos forman parte del reproductor multimedia y pueden ser computadoras,

set-top-box, móviles, televisores, etc.

1.1. Codecs

Una codificación o compresión de video, es una técnica diseñada para la eliminación de redundancias en la señal de video. En una señal de video hay esencialmente dos tipos de información. Existe aquella que es nueva e impredecible y aquella que puede ser anticipada o deducida. La primera es llamada entropía y la otra redundancia. En una secuencia de imágenes digitales (como en el caso de señal de video digital) existen redundancias tanto temporales como espaciales. Las redundancias temporales se dan cuando una imagen y la siguiente en la secuencia son similares, y son frecuentes dado que en general imágenes continuas corresponden a movimientos en objetos de la escena (excepto en cambios de escena). La redundancia espacial, por su lado, consiste en que dentro de una misma imagen, bits adyacentes tienden a no variar o ser similares. La compresión de la señal funciona separando la entropía de la redundancia. Solo la entropía es almacenada o transmitida mientras que la redundancia se computa o se genera a partir de la información almacenada o transmitida, este concepto puede apreciarse en la Figura 5.

Un codificador ideal extraería toda la entropía y solo eso sería transmitido al decodificador. Un decodificador ideal luego reproduciría la señal original. En la práctica este ideal no puede ser alcanzado. Un codificador ideal sería muy complejo y causaría mucho retardo (delay) para aprovechar la redundancia temporal. En ciertas aplicaciones de broadcasting es aceptado cierto delay pero en otras como lo sería una video conferencia no lo es tanto. Además, puede lograrse una compresión muy significativa afectando imperceptiblemente la calidad del medio. Es por estas razones que no existe un sistema de compresión de video ideal. Estos métodos de compresión son por lo general con pérdida, es decir que de la señal codificada no es posible recuperar la señal original de video.

Un método de compresión es definido por su decodificador. Comúnmente el decodificador es llamado Codec por su abreviatura en inglés (COder/DECder) [24] haciendo referencia también al codificador. En general todo codec comparte los mismos principios de compresión y es por esto que el entendimiento de uno o dos de ellos se extiende, a alto nivel y sin entrar en detalles, al resto de estos. A continuación se presenta una pequeña reseña del funcionamiento de la compresión de video en los estándares MPEG y Theora.

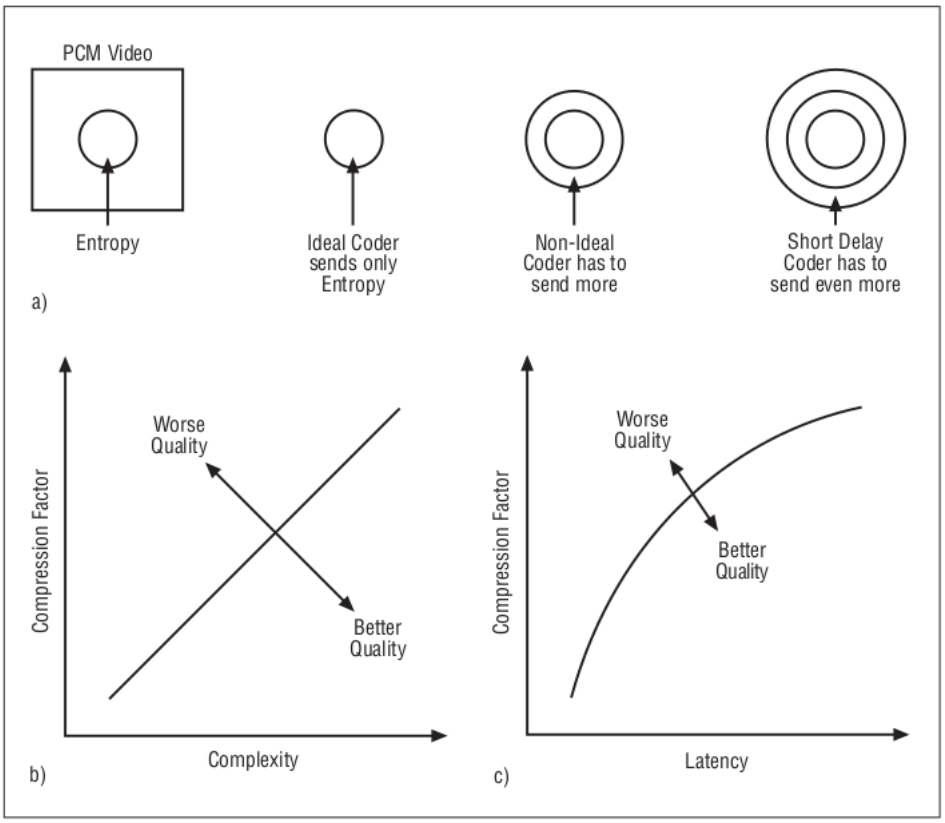

Figura 5: Codificadores

MPEG El Moving Picture Experts Group o mejor conocido como MPEG, ha desarrollado un conjunto de estándares para la compresión y transmisión de señales de audio y video, estos son MPEG-1, MPEG-2 y MPEG-4. Este último fue introducido en 1998 y reconocido como estándar internacional a comienzos de 1999 [41], este estándar comunmente utiliza como codec de video al H.264 y codec de audio al AAC.

Los codecs MPEG son un esquema híbrido de compresión para imágenes que usa codificación inter-trama, la cual explota la redundancia espacial contenida en una figura, y codificación intra-trama, que explota la redundancia temporal entre imágenes, combinando la codificación con la transformada coseno discreta (DCT)[55] para la inter-trama y la codificación predictiva para la intra-trama. Esta transformada es un algoritmo matemático que es aplicado típicamente sobre los bloques de 8x8 píxeles de la imagen. La DCT elimina redundancia en la imagen a través de la compresión de la información contenida en 64 píxeles. El cuantizador otorga los bits para los coeficientes DCT más importantes, los cuales son transmitidos.

Maneja la compresión de video a nivel de GOP (Group of Pictures o en

español Grupo de Imágenes) y para esto define 3 tipos diferentes de imágenes (también llamados frames), los frames I, P y B como podemos observar en la Figura 6.

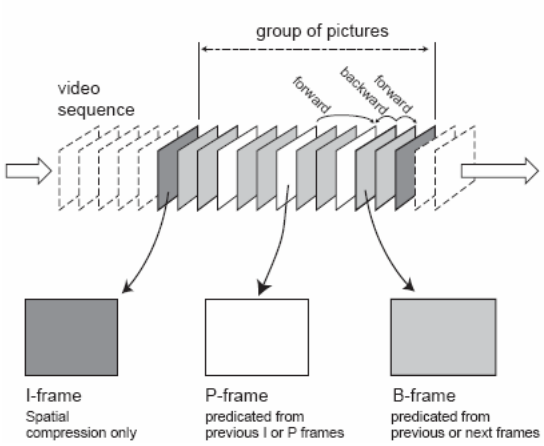

Figura 6: Tipos de frames I, P y B

Frame-I Estas se codifican como si fuesen imágenes fijas utilizando la norma JPEG, por tanto, para decodificar una imagen de este tipo no hacen falta otras imágenes de la secuencia, sino sólo ella misma. Es por esto que esta clase de frames son los de mayor tamaño y solo aplican compresión espacial de la original.

Frame-P Predecible posterior, estas están codificadas como predicción de la imagen I ó P anterior usando un mecanismo de compensación de movimiento. Para decodificar una imagen de este tipo se necesita, además de ella misma, la I ó P anterior y alcanzan a ser un 10 % del tamaño que un frame de tipo I [57].

Frame-B Predecible bidireccional, estas se codifican utilizando la I ó P anterior y la I ó P siguiente como referencia para la compensación y estimación de movimiento. Para decodificarlas hacen falta, además de ellas mismas, la I ó P anterior y la I ó P siguiente. Estas imágenes consiguen los niveles de compresión más elevados y por tanto son las más pequeñas alcanzando a ser un 2 % de un frame de tipo I [57].

A alto nivel, con estos tres tipos de imágenes es que se codifica toda la señal de video y con los frames P y B es que se comprime la misma quitando la redundancia temporal, mientras que en los frames I que son los más pesados se le quita la redundancia espacial a la original.

Theora Theora es un codec de video con pérdida de propósito general. Esta basado en el codec de video VP3 desarrollado por On2 Technologies[1], On2 donó el código fuente VP3.1 a la Fundación Xiph.Org y fue liberado bajo una licencia similar a BSD [2], renunciando irrevocablemente a todos los derechos y patentes que tenía sobre éste.

Theora es un codec con pérdida basado en bloques de transformación que utiliza bloques de 8×8 píxeles usando la Transformada Discreta del Coseno (DCT) y compensación de movimiento basada en bloques. Esto lo coloca a en la misma clase de codecs que MPEG-1, MPEG-2, MPEG-4, y H.263. Los detalles de cómo bloques individuales son organizados y cómo coeficientes DCT son almacenados difieren sustancialmente de estos codecs. Theora sólo admite cuadros intra (cuadros I en MPEG) y frames-ínter (frames-P en MPEG). No existiendo un equivalente al los frames-B, bi-predictivos encontrados en codecs MPEG [34].

Típicamente se utiliza a Theora como codec de video junto con Vorbis[33] como codec de audio, pero se pueden utilizar otros codecs de audio tales como Speex[49] y Flac (Free Lossless Audio Codec o en español codec de audio sin pérdida)[26].

1.2. Protocolos de streaming de tipo cliente-servidor

1.2.1. RTP

RTP [47] es un estándar, definido en el RFC 1889, para el encapsulamiento de información de video a enviarse a través de la red mediante UDP. Define un encabezado con información útil para la aplicación de video que permite realizar una transmisión de video almacenado en tiempo real (es decir visualizar el video a medida que se recibe).

[1] On2 Technologies fué adquirida junto con todos sus productos y tecnología por Google (www.google.com) en febrero de 2010

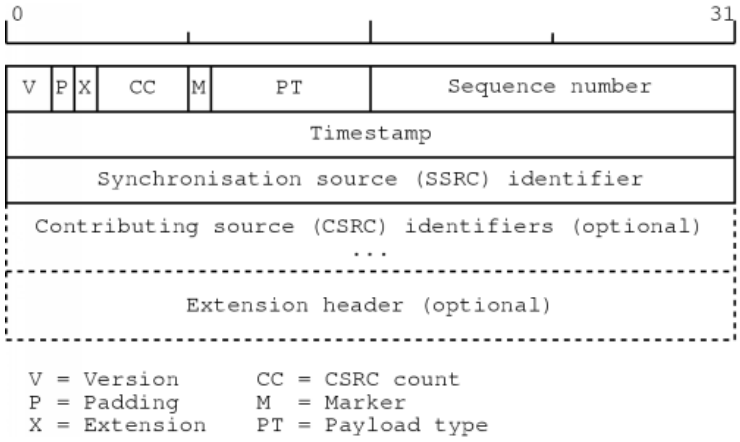

Figura 7: RTP Header

El header RTP permite a la aplicación, entre otras cosas, conocer el tipo de codificación que tienen los datos de audio o video, detectar la pérdida de paquetes y sincronizar la reproducción del video a cierta tasa, solucionando el jitter introducido por la red. Algunos de los campos más importantes de dicho header son:

- Sequence number (número de secuencia): Número entero de 16 bit que se incrementa en uno cada vez que un paquete RTP es enviado. Puede ser usado por el receptor para detectar pérdida de paquetes o reordenarlos.
- Timestamp: Es un número de 32 bits producido por un reloj del lado del emisor que se incrementa en cada tiempo de muestreo a una tasa constante. El valor refleja el momento en el que la primera muestra del paquete fue tomada. Permite la sincronización y el cálculo de jitter.
- Synchronization Source Identifier SSRC: Es un identificador que distingue entre distintos flujos RTP y es elegido de forma aleatoria por el emisor. Si se desea enviar dos flujos RTP distintos, la aplicación receptora podrá agrupar los paquetes RTP pertenecientes a cada uno de los flujos mediante su SSRC.

1.2.2. RTCP

Es un protocolo de control que suele acompañar a RTP y es definido en el RFC 1889 al igual que este último [47]. En lugar de encapsular audio o video, los paquetes RTCP contienen información estadística acerca de cantidad de paquetes enviados por el servidor, paquetes perdidos por el cliente, jitter, etc. Esta información es intercambiada en paralelo al flujo RTP y puede resultar muy útil tanto para el cliente como para el servidor, aunque el RFC no define acciones a tomar ni especifica qué hacer con estos datos.

1.3. GoalBit

GoalBit es el primer sistema *peer to peer* de código abierto (open source) para realizar transmisión de flujos de video en tiempo real [22]. El diseño y funcionamiento del sistema está inspirado en el funcionamiento de BitTorrent [27], el cual está orientado a trabajar con video almacenado.

En GoalBit el video se divide en varios flujos los cuales son enviados y recibidos por los clientes. El cliente (*peer*) recibe varios de estos flujos desde distintas fuentes y luego es capaz de reconstruir el contenido para su reproducción. Para encontrar *peers* que posean los flujos de video que le interesan, el cliente consulta a un servidor llamado *tracker*, y se registra con él declarando los flujos de video que ya ha obtenido y que ofrece a otros *peers*. Este aspecto es análogo al funcionamiento de BitTorrent[27] para la distribución de archivos.

Como se comenzó a mencionar anteriormente, la arquitectura de GoalBit está formada por distintos componentes. Éstos se pueden resumir en *broadcasters* (emisores), *peers*, *super-peers* y el *tracker*.

El *broadcaster* es quien obtiene por algún medio la información de video, la procesa y la inyecta dentro del sistema. Tiene un rol fundamental ya que si el *broadcaster* falla el flujo de video se corta y a la larga nadie recibirá más información de video. Por estos motivos usualmente es deseable no sobrecargar al *broadcaster* con otras tareas, por lo cual no se encarga de la distribución del contenido.

El *broadcaster* envía el *streaming* de video a una serie de *peers* especializados, con alta disponibilidad, capacidad de procesamiento y ancho de banda (en especial de subida), llamados *super-peers*. Estos *super-peers* son los encargados de la distribución inicial del contenido en la red. En un funcionamiento normal, los *peers* casi no se comunican directamente con el *broadcaster*, mientras que si lo hacen con los *super-peers* de forma bastante frecuente. A pesar de lo anterior, no debe confundirse esta arquitectura con una arquitectura cliente-servidor en donde los *peers* descargan el contenido de los *super-peers* únicamente. Los *super-peers* incluso no suelen poseer todo el contenido en todo momento sino que también participan del intercambio, incluso entre ellos. Uno de los roles fundamentales de los *super-peers* por ejemplo, es ayudar a *peers* que ingresan recientemente a la red sin piezas de video para intercambiar. El *super-peers* rápidamente le brinda piezas de video para que pueda comenzar a intercambiarlas con otros *peers* a fin de conseguir todo el *streaming*. También ayuda de esta manera a *peers* que se quedan atrás en el intercambio y no contienen piezas útiles para otros *peers*.

Los *peers*, por su lado, son los usuarios finales del sistema. Ellos intentan descargar todas las piezas del *streaming* con el fin de reproducirlas con un reproductor multimedia. Mientras que los *super-peers* son pocos y tienen una naturaleza más bien estática dentro del sistema, los *peers* son el componente más abundante del mismo y tienen una naturaleza muy dinámica, o sea, se conectan y desconectan con frecuencia.

Para que la conexión entre *peers* y el intercambio sea posible, es necesario un sistema que permita descubrir quién tiene cierta pieza del *streaming* y cómo

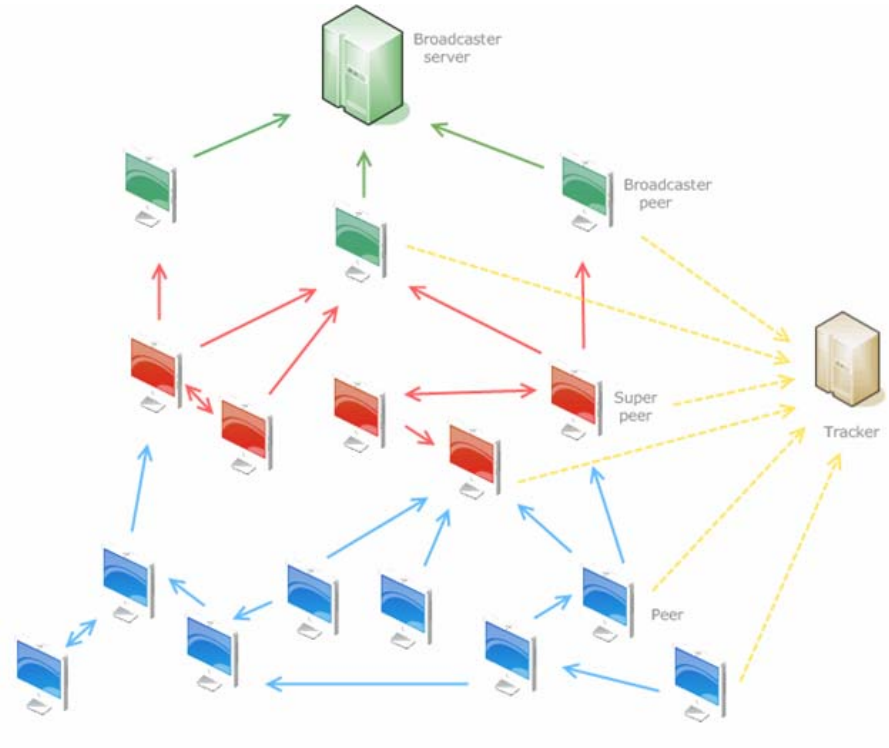

Figura 8: Componentes de GoalBit

contactarlo para poder obtener la pieza. El *tracker* es un servidor que juega un papel fundamental en este mecanismo. Guarda una lista con todos los *peers* (también *broadcasters* y *super-peers*) conectados a cada canal. Los *peers* deben reportarse periódicamente con el tracker, informándole acerca de las piezas que tienen disponibles para intercambiar y el canal al que pertenece. El *tracker*, a su vez, le entrega al peer una lista que contiene un subconjunto de los peers conectados a ese canal, del cual el *peer* luego eligirá los cinco mejores según su criterio para intercambiar piezas. Una vez obtenida esta información del *tracker*, los *peers* se comunican directamente entre ellos (de ahí el nombre *peer to peer*) sin necesidad de la intervención de otros agentes. Por otro lado, el *tracker* es la única forma por la cual un peer puede obtener éste tipo de información. Por estas razones el *tracker* es otro componente fundamental del sistema. Si el mismo cae no es posible el intercambio de piezas, al menos para nuevos *peers* que no dispongan de esa información.

1.3.1. Protocolo de transporte GoalBit (GBTP)

GBTP o GoalBit Transport Protocol, es un protocolo de transporte para el sistema *peer to peer* que funciona sobre otros protocolos de transporte (actualmente solo sobre TCP) [22] y guarda cierta semejanza con BitTorrent [27].

Como se mencionó anteriormente, GoalBit permite distribuir varios flujos de video entre los nodos del sistema, los cuales son llamados *peers* de forma general (más allá de la clasificación mencionada en la Sub-sección anterior). Para su distribución, GoalBit divide el flujo en piezas de igual tamaño llamados *chunks*. Cada *chunk* está identificado por un número de serie que lo posiciona dentro del flujo de video, contiene datos de video y datos útiles para el protocolo GBTP [22].

A los efectos de reproducir el *streaming* el cliente almacena tales *chunks* en un *buffer*, sobre el cual rige una política de ventana deslizante. En determinado momento un cliente solamente podrá descargar *chunks* cuyo identificador esté dentro de cierto rango, el cual está acotado inferiormente por un número de *chunk* mínimo al que se llama *base* y superiormente por la disponibilidad del *buffer* o tamaño establecido de la ventana al cual se llama *length*. El reproductor multimedia del lado del cliente reproducirá uno a uno los *chunks* consecutivos a medida que se descargan, desempaquetan y se introducen en el *buffer* de reproducción. Se le llama *Execution Index*, al ID del próximo *chunk* a reproducir, y se le llama *ABI* (Active Buffer Index) al ID del *chunk* para el cual se han obtenido todos los *chunks* anteriores, es decir, la pieza de video hasta la cual se puede reproducir sin interrupción.

Como fue mencionado anteriormente, el *tracker* mantiene información sobre canales a los cuales se encuentran subscritos varios *peers*. Más aún, mantiene información sobre cada *peer* conectado al sistema. Esta información se puede dividir en información estática, por ejemplo IP del *peer*, puerto en el que recibe conexiones entrantes y el tipo de *peer* (*peer*, *super-peer*, *broadcaster* o *broadcaster-super-peer*), e información dinámica como por ejemplo el *ABI*, fecha y hora del último reporte con el *tracker* y estadísticas sobre la calidad de servicio

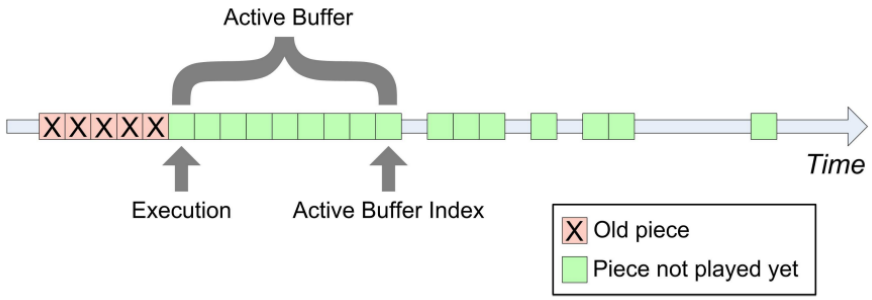

Figura 9: Active Buffer Index

experimentada.

Los *peers* se comunican con el *tracker* a través de un proceso llamado *announce*. Mediante esta especie de saludo inicial o *handshake,* el *tracker* obtiene información sobre el *peer* y la pone a disposición de los demás *peers*, mientras que el mismo *peer* obtiene información necesaria para el intercambio.

Hay dos tipos de *announce*: el *announce* inicial y el periódico. En el *announce* inicial el *peer* solicita la lista de otros *peers* suscritos a su canal (*swarm*) y debe obtener también un ID de *chunk* a partir del cual debe empezar a pedir piezas. Este ID será la base inicial de la ventana deslizante. Adicionalmente el *tracker* asigna a cada *peer* el "tipo de *peer*", y el *swarm* devuelto por el *tracker* dependerá del tipo de *peer* asignado (para un *broadcaster* no existe el *swarm*, para un *super-peer* el *swarm* está formado por el o los *broadcasters* y otros *super-peers*, mientras que los *peers* deben recibir una lista con pocos *super-peer* y muchos *peer*). Existen diversos criterios para elaborar la lista, algunos basados por ejemplo en la proximidad geográfica entre *peers*. A los efectos de que el *tracker* mantenga la información dinámica de cada *peer* debe existir un *announce* periódico en el cual el *peer* informa al *tracker* acerca de su *ABI*. Durante estos *announce* un *peer* puede solicitar o no una nueva lista de *peers*.

Una vez recibido el *swarm* desde el *tracker*, los *peers* deben comunicarse entre sí para poder lograr su objetivo. La comunicación entre *peers* puede dividirse en dos actividades básicas: intercambio de información de control e intercambio de piezas. Existen a estos efectos varios tipos de mensaje dentro del protocolo. Inicialmente un *peer* determinado deberá enviar un mensaje de tipo *HANDSHAKE* a todos los *peers* conectados en su *swarm* con el fin de establecer una conexión con ellos. En este mensaje el *peer* le comunica a los demás su tipo de *peer* y los límites de su ventana deslizante. Esta información es usada por los demás *peers* para aceptar la conexión o no.

Usando la información proporcionada durante el *handshake*, una vez establecida la conexión los *peers* intercambian mensajes de tipo *BITFIELD* en el cual indican mediante un mapa de bits los IDs de los *chunks* que poseen dentro del rango establecido por la ventana del otro *peer*. De esta forma cada *peer* posee

información acerca de quién tiene las piezas que debe obtener. Con el fin de mantener esta información actualizada, cada vez que un *peer* recibe una pieza éste debe enviar un mensaje de tipo *HAVE* indicando el número de la pieza a aquellos *peers* conectados cuyo rango de ventana contiene dicho número. Además, sólo se enviará el mensaje *HAVE* si el peer destinatario del mensaje no tiene la pieza a disposición.

En caso de que un *peer* deslice su ventana debe enviar a los demás un mensaje de tipo *WINDOW_UPDATE* indicando los nuevos límites de la ventana, al que los demás peers deberán contestar con el correspondiente mensaje *BITFIELD*. Los peers deben comunicarse de forma regular, por lo que tras cierto tiempo en el cual no se intercambian mensajes se envían mensajes de tipo *KEEP_ALIVE*.

Para intercambiar piezas, un *peer* interesado en descargar cierta parte del *streaming* desde otro *peer*, le envía a este último un mensaje de tipo *INTERESTED* indicándole la pieza que desea descargar. El *peer* que recibe un mensaje *INTERESTED* debe autorizar la descarga mediante un mensaje de tipo *UNCHOKE*. Una vez aceptada la descarga, el *peer* puede solicitar partes de un *chunk* mediante un mensaje de tipo *REQUEST*, ya que todos los *chunks* se envían fragmentadamente en partes llamadas *slices*. Ante un mensaje de tipo *REQUEST* el *peer* responde enviando un *slice* al otro *peer* mediante un mensaje de tipo *PIECE*. Siempre es posible cancelar estas acciones por lo que existen tipos de mensaje contrapuestos a los anteriores, a saber, *NOT INTERESTED*, *CHOKE* y *CANCEL* (que cancela el pedido de una pieza).

La definición de este protocolo implica algunos aspectos importantes a resolver por quien lo implementa. En particular, es de vital importancia para el rendimiento del sistema la selección de *peers*, los criterios para aceptar conexiones de otros *peers* y también aquellas políticas para determinar qué pieza del *streaming* solicitar en cada momento. Más información acerca del protocolo puede encontrarse en [22] y en [51].

1.3.2. Modos de broadcast

GoalBit dispone de dos modos principales de funcionamiento que lo hacen adecuado a un mayor número de escenarios.

En uno de estos modos, un usuario puede transmitir un contenido por sí mismo, siendo *broadcaster* y *super-peer* al mismo tiempo, necesitando únicamente de un *tracker* para que otros usuarios puedan consumir el *streaming*. Este modo de funcionamiento es conocido como *broadcast yourself*, y permite utilizar gran parte de la funcionalidad de GoalBit sin mayor despliegue de infraestructura.

El otro modo de funcionamiento hace uso de la *GoalBit Suite*, la cual brinda acceso controlado a la infraestructura de GoalBit (en la que se pueden encontrar distintos tipos de servidores y peers) y mediante la cual es posible mantener un monitoreo centralizado de los canales de *streaming*. A este modo se le llama *Professional Broadcast*.

Broadcast yourself Broadcast yourself se refiere a una de las principales funcionalidades de GoalBit, la cual permite a un usuario común crear su propio

flujo de video. El usuario puede elegir la fuente multimedia, la codificación, el método de distribución (con o sin tracker) y si se utilizará o no cifrado. En este modo de broadcast, el broadcaster tiene un rol activo en la distribución del video, por lo cual entra dentro de la clasificación de "broadcaster super-peer".

Professional Broadcast Esta modalidad de broadcast está orientada más que nada a servicios de broadcast especiales, los cuales deben cumplir con ciertos requisitos de calidad de servicio, seguridad, compatibilidad y control, en contraposición con el broadcast yourself, el cual está más orientado a brindar un servicio de broadcast al alcance de todos los usuarios cuya eficiencia depende principalmente de la capacidad del *broadcaster*.

La mayor diferencia entre los dos modos de transmisión es que en el profesional se hace uso de la *GoalBit Suite* que, como ya se ha mencionado, es un conjunto de aplicaciones PHP [19] que proveen los servicios que se enumeraron recientemente. A continuación se hace una breve reseña de sus componentes principales y arquitectura.

1.3.3. GoalBit Suite

El objetivo de la Suite es proporcionar las funcionalidades básicas requeridas por el sistema de *streaming* por lo cual se la considera un *backoffice* que brinda API's mediante las cuales pueden elaborarse distintas capas de presentación, a las que se les llama *Branches*. Un *branch* es simplemente un *front-end* de la Suite y pueden existir varios *branches* para una misma Suite. Mientras que la Suite se encarga de los aspectos relacionados con la transmisión de contenido, la información de usuario y lógica de negocio es responsabilidad del *branch*.

La GoalBit Suite está formada principalmente por cinco sub-sistemas:

- Access
- Controller
- Branch Services
- *Streaming Services*
- User Services

Estos sub-sistemas (a excepción de los *Streaming Services*) se comunican entre sí mediante Servicios Web los cuales son asegurados mediante el uso de certificados para cada sub-sistema y el protocolo SSL o TLS [46]. En la Figura 10 se puede apreciar un diagrama que muestra en alto nivel como se relacionan los distintos componentes de la *Goalbit Suite*.

Access El sub-sistema denominado GoalBit Access, como su nombre lo sugiere, permite realizar el control de acceso a los distintos componentes del sistema. Mediante este componente un administrador general puede asignar permisos y

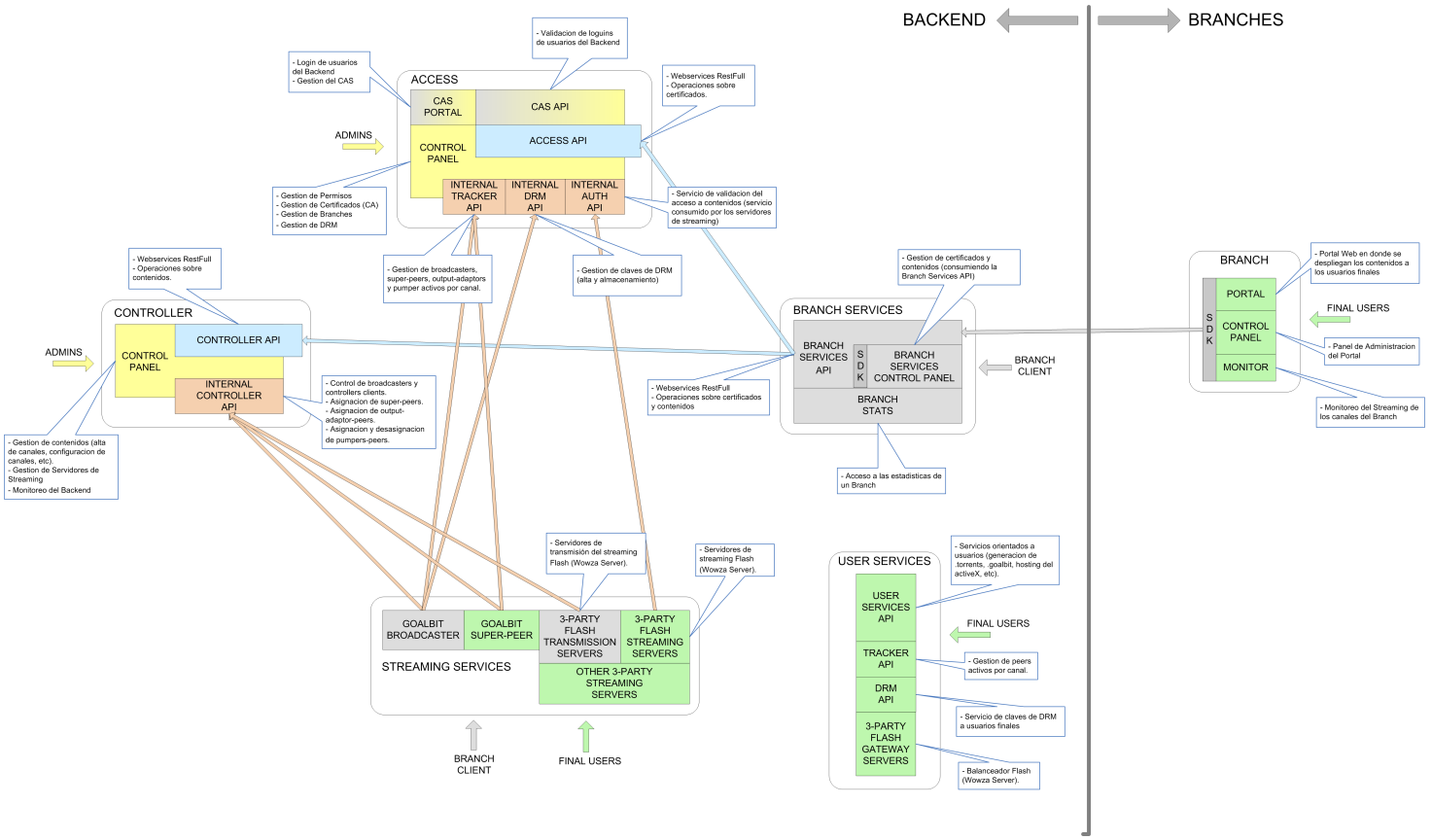

Figura 10: Diagrama GoalBit Suite

privilegios a otros usuarios o administradores, se puede limitar el acceso a ciertos contenidos y se administran las claves generadas por el *broadcaster* en los casos de Professional Broadcast con cifrado a través de un servidor de DRM que pertenece al sub-sistema, así como también permite generar los distintos certificados que se utilizan en la comunicación con la Suite y entre los distintos componentes de la Suite. Además, el *tracker* del sistema reside en este módulo, ya que es el encargado de mantener la información sobre todos los peers que acceden al sistema. La Figura 11 se ilustra al componente *Goalbit Access*.

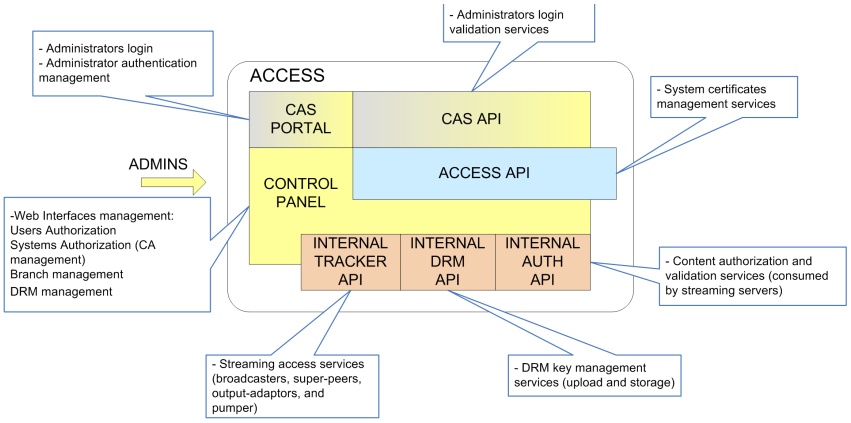

Figura 11: El sub-sistema denominado GoalBit Access permite realizar el control de acceso a los distintos componentes del sistema. Mediante este componente un administrador general puede asignar permisos y privilegios a otros usuarios o administradores.

Controller Este componente de la Suite se encarga de controlar por completo el proceso de transmisión de video, tanto dentro de la Suite como hacia el usuario final. Su funcionalidad va más allá de un *branch* determinado, manteniendo el control de todos los flujos de video que utilizan la Suite. Entre sus tareas principales se encuentra la de asignar *super-peers* a flujos de video automáticamente, asignar *output-adaptors* para proporcionar compatibilidad con otros servidores de streaming, administrar las tasas de transmisión o bitrate de un flujo, asignar más o menos ancho de banda de subida a los *super-peers* ante cambios en el sistema (usualmente cambios en la popularidad de un contenido).

En la nomenclatura de GoalBit suele llamarse "Controller Server" al componente aquí descrito, ya que se le llama "controllers" de forma genérica a los procesos que se anuncian con el Controller Server tomando órdenes del mismo y realizando una tarea determinada. Estos últimos han sido identificados individualmente en este trabajo como *super-peers*, *pumper-peers*, y *output adaptors*. Ver Figura 12.

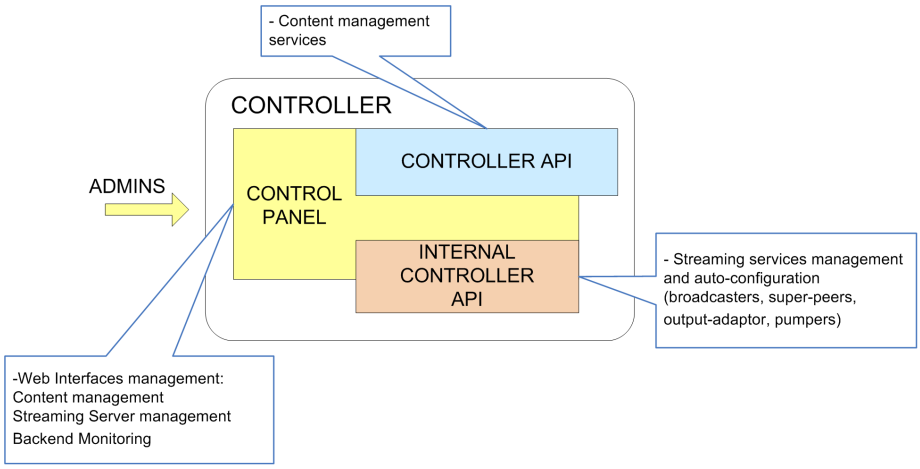

Figura 12: El GoalBit Controller Server se encarga de controlar por completo el proceso de transmisión de video, tanto dentro de la Suite como hacia el usuario final. Su funcionalidad va más allá de un *branch* determinado, manteniendo el control de todos los flujos de video que utilizan la Suite.

Branch Services Los *Branch Services* administran los contenidos organizados por proveedor de contenido o branch. Conocen información tal como si determinado contenido está siendo transmitido, las distintas tasas de transmisión de las señales, los certificados utilizados por el contenido, etc. Además, proporciona una API, a la cual se le llama SDK (Software Development Kit), mediante la cual los *branches* pueden obtener información de este tipo, y se comunica con el resto de la Suite utilizando las API's internas de los distintos componentes. En este sentido representa una interfaz entre el *branch* y el resto de la Suite. Ver Figura 13.

Streaming Services Son aquellos servicios que generan un *streaming* y lo transmiten. Entre estos servicios se engloba al broadcaster (quien produce el *streaming* de video, lo codifica, lo paquetiza y lo transmite), al *super-peer* (quien se encarga de la distribución primaria), *output-adaptors* (quienes pueden enviar un *streaming* de GoalBit a servidores de *streaming* de terceros, transformándolo para cumplir con otros protocolos según el caso), a los mismos servidores de *streaming* de terceros (por ejemplo servidores flash o http). Son, entonces, los servicios con los cuales se comunica el usuario final en los aspectos referentes a enviar o recibir información de video.

User Services Estos servicios acompañan a los *Streaming Services* y son servicios brindados al usuario final que no están directamente relacionados con el *streaming* de video sino con aspectos de funcionamiento del sistema. Por ejemp-

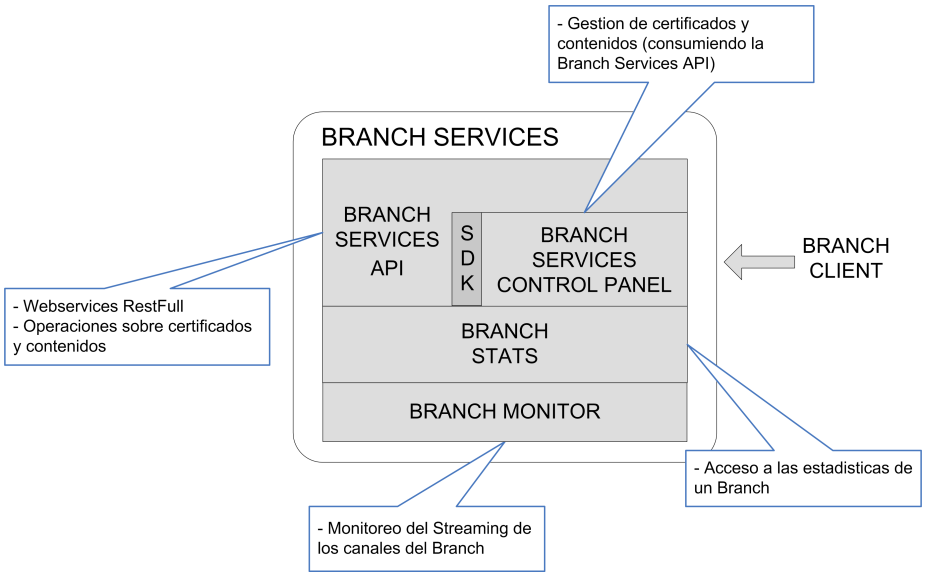

Figura 13: Los GoalBit Branch Services permiten a los *branch* obtener información sobre las distintas señales que administran.

lo, dentro de los *User Services* se encuentra la API del *tracker* mediante la cual el cliente se debe comunicar para anunciarse e interactuar con el mismo, la DRM API encargada de brindar las claves necesarias para decodificar los *streaming* de video que utilizan cifrado y servicios de hosting brindados al usuario final, como por ejemplo almacenamiento de archivos ".goalbit".

2. Seguridad en Internet y en sistemas multimedia

2.1. DRM

DRM (por sus siglas en inglés Digital Rights Management o Manejo de Derechos Digitales en español) es un término genérico que refiere a las tecnologías de control de acceso usadas por editoriales y dueños de derecho de autor para controlar el uso de contenido digital[56]. Este término se utiliza para describir cualquier tecnología que prohíba el uso de contenido digital no autorizado por el proveedor de éste. Los sistemas DRM definen políticas de consumo del contenido, autentican al usuario y validan cada instancia de reproducción. Permite una amplia variedad de servicios para la distribución, compra y promociones de contenido digital de la red en forma segura. Es utilizado típicamente para controlar el uso de contenido multimedia (música, películas, libros digitales, etc), y no debe confundirse con otros mecanismos de control de copia como los que se

usan para controlar las licencias de software en general que no necesariamente modifican el contenido provisto.

DRM es usado más que nada en la industria del entretenimiento por empresas tales como Sony, Amazon, Microsoft, la BBC (British Broadcasting Corporation de Inglaterra) y Apple Inc (utiliza su propio DRM llamado FairPlay). Por ejemplo, muchas de las tiendas de música en linea, como la tienda iTunes de Apple Inc [11], así como también las editoriales de libros digitales (e'books) han implementado DRM.

En los últimos años una cierta cantidad de productoras de televisión han implementado DRM en los dispositivos digitales de los consumidores para controlar el acceso al contenido de sus shows, en respuesta a la creciente popularidad de dispositivos como TiVo[2] que permiten grabar el contenido de la televisión.

Modo de funcionamiento El DRM cifra el contenido multimedia, para que éste sea reproducido solo por los receptores que cuenten con la licencia correspondiente, siendo esta licencia básicamente la clave para descifrar el contenido. Con esto se evita que el contenido digital sea copiado, puesto que el mismo nunca deja de estar cifrado. Si un subscriptor copiara el contenido digital que recibe a otro usuario que no cuenta con la clave para descifrarlo éste último no podría reproducirlo.

Oposiciones al uso de DRM El uso de DRM es controversial. Los que lo apoyan dicen que es necesario para que los dueños de derechos de autor puedan prevenir la copia no autorizada de su trabajo, tanto para mantener la integridad artística como para asegurar flujos continuos de ingresos. Mientras que los que se oponen, como la Free Software Foundation [32] sostienen que el uso de la palabra "rights" (derechos) en DRM es engañoso y que el término Digital Restrictions Management (en español sería Manejo de Restricciones Digitales) es más adecuado. La posición de los que se oponen es básicamente que los dueños de derechos de autor ponen restricciones sobre el contenido digital que están más allá de las existentes leyes de *copyright* (derechos de copia)

De acuerdo con Steve Jobs[3], Apple se opone al uso de DRM con la música despúes de publicar una carta exhortando a sus sellos discográficos a no exigir DRM en la tienda iTunes. El 6 de enero del 2009 la iTunes Store se volvió DRM-free para las canciones, aunque Apple continua utilizando DRM para el contenido de video, tratando a este tipo de contenido multimedia como un tema aparte.

[2] TiVo es una tecnología que permite grabar el contenido de la televisión, pero a diferencia de los clásicos vídeos, lo hace en un disco duro que permite almacenar entre 80 y 300 horas de programación recibida a través del cable, cable digital, transmisión satelital o la tradicional antena [65].

[3] Paul Steven "Steve Jobs" (nacido 24 de febrero 1955) es un magnate de los negocios estadounidenses e inventor. Es co-fundador y director ejecutivo de Apple Inc.

Figura 14: Miembro de la organización Defective by Design (creada por la Free Software Foundation) en una protesta en contra de DRM el 25 de mayo de 2007.

2.2. Firmas digitales

Una firma digital o el esquema de firma digital es un esquema matemático utilizado con el fin de demostrar la autenticidad de un mensaje digital o un documento. Si la firma digital de un mensaje es válida, el destinatario tiene la seguridad de que el mensaje fue creado por un remitente conocido y que no fue alterado durante la transmisión. Las firmas digitales se utilizan comúnmente para la distribución de software, las transacciones financieras, y en otros casos donde es importante para la detección de falsificación o manipulación fraudulenta.

Con una firma digital se pretende tener un equivalente a la firma manual de un documento. Una firma digital debe asegurar:

- Verificar el autor y la fecha y hora de la firma
- Autenticar el contenido cuando fue firmado
- Verificable por terceros
- Incluye la función de autenticación

Algo importante a tener en cuenta es que las copias de un documento digital no son distinguibles del original, una firma digital nos dice que un programa, que tuvo acceso a la información privada del individuo que "firma", realizó esas operaciones.

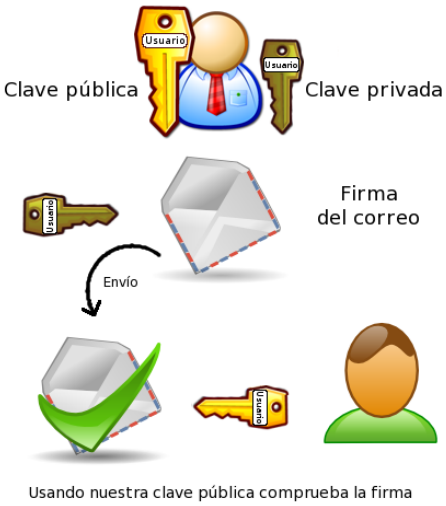

Figura 15: Esquema básico de firma digital. El firmante utiliza su clave privada para firmar el mensaje y el receptor utiliza la clave pública del firmante para verificar la autenticidad de dicho mensaje.

2.3. DSA (Digital Signature Algorithm)

Es un estándar norteamericano para firmas digitales y fue desarrollado por el NIST (National Institute of Standards and Tecnology) bajo el nombre de DSS (Digital Signature Standard) en agosto de 1991, siendo especificado en el FIPS 186[4] [45].

Generación del par de claves La generación de una clave DSA consta de dos etapas. En la primera se eligen parámetros del algoritmo los cuales son públicos y podrán ser compartidos entre los usuarios del sistema. Esta etapa consta de los siguientes pasos:

- En primer lugar, debe elegirse una función de resumen (H) aprobada por el estándar. Originalmente H era siempre SHA-1 pero el estándar se modificó para permitir el uso de su versión mejorada y más segura, SHA-2. Es posible truncar el hash al tamaño de la clave si se considera adecuado.
- Uno de los parámetros más importantes del algoritmo es el largo de la clave ya que da una idea de la seguridad del algoritmo (cuanto más larga la clave es en general más difícil de descifrar). Deben elegirse dos números

[4] FIPS equivale a Federal Information Public Standard y es un documento público que emite el gobierno de los EEUU para definir estándares a utilizar en todos los sistemas informáticos de agencias gubernamentales no militares y contratistas del gobierno.

L y N, donde L es el frecuentemente múltiplo de 64 y N es menor o igual que el largo del hash producido por H.

- Se elige un número primo q de N bits.
- Se elige un módulo primo de L bits p tal que p–1 sea múltiplo de q.
- Se elige g congruente con q módulo p. Por ejemplo, puede elegirse $g = \left|h^{(p-1)/q}\right|_p$ tal que (1 < h < p-1), probando con distintos valores de h si el resultado es 1. La mayoría de los valores de h producen un valor usable de g. Comúnmente se utiliza h=2.

La segunda fase del algoritmo computa las claves pública y privada para un usuario determinado:

- Se elige un número x entre 0 y aleatoriamente.
- $y = \left|g^x\right|_p$ La clave pública es (p, q, g, y).
- La clave privada es x.

La generación de la clave es relativamente rápida ya que existen algoritmos eficientes para calcular las potencias $\left|g^x\right|_p$ y $\left|h^a\right|_p$, por ejemplo la exponenciación por cuadrados[5].

Firmado Sea H la función de hash y m el mensaje:

- Generar un valor aleatorio k por cada mensaje tal que 0 < k < q
- Calcular $r = \left|\left|g^k\right|_p\right|_q$
- Calcular $s = \left|k^{-1}(H(m) + x.r)\right|_q$. Para poder realizar este cálculo H(m) (resultado de aplicarle la función de hash al mensaje) debe ser convertido a un formato numérico. La forma en que debe realizarse dicha conversión está especificada en un apéndice del estándar DSS de NIST.
- Debe recalcularse la firma en el improbable caso de que r = 0 o s = 0
- La firma está formada por los valores r y s

[5] Algoritmo recursivo que se basa en la observación de que $x^n = x.\left(x^{\frac{n-1}{2}}\right)^2$ cuando n es impar y $x^n = \left(x^{\frac{n}{2}}\right)^2$ cuando n es par

Verificación Antes de verificar la firma en un mensaje firmado, p, q, g y la clave pública del firmante (y) deben haber sido comunicados al verificador de forma autenticada.

- La firma se rechaza si no se satisface $0 <$ r $<$q o $0 <$ s $<$ q.
- Calcular $w = \left|s^{-1}\right|_q$
- Calcular $u_1 = |H(m).w|_q$
- Calcular $u_2 = |r.w|_q$
- Calcular $v = \left|\left|g^{u_1}.y^{u_2}\right|_p\right|_q$
- La firma es considerada válida si $v = r$

RSA vs DSA La encriptación utilizando métodos asimétricos es en general muy lenta, especialmente si se trata de grandes cantidades de datos. Es por eso que al utilizar estos métodos resulta pertinente realizar algunas consideraciones de rendimiento al diseñar el sistema.

RSA (Rivest, Shamir y Adleman) y DSA son dos algoritmos criptográficos asimétricos de gran difusión y éxito. Mientras que DSA es un algoritmo de generación de firmas digitales, RSA puede ser utilizado tanto para encriptación como para firmado. Es por esto que es interesante comparar el rendimiento de ambos algoritmos para determinar cual se adecua mejor al sistema que se desea construir.

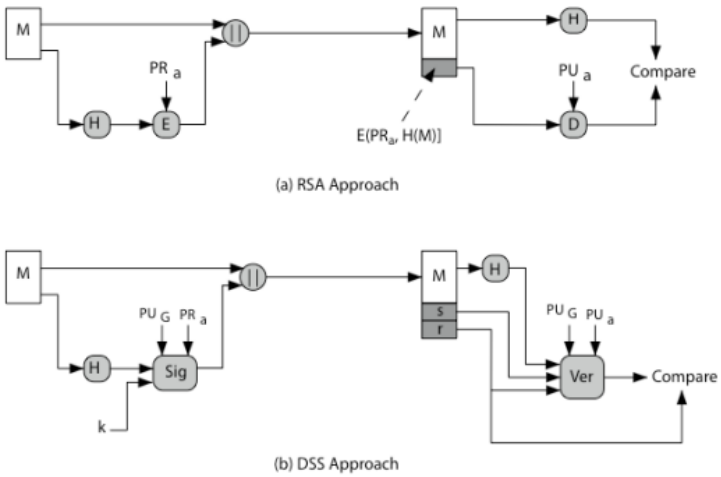

Figura 16: Esquema de funcionamiento de RSA y DSA

Existen muchas comparaciones y estudios. En este caso se tomó parte de un estudio publicado por Microsoft en Octubre del 2002, y que se encuentra

disponible en http://msdn.microsoft.com/en-us/library/ms978415.aspx. En este estudio se comparan los algoritmos DSA y RSA basándose en qué tan rápido generan una firma digital y qué tan rápido la verifican[6].

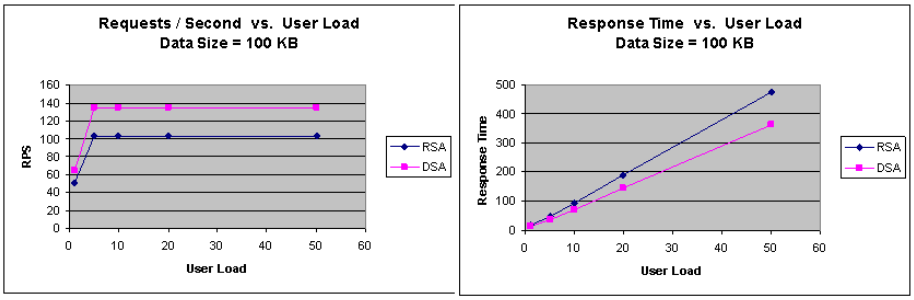

Figura 17: Creación de una firma para 100Kb de datos. Cantidad de solicitudes atendidas por segundo y tiempo de respuesta en función de cantidad de usuarios simultáneos

Creación de la firma En la Figura 17 puede apreciarse que DSA soporta una mayor carga de usuarios y tiene un tiempo de respuesta menor que RSA (aproximadamente un 29 %). Esto no resulta demasiado sorprendente, ya que se debe a que RSA simplemente encripta el resumen del mensaje mientras que DSA, como se ha mostrado anteriormente, crea una firma que no consiste en el resumen encriptado sino en un número derivado del mensaje, la clave privada y otros parámetros del algoritmo.

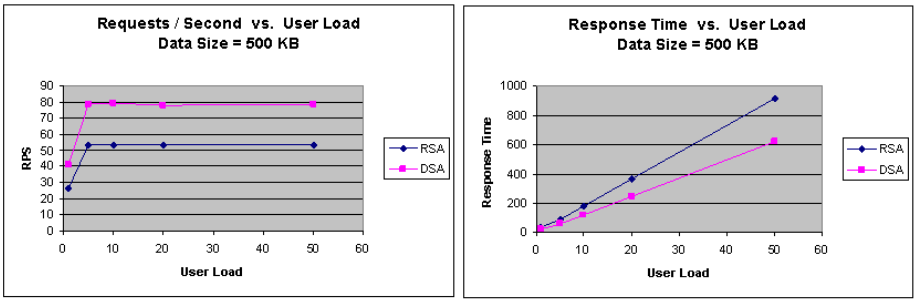

Figura 18: Creación de una firma para 500Kb de datos. Cantidad de solicitudes atendidas por segundo y tiempo de respuesta en función de cantidad de usuarios simultáneos

[6]Se utilizaron RSACryptoServiceProvider y DSACryptoServiceProvider disponibles en System.Security.Cryptography, que son envoltorios (wrappers) de las implementaciones de RSA y DSA, provistas por CryptoAPI.

Para RSA se utilizó una clave de 1024 bits mientras que para DSA se utilizó una de 512 bits.

En la Figura 18 se puede apreciar que con mayores cantidades de datos DSA continúa siendo más rápido.

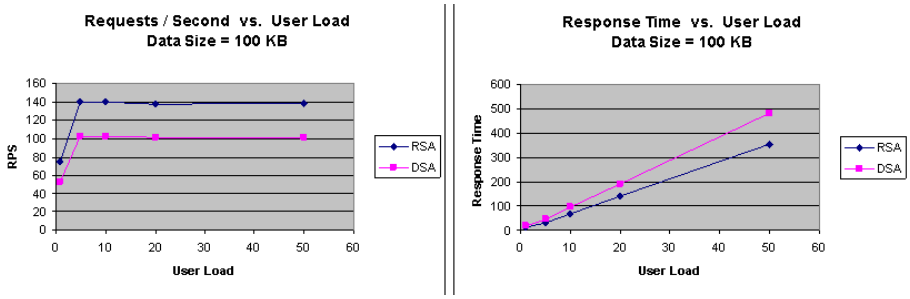

Figura 19: Verificación de una firma para 100Kb de datos.

Verificación de la firma Los resultados muestran que en el caso de la verificación los papeles se invierten, siendo RSA mucho más rápido que DSA (otra vez aproximadamente un 29 %), ver Figura 19. Para verificar una firma digital RSA genera un nuevo resumen del mensaje, desencripta la firma con la clave pública y luego compara la firma desencriptada con el resumen generado, si coinciden la firma es válida. Este proceso autentica al emisor del mensaje dado que la clave pública solo puede desencriptar datos que hayan sido encriptados con la clave privada del emisor. DSA, por otro lado realiza un proceso más complejo para verificar la firma lo cual resulta en mayores tiempos de respuesta. Sin embargo, a medida que aumenta el tamaño de los datos a firmar, la diferencia deja de ser significativa (Ver Figura 20).

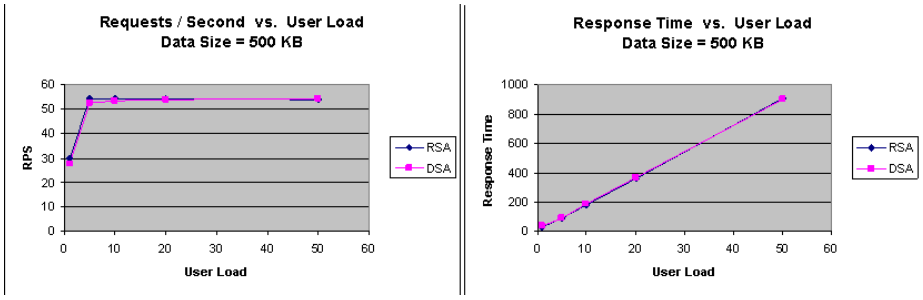

Figura 20: Verificación de una firma para una data de 500Kb..

2.4. Certificados y Autoridades Certificadoras.

Un certificado es un documento electrónico cuyo propósito es asociar a cierta entidad con una clave pública dentro de una Infraestructura de Clave Pública (PKI - Public key infrastructure). Por ejemplo, el uso de certificados permite

confiar en las firmas realizadas con la clave privada asociada a dicha clave pública.

Una Autoridad Certificadora es una entidad encargada de emitir dichos certificados. Debe ser una entidad de confianza tanto para el sujeto del certificado como para quien debe confiar en el certificado. Gracias a la necesidad de esta relación de confianza, existen Autoridades Certificadoras que cobran por emitir certificados en los que confíen automáticamente todos los navegadores de Internet[54].

2.5. SSL (Secure Socket Layer) - TLS (Transport Layer Security)

SSL es un protocolo, desarrollado originalmente por Netscape Communications Corporation, que provee autenticación, integridad y privacidad en comunicaciones cliente-servidor a nivel de capa de transporte. Es independiente de la aplicación y su última versión es SSL v3. Es reemplazado actualmente por TLS, su versión mejorada, que es muy similar a SSL v3 por lo cual se hablará de TLS y SSL de forma indistinta. Actualmente TLS se encuentra en su versión 1.1 definida en el RFC 4346 de Abril 2006. Sin embargo, la versión más implementada es la 1.0 definida en el RFC 2246 de 1999 .

Para garantizar integridad de los mensajes, SSL utiliza un MAC[7] con clave secreta compartida mientras que TLS utiliza HMAC[8].

Para asegurar la confidencialidad de la comunicación se emplea criptografía simétrica, ya que los métodos asimétricos son demasiado costoso computacionalmente y se vuelven lentos conforme aumenta la cantidad de información a cifrar o descifrar. La clave simétrica es definida mediante un protocolo de "handshake" al igual que los algoritmos a utilizar. Dichos algoritmos pueden ser, por ejemplo, AES, IDEA, RC2-40, DES-40, DES, 3DES, Fortezza, RC4-40, RC4-128.

Opcionalmente SSL soporta que el mensaje se comprima antes de encriptarse [29].

Funcionamiento

Handshake Usualmente, un cliente establece una conexión segura con un servidor en la cual este último se autentica. Esto se logra mediante un proceso llamado "handshake" que consta de los siguientes pasos [30]:

- El cliente inicialmente envía al servidor datos como por ejemplo la versión de TLS/SSL que soporta, protocolos soportados de intercambio de claves,

[7] MAC significa Message Authentication Code y es un algoritmo que, dado un mensaje (de largo arbitrario) y una clave (de largo fijo), produce un bloque de bits de largo fijo. Es similar a una función de resumen o hash, exceptuando la clave. Es usado para proteger la integridad de un mensaje en una comunicación.

[8] HMAC significa Hash-based MAC. Es un algoritmo con un objetivo similar al de MAC que utiliza un resumen del mensaje (por ejemplo MD5 o SHA1) en conjunción con una clave simétrica para garantizar la integridad de los mensajes en una comunicación.

métodos de cifrado, algoritmos de hash, y un número aleatorio llamado "nonce".

- El servidor envía luego su certificado, los algoritmos elegidos y un "nonce". En caso de que se requiera autenticación mutua puede exigir al cliente el envío de un certificado.
- El cliente chequea el certificado del servidor, genera una clave secreta ("premaster key"), la encripta con clave pública del servidor y se la envía. También debe enviar un certificado que lo autentique si el servidor lo requiere.
- El servidor desencripta la premaster key y, a partir de ésta y de los nonces intercambiados, genera cuatro claves, a saber, dos claves para cifrado de datos (una para cada sentido de la comunicación) y dos claves MAC (también una para cada sentido de la conexión). Estas claves serán usadas a partir de ahí para cifrar y descifrar los datos mediante el algoritmo simétrico elegido, así como también para garantizar la integridad de los mensajes.

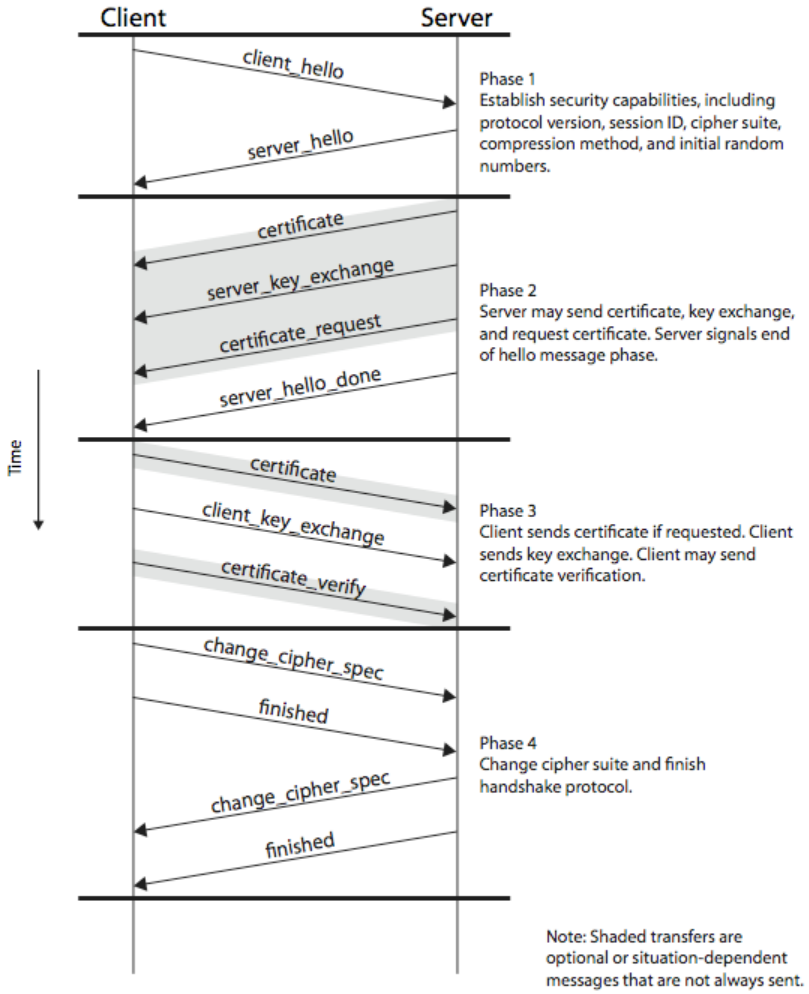

Figura 21: SSL Handshake

Medidas de seguridad El funcionamiento de TLS/SSL se basa en diversas medidas de seguridad [61]:

- Utilización del número de secuencia en el MAC.
- Utilización de HMAC.
- Protección contra varios ataques conocidos (incluyendo ataques man-in-the-middle).
- El mensaje que finaliza el protocolo handshake (Finished) envía un hash de todos los datos intercambiados y vistos por ambas partes.

- La función pseudo aleatoria divide los datos de entrada en 2 mitades y las procesa con algoritmos hash diferentes (MD5 y SHA), después realiza sobre ellos una operación XOR. De esta forma SSL evita utilizar un algoritmo de hash específico para no depender de sus vulnerabilidades.

Aplicaciones SSL trabaja entre la capa de aplicación y de transporte, proveyendo a aplicaciones que utilizan TCP de una forma para establecer conexiones seguras. Por ejemplo, puede trabajar en conjunto con los protocolos SMTP, NNTP o HTTP. Cuando se asocia SSL con éste último se habla frecuentemente del protocolo HTTPS (HTTP Secure).

El objetivo de HTTPS es proteger un intercambio de información HTTP y es frecuentemente utilizado en aplicaciones Web de e-commerce, e-banking, etc. Su funcionamiento se basa en que los navegadores Web traen instalados los certificados de ciertas autoridades certificadoras en quien confían. Durante la etapa de handshake entre el navegador (cliente) y el servidor Web, el primero recibe el certificado del segundo y confía en él siempre y cuando esté firmado por una de las autoridades certificadoras reconocidas por el navegador. En otras palabras, son las empresas que desarrollan los navegadores Web quienes deciden en qué sitios debería confiar el usuario. Usualmente, si un sitio no es confiable para el navegador, el usuario recibe una notificación y se le da la opción de confiar en el sitio o no.

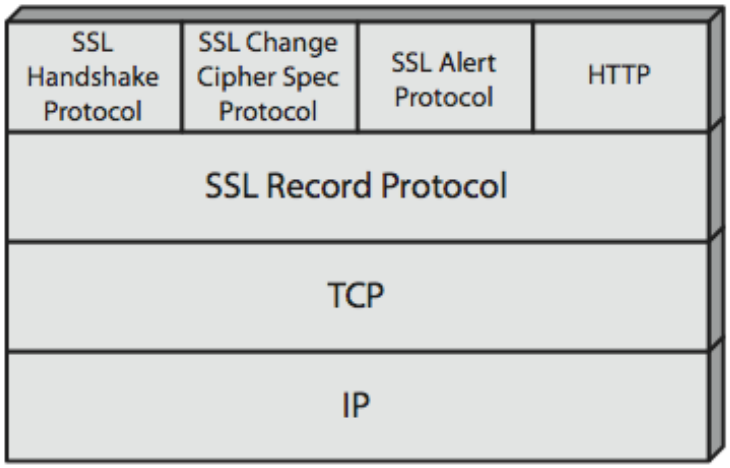

Figura 22: Ubicación del protocolo SSL en el stack de protocolos de una red TCP-IP

SSL también es utilizado en la creación de redes privadas virtuales (VPN), como en el caso de OpenVPN[7].

3. Mozilla Firefox Web Plugins - NPAPI

NPAPI (Netscape Plugin Application Programming Interface) es una arquitectura para soporte de plugins usada por muchos navegadores Web. Comenzó con el navegador Web Netscape en su versión 2.0 y hoy en día es implementada por varios navegadores Web como por ejemplo Mozilla Firefox [16], Opera, Safari y Google Chrome entre otros [59]. Los plugins (o plug-in) son bibliotecas compartidas que los usuarios pueden instalar para manejar tipos de datos "content-types" que el navegador no puede manejar en forma nativa. Por ejemplo los plugins de Adobe Reader [1] le permiten al usuario abrir un archivo PDF directamente dentro del navegador, y los plugins de QuickTime y RealPlayer son usados para ver videos en una página Web. El plugin hace el trabajo en lugar del navegador, pero completamente integrado con la página Web, otra forma que ya se usaba por los navegadores más viejos para manejar tipos de datos "desconocidos" era levantar una aplicación externa. La API requiere que cada plugin implemente y exponga ciertas funciones, aproximadamente 15, para la inicialización, creación, destrucción etc. Para poder ver por ejemplo en el navegador Mozilla Firefox la lista de plugins que se tienen instalados así como los tipos de datos aceptados se puede lograr digitando about:plugins en la barra de direcciones.

3.1. Ciclo de vida de un plugin

El ciclo de vida de un plugin es completamente controlado por la página Web que lo invocó. Esta Sección da una visión general acerca de la forma en que estos plugins operan en los navegadores basados en Gecko [43] como Mozilla Firefox. Cuando el navegador se levanta, busca los módulos de plugins en un lugar en particular del sistema y estos declaran que tipos de datos "content types" (ej: "application/pdf") manejan. Cuando un usuario abre una página Web conteniendo las etiquetas HTML *EMBED* u *OBJECT* de un tipo de datos que invoca un plugin, el navegador responde con la siguiente secuencia de acciones:

1. Busca a un plugin que soporte el *MIME type* o *content-type* asociado.
2. Carga el código del plugin en memoria.
3. Inicializa el plugin, se invoca la función NP_Inicialize.
4. Crea una nueva instancia del plugin, se invoca la función NPP_New.

Ahora el plugin es responsable de procesar los datos como corresponda, sean estos una imagen, audio, video o de otro tipo. El navegador puede cargar múltiples instancias de un mismo plugin en la misma página Web. Si una página Web tiene varias etiquetas HTML *EMBED* de un determinado *MIME type*, este creará tantas instancias del plugin como sean necesarias. Cuando el usuario abandona la página o cierra la ventana (en caso de que este se haya abierto en en una nueva) la instancia es borrada, invocando la función NPP_Destroy. Cuando la última instancia del plugin es borrada (Se invoca la función NP_Shutdown),

el código del plugin se saca de la memoria, los plugins no consumen más que espacio en disco cuando no están cargados. La comunicación entre el plugin y el navegador es bidireccional. Para tener una completa referencia de las funciones que implementa el plugin y son invocadas por el navegador, o las que expone el navegador y son accesibles desde el plugin ver [28].

3.2. NPRuntime

NPRuntime es una extensión que se le agregó a la API NPAPI para agregar soporte de scripting, permitiendo usar Javascript para interactuar con el plugin. Esta fue desarrollada por un grupo de compañías, incluyendo Mozilla Fundation [12], Adobe [1], Apple [11], Opera [13] y la ex Sun Microsystems [64]. A continuación se explica como hacer que un plugin use esta extensión para que sea accesible desde los scripts javascript y como acceder a sus objetos. El navegador ahora chequea si el plugin soporta esta nueva extensión a la API y si expone objetos accesibles desde un script. Esto lo hace invocándole al plugin la función de la API NPP_GetValue, la misma tiene la siguiente firma:

```
NPError NPP_GetValue(void *instance, NPPVariable variable, void *value);
```

instance corresponde a la instancia del plugin.

variable es un enumerado y especifica el dato que se desea, por ejemplo para obtener el nombre del plugin se pasa el valor NPPVpluginNameString y para saber si el plugin soporta la extensión se le pasa por parámetro NPPVpluginScriptableNPObject.

value es el valor de retorno, en el caso anterior, para que el plugin sea accesible vía un script debe ser un apropiado objeto del tipo NPObject, el que expone los objetos accesibles desde los scripts, así como los métodos y propiedades.

Finalmente se retorna NPERR_NO_ERROR en caso de que la operación sea exitosa.

3.3. Consideraciones de seguridad

Agregar un plugin NPAPI es peligroso porque estos tienen acceso sin restricciones a la maquina local, pudiendo acceder al sistema de archivos y uso de la red; es por esto que el usuario debe ser consciente de los riesgos e instalar estos plugins solo si sabe que los mismos son confiables.

4. El Plan Ceibal

El Proyecto de Conectividad Educativa de Informática Básica para el Aprendizaje en Línea (CEIBAL) [8] se trata de un plan educativo integrado, implementado en Uruguay, que tiene como valores fundamentales la democratización

del conocimiento y la equidad. Dentro de este plan se engloban varias medidas gubernamentales, siendo unas de las más importantes la entrega de un PC portátil de bajo costo (conocido como XO) a cada usuario de la enseñanza primaria pública, y la mejora en infraestructura que permita mayor conectividad, sobre todo en el interior del país. En este sentido el plan se encuentra enmarcado dentro del proyecto OLPC presentado en 2006 por su fundador Nicholas Negroponte [6].

Por otro lado, en el entorno del Plan Ceibal se ha generado una importante contribución desde el ámbito académico y también contribución independiente motivada por la importancia desde el punto de vista social que se le atribuye al proyecto. Este apoyo al plan se manifiesta más que nada en la capacitación de maestros, asesoramiento técnico y desarrollo de software para las XO. Dentro de esa línea se encuentra este proyecto.

Figura 23: Logo del Plan Ceibal

4.1. Hardware de las XO

La mayoría de las XO distribuidas en el Uruguay, tienen una velocidad de reloj de la CPU de 433 MHz, memoria DRAM: 256 MB Data rate Dual – DDR333 – 166 MHz y almacenamiento central de 1024 Mb SLC NAND flash [39] lo cual se encuentra muy por debajo de una PC de escritorio promedio. Se espera vayan mejorando con el tiempo, de igual forma este proyecto supone la capacidad antes mencionada .

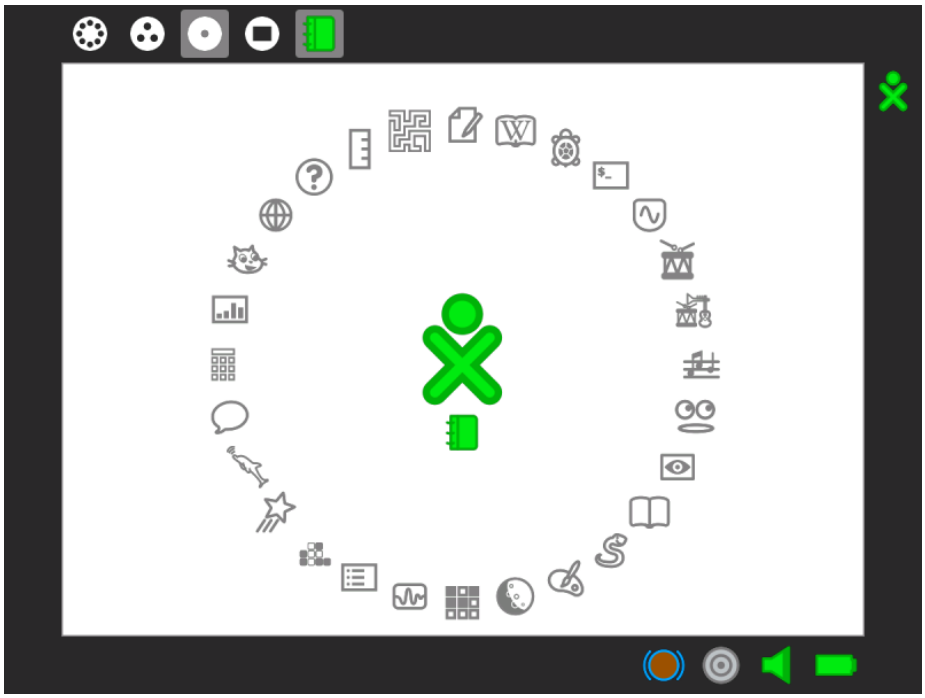

Figura 25: Entorno Sugar

Figura 24: Ilustración de una XO

4.2. Software

4.2.1. Sugar

Sugar [44] es un entorno gráfico diseñado especialmente para el proyecto OLPC y es la interfaz gráfica principal del Plan Ceibal, aunque recientemente se han liberado imágenes del sistema operativo que cuentan también con el manejador de escritorio Gnome [36]. El objetivo de Sugar es brindar una forma de interacción divertida y fácil de usar incentivando la creación de un ambiente de aprendizaje en grupo a través de la interacción entre distintas XO.

Para lograr los objetivos mencionados, la interfaz plantea un cambio de paradigma respecto a los entornos gráficos habituales de las PC de escritorio. En este sentido presenta algunas características que lo distiguen del resto de los entornos gráficos tradicionales :

- En Sugar no se habla de aplicaciones sino de reunir a los niños entorno a "actividades". El concepto de actividad en Sugar implica más que un simple cambio de nombre y hace referencia al enfoque hacia la colaboración entre las mismas, al objetivo educativo, la interacción con el usuario y la utilización de un "diario" que guarda el estado de la actividad cuando esta se suspende [48].

- Para facilitar la creación de un ambiente de aprendizaje colaborativo, las XO emplean una red de tipo malla (mesh) que interconecta todas las laptops que se encuentren lo suficientemente cerca. Sugar explota esta conectividad permitiendo que las actividades tengan la posibilidad de trabajar en red. Se aspira a que todas las actividades sean diseñadas de forma tal que aprovechen de alguna manera la red malla. Podría considerarse, por ejemplo, el caso del navegador Web distribuido con las XO. Usualmente, en un navegador común, un usuario navega por la red en solitario, y a lo sumo recomienda algún link a un amigo mandando un e-mail o a través de una red social. En la XO existe una funcionalidad que permite compartir links con otras laptops conectadas a la red malla, la cual está integrada al navegador, transformando la actividad de navegar por la Web en una actividad colaborativa. Se incentiva a que, en lo posible, todas las actividades aprovechen la red y hagan énfasis en facilitar estos procesos colaborativos [48].

- Sugar incentiva la creación de contenidos así como el desarrollo y crítica colaborativa de los mismos. De esta forma la mayor parte de las actividades se concentran en la creación de algún tipo de objeto, por ejemplo un dibujo, una historia, una pieza musical, etc. Es por esto que, de forma similar a como ocurre con las actividades y las aplicaciones, se habla de objetos en lugar de archivos, es decir, en lugar de un archivo de sonido hablamos de una canción, en lugar de un archivo de texto se habla de una historia. La *objetización* del sistema de archivos permite además una relación más directa de los contenidos con los objetos del mundo real [48].

- Los objetos y actividades son organizados en un diario. Esta es la visión del sistema de archivos que tiene un usuario de Sugar aunque el sistema de archivos subyacente sea muy similar a uno convencional. Un diario típicamente registra las actividades realizadas durante el día y es por esto que se ha adoptado dicha metáfora al desarrollar Sugar. Un usuario puede explorar el diario para continuar una actividad que comenzó en el pasado o comenzar una nueva actividad que se escribirá en el diario. Mediante el diario un niño tiene un registro cronológico de las actividades que ha iniciado así como de las cuales ha participado, es decir, no solo se registran las interacciones con su computadora sino también aquellas con sus pares [48].

4.2.2. Fedora

Linux Fedora es una de las varias distribuciones del sistema operativo Linux, y es el sistema operativo subyacente de la XO. Es desarrollado por el Proyecto Fedora (Fedora Project) bajo el patrocinio de Red Hat. Este proyecto es llevado a cabo por una gran comunidad de desarrolladores en todo el mundo y su objetivo es crear y difundir un sistema operativo que refleje el estado del arte en software libre, utilizando solamente software libre.

El Proyecto Fedora no solo permite sino que motiva a usuarios, empresas y comunidades a tener su propia visión del sistema operativo. Cualquiera puede crear un nuevo producto a partir de Fedora y llamarlo con otro nombre. Así, OLPC ha tomado versiones de Fedora y las ha adaptado a sus necesidades, convirtiéndolo en el sistema operativo de las XO. Sobre este sistema operativo fue construido Sugar, por lo cual es una referencia para cualquier otra distribución de Linux que intente correr en el hardware de las XO o presentar la interfaz gráfica Sugar en un PC convencional [6, 31].

4.2.3. Actividades

Como se ha dicho anteriormente, en Sugar no se habla de aplicación sino de "actividad". Las actividades son esencialmente paquetes de software que guardan su estado una vez que se cierran, creando varias instancias específicas de las mismas, de modo que el usuario pueda luego retomar la actividad mediante alguna de esas instancias. Una instancia de una actividad es típicamente una entrada en el "diario" mantenido por Sugar. Muchas actividades soportan además un modo de funcionamiento colaborativo, en el cual un niño puede invitar a otros a participar de la actividad.

5. Trabajo relacionado

5.1. VLC

VLC (VideoLAN Client) es un reproductor multimedia creado en 1996 por el proyecto VideoLAN. Es compatible con una amplia variedad de formatos de

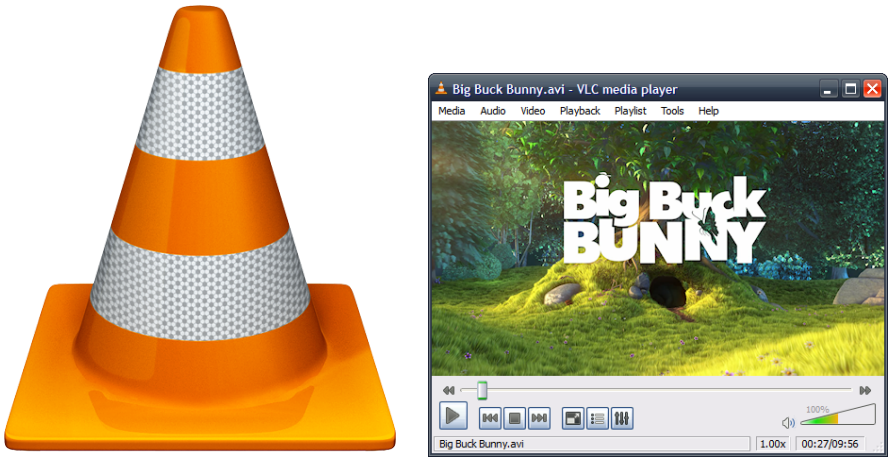

Figura 26: Icono de VLC e interfaz clásica

audio y video por lo que ha cobrado cierta popularidad principalmente en el ámbito del Software Libre.

Tiene la particularidad de ser multiplataforma, encontrándose disponible en versiones para Linux, Microsoft Windows, Mac OS X, BeOS, BSD, PocketPC y Solaris [21].

Su nombre refiere a los orígenes del proyecto, en el cual el reproductor fue concebido como dos aplicaciones separadas, un cliente y un servidor. Actualmente cliente y servidor se encuentran integrados en una misma aplicación, por lo cual el reproductor de VLC permite tanto la reproducción como la generación de un *streaming* de video.

VLC se distribuye bajo la licencia GPL por lo que su código es abierto. Además es fácilmente extensible y personalizable gracias a las características de su arquitectura. Esto lo vuelve una buena opción como punto de partida para el desarrollo de un reproductor multimedia de código abierto con características particulares como es el caso de GoalBit Media Player.

5.1.1. Arquitectura de módulos

Desde un punto de vista superficial podría decirse que VLC presenta una arquitectura modular con tres componentes principales: el "*core*" de VLC, los módulos y las bibliotecas externas.

El *core* de VLC se encarga de implementar una especie de *framework* orientado a objetos (aunque la mayor parte del código está escrito en lenguaje C convencional) en el cual se definen y manipulan los llamados *VLC_Objects* y se administran distintas estructuras que permiten manejar módulos de forma abstracta. Por ejemplo, todo módulo puede ser visto como un *VLC_Object* pero a su vez un *struct* de tipo *intf_t* representa una interfaz. De esta forma el *core* de VLC tiene estructuras especiales para distintos tipos de objeto en su modelo

de dominio, e implementa una relación de generalización-especialización entre *VLC_Object* y las otras estructuras. El *core* de VLC es llamado *libVLCcore* y es posible utilizarlo desde aplicaciones externas por medio de la biblioteca *libvlc* quien expone una API con las funcionalidades de *libVLCcore*.

El manejo de los módulos de VLC es realizado por *libVLCcore* mediante la utilización de módulos genéricos que luego podrán ser especializados por módulos concretos. Por ejemplo, VLC cuenta con módulos que proveen funcionalidades abstractas como *muxing/demuxing*, *video ouput, audio ouput*, *ssl/tls*, etc. Luego estas funcionalidades deben ser implementadas por módulos concretos.

Los módulos son el componente que le brinda la funcionalidad a VLC. Típicamente VLC carga entre 200 y 400 módulos al iniciarse, dependiendo de las opciones con las que se haya compilado. Estos módulos están clasificados según sus *capabilities* (capacidades) y son cargados dinámicamente por el *core*. Éste último provee una interfaz que los módulos deben implementar para comunicarse correctamente con él y entre sí. El *core* también provee diversos mecanismos de comunicación entre los módulos, por ejemplo, un robusto sistema de variables globales y manejo de eventos.

Por último VLC hace uso de un gran número de bibliotecas externas (debido a la gran diversidad de módulos que presenta) por lo cual debe cumplirse con un gran número de dependencias a la hora de compilarlo e instalarlo[37].

5.2. Veetle

Veetle es una compañía de multimedia digital establecida en Silicon Valley. Fue creada por graduados de la universidad de Stanford (EE. UU.) y provee una plataforma de transmisión de video en vivo de alta calidad que se caracteriza por su eficiencia y escalabilidad[50].

En el portal Web http://www.veetle.com/ se encuentra disponible la funcionalidad que permite utilizar la plataforma mediante un Web Plugin. Este plugin es una versión modificada del reproductor multimedia VLC y es necesario descargarla desde la página de Veetle para luego instalarla en el equipo siguiendo el instructivo. Dicho portal Web permite la transmisión de diversos tipos de contenido multimedia, la visualización de canales en vivo y la creación de foros de discusión.

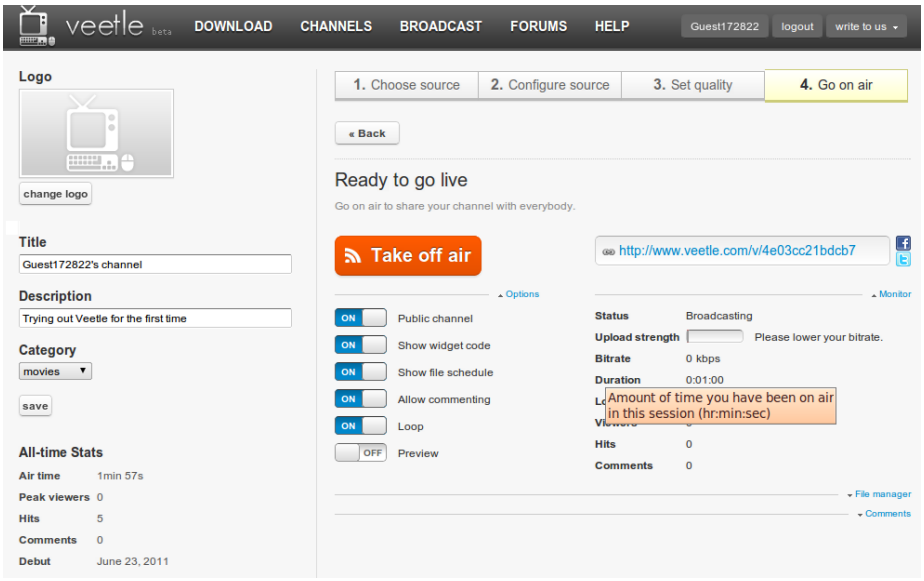

Figura 27: Portal de Veetle

5.3. JAMedia antecedente en las XO

JAMedia es un proyecto impulsado por la organización CeibalJAM que surge en respuesta a la necesidad de contar con un reproductor multimedia en la XO. Últimamente ha ganado renombre gracias su capacidad de reproducir canales de video en vivo aunque cuenta con la funcionalidad de cliente de *streaming* HTTP/FTP, RTP/RTSP, MMS/MMST, MPST y SDP[42].

La actividad JAMedia.activity está basada en el reproductor multimedia mplayer [42] precompilado y utilizado como backend. A este reproductor multimedia se le agrega una interfaz gráfica PyGTK[35] construida para el entorno gráfico Sugar que controla el reproductor a través del módulo *subproceso* de python. Su tamaño es tan solo de 8Mb.

Actualmente la actividad se encuentra en la versión 4. Esta versión cuenta con las siguientes funcionalidades:

- Reproducción de audio y video almacenado.
- Reproducción de contenido on-line tal como canales de video en vivo, videos almacenados remotamente y radios on-line.

La actividad no cuenta en la actualidad con la posibilidad de realizar *streaming* de video.

Figura 28: Logo de la actividad JAMedia y el reproductor multimedia MPlayer

Parte III
Solución propuesta

Adaptar un producto como GoalBit a la realidad del Plan Ceibal es una propuesta compleja y ambiciosa dado que dentro de dicha consigna se engloban muchas cosas que implican sobreponer diversas dificultades inherentes a dicha realidad. En este proyecto se trabajó sobre algunos aspectos incluidos dentro de esta consigna, generando una solución inicial que proporciona la funcionalidad básica requerida por el Plan Ceibal, atendiendo los requerimientos más importantes impuestos por dicho cliente y marcando varias líneas sobre las cuales se puede extender el trabajo y mejorar el producto, como fue acordado desde un principio en la definición del alcance. La solución, entonces, se compone de diversos artefactos, los cuales resuelven distintos aspectos del problema general. Algunos de estos artefactos ya formaban parte de la plataforma GoalBit y fueron modificados para satisfacer ciertos requerimientos mientras que otros son nuevos. Se ha agregado funcionalidad a GoalBit y también se han limitado algunas de sus funcionalidades a fin de adecuarlo a la plataforma objetivo. También se han realizado diversas pruebas y se han discutido aspectos relacionados con el objetivo y la utilidad del producto, a fin de determinar un balance entre el grado de funcionalidad deseada y la funcionalidad permitida por el limitante hardware objetivo.

En este caso, de todos los componentes que forman la solución, quizá uno de los más importantes es la interfaz. La interfaz no es solo el conjunto de controles y mensajes que permiten al usuario comunicarse con el sistema, ni tampoco el diseño de la interfaz se reduce a un problema estético. Para el usuario, la interfaz es el sistema ya que todo lo que se puede hacer con el sistema y todo lo que el sistema hace se manifiesta a través de la interfaz. Es uno de los componentes (sino el componente) que determina la calidad de la experiencia del usuario. En esto último, las consideraciones acerca de la usabilidad del sistema tienen un rol fundamental, más aún cuando el usuario objetivo pertenece a la franja etaria a la que pertenece en este caso. Muchas veces incluso, aunque no en este caso, el trabajo de adaptar determinado software para su uso en el Plan Ceibal se reduce a proporcionar una interfaz adecuada para dicho software.

En este caso la interfaz de usuario está compuesta por un Web Plugin para Firefox que es compatible con la actividad Browse de las XO del Plan Ceibal, un portal Web en donde los usuarios podrán acceder a las funcionalidades del sistema y un instalador que permite instalar la aplicación de forma sencilla y como el usuario está acostumbrado. La solución es presentada al usuario como una actividad del entorno Sugar. Esta actividad es una versión de su navegador Web modificado. Una vez instalada la actividad, puede accederse a una versión del portal Goalbit-Ceibal instalado en Antel, a través de la URL http://ceibal-multimedia.antel.net.uy.

El plugin de Firefox está basado en el plugin de VLC para éste navegador, el cual ya era distribuido junto con GoalBit. En este proyecto se agregó al plugin

la funcionalidad de broadcast utilizando GoalBit, ya que el plugin funcionaba únicamente como un reproductor multimedia. Esto al mismo tiempo le proporciona a GoalBit la posibilidad de realizar broadcast mediante una interfaz Web, funcionalidad con la cual no contaba. Esta fue una de las motivaciones para elegir una interfaz enteramente Web. Se está marcando cada vez más un cambio de paradigma en cuanto a cierto tipo de aplicaciones, en el cual se tiende a abandonar la interfaz de escritorio y proporcionar una interfaz Web para acceder al servicio. Las aplicaciones multimedia y sobre todo las que trabajan con video *streaming* son un claro ejemplo. Basta ver como se han popularizado portales del estilo de Youtube [10] o Veetle [14], los cuales brindan acceso a contenido, proporcionan un reproductor multimedia y en algunos casos, por ejemplo el de Veetle, permiten incluso realizar broadcast de un contenido. Este tipo de portales han resultado ser interfaces idóneas para los mencionados sistemas en la medida de que vuelven más accesible el producto y proporcionan una comunicación natural con los servidores que publican o administran el contenido.

En lo que se refiere a usabilidad, el portal fue diseñado siguiendo las pautas de usabilidad de portales similares y teniendo en cuenta que el mismo debe ser simple e intuitivo a los efectos de ser utilizado por niños satisfactoriamente. Se hizo una evaluación del portal de Veetle y se diseñaron prototipos. El diseño final del portal fue conversado y validado con el cliente del Plan Ceibal y difiere en algunos aspectos con el prototipo primario, que se realizó debido a razones que se explicarán en detalle más adelante.

El portal está integrado con la Suite de GoalBit y provee un panel de control para monitorear los *streamings* realizados a través de él. Mediante la Suite es posible mantener un control centralizado sobre los flujos de video que se realizan a través del portal, por ejemplo se pueden dar de alta canales, borrarlos, saber si un canal está activo, si es público o es privado, en cuyo caso se controla el acceso seguro al mismo, etc. También proporciona acceso a la infraestructura de GoalBit gestionando, por ejemplo, la asignación de super-peers, pumper-peers y output-adaptors, regulando el bitrate del *broadcaster* o realizando *transcoding*[9] (transcodificar) y *transrating* (conversión de un *bitrate* en otro) cuando es adecuado.

Otro de los aspectos más significativos de la solución tiene que ver con la seguridad. Dados los objetivos del Plan Ceibal, no es aceptable una solución en la que no exista garantía de que la información recibida por un canal fue emitida realmente por el broadcaster de ese canal.

El diseño original de GoalBit permite que cualquier peer puede inyectar tráfico malicioso en un canal y, si se satisfacen ciertas condiciones, ese tráfico es reproducido por el reproductor. GoalBit ofrece la posibilidad de encriptar el contenido de un canal como forma de evitar el consumo no autorizado del canal pero no se soluciona el problema de la inyección de chunks en un canal dado ya que un peer no recibe los chunks de video directamente del broadcaster sino de otros peer de diversos orígenes. La solución propuesta para resolver este problema fue implementar una firma digital que acompañe al chunk y que

[9] El proceso de transcodificación es la conversión de un *encoding* en otro

identifique los chunks de determinado canal.

A continuación se describen en detalle los elementos de la solución mencionados anteriormente y se hace referencia al proceso mediante el cual se llegó a tales resultados. Parte importante de este proceso no se traduce directamente en un entregable pero juega un papel más que significativo en el resultado final. Existió una etapa inicial de aprendizaje, en la que se estudiaron varios aspectos tecnológicos referentes al proyecto. En esta etapa se realizó una mitigación de riesgos técnicos que implicaron extensivas pruebas y prototipos, algunos de los cuales se continuaron en posteriores iteraciones y otros que se descartaron. Definir el ambiente de desarrollo no fue una tarea despreciable en el proyecto y merece mención en las secciones subsiguientes. La dificultad para definir el ambiente de desarrollo radica en la complejidad y heterogeneidad tanto de los componentes de la solución, como de los ambientes de producción. Estos aspectos son ampliados en la Sección siguiente.

6. Definición del ambiente de desarrollo

En la definición del ambiente de desarrollo se establecen los medios y las herramientas por los cuales se desarrollará determinado software. Es importante que el mismo asegure que cierto código fuente que compila y funciona bien en el ambiente de desarrollo también lo haga en la plataforma objetivo, por lo cual no es una tarea trivial, en especial cuando el ambiente de producción implica el uso de un hardware específico con serias restricciones en cuanto a capacidad de almacenamiento y cómputo.

GoalBit es un software multiplataforma que es compilado nativamente en Linux. Existen manuales acerca de cómo compilar e instalar GoalBit en algunos sistemas Linux como por ejemplo Debian, Fedora, CentOs, Gentoo, e incluso está documentado como compilarlo para Microsoft Windows [9] desde un ambiente Linux (*cross-compile*). Sin embargo requiere cierto esfuerzo adaptar el software a los distintos escenarios y documentar el proceso de compilación e instalación incluso tratándose de sistemas Linux muy similares entre sí. Algunas de las razones por las cuales surgen dificultades tienen que ver con que es necesario instalar un número significativo de dependencias, que en general son bibliotecas desarrolladas por terceros, y que no se encuentran o se encuentran bajo distintos nombres en los repositorios principales de las distintas distribuciones de Linux. Por estas razones, para ciertas distribuciones de Linux es necesario descargar algunas bibliotecas desde la página oficial, compilarlas e instalarlas manualmente.

Como se ha mencionado antes, GoalBit se compone de tres elementos principales: el *GoalBit Media Player* (GMP) el *GoalBit Media Server* (GMS) y la *GoalBit Suite*. También tiene un Web Plugin para Firefox, el cual fue extendido para soportar las funcionalidades deseadas. Los elementos a ser implantados en la XO, por lo tanto aquellos que están condicionados por sus características tanto de hardware como de software, son principalmente el GMP y el Web Plugin.

Dado que GoalBit interactúa con bibliotecas dinámicas del sistema operativo, es deseable que en lo referente al desarrollo del GMP y el plugin, el ambiente de compilación se asemeje lo más posible al ambiente de producción, el cual se trata de un Linux, basado en Fedora Core 9 y 11, modificado para cumplir con las restricciones de hardware que impone la XO y de seguridad establecidos por el Plan Ceibal [8]. De esta forma se evita que existan problemas por ejemplo al *linkeditar* con una versión de una biblioteca que es distinta a la disponible en el SO de producción, lo cual puede generar efectos inesperados. A los efectos de lograr lo anterior existen varias alternativas. A continuación se enumeran aquellas que fueron consideradas en este proyecto:

- Utilizar la misma XO para el desarrollo y la compilación del software.
- Emular el comportamiento de una XO mediante un emulador o máquina virtual.
- Emular en un sistema Linux ordinario tan solo el sistema de archivos de la XO de forma de utilizar las mismas bibliotecas a la hora de compilar.

Como se mencionó en la Sección 4.1 la XO cuenta con un almacenamiento secundario con una capacidad de 1 Gb. Instalando las herramientas de desarrollo y las bibliotecas necesarias durante la compilación se excede dicha capacidad lo cual deja como única alternativa utilizar un medio extraíble para el almacenamiento. Además, dado que el proceso de compilación es costoso computacionalmente y compilar el *GoalBit Media Player* en un PC Intel(R) Core(TM) 2 Quad 2.33GHz y 2GB RAM consume aproximadamente quince minutos, utilizar la propia XO para el desarrollo y la compilación implicaría tiempos de compilación del orden de varias horas. Por estos motivos no es una buena opción montar un ambiente de desarrollo en la XO.

Una máquina virtual que sea capaz de correr una imagen del sistema operativo de la XO, en cambio, contaría con una capacidad de cómputo mucho mayor y podría llegar a ser un ambiente adecuado para la compilación del software.

Existen dos tipos de imágenes disponibles en la Web. Unas con sistema de archivos ext3 [63] y otras con sistema de archivos jffs2 [15, 69]. El sistema de archivos jffs2 es un sistema de archivos diseñado para trabajar con memorias NAND flash y es utilizado comúnmente en sistemas embebidos (como es el caso de las XO). Los sistemas de archivos como jffs2 no funcionan en una máquina virtual que utilice un dispositivo de bloques como almacenamiento secundario. Una alternativa posible en este caso es trabajar con un *LiveCD* que genere una RAMDrive la cual cuenta con unos pocos Mb disponibles y no es configurable. Por otra parte, las imágenes disponibles para uso público que cuentan con un sistema de archivos ext3 tienen bibliotecas cuya versión difiere (por lo general es más antigua) con las actuales del Plan Ceibal. También cuentan con distintas versiones del sistema operativo, el cual es basado en Fedora 7 en lugar de Fedora 9 y 11.

La opción que se encontró más adecuada, basándose en experiencias similares encontradas en la Web , fue la de copiar íntegramente el directorio "/" de la XO a

un PC de escritorio común para luego compilar en una "jaula chroot" utilizando el comando chroot de Linux . Este comando permite cambiar la ubicación del directorio raíz de Linux para cierto proceso (por ejemplo la consola) y sus hijos. Los procesos que corren bajo el chroot no pueden acceder a archivos fuera del nuevo directorio raíz. Mediante este comando es posible compilar el código fuente de GoalBit utilizando las mismas bibliotecas y objetos compartidos que la XO, pero aprovechando la capacidad de cómputo de un PC convencional.

Para el testing se dispone de dos XO proporcionadas por Ceibal. El chroot no es apropiado para esta tarea, debido a que no emula la arquitectura, el hardware de la XO ni tampoco el ambiente Sugar.

6.1. Compilación e instalación de GoalBit Media Player

Para la instalación de un programa en una XO, que llamamos actividad, se debe crear un paquete zip con cierta estructura interna de archivos y su extensión es XO. Este paquete XO debe contener al menos los siguientes archivos:

- Activity.info: Un archivo de propiedades que describe al paquete, nombre de la actividad, identificador, clase principal, nombre del icono y numero de versión entre otras.
- Icono de la actividad, en formato SVG W3C [53].
- Una clase principal que extienda a la clase activity.Activity en lenguaje Python.
- MANIFEST: Contiente la ruta relativa de todos los archivos en el paquete.

Dentro del directorio raíz de la nueva actividad se cuenta con la aplicación GoalBit desarrollada en lenguaje C, compilada para la arquitectura de la XO en una PC de escritorio con la facilidad que brinda el comando de Linux chroot mencionado anteriormente. Dado que GoalBit para su funcionamiento necesita ciertas bibliotecas del sistema y puesto que las XO no cuentan con ellas, al paquete se agrega un nuevo directorio lib en el que se tienen aquellas bibliotecas necesarias que no se encuentran en la XO en los directorios estándar de Linux como el directorio /lib.

Una vez creado el paquete XO de la nueva actividad se lo instala usando el script de instalación Sugar-install-bundle en el que únicamente se le pasa por paramento la ruta al paquete XO. Finalmente luego de reiniciar Sugar la nueva actividad aparece en la interfaz para su uso corriente. De la actividad generada en este proyecto se hablará en la Sección 8.

7. GoalBit Media Player, edición Web Plugin

GoalBit Media Player cuenta con varias interfaces de usuario, entre las cuales están la consola de Linux, una interfaz Qt[10], es capaz de recibir comandos remotos mediante una interfaz http e incluso cuenta con un plugin para navegadores

[10]Qt es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario.

Web, con el objetivo de poder embeber al GMP (GoalBit Media Player) en una página HTML. Del plugin Web o Web Plugin de GoalBit es de lo que se hablará en esta Sección.

Con el plugin Web es posible comandar al GMP utilizando el lenguaje de scripting javascript. El mismo implementa la interfaz NPAPI-NPRuntime para los navegadores que dan soporte a esta API. Adicionalmente, GoalBit cuenta con un control Activex para los navegadores Internet Explorer de Microsoft. Este control Activex implementa la misma interfaz que el plugin NPAPI por lo que es posible utilizar el mismo código javascript para manipular el reproductor multiplataforma GoalBit independientemente del navegador Web.

Si bien en este proyecto se actualizó este control Activex agregándole las mismas funcionalidades que se le agregaron al plugin para los navegadores que utilicen la plataforma de plugins NPAPI, con el fin de no dejar desactualizado un plugin respecto al otro, nos enfocaremos en detallar la implementación y arquitectura del plugin NPAPI, puesto que es el que se utiliza por el Navegador Web (la actividad GoalBit) de las XOs.

A lo largo de esta Sección, con el fin de esclarecer cómo se implementó esta solución, se explicará por medio de diagramas, pseudo código y fragmentos de código cercanos al código real, ciertos ítem relevantes.

En la Figura 29 se muestra un diagrama simplificado que muestra la relación entre los distintos componentes que componen el plugin de GoalBit.

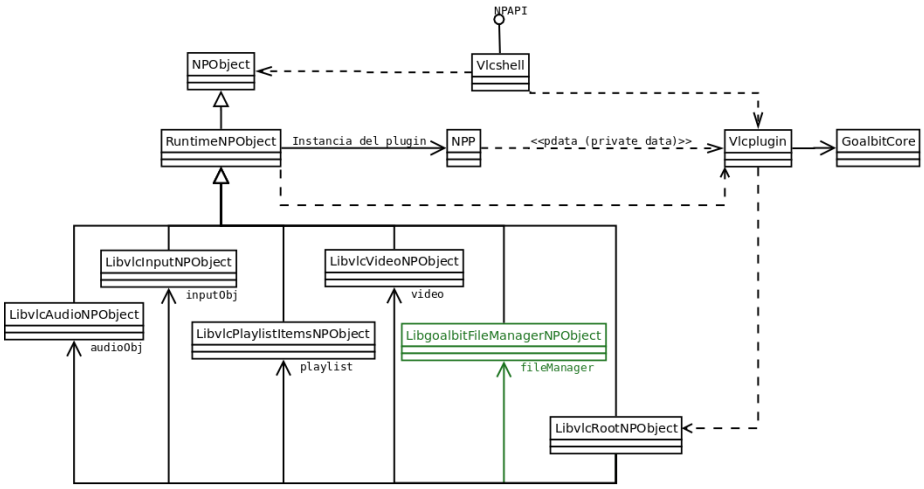

Figura 29: Diagrama de componentes del plugin. Se señala en verde el objeto FileManager (agregado en este proyecto) el cual permite implementar el selector de archivos.

En la Figura 29 podemos apreciar a *Vlcshell*, que corresponde a un archivo escrito en C++ (vlcshell.cpp). Este archivo es muy importante porque es donde se implementa la interfaz NPAPI. Es quien crea la instancia de la clase *Vlcplugin*, y por medio de esta última que es posible acceder a *LibvlcRootNPObject* cuando

se le invoca la función *getScriptClass()*. *LibvlcRootNPObject* es la clase raíz desde la cual se accede a todas las propiedades y funciones del plugin Web.

VlcPlugin actúa como el punto de entrada al GMP, de quien tiene una referencia directa, y es accesible para todos los componentes del plugin. Es creado e inicializado en la función *NPP_New()* de *vlcshell*. Aquí mismo se guarda la referencia a *VlcPlugin* como un dato privado de la instancia del plugin (atributo pdata de la clase *NPP* que representa la instancia del plugin). Este objeto es visible para los objetos que extienden *RuntimeNPObject* y lo pueden obtener invocando al método *GetPrivate()*.

Si bien es algo compleja la forma en que se llega desde las invocaciones a funciones y propiedades javascript del plugin, a su correspondiente implementación en código C/C++, en el proyecto VLC de donde GoalBit heredó la arquitectura del plugin, se desarrolló una estructura de clases que permite abstraerse de esta implementación y a los desarrolladores agregar nuevas propiedades y métodos al plugin sin mayor esfuerzo. En esta arquitectura se definen para toda clase que extienda a RuntimeNPObject los métodos y propiedades que son expuestas hacia javascript con dos arreglos:

- *propertyNames* es la lista de propiedades (tanto de solo lectura como de lectura y escritura)
- *methodNames* corresponde a la lista de métodos accesibles desde javascript del objeto.

En el diagrama de la Figura 29 podemos apreciar al objeto raíz del plugin que es LibVlcRootNPObject. Esta clase tiene el arreglo *propertyNames* definido como *["audio", "input", "playlist", "video", "VersionInfo", "fileManager"]*, de esta forma cuando se accede a la propiedad playlist del plugin (plugin.playlist desde javascript) se estará accediendo al objeto LibVlcRootNPObject indicándole mediante un string que debe devolver una instancia de clase LibVlcPlayListItemsNPObject. Está última es también una sub-clase de RuntimeNPObject y por lo tanto describe los métodos y propiedades que expone hacia javascript de la misma manera. Dado lo anterior, el proceso al acceder un nivel más, por ejemplo "plugin.playlist.play", es análogo.

Al invocarse un método javascript o acceder a una propiedad del plugin, el navegador Web inicia una invocación según se encuentra especificado por NPAPI. Luego, gracias al framework mencionado anteriormente, al objeto NPRuntimeObject correspondiente se le termina invocando el método invoke(índice,...) o GetProperty(índice,...) según el caso, con el índice del método o propiedad correspondiente, junto con los parámetros adicionales que sean necesarios.

El plugin de GoalBit ya contaba con la estructura de objetos marcados en negro descrita en la Figura 29. Los objetos accesibles desde javascript eran:

- LibvlcAudioNPObject, accesible como la propiedad *audio* del plugin.
- LibvlcInputNPObject, accesible como la propiedad *input* del plugin.
- LibvlcPlaylistItemsNPObject, accesible como la propiedad *playlist* del plugin.

- LibvlcVideoNPObject, accesible como la propiedad *video* del plugin.

En la Figura 30 se muestra un diagrama de colaboración que ilustra como se obtiene la clase raíz del plugin y en la Figura 31 como se accede a las propiedades/objetos de éste.

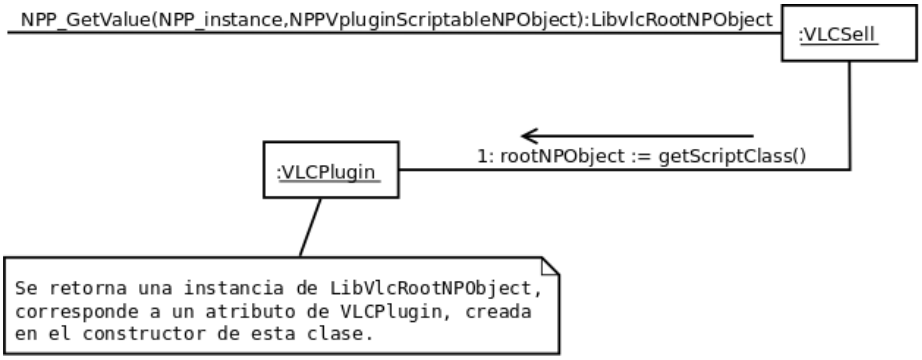

Figura 30: Diagrama de colaboración que representa como se obtiene una instancia de la clase raíz del plugin LibVlcRootNPObject.

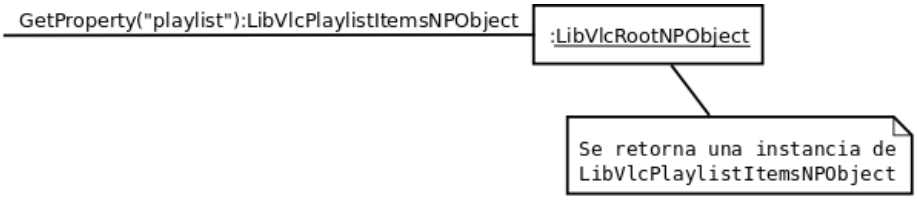

Figura 31: Diagrama de colaboración que representa como se obtiene la propiedad *playlist* del plugin, correspondiente a una instancia de la clase LibVlcRootNPObject.

Para ver en forma exhaustiva los parámetros/atributos soportados por el Web plugin así como también los métodos y propiedades expuestos hacia javascript que éste ofrece, ver el Anexo C. El código fuente asociado al Web-plugin se encuentra en el sub-directorio projects/Mozilla, dentro del proyecto GoalBit.

7.1. Inicialización del plugin

El plugin Web es inicializado una sola vez por ocurrencia de la etiqueta *EMBED* de HTML, al cargar la página Web, como fué explicado en la Sección 3 del Marco Teórico. La función de la clase VlcPlugin a la que se invoca cuando se inicializa el plugin es VlcPlugin::init(), en esta función principalmente se crea una instancia del GMP (GoalBit Media Player), especificándole los parámetros obtenidos desde el objeto del DOM (Domain Object Model) EMBED.

```
libvlc_instance = libvlc_new(ppsz_argc, ppsz_argv, &ex);
```

En donde libvlc_instance es un atributo de VlcPlugin, por medio del cual se conserva la referencia al GMP creado.

Los nuevos parámetros/atributos soportados por el Web plugin, agregados a lo largo de este proyecto, son principalmente "b_yourself" y "broadcast_prof". Estos fueron agregados para especificar el modo en el que se desea utilizar al plugin Web.

b_yourself Este atributo es de tipo booleano y cuando su valor es "true" indica que se utilizará al plugin Web en modo *Boradcast Yourself*. En este caso se crea adicionalmente una nueva interfaz GoalBit que corresponde al bt_manger.

```
if( !strcmp( argn[i], "b_yourself" ) ){
    broadcaster_yourself = boolValue(argv[i]);
}
...
if ( broadcaster_yourself ){
    libvlc_add_intf( libvlc_instance, "goalbit_bt_manager", &ex );
}
```

Esta interfaz es levantada con el fin de mejorar el rendimiento cuando se interactúa con el *Broadcaster*. Cuando se decide comenzar a transmitir el *streaming* multimedia (broadcasting) se crean los peers del tipo que corresponda, ya sean broadcaster-peers o broadcaster-super-peers, pero en ambos casos se necesita tener un Bittorrent Manager que los maneje. Habiendo levantado esta interfaz desde un principio hace que no sea necesario crearla y destruirla cada vez que se comienza y se detiene el *streaming* multimedia por medio de las directivas "play" y "stop" del GMP respectivamente.

broadcast_prof Este atributo es de tipo booleano y cuando su valor es "true" indica que se quiere utilizar al plugin Web en modo *Professional Broadcast* . En este caso al igual que con el parámetro anterior se crea una nueva interfaz GoalBit, que corresponde al *Web Professional Broadcaster Manager*. Esta interfaz a diferencia de la anterior no es levantada solo por razones de rendimiento sino que es necesario crearla una sola vez por cada instancia de plugin y debe estar levantada desde el principio, puesto que se interactúa con ella a lo largo de la ejecución.

```
if( !strcmp( argn[i], "broadcast_prof" ) ){
    broadcaster_prof = boolValue(argv[i]);
}
...
if ( broadcaster_prof ){
    libvlc_add_intf( libvlc_instance, "goalbit_web_prof_broadcaster_manager", &ex );
}
```

Dado este modo de funcionamiento del Web Plugin deben especificarse adicionalmente los parámetros: *controller-url*, *cn-suffix*, *certs-url* y *certs-suffix*.

En la Sección 10 se especifica lo referente al *Web Professional Broadcasting* .

7.2. Manejador del sistema de archivos

En este proyecto se agregó un nuevo objeto al plugin que se encarga del manejo del sistema de archivos que es LibgoalbitFileManagerNPObject, este permite desde javascript poder explorar el Sistema de archivos del cliente Web.

Para agregarle un nuevo objeto/propiedad al LibVlcRootNPObject se creó la clase LibgoalbitFileManagerNPObject que extiende a NPRuntimeObject. Desde javascript el objeto fileManager es invocado como una propiedad del objeto principal (plugin.fileManager). Es por esto que debe agregarse también una propiedad en el objeto LibvlcRootNPObject. Se agrega, entonces, "fileManager" al atributo *propertyNames* del objeto raíz.

```
const NPUTF8 * const LibvlcRootNPObject::propertyNames[] = {
    "audio",
    "input",
    "playlist",
    "video",
    "VersionInfo",
    "fileManager",
};
```

También se modifica adecuadamente el método GetProperty de este objeto, que es el método que será invocado cuando se acceda desde javascript a plugin.fileManager, devolviéndose una instancia de la nueva clase LibGoalbitFileManagerNPObject.

```
InvokeResult LibvlcRootNPObject::getProperty(index, result) {
    if( isPluginRunning() ){
        switch(index){
            case ID_root_fileManager:

              result = new LibgoalbitFileManagerNPObject();
              return INVOKERESULT_NO_ERROR;
            case ID_root_audio:
              ...
            ...
            default:;
```

```
        }
    }
    return INVOKERESULT_GENERIC_ERROR;
}
```

Luego, de forma análoga al objeto raíz, deben declararse los nombres de los métodos y propiedades accesibles desde javascript que tendrá el nuevo objeto.

```
const NPUTF8 * const LibgoalbitFileManagerNPObject::propertyNames[] = { };
const NPUTF8 * const LibgoalbitFileManagerNPObject::methodNames[] = {
    "ls",
    "getRootPath",
    "getTmpPath",
};
```

También se deben implementar los métodos *getProperty y setProperty* para obtener y especificar el valor de las propiedades.

```
InvokeResult LibgoalbitFileManagerNPObject::getProperty(index, result) {
    return INVOKERESULT_GENERIC_ERROR;
}
InvokeResult LibgoalbitFileManagerNPObject::setProperty(index, value) {
    return INVOKERESULT_GENERIC_ERROR;
}
```

Fianlmente se implementa el método *invoke* para las funciones expuestas a javascript:

```
InvokeResult LibgoalbitFileManagerNPObject::invoke(index, args, result){
    if( isPluginRunning() ){
        switch( index ){
            case ID_ls:
              ...
              return INVOKERESULT_NO_ERROR;
            case ID_getRootPath:
              ...
              return INVOKERESULT_GENERIC_ERROR;
            case ID_getTmpPath:
```

```
            ...
            return INVOKERESULT_GENERIC_ERROR;
          default:
            ;
        }
    }
    return INVOKERESULT_GENERIC_ERROR;
}
```

Función "ls" La función principal que ofrece este nuevo objeto del plugin es "*ls*". Esta función recibe como parámetro la ruta de un directorio y devuelve en formato JSON (Javascript Object Notation) una colección de objetos que representan el contenido de éste. Un ejemplo de salida es: [{}, {is_dir:1, is_hidden:1, file_name:'.config'}], en donde el primer elemento es auxiliar, y los atributos de los demás objetos de la colección especifican si es un directorio con el atributo *is_dir* en 1, si es un archivo oculto con el atributo *is_hidden* en 1 y por último el nombre del archivo con el atributo *file_name*. Esta función fue creada principalmente para dar soporte a un selector de archivos[11].

Esta funcionalidad por otra parte introduce un problema de seguridad, porque expone por completo el sistema de archivos del cliente Web que tenga instalado el plugin de GoalBit.

Luego de una pequeña investigación determinamos que Veetle5.2, quien también implementa un selector de archivos en Linux, resuelve este problema de la misma manera, utilizando una función de su plugin (que al igual que GoalBit es basado en VLC) y adolece de la misma vulnerabilidad[12].

Una solución posible al problema de seguridad planteado anteriormente podría ser tener un archivo de configuración en donde se especifique la ruta base desde la cual el usuario quiere explorar su sistema de archivos, ejemplo "/home/olpc/Videos" en el caso de las XOs. De esta manera en la función que expone el plugin se controlaría que la ruta pasada por parámetro contenga como base a la especificada en dicho archivo y por lo tanto no se expone todo el sistema de archivos del usuario a un potencial atacante. Esta ruta sería la que devolviera la función *getRootPath()* ofrecida por este mismo objeto.

7.3. Capacidad de broadcast desde el Web plugin

Al comienzo de este proyecto el plugin de GoalBit brindaba las funcionalidades de un reproductor multimedia. En este proyecto se agregó la funcionalidad que permite manejar a un broadcaster desde javascript.

[11] En el portal creado en el proyecto se utiliza con estos fines

[12] La función que expone este plugin para listar el contenido de un directorio es "dirInfo". Es notable recalcar que no sabemos ciertamente si en el código de esta función se hace algún control de seguridad (dado que no es de código abierto), pero se instaló el plugin en una maquina con ubuntu versión 10.10 y por medio del plugin de Veetle y utilizando la función anterior fué posible explorar el sistema de archivos por completo, dando una excepción solo cuando se intentaba listar el contenido del directorio raíz de Linux "/".

En un principio se creó un nuevo objeto del plugin, tal como *LibgoalbitFileManagerNPObject*, que brindaba las funciones básicas que permitían manejar un broadcaster, pero luego al evaluar el prototipo y otras soluciones posibles, se observaron otras aplicaciones similares como por ejemplo el portal de Veetle (Sección 5.2), el cual ofrece la posibilidad de especificar una lista de fuentes multimedia como entrada para hacer *streaming*, similar a la lista de reproducción. Si bien en el portal GoalBit-Ceibal (Sección 11) se optó por tener como única fuente multimedia a la cámara Web con la que cuenta la XO, se decidió modificar radicalmente la solución dándole al plugin la potencialidad que otros ofrecen. Dejando la puerta abierta para que en este portal (así como cualquier otro portal Web en el que se utilice el plugin de GoalBit) pueda ser posible manipular múltiples fuentes multimedia, con todas las funcionalidades con las que cuenta el reproductor con la lista de reproducción. Dado que el plugin ya contaba con la funcionalidad de la lista de reproducción, se logró el cometido modificando la implementación de ciertas funciones del plugin de GoalBit para que soportaran esta nueva funcionalidad.

Para poder, a estos efectos, realizar un broadcast primeramente se debe poder especificar la fuente multimedia de la que se desea tomar el *streaming*, luego se tiene que levantar el broadcaster y durante el transcurso del envío de la señal se tiene que poder controlar al mismo, al menos con las directivas de *play* y *stop*.

7.3.1. Especificación de la fuente multimedia

Se decidió utilizar el mismo objeto playlist que permite mantener una lista de reproducción para mantener una lista de fuentes multimedia a transmitir. Para esto se modificó la función *playlist_add_extended_untrusted* de la clase *VlcPlugin*. Esta función agrega un elemento a la playlist de GoalBit y recibe como parámetro una MRL (Multimedia Resource Location), el nombre del recurso que se desea agregar a la playlist y una lista de opciones que determinan la manera en la que será reproducido. El siguiente pseudo código describe la implementación de esta función:

```
playlist_add_extended_untrusted( mrl, nombre, opciónes ){
  media = libvlc_media_new( mrl )
  para cada opción en opciónes
    libvlc_media_add_option_untrusted( media, opción );
  libvlc_media_list_add_media( playlist, media );
}
```

A la función anterior se le sustituyó el llamado a la función libvlc_media_add_option_untrusted por libvlc_media_add_option. La principal diferencia entre estas dos funciones es que la última no realiza algunos chequeos sobre las opciones que se pasan por parámetro "confiando" en ellas. Por ejemplo, libvlc_media_add_option_untrusted no permite especificar entre las opciones de la fuente multimedia la opción *:sout,* la cual permite ajustar los parámetros relativos al *streaming* de salida, indicando cosas como por ejemplo los destinos, transcodificación, etc. Se considera que

permitir el uso de esta opción a un plugin Web es un riesgo de seguridad, ya que por ejemplo permitiría la sobreescritura de archivos locales si se pasara por parámetro como destino de la transmisión la ruta de un archivo conocido.

Para ilustrar mejor la especificación de la fuente y cómo se agrega a la playlist ofrecemos un ejemplo. Para agregar la fuente multimedia correspondiente a la cámara Web en un entorno Linux desde javascript, de manera similar a como se hace en el portal ceibal-goalbit (Sección 11) se utiliza el siguiente código:

```
goalbit.playlist.add(
    “v4l2://”,
    null,
    ":sout=#transcode{
       vcodec=theo, vb=300, scale=1, acodec=vorb, ab="+ab+", channels=2
    }
    :duplicate{dst=display, dst=qoe_injector_duplicate{ dst=std{
    access=goalbit_broadcaster{
       piece-len=65536, base-dir='[dir]', tracker-url='[URL]', bitrate=300,
       channel-name='[Nombre_canal]'}, mux=ogg, dst='...'
    }}}"
);
```

Explicaremos a alto nivel que es cada parámetro pasado a la función del plugin y adicionalmente como se desencadenan las llamadas hasta llegar al GMP. El primer parámetro corresponde al MRL en el que se especifica que se desea usar la cámara Web, más particularmente usando la biblioteca v4l2 (Video for Linux versión 2) , el segundo parámetro corresponde al nombre que se le quiera dar al elemento en la lista y el tercero a las opciones extras. En estas opciones extras es que se especifican todas las demás opciones que se desean del broadcaster.

En las opciones extras se encuentran “sout” y “duplicate”. La opción “sout” es utilizada para permitir ajustar los parámetros relativos al *streaming* de salida como ya se dijo anteriormente, y en este caso se puede ver que se especifica el parámetro “transcode” para el *Transcoding*, el cual sirve únicamente para el *Boradcast Yourself* puesto que para el *Web Profesional Broadcasting* que es el que se utiliza en el portal GoalBit-ceibal (ver Sección 11) se determina centralmente desde la Suite qué codecs se utilizarán en la emisión de audio, video y los bitrates de estos por cada canal. Para el *Transcoding,* en el fragmento de código se especifica que se desea utilizar al codec Theora para el video, con un bitrate de 300Kb/s, para el audio el codec Vorbis y con un bitrate de 32Kb/s y 2 canales. La opción “duplicate” duplica la salida, permitiendo al usuario visualizar lo que se esta transmitiendo. Aquí se manda al display para visualizar y la segunda salida va hacia el GoalBit *Broadcaster*, quien recibe por parámetro lo especificado como *pice-len*, *base-dir*, *tracker-url*, *bitrate*, otra vez aquí hay parámetros que solo se utilizan en el modo *Broadcast Yourself* que son el *tracker-url* y el *bitrate* ambos al igual que el transcode son determinados en la GoalBit Suite para cada canal.

La implementación en código C/C++ que hay detrás de la invocación javascript, es ilustrada en el diagrama de colaboración de la Figura 32, en la cual se puede apreciar la función *playlist_ add_ extended_ untrusted* anteriormente descripta, perteneciente a la clase *VLCPlugin*.

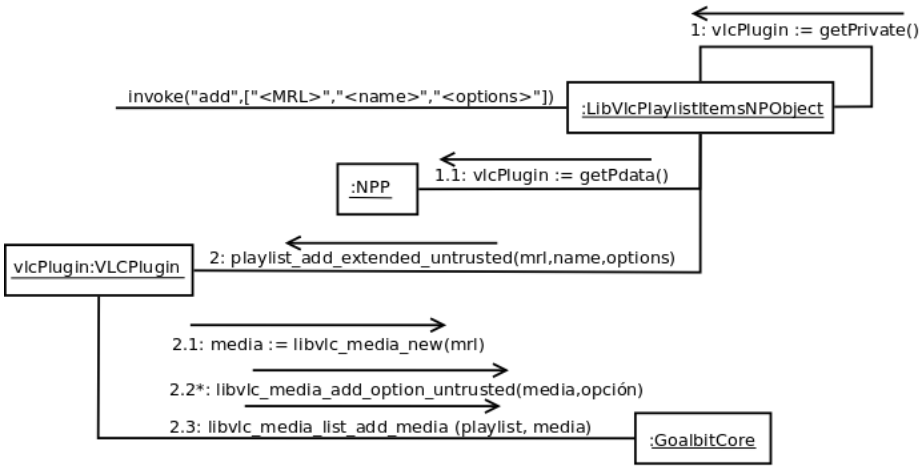

Figura 32: Diagrama que representa la pila de llamadas realizadas a raíz de invocar a la función *add* del objeto *playlist* del plugin.

Con la intención de simplificar el diagrama de la Figura 32 anterior, se muestra al GoalBitCore como un único componente y no con todas las clases que constituyen al GMP.

7.3.2. Control de la señal trasmitida

Para controlar la señal transmitida se tienen las operaciones, "play", "playItem" y "stop". Estas funciones no fueron modificadas en este proyecto y son las mismas que se utilizan para el reproductor multimedia corriente. Pero para tener presente como es la implementación de estas llamadas en el plugin y como se invocan desde javascript se muestran extractos de código y diagramas de colaboración.

Para comenzar a trasmitir la señal se debe invocar a "play" o "playItem", dado que en el portal Web se emite unicamnete un elemento (la cámara Web), es lo mismo usar "play", que "playItem" con el índice igual a 0.

```
goalbit.playlist.playItem(0);
```

El diagrama que muestra la pila de llamadas correspondiente a la línea de código javascript anterior se puede ver en la Figura 33.

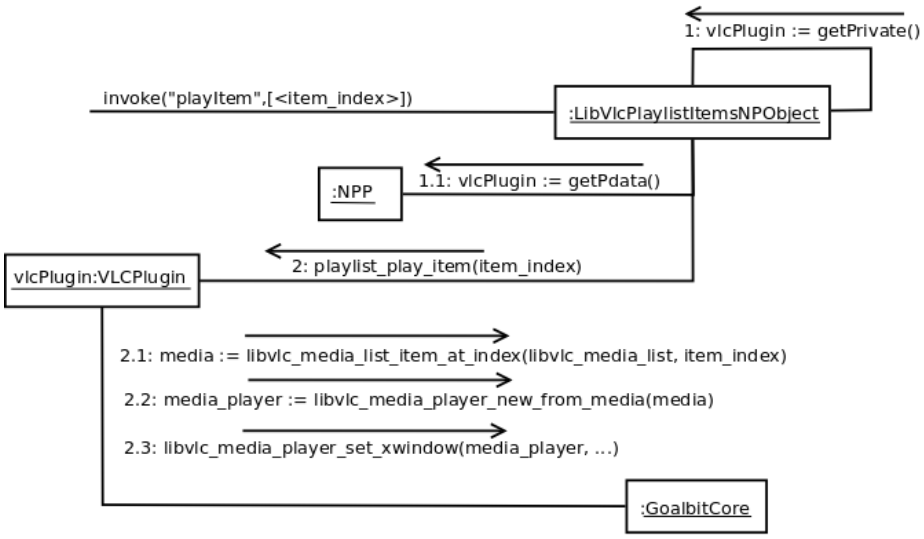

Figura 33: Diagrama que representa la pila de llamadas realizadas a raíz de invocar a la función *play* del objeto *playlist* del plugin.

Para dejar de transmitir se usa la función de la lista de reproducción "stop":

```
goalbit.playlist.stop();
```

El diagrama correspondiente a esta línea se puede apreciar en la Figura 34.

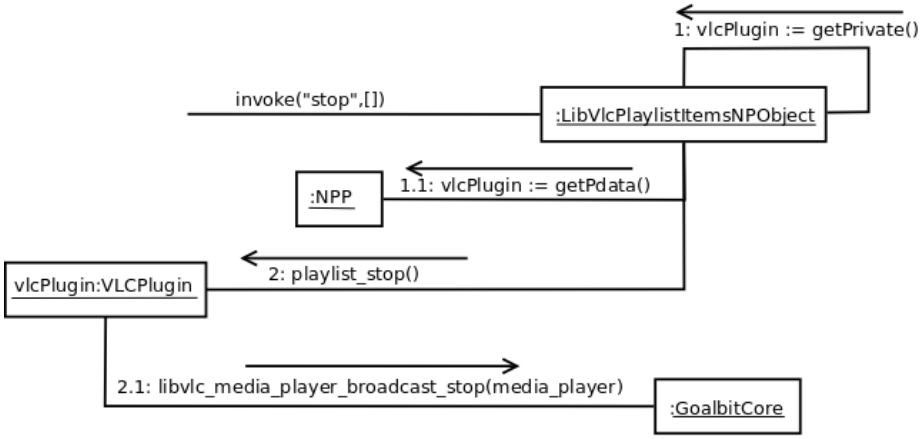

Figura 34: Diagrama que representa la pila de llamadas realizadas a raíz de invocar a la función *stop* del objeto *playlist* del plugin.

8. Actividad GoalBit

En la Sub-sección 6.1 se mencionó que todos los programas que el usuario manipula en el entorno Sugar son llamados actividades y adicionalmente se especificó como se genera el paquete de extensión XO para su instalación. Parte del producto generado a lo largo de este proyecto es una actividad que utiliza el GMP, y es por esto que la llamamos "Actividad GoalBit". Esta actividad se puede decir que tiene dos grandes modos, uno cuando se tiene conexión a Internet y otro cuando no se tiene:

1. Cuando no se tiene conexión a Internet se utiliza al GMP como un reproductor multimedia, pudiendo escuchar música o ver películas, manejando una lista de reproducción de archivos locales. En este caso la interfaz de la aplicación consiste en una página de contenido estático en la que se encuentra embebida el plugin de GoalBit.

2. Si se detecta que se tiene conexión a Internet el usuario es re-dirigido a un portal Web (ingresándolo en el mismo automáticamente) en donde no solo cuenta con la funcionalidad de un reproductor multimedia como en el caso anterior, sino que también se le permite consumir *streamings* de video creados por Ceibal, generar su propio *streaming* utilizando la cámara Web y el de la XO, o ver a otros niños que están generando esta clase de *streaming*.

El GoalBit Media Player como parte de la actividad Esta actividad tiene el GMP en el sub-directorio "goalbit" dentro de su directorio principal. El mismo fue compilado en el ambiente de desarrollo descripto en la Sección 6, utilizando el comando *chroot* de Linux. La versión de GoalBit con la que se creó este paquete es la 0.7.6, y si se quisiera actualizar la versión de GoalBit para esta actividad se tienen dos posibles opciones: Una sería reinstalar la actividad a partir de otro paquete XO, con el GMP compilado actualizado y la otra es simplemente sustituir el GMP viejo por la nueva versión compilada de la misma manera.

8.1. Interfaz

Debido a la popularidad que ha ganado Internet, existe una creciente tendencia a crear cada vez más aplicaciones Web e incluso migrar aplicaciones de escritorio a este nuevo entorno. Una de las principales ventajas que presentan las aplicaciones Web ante las aplicaciones de escritorio, es el hecho de que no dependen de ningún sistema operativo ni configuración de hardware específica. Para su ejecución simplemente basta digitar su dirección URL (Uniform Resource Locator) en cualquier navegador Web. De igual manera sus actualizaciones se hacen de una manera muy sencilla, sin necesidad de hacer descargas ni instalaciones.

Si bien la mayoría de las actividades de las XOs utilizan una interfaz de escritorio hecha en python, en este proyecto se decidió utilizar una interfaz

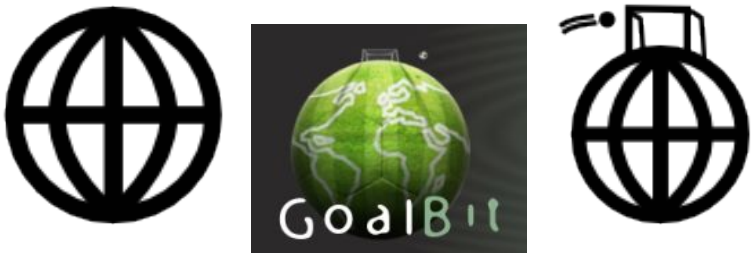

Figura 35: Icono de la actividad Navegar, GoalBit y Actividad GoalBit

Web. Al elegir este tipo de aplicación para este proyecto, no solo cuenta con los beneficios de las aplicaciones Web, sino que también es posible interactuar de una forma más directa con la GoalBit-Suite.

Dentro de las actividades que vienen instaladas con las XOs está Navegar, una actividad hecha en python que consiste en un navegador Web basado en Mozilla Firefox. Esta actividad utiliza un control llamado hulahop, que transforma a Gecko 1.9 (el motor de *renderizado* utilizado por Firefox 3.0) en un control simple y accesible desde python. Esto permite embeber un navegador Web similar a Mozilla Firefox en una aplicación de escritorio escrita en python. La actividad Navegar cuenta con soporte para javascript y plugins basados en la interfaz NPAPI-NPRuntime, tal como un navegador Firefox convencional. Como se mencionó en la Subsección 3, la API NPAPI (Netscape Plugin Application Programming Interface) creada para el navegador Web Netscape, permite crear plugins para el navegador Web, y con la extensión realizada posteriormente NPRuntime es posible tener plugins como GoalBit, con el que es posible interactuar utilizando javascript.

La actividad Navegar con el plugin de GoalBit instalado es esencialmente la Actividad GoalBit. Con el fin de agregar las funcionalidades deseadas y cumplir con los requerimientos del cliente, a esta actividad Navegar le fueron hechas ciertas modificaciones.

La actividad GoalBit incluye el GMP y este plugin es instalado automáticamente al ejecutarla.

8.1.1. Icono de la actividad

Para diferenciar la actividad GoalBit de la actividad Navegar original se diseño un nuevo icono. El mismo es una combinación entre el logo de GoalBit y el de la actividad Navegar.

Los iconos para las actividades de Sugar tienen un comportamiento especial. Estan hechos con dos colores (típicamente con trazos de un color sobre fondo blanco) y mientras la actividad es cargada por Sugar el usuario ve el ícono de la actividad cambiar de color gradualmente. Además, todos los íconos de Sugar toman los colores elegidos por el usuario para distinguir su XO (los cuales son elegidos en el panel de control de Sugar).

Para poder generar un ícono que en Sugar se comporte adecuadamente, el mismo debe tener ciertas características:

- Deben ser un dibujo vectorial expresado en formato SVG. Para generar el ícono se utilizó en este caso el editor de gráficos vectoriales libre Inkscape Wixson [68]. El formato SVG representa el dibujo mediante un archivo XML. Para que el archivo XML producido por Inkscape sea simple y legible, a fin de poder editarlo fácilmente luego, es necesario guardar el dibujo como SVG optimizado.

- Los íconos en Sugar deben representarse dentro de un cuadrado de 55 píxeles. Ésto debe especificarse en la etiqueta <svg> del archivo SVG mediante los atributos height y width. Si estos atributos no están especificados Sugar no interpretará de forma correcta el ícono, produciendo un ícono en blanco.

- Para que el ícono cambie de color al cargar la actividad y que el mismo adopte los colores de Sugar deben definirse dos entidades (<!ENTITY ...>) en el XML del archivo SVG. Una para el color del trazo y otra para el color del fondo.

8.2. Script inicial de la actividad

Como se dijo antes en el archivo de configuración activty/activity.info, dentro de la actividad, se especifica entre otras cosas el script que se ejecutara al iniciar la actividad. Para la actividad GoalBit, el script inicial es run, se muestra el código del script y una breve explicación:

```
1  #!/bin/sh
2
3  #init
4  GOALBIT_PATH=/home/olpc/Activities/Goalbit.activity
5  PLUGINS_PATH=$GOALBIT_PATH/goalbit/lib/mozilla/plugins
6  PLUGINS_DEST=/home/olpc/.mozilla/plugins
7
8  #symbolic links - plugin
9  ln -s $PLUGINS_PATH/libgoalbitplugin.so $PLUGINS_DEST/
       libgoalbitplugin.so
10 ln -s $PLUGINS_PATH/libgoalbitplugin.la $PLUGINS_DEST/
       libgoalbitplugin.la
11
12 #uuid
13 echo -n 'uuid="'␣>␣$GOALBIT_PATH/data/uuid_content.js
14 cat␣/ofw/mfg-data/U#␣>>␣$GOALBIT_PATH/data/
       uuid_content.js
15 echo␣'";' >> $GOALBIT_PATH/data/uuid_content.js
16
```

```
17 #Sugar activity
18 SUGAR_BUNDLE_PATH=/home/olpc/Activities/Goalbit.
       activity
19 _ACTIVITY=goalbit.GoalbitActivity
20 _BUNDLE=uy.edu.fing.Goalbit
21 Sugar-activity ${_ACTIVITY} -b ${_BUNDLE} -a GoalBitId
```

En las líneas 9 y 10 se crean los links simbólicos para "instalar" el plugin de GoalBit para el Navegador, esto hace que quede instalado tanto para la actividad GoalBit, como para la actividad Navegar. Ambos navegadores mostraran entre los plugins instalados a GoalBit.

En las lineas 13,14 y 15 se crea el archivo data/uuid_content.js a partir del archivo /ofw/mfg-data/U#, utilizando los comandos "echo" y "cat" de la consola. Este archivo creado es utilizado luego para poder hacer el login automático al portal Web.

8.3. Login

Uno de los requerimientos del cliente consiste en que no sea necesario para el escolar recordar un usuario y contraseña para hacer uso del portal Web, por otra parte es necesario identificar a los usuarios que hacen uso del portal para recordar el estado de cada usuario. Para solucionar el problema, se utilizó un archivo que esta presente en toda XO y que tiene un identificador único (UUId de las siglas en ingles Universally Unique Identifier) entre las XOs, en lugar de usar un usuario y contraseña. El archivo mencionado es /ofw/mfg-data/U#.

1. Durante la inicialización de la actividad, más particularmente en el script run se lee el archivo /ofw/mfg-data/U#, y a partir de éste se crea otro que es /home/olpc/Activities/Goalbit.activity/data/js/uuid_content.js. El javascript resultado consiste en asignar a una variable javascript llamada uuid, de tipo alfanumérico, el valor UUId obtenido del primer archivo. Un ejemplo del resultado como uuid.js es:

   ```
   uuid="XXXXXXXXXX";
   ```

2. Luego para utilizar el uuid se incluye el javascript uuid.js en la página inicial de la actividad "index.html", para utilizarlo como usuario y contraseña en el login del portal. En el siguiente fragmento de código (el cual esta reducido para mostrar la funcionalidad) de la página inicial de la actividad, se hace una prueba de conexión tratando de obtener una imagen del portal. En caso de que falle esta carga, se ejecuta la función javascript correspondiente al evento "onerror" de la imagen (línea 17) que carga el reproductor local. Si la carga fuera exitosa, se considera que se tiene conexión a Internet y se asigna a los objetos del DOM de tipo input correspondientes al usuario (login del formulario) y contraseña (paswd del formulario) el valor de la variable javascript uuid. Adicionalmente se asigna el atributo 'action' del formulario y se ejecuta la función submit para enviar los datos al servidor y que se cargue el portal.

```
<html>
<head>
<script type="text/javascript">
   var uuid="";
</script>
<script type="text/javascript" src="js/
   uuid_content.js"></script>
<script type="text/javascript">
   var uri_portal = "...";
   function ready_function(){
      var i = new Image();
      i.onload = function(){
         document.getElementById("paswd").value =
            uuid;
         document.getElementById("login").value =
            uuid;
         document.forms[0].action = uri_portal+"
            login/";
         document.forms[0].submit();
      };
      i.onerror = function (){
         self.location.href = "reproductor_local.
            html";
      };
      //Evitando la cache
      i.src = uri_portal+'images/spacer.gif?d=' +
         escape(Date());
   }
</script>
</head>
<body onload="ready_function();" style="...">
   <form action="" method="post">
      <input type="hidden" name="login" id="login"
          value="uuid"/>
      <input type="hidden" name="paswd" id="paswd"
          value="uuid"/>
   </form>
</body>
</html>
```

8.4. Reproductor Local

En la Subsección 8.3 se explica cómo fue resuelto el problema del login de los escolares al portal y cómo, cuando falla la prueba de conexión a Internet (más particularmente al portal Web) se redirige al navegador a la página html

local “reproductor_local.html”. Esta página tiene el aspecto del portal pero únicamente brinda la funcionalidad de reproductor multimedia utilizando al GoalBit Media Player con este fin a través del plugin. Junto con esta página se alojan en el mismo directorio archivos javascript, hojas de estilo e imágenes necesarias para dar todas las funcionalidades y aspecto que el reproductor local le ofrece al usuario.

Para poder lograr que la página inicial del navegador sea index.html, fue modificada la clase principal de la anterior actividad Navegar y este corresponde al archivo ahora llamado GoalBit.py.

Este reproductor tiene una lista de reproducción a la que se le pueden agregar nuevos items a reproducir utilizando un selector de archivos, permite reproducir en el orden que se quiera cada uno de los items y automáticamente se reproducen en forma cíclica. Como todo reproductor se le puede pausar, parar y volver a reproducir. Cuenta con el modo “pantalla completa” que se obtiene dando doble click sobre el video. Y en todo momento de la reproducción se le permite al usuario adelantar el video por medio de una barra de desplazamiento que además indica el avance en el video. El reproductor local se puede apreciar en la Figura 36.

Figura 36: Reproductor local

8.4.1. Selector de archivos

Este selector permite escoger los archivos para la lista de reproducción desde el sistema de archivos del cliente Web. Adicionalmente es posible filtrar los mismos por su extensión separando los posibles valores por una coma, ej: “mpg,mp3,ogg,avi,mp4”. El mismo fue implementado especialmente para este proyecto utilizando para la interfaz ciertas bibliotecas javascript como jquery

[5] y jqueryFileTree.js [4]. Para reproducir un archivo multimedia es necesario que el plugin reciba la ruta completa del archivo. Como la interfaz es Web, se debe invocar al plugin proporcionando este parámetro desde un script javascript, por lo cual es necesario que se conozca la ruta completa de los archivos seleccionados en el script mencionado. HTML y javascript no permiten por razones de seguridad explorar el sistema de archivos del cliente Web. Para obtener esta información se utiliza el mismo plugin, al cual se le agregó una operación que dada la ruta escogida liste los archivos así como lo hace el comando "ls" de Unix, de aquí que esta nueva operación fue llamada "ls", ver Sección 7 en la que se habla a cerca del plugin de GoalBit. En la Figura 37 se muestra el selector de archivos.

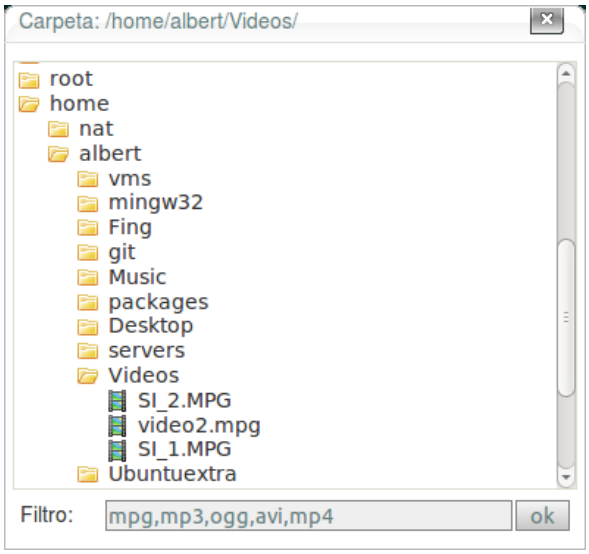

Figura 37: Selector de archivos

9. Seguridad

Teniendo en cuenta que la motivación final del proyecto es lograr, quizá en el futuro, la implantación del sistema para su uso masivo por el Plan Ceibal, y considerando que en ese caso la mayor parte de los usuarios finales serían estudiantes de escuela primaria, la seguridad ha sido un aspecto que se ha discutido durante todo el proyecto. En particular interesa asegurar que el contenido reproducido por el usuario final es apropiado, o por lo menos es el contenido que el usuario eligió reproducir. En un sistema de flujo de video con una arquitectura cliente-servidor lo anterior parece trivial, sin embargo, en un sistema con arquitectura peer to peer, el flujo de video es recibido de varias fuentes distintas que además cambian constantemente, lo cual transforma el problema en algo más que asegurar una conexión entre un cliente y un servidor.

Antes de comenzar este proyecto GoalBit ya implementaba medidas relacionadas a la seguridad. Por ejemplo proporciona la posibilidad de encriptar las piezas de un flujo de video de forma de proteger el contenido de un canal privado, mediante un sistema de DRM. Mediante este sistema un usuario es autorizado a consumir el canal mediante un código (*token*) que luego utilizará para descargar una clave simétrica que le permita descifrar las piezas de video. El proceso es automático y transparente al usuario, es relativamente liviano (ya que entre otras cosas utiliza criptografía simétrica) y es adecuado para algunos escenarios. Sin embargo, existen algunos problemas al aplicarlo al caso del Plan Ceibal.

Se llegó a la conclusión junto con el usuario responsable de que en el marco del Plan Ceibal no es tan importante proteger la confidencialidad de los flujos de video y éstos podrían ser públicos en un principio. Por otro lado, es sumamente importante identificar la fuente de un flujo y qué piezas de video en un canal fueron emitidas por los broadcasters del mismo. De esta forma es posible descartar contenido intruso en un canal e identificar a los broadcasters de contenido inadecuado. El sistema de DRM está diseñado para evitar la reproducción del contenido de canales privados por parte de usuarios no autorizados, pero no brinda las funcionalidades descritas. Además, teniendo en cuenta las limitaciones de cómputo de las XO y las necesidades descritas anteriormente, al ser el cifrado un proceso costoso, se decidió prescindir del mismo.

Por estas razones se implementó un sistema de firmas digitales para solucionar los problemas mencionados anteriormente. Se siguieron los lineamientos generales de diseño e implementación del sistema de DRM existente en GoalBit. Se hicieron cambios tanto en el GMP como en la Goalbit Suite.

En la Sección siguiente se describe brevemente el sistema de DRM de GoalBit y sus componentes. Luego, en la Sección 9.2 identifican los problemas de seguridad más importantes. Por último, en la Sección 9.3 se describe la solución implementada.

9.1. Goalbit Encrypt - sistema DRM de GoalBit

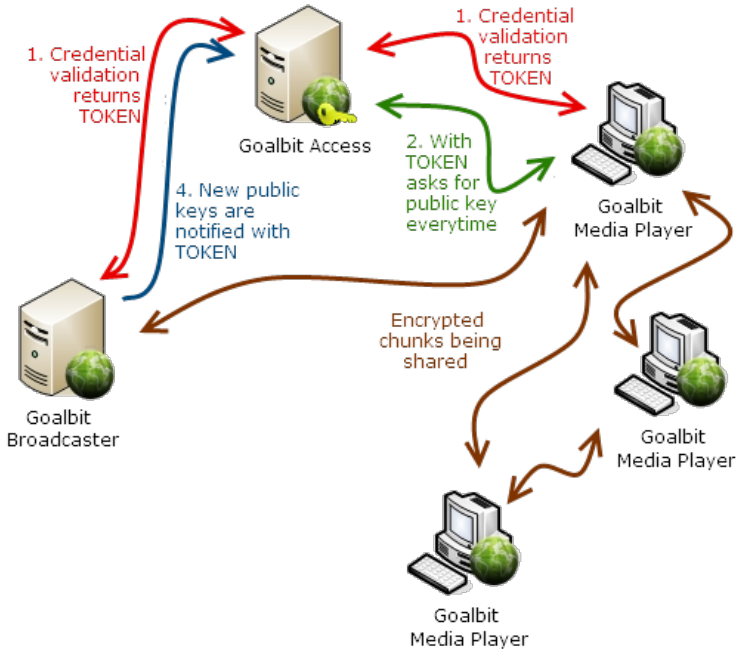

Figura 38: Sistema de DRM de GoalBit

En la Figura 38 se pueden apreciar los componentes más importantes (aunque no todos) que participan en el sistema de DRM. Entre ellos se destaca el llamado GoalBit Access, quien forma parte de la Suite de GoalBit y del que se habló en la Sección 1.3.3. Si el broadcaster desea que un canal sea privado, él mismo genera una clave con la cual cifrará las piezas del flujo de video pertenecientes al canal. La misma clave sirve para descifrar las piezas por tanto decimos que se trata de criptografía simétrica. Uno de los problemas inherentes al uso de este tipo de criptografía es el del intercambio de claves, ya que los usuarios finales deben obtener la clave generada por el broadcaster para ser capaces de descifrar las piezas que reciben.

Como se mencionó anteriormente, GoalBit tiene dos modos de funcionamiento o formas de uso. En el modo al que llamamos Professional Broadcast en la Sección 1.3.2, la GoalBit Suite oficia de intermediario entre el broadcaster y los usuarios finales para solucionar el problema de la distribución de las claves. Antes de comenzar la transmisión de video el broadcaster se registra con la Suite de GoalBit obteniendo de alguna forma segura un certificado de la Suite (clave pública de la Suite firmada por la autoridad certificadora de GoalBit), una clave privada y el certificado de la autoridad certificadora de GoalBit. Con estas credenciales, el broadcaster será capaz de establecer una conexión con la

Suite utilizando el protocolo TLS/SSL, y a través de esta conexión, utilizando la API interna de la Suite, le entregará a la misma la clave que ha generado y con la cual cifrará las piezas de video a emitir. El usuario, por su parte, no debe comunicarse con la API interna de la Suite de GoalBit, obteniendo las claves a través de una API especial que se le brinda, a la cual en la Sección 1.3.3 llamamos "Goalbit User Services". Para que el intercambio sea seguro y solo usuarios autorizados puedan consumir el canal, el sub-sistema GoalBit Access se encarga de verificar las credenciales de un usuario que desea consumir un flujo de video (por ejemplo usuario y contraseña) y, si dichas credenciales son válidas, el usuario autenticado obtiene un código temporal llamado token. Este token es asociado a dicho usuario y es su forma de autenticarse con el sistema GoalBit Access y poder descargar las claves con las que descifrar las piezas.

A medida que se obtienen y acumulan muchas piezas de video encriptadas, aumenta la posibilidad de encontrar la clave con la cual han sido encriptados, en especial si puede inferirse parte de los datos a partir del muxer o codec que se utilice. Debido a esto, para reforzar el sistema se utiliza una política de rotación de claves, en la cual el broadcaster cambia la clave de cifrado cada cierto número de piezas y publica las claves correspondientes a cada rango a través del GoalBit Access.

En el modo "broadcast yourself" la clave se distribuye mediante el mismo token, solo que se prescinde de la Suite y el usuario recibe este token directamente del broadcaster a través de un meta-archivo de GoalBit (archivo .goalbit) publicado de alguna forma. En este modo no existe la rotación de claves.

El sistema ha sufrido algunos cambios a partir de la versión 0.7 de la Suite de GoalBit. La mayoría de estos cambios están orientados a adecuar el sistema al broadcasting desde la Web, por lo tanto son descritos con más profundidad en la Sección 10 de este documento. Sin embargo, lo anterior ilustra la idea general del sistema de DRM y brinda un marco adecuado para profundizar sobre los aspectos de seguridad que mencionábamos en la introducción.

9.2. Consideraciones de seguridad

Si bien el uso del sistema de DRM de GoalBit es adecuado en muchos casos, se han identificado por lo menos dos aspectos que lo vuelven inadecuado para su uso en el Plan Ceibal:

- El sistema evita que un usuario no autorizado inyecte flujo de video malicioso en el canal.
- El broadcaster indica si la pieza está o no encriptada a través de un bit en el header de la pieza, por lo que parte de la seguridad del sistema se basa en información que es mandada en claro.

A continuación se describe en detalle estos dos problemas.

9.2.1. Inyección de piezas en un canal

En la Sección 1.3.1 del documento se habló brevemente acerca del protocolo de transporte de GoalBit (GBTP). El diseño de este protocolo, si bien es la característica principal de GoalBit y lo que lo distingue frente a otros sistemas de distribución de video, da lugar a ciertos escenarios que resultan en un comportamiento no deseado.

Para que un cliente o peer pueda participar del intercambio de piezas de determinado canal, lo único que debe conocer en principio es la dirección del tracker a los efectos de registrarse con él. Como se mencionó en las secciones 1.3 y 1.3.1, el tracker es una entidad a la que deben reportarse todos los usuarios del sistema, sean clientes (peers, super-peers, etc.) o broadcasters. Es esta entidad quien le brinda a un peer una lista con otros peers de los cuales puede obtener piezas. Si un broadcaster malicioso se registra con el tracker, éste lo incluirá en algunas de las listas de peers (swarm) que le entregará a los demás. Luego, los peers intentarán contactar a todos los peers de su swarm, y descargarán de aquellos peers que contengan piezas que pertenezcan a la ventana del primero y que el primero no tenga. Esta condición puede ser satisfecha por un broadcaster sin ningún tipo de trabajo adicional e, incluso, al ser un sistema de código abierto, un usuario experto y mal intencionado podría modificar GoalBit para que esta condición se cumpla sistemáticamente para ciertos peers. De esta forma, algunos peers que cuentan en su swarm con el broadcaster malicioso descargarían algunas piezas de éste, lo cual produciría una mezcla de los dos canales en el reproductor multimedia. Más aún el broadcaster podría implementar una política que le permita enviar piezas sistemáticamente a determinados peers de forma que estos únicamente puedan ver el contenido no deseado. El peer que recibe este contenido además comenzará a compartirlo con los demás peers de su swarm extendiéndolo por el sistema.

El problema, entonces, radica en que un peer recibe piezas de video de muchas fuentes distintas, de las cuales no tiene demasiada información. En particular, es imposible saber si una pieza que llega de determinado peer, fue emitida por el broadcaster auténtico de ese canal.

Una solución parcial a este problema está incluida dentro de lo que en la Sección 1.3.2 se llama "Professional Broadcast". Aquí solo usuarios autorizados tienen acceso al tracker y se utiliza un sistema de DRM similar al que fue descrito en la Sección anterior. Al exigir el uso de certificados por parte del broadcaster para comunicarse con la GoalBit Suite, el sistema de DRM evita que usuarios no autorizados se registren como broadcasters del sistema. Sin embargo, podría modificarse el código de GoalBit para que, sin contar con los certificados, un usuario autorizado a ver el contenido comparta piezas que no pertenecen al contenido original, interfiriendo con el funcionamiento del sistema. Para evitar esto, el broadcaster cifra las piezas y publica las claves mediante el sistema descrito en la Sección anterior, pero ésto no soluciona el problema del todo. Un usuario autorizado del sistema conocerá la clave con la cual debe descifrar las piezas de video y a que piezas pertenece dicha clave (recordando que cada clave sirve únicamente para determinado rango de piezas), pero al utilizar criptografía

```
struct bt_chunk_header_t {
    uint32_t  i_data_start;
    uint32_t  i_data_end;
    uint32_t  i_mux_header;
    uint32_t  i_extra_data;
    uint8_t   i_flags;
    uint32_t  i_key_frame;
    char      psz_sign[80]; //Agregado en este
       proyecto
};
```

Figura 39: Header de una pieza

simétrica, dicho usuario podría usar esas mismas claves para cifrar chunks con contenido malicioso y compartirlos con otros peers haciéndolos pasar por los originales.

Adicionalmente, el Professional Broadcast tiene la posibilidad de utilizar cualquier tracker, incluso un tracker público, al cual accederá cualquier usuario, no solo usuarios autorizados. Por más que un usuario no autorizado no podrá obtener las claves para descifrar los chunks, el mismo puede compartir "piezas basura" con otros peers.

9.2.2. Modificación de la bandera "encrypted"

Como se mencionó en la Sección 1.3.1, y repetidas veces a lo largo del documento, GoalBit distribuye un flujo de video dividiéndolo en chunks o piezas. Estas piezas contienen un *header*, en el cual se incluye información de control útil para el cliente, y un *payload* en donde va el contenido de video.

Como se aprecia en la Figura 39, el *header* de la pieza contiene un campo de banderas. Mediante este campo un broadcaster indica, entre otras cosas, si la pieza está cifrada.

Por una decisión de diseño el *header* del chunk se envía en claro. Esto propicia ataques del tipo man-in-the-middle (o intermediario) en los cuales este header puede ser modificado ya sea por un cliente GoalBit modificado, o simplemente por una herramienta de tipo *proxy* que capture el tráfico del una red en donde circulan piezas. En caso de que la pieza esté cifrada y se modifique la bandera, el cliente GoalBit pondrá en el buffer de reproducción información de video cifrada, inentendible para el demuxer y el decoder. En el caso contrario, en el que el chunk no está cifrado y se altera la flag, el mismo se manda a descifrar provocando un error en el módulo de cifrado o en el mejor de los casos enviando al buffer de reproducción información totalmente corrupta por el proceso de descifrado a la que fue sometida.

9.3. Sistema de firmas digitales

Mediante este sistema, el broadcaster incluye una firma digital en cada chunk que se envía y publica la clave mediante la cual los clientes pueden verificar su validez. Es adecuado entonces distinguir estas dos tareas (la de firmado y la de publicación de la clave), ya que el sistema varía según el modo de broadcast (*Professional* o *Yourself*) más que nada en la forma de publicar la clave.

Independientemente del modo de transmisión, siempre es el emisor quien genera las claves para el firmado. Al tratarse de una firma digital son generadas dos claves (una privada para firmar y una pública para verificar la validez de la firma). Una vez generadas las claves, el broadcaster podrá enviárselas a la Suite a través del protocolo TLS/SSL, o simplemente escribir un meta-archivo .goalbit que contenga la clave pública del firmado. Este archivo se distribuirá de alguna forma para que quienes lo obtengan puedan consumir el flujo firmado verificando la firma de cada pieza adecuadamente. Luego, el broadcaster firmará cada chunk antes de que sea introducido en la red P2P.

El cliente por otro lado, obtiene un archivo .goalbit ya sea de la GoalBit Suite en caso del *Professional Broadcast* o de algún medio independiente en el caso de *Broadcast Yourself*. Si el archivo contiene una entrada con la clave pública del firmado el cliente verificará la validez de la firma de las piezas que reciba. En caso contrario se actuará como si el canal no fuera firmado garantizando la compatibilidad hacia atrás con archivos .goalbit de versiones anteriores.

La implementación del sistema está divida en dos módulos principales:

- El módulo crypt proporciona la interfaz mediante la cual otros módulos deben acceder a las funcionalidades de firmado. En este módulo se mantiene el estado del sistema, es decir, datos como las claves públicas y privadas e información del canal.

- El módulo encryption se encarga de proveer una interfaz con la biblioteca criptográfica gcrypt. Encapsula el acceso a las funciones básicas de criptografía y convierte los datos a la entrada esperada por la biblioteca, la cual consiste en s-expressions[13]. A este módulo se le agregaron las funciones que permiten implementar el algoritmo DSA de firmas digitales.

A continuación se detalla la implementación del sistema, la cual se ilustra en la Figura 40. Las siguientes secciones se organizan según las distintas actividades dentro del sistema de firmas digitales, a saber: la generación e intercambio de las claves, el firmado y la verificación.

[13]S-Expression es una expresión parentizada que denota de forma estándar una estructura jerárquica. Cumple la misma función que expresiones del estilo de XML o JSON

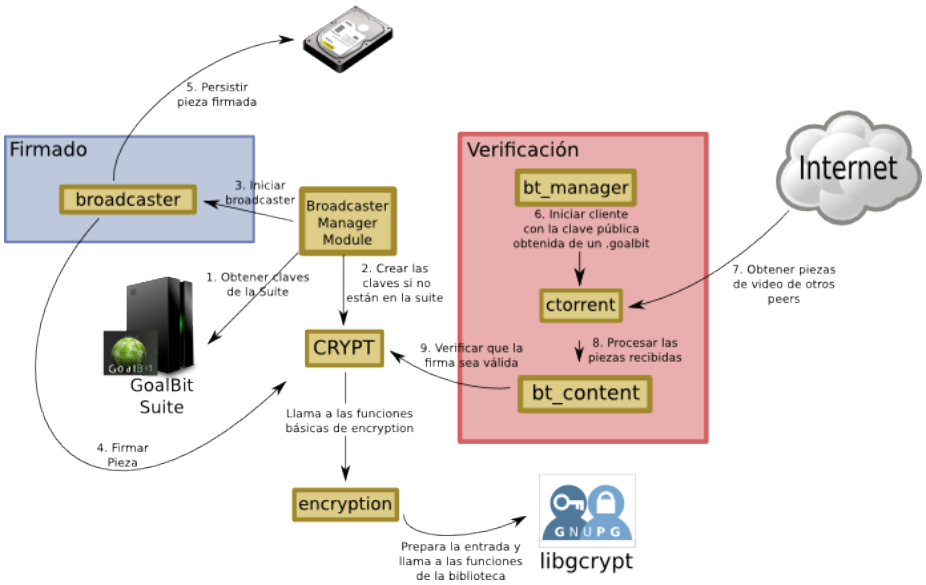

Figura 40: Esquema del sistema de firmas digitales.

9.3.1. Generación de claves

La fase de generación de claves varía ligeramente si se utiliza o no la GoalBit Suite. La diferencia fundamental consiste en que en el caso "profesional", el broadcaster debe encargarse de compartir las claves con la Suite mientras que en el otro caso corre por cuenta del usuario publicarlas por algún medio.

- En el caso de no utilizar la Suite, el broadcaster simplemente tiene la responsabilidad de generar el par de claves para el firmado. Dichas claves se asocian al canal y son mantenidas por el módulo crypt. Para no crear un nuevo par de claves cada vez que se inicia el broadcaster asociado a un canal, primero se intenta obtenerlas de un archivo que es guardado al momento de crearlas por primera vez en un directorio de configuración de GoalBit. Si no existe dicho archivo se procede a crear las claves nuevamente persistiéndolas en dicho directorio. La razón por la cual se hace esto es que, si se crea un nuevo par de claves, todos los meta-archivos .goalbit que contienen la antigua clave pública quedan obsoletos. Los usuarios que disponían de esos archivos para poder ver el canal deberán obtener uno nuevo. Esto no es problema en un escenario en el cual se descarga el archivo cada vez que se visualiza el canal de forma transparente al usuario, por ejemplo en un portal Web, pero sí lo es en el caso de que el usuario deba descargar manualmente el archivo o lo obtenga por otro medio.

- En el caso de utilizar la Suite, el broadcaster genera una única vez las claves pública y privada para luego enviárselas a ésta. En ocasiones subsiguientes en las cuales se inicialice el broadcaster de un canal profesional, la Suite,

si dispone de las claves en su base de datos, se las enviará al broadcaster para que las utilice. Este método intenta solucionar el mismo problema que se solucionaba mediante el archivo en el caso del "broadcast yourself", pero lo hace de manera más robusta. En el caso anterior, si se quisieran tener varios broadcasters para un mismo canal en caso de que alguno falle, quizá incluso en servidores independientes, sólo el que generó las claves en primera instancia dispondrá del archivo, mientras que los otros volverán a generar claves distintas al iniciarse. Por lo dicho anteriormente, entonces, el evento recibe como parámetro el par de claves. Si la Suite no dispone de ellas y las mismas no fueron enviadas las recibirá vacías, en cuyo caso las genera y envía a la Suite.

9.3.2. Firmado

El broadcaster simplemente recibe los datos a firmar junto con la longitud de los mismos y el ID del canal. Mediante este ID se obtiene la clave privada y transmite toda esta información a la función que realiza el firmado en el módulo encryption.cpp de quien GoalBit Crypt posee una instancia como atributo.

Los datos firmados contienen además dos datos que no viajan con la pieza y son agregados sólo al momento de generar la firma. Los datos de la firma están compuestos por la concatenación de:

- Un número de secuencia de la pieza
- El nombre del canal.
- La pieza de video generada, incluyendo su cabezal.

Esto provocaría que la firma sea rechazada en el caso de que las piezas de video lleguen desordenadas o que pertenezcan a otro canal del mismo emisor.

Naturalmente, al momento de generar la firma, es necesario excluir del cabezal de la pieza el espacio donde viajará la misma. Este espacio se rellena con ceros.

9.3.3. Verificación

Similar al caso del firmado, el receptor recibe los datos y la firma que se quieren comprobar, obtiene la clave pública del canal a través de su ID y llama a la función para verificar la firma en el módulo encryption.cpp.

Para verificar la firma de una pieza es necesario extraer la misma y luego reemplazar con ceros los 40 bytes del cabezal destinados a ésta. También debe obtenerse el nombre del canal y número de secuencia del chunk y pasarlo como parámetro a la función que verifica la firma.

9.3.4. Modificaciones a la GoalBit Suite

Para agregar firmado a los canales se hicieron también cambios en la GoalBit Suite.

En uno de los esquemas de la base de datos de la Suite se debieron modificar algunas tablas y crear una nueva:

- Se creó la tabla channel_sig_keys cuyos atributos son:
  - *sig_keys_id* (clave primaria)
  - *sig_pub_key* (almacena la clave publica)
  - *sig_priv_key* (almacena la clave privada)
- Se modificó la tabla goalbit_channel_conf:
  - Se agrega el atributo *sig_keys_id* (referencia a la clave primaria de la tabla channel_sig_keys)
  - Se agrega el atributo *sign_active* (un bit que indica si el canal está firmado o no).

Como se dijo anteriormente, en el modo de Professional Broadcast el broadcaster genera las claves del sistema de firmado y se las comunica a la Suite. Es importante recordar que el broadcaster hace ésto únicamente cuando la Suite no dispone de las claves.

Para que lo anterior sea posible se debió extender el módulo GoalBit Access realizando los siguientes cambios:

- Se crea una función que permite almacenar las claves en la tabla descripta anteriormente y se actualiza la referencia en la tabla de canales para vincular al canal con las claves insertadas.
- Se crea un servicio que se expone y permite crear claves para un nuevo canal, el cual hace uso de la función anterior.

Luego el broadcaster debe ser capaz de obtener las claves desde la Suite. Para poder lograr lo anterior, la Suite debe exponer un servicio capaz de retornar dichas claves. Con este fin se han realizado los siguientes cambios:

- Se modificó el componente *GoalBit Controller* de la *Suite* para que en la función que atiende el *announce* del broadcaster se retornen los siguientes parámetros:
  - *sign_active*, indica si esta activo el firmado.
  - *sig_pub_key*, corresponde a la clave pública utilizada en el proceso de firmado.
  - *sig_priv_key*, corresponde a la clave privada utilizada en el proceso de firmado.

Para que un branch pueda crear un canal con firmado se realizaron las siguientes modificaciones al *GoalBit Controller*:

- Se modificó la función encargada de la creación de las señales para que contemplen el nuevo atributo "sign_active", referente al firmado.

Para incorporar la clave pública de un canal firmado en el archivo .goalbit generado por la Suite se modificó la función encargada de generarlo, para que agregue la clave pública en el caso de que el canal tenga el firmado activo:

```
<sig-pubkey>[SIG_PUB_KEY]</sig-pubkey>
```

10. Web Profesional Broadcasting

El hecho de haber elegido una interfaz Web para la interacción con el usuario implicó agregar a GoalBit la capacidad de realizar un "Professional Broadcast" desde la Web. Esta funcionalidad no existía en GoalBit y fue uno de los objetivos principales del proyecto.

El "web Professional Broadcast", además de ser una funcionalidad nueva, tiene la particularidad de introducir a GoalBit en un nuevo modelo de negocio, con otra clase de usuarios (web broadcasters) que utilizarán el sistema de forma totalmente distinta a la acostumbrada hasta el momento.

Hasta antes de este proyecto, el "Professional Broadcast" era utilizado de forma puntual por clientes para los cuales se instala una infraestructura y se ajustan parámetros de forma manual. Por ejemplo, una empresa podría querer transmitir eventos en vivo por un canal de GoalBit. Para ello debe crearse un canal en la GoalBit Suite y generar los certificados x509 correspondientes al broadcaster del mismo. Con este certificado el broadcaster será capaz de comunicarse con el Controller Server de la Suite. La creación del canal y los ajustes al mismo debían hacerse desde un panel de control de la propia GoalBit Suite por un usuario autorizado que debe crearse para el cliente, típicamente quien crea los canales es el mismo personal de GoalBit.

En el nuevo escenario Web, los broadcasters del sistema serán posiblemente personas que navegan por Internet en lugar de empresas o usuarios puntuales que contratan el servicio. Es esperable que esto tenga influencia directa en la cantidad de usuarios que utilizan el sistema (por lo menos sobre los broadcasters) y sobre la permanencia de los broadcasters en el sistema (es esperable que los usuarios Web realicen broadcast más esporádicos y cortos). También el hecho de que el servicio de broadcasting sea disponible desde Internet expone una parte de GoalBit a la que anteriormente accedían solamente usuarios puntuales, conocidos y autorizados. Esto introduce consideraciones de carga del sistema, rendimiento y seguridad a la hora de realizar el diseño de la funcionalidad.

10.1. Descripción de la solución

Para poder realizar un broadcast desde la Web, un cliente deberá crear una cuenta en un branch de la GoalBit Suite, por ejemplo registrándose en un portal Web como un nuevo usuario. En este caso la Suite creará para el usuario un juego de credenciales que contiene el certificado, la clave privada del usuario y

el certificado de la CA de GoalBit. Estas credenciales son las que luego utilizará el broadcaster del usuario para comunicarse con el controller. En este sentido existe una leve diferencia con el modelo anterior, en el cual las credenciales se asociaban a una señal y no a un usuario. Si bien el portal desarrollado en este proyecto no lo permite, un usuario podría ser responsable de varias señales.

Al momento de realizar un broadcast el portal debe desplegar una página Web en la cual se encuentre embebido el plugin de GoalBit. Como se menciona en la Sección 7, lo anterior significa que el GMP es inicializado al cargarse la página. Desde luego será el GMP quien realice el broadcast y se comunique con el Controller Server de la GoalBit Suite, por lo tanto debe obtener de alguna manera las credenciales de las que se ha hablado recientemente. Esto lo realiza también a través del branch. El branch obtiene de la Suite información útil para el broadcaster y se la hace llegar al GMP pasándola como parámetros de la etiqueta <EMBED>. Entre esta información se encuentra la URL de la API que expone el Controller Server para el broadcaster y una URL de donde este último puede descargar sus credenciales. Estas URLs son generadas dinámicamente por la Suite y contienen tokens asociados al usuario para hacer más difícil el robo de las credenciales. También hay información para el mismo plugin en dicha tag como, por ejemplo, si se trata de un Professional Broadcast, en cuyo caso se crea una instancia de la interfaz Web Professional Broadcaster Manager de GoalBit, de la cual se hablará más adelante.

Debido a que se espera una mayor cantidad de usuarios, lo cual significa una carga mayor en el Controller Server, la URL que se proporciona al broadcaster no es la URL de la api interna del controller como lo era en un principio. En cambio, el broadcaster obtiene una URL de User Services desde la cual se hace el pedido real al Controller Server. Esta indirección permite realizar un balance de carga ya que los User Services actúan como proxy y están diseñados para poder distribuirse en varios servidores.

Una vez que el plugin Web es iniciado por el navegador y el usuario navega hasta la página de emitir, el branch pasa a segundo plano. El resto de las acciones implicadas en el caso de uso de realizar un broadcast por la Web son efectuadas mediante la colaboración del GoalBit Media Player(cliente) y la GoalBit Suite.

En la Figura 41 puede apreciarse la interacción entre los distintos componentes: el *GMP* , la *Goalbit Suite* y el Navegador Web.

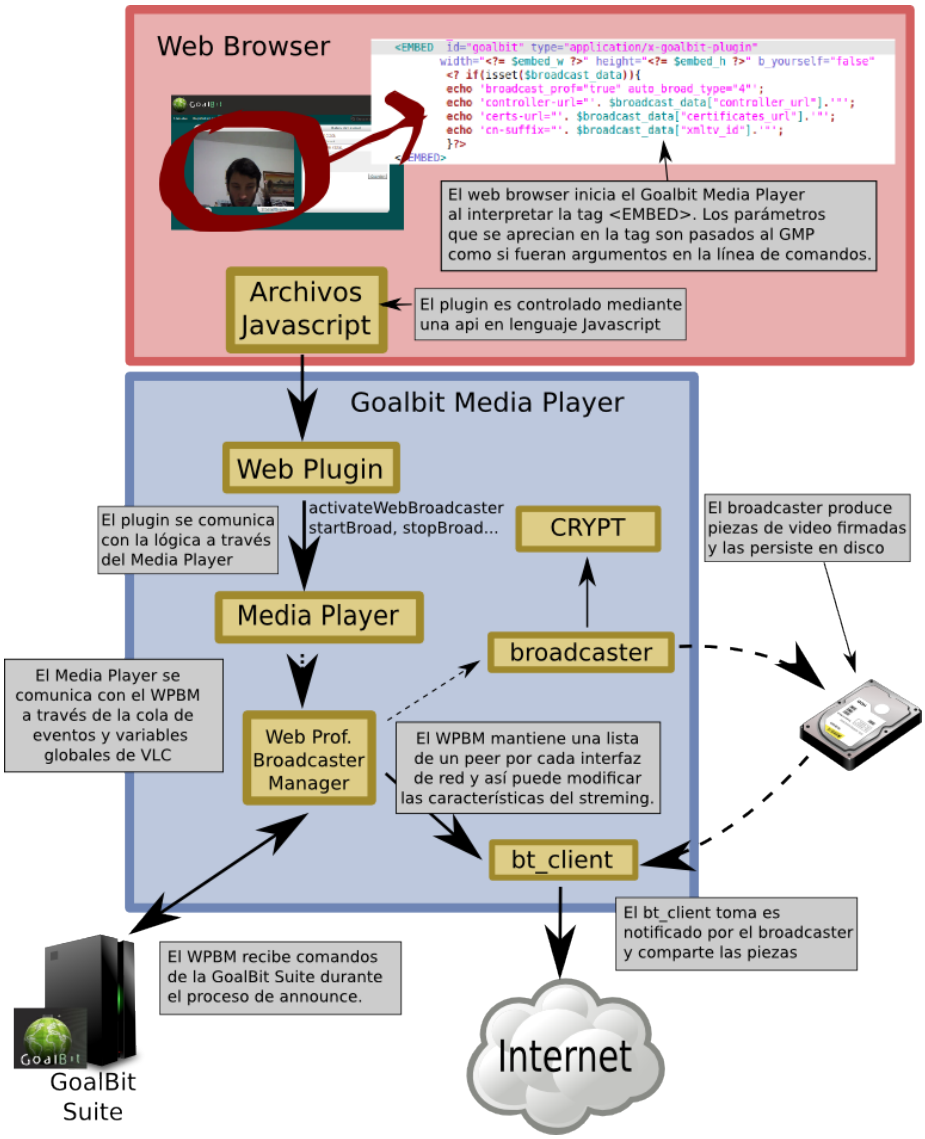

Figura 41: Esquema de los componentes principales del Web Professional Broadcast.

En el GMP intervienen varios módulos a la hora de realizar el broadcast. En primer lugar actúa el plugin, quien es invocado a través de la API en javascript como se describió en la Sección 7. Si bien se incluye dentro de lo que se llama GoalBit Media Player, el plugin podría verse como una capa de presentación que debe invocar las funciones pertenecientes a la capa lógica. Con este fin incluye varios archivos de cabecera que le permiten invocar funciones públicas

de los demás módulos de GoalBit, como por ejemplo el "Media Player" (media_player.c) así como de la misma biblioteca libvlc. El plugin es responsable también del inicio de algunos módulos llamados interfaces, con los que interactuará luego de forma indirecta. Concretamente en el caso del Professional Broadcast se inicia la interfaz "Web Professional Broadcaster Manager". Decimos que el plugin interactúa de forma indirecta con esta interfaz porque el mismo (por lo menos la versión modificada y utilizada en este proyecto) no tiene acceso al manejo interno de eventos de VLC y se comunica con el módulo principalmente a través de llamados a funciones públicas del Media Player. La interfaz Web Professional Broadcaster Manager, a su vez, interactúa con otros módulos para realizar un broadcast. Por ejemplo, debe iniciar los clientes bittorrent por lo que interactúa con la interfaz Bittorrent Manager. También debe inicializar parámetros de seguridad por lo cual se comunica con el módulo GoalBit Crypt (crypt.cpp) y debe dar la orden para que se inicie la transmisión, por lo cual se comunica con el módulo Input, gracias a quien se inicia finalmente un broadcaster (broadcaster.cpp) para comenzar a producir chunks de video.

El caso de uso "Realizar Professional Broadcast" comienza cuando un usuario selecciona la opción "Emitir" en el portal Web y luego presiona el botón de play en el reproductor. Esta acción resulta en el cambio de la variable "web-broadcast-state" al valor WB_PLAY. En la Figura 42 puede apreciarse un diagrama de comunicación de la función "broadcast_play" mediante la cual se realiza el cambio de estado mencionado anteriormente. Este cambio dispara un evento que será atendido por el Web Professional Broadcaster Manager y que resulta en el comienzo de la transmisión. El mismo se detalla en la Figura 42 con la función "BroadcastStateChanged".

En lo que resta de esta sección se describen los tres módulos más importantes de este caso de uso:

- Web Professional Broadcaster Manager
- Media Player
- Broadcaster

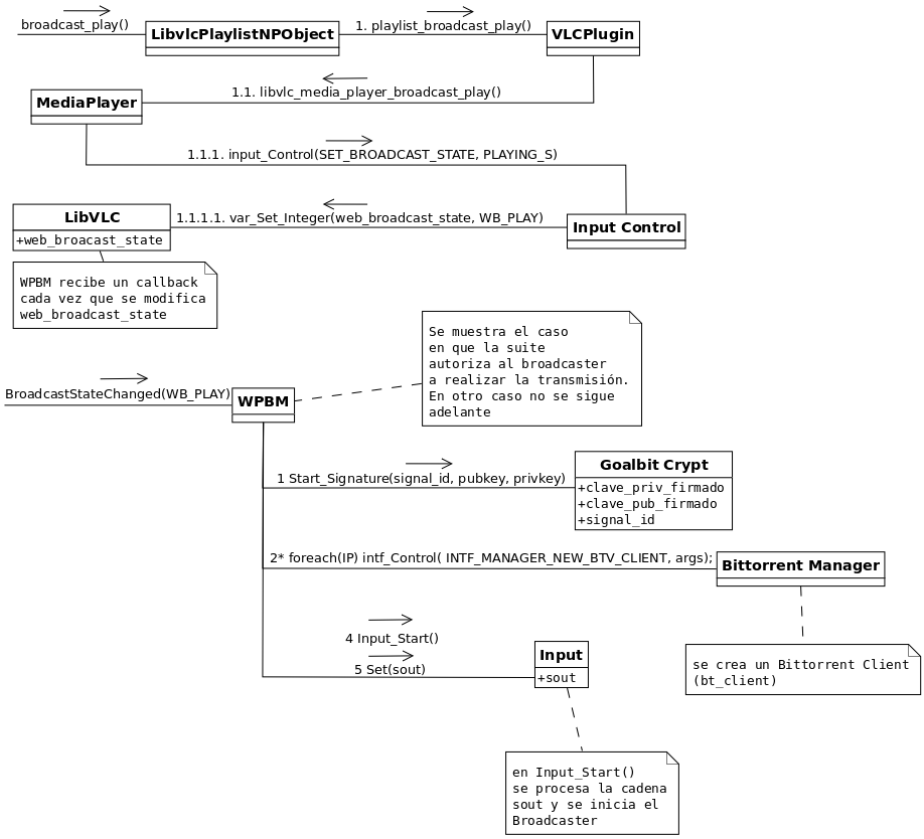

Figura 42: Diagramas de comunicación para el caso de uso "Professional Broadcast". El primer diagrama corresponde a lo que sucede inmediatamente luego de la acción del usuario. El segundo es la atención del evento que termina iniciando el broadcast. Se muestra el caso en que el canal se encuentra habilitado por la Suite.

10.1.1. Web Professional Broadcaster Manager

Como se ha mencionado, al iniciar el plugin se inicia también una interfaz de GoalBit llamada Web Professional Broadcaster Manager. Esta interfaz fue creada para poder realizar el Professional Broadcast desde la Web. Como su nombre lo indica es quien administra todas las etapas del caso de uso colaborando con otros módulos como módulo principal. Aunque no es una tarea simple, la tarea de éste módulo se podría resumir en obtener las credenciales correspondientes al broadcaster, iniciar una conexión segura con el Controller Server de la GoalBit Suite, recibir comandos de éste y ejecutarlos. Adicionalmente debe recibir las señales de play y stop desde el Web Plugin y el Media Player.

Al cargar el plugin el módulo comenzará inhabilitado pudiendo habilitarse mediante un método de la API en javascript. La razón por la cual es inhabilitado

al inicio consiste en que el módulo consulta periódicamente al Controller Server para recibir comandos e información sobre la señal que se desea emitir. Como se desea utilizar el plugin como reproductor de señales en vivo, reproductor de multimedia almacenada localmente y emisor (broadcaster) indistintamente, en caso de que no se deshabilite el Web Professional Broadcaster Manager, éste se estará comunicando con el Controller aún cuando se esté reproduciendo un medio local. Además, es deseable que la interfaz sea iniciada una única vez, ya que su inicialización ocupa un tiempo no despreciable en un computador XO.

WebProfBroadcasterManager es una interfaz de VLC y por lo tanto posee sus métodos básicos:

- Open()
- Close()
- Run()
- Control()

Las funciones de apertura y clausura del módulo (usualmente llamadas Open() y Close()) deben estar definidas en una Sección del módulo llamada descriptor. Esta Sección está delimitada por las macros vlc_module_begin() y vlc_module_end() y en la misma se definen parámetros del módulo que son útiles para que VLC pueda cargarlos. Entre estos parámetros se encuentra el nombre del módulo, descripción, nombre corto, categorización del módulo (por ejemplo CAT_INTERFACE en caso de tratarse de una interfaz), y una declaración de las "capacidades" (capabilities) del módulo. En VLC es posible invocar un módulo pidiendo un módulo con determinadas capacidades mediante la función vlc_module_need(). VLC luego inicia el módulo que mejor cumpla con las capacidades requeridas. Por ejemplo, uno podría requerir un módulo con la capacidad de realizar una comunicación TCP segura, en cuyo caso podría llamar a module_need(...,"tls") y VLC respondería iniciando un módulo que declare dicha capacidad, como por ejemplo el módulo gnutls. En el caso del módulo Web Professional Broadcaster Manager el mismo se inicia directamente mediante la función libvlc_add_intf, por lo que en sus capabilities solo figura la de "Interface".

También deben indicarse en esta Sección cuales son las funciones que VLC debe llamar al inicio y al cierre del módulo. Esto se realiza mediante la función set_callbacks(<Callback de apertura>,<Callback de clausura>) [52].

A continuación se detalla cada uno de estos métodos:

- Open(vlc_object * p_this):
  Esta función es llamada cuando VLC intenta cargar el módulo. En la función Open deberían inicializarse las estructuras internas del módulo así como cargar otros módulos necesarios, realizar chequeos de dispositivos. En caso de que sea exitosa, la función debe retornar el valor VLC_SUCCESS. En cualquier otro caso VLC considera que el módulo no se cargó y muestra el mensaje de error correspondiente.
  El principal trabajo de la función Open del módulo Web Professional

Broadcaster Manager es inicializar una estructura con los datos privados del módulo. Entre otras cosas, se descargan de la Suite las credenciales que necesita el broadcaster para comunicarse con el Controller Server de la misma. La URL de la que cual deben descargarse dichas credenciales es proporcionada por parámetro al iniciar el módulo. Actualmente las credenciales se descargan utilizando el protocolo HTTP pero también se soporta HTTPS.

- Close(vlc_object * p_this):
  Esta función es llamada al descargar el módulo y básicamente es responsable de desinicializar y liberar toda la memoria que fue asignada durante el Open().

- Run(intf_thread_t * p_intf):
  La función Run es la que brinda verdadera funcionalidad al módulo y desde luego es la más compleja. Es invocada una vez que el módulo se encuentra cargado y se ha inicializado correctamente luego del llamado a la función Open.
  Usualmente la responsabilidad de la función Run puede resumirse en recibir la entrada del usuario y realizar acciones en un bucle, consultando regularmente la variable p_intf->p_die, que pasa a valer VLC_TRUE cuando se está intentando finalizar el módulo desde afuera. En este caso mientras no se cumpla esa condición la función Run permanecerá en un bucle. En cada período la función chequea si se han ingresado eventos en la cola de eventos del módulo. Luego, si el módulo se encuentra activo (recordar lo que se decía al principio de esta sección) realizará (entre otras cosas) cada cierto tiempo el proceso de announce con Controller Server del cual se hablará brevemente a continuación.

- Control(vlc_object * p_this, int i_query, va_list args):
  Esta función proporciona otra forma de comunicación entre módulos de VLC/Goalbit. Resulta invocada luego de un llamado a intf_Control(interfaz, valor, args) donde *interfaz* es una estructura de tipo intf_thread que describe al módulo, *valor* es un entero que puede ser utilizado como código de operación dentro de la función Control que resulta invocada y *args* son argumentos que se pueden pasar a la función.
  En la función Control de este módulo se permite que el módulo broadcaster, al iniciarse, pueda registrarse con el Broadcaster Manager. En este caso recibe el código de operación INTF_BROAD_SET_ACCESS y una referencia al módulo Broadcaster la cual apunta desde la estructura privada del Broadcaster Manager.

Controller Announce Process Mediante el proceso de announce el Controller Server es capaz de controlar al broadcaster. El Controller puede permitir o prohibir que un broadcaster transmita un *streaming* y comunicarle parámetros como por ejemplo bitrate, transcoding, claves pública y privada en caso de utilizar firmas digitales, etc.

En el proceso de announce, el broadcaster envía un mensaje al Controller mediante el protocolo HTTPS para luego leer la respuesta de este último. Dicho mensaje contiene un tipo de mensaje (en este caso el mensaje es de tipo "report"), una lista de entradas de tipo IP:Puerto, la MRL del medio que se intenta transmitir y el CN (Common Name) de su certificado, el cual identifica al usuario dentro de la Suite. En la lista de IPs y puertos el broadcaster le comunica al Controller IP y puerto en el que cada peer activo escucha conexiones. También se envían los archivos correspondientes a las capturas que se hacen periódicamente del *streaming* a los efectos de mostrar miniaturas describiendo el contenido de la señal en los branches.
Toda la información mencionada es mandada como parámetro de un mensaje de tipo POST a la URL del Controller que fue obtenida por el branch antes de iniciar el GoalBit Media Player.
Una vez que la Suite procesa el announce, el Broadcaster Manager lee la respuesta de la Suite. Esta respuesta es de la siguiente forma:

```
(comando[^parámetros])*
```

donde comando puede ser broadcaster_stop, interval o broadcaster_start.

- interval:
  Mediante este comando el Controller le indica al Broadcaster Manager cada cuanto tiempo debe reportarse realizando el announce. Como parámetro de este comando es enviado dicho intervalo.

- broadcaster_stop:
  Este mensaje es enviado cuando el broadcaster ha sido deshabilitado desde la Suite y no le es permitido realizar una transmisión. No es acompañado por ningún parámetro.

- broadcaster_start:
  Este mensaje le indica al Broadcaster Manager que está habilitado para realizar una transmisión. En caso de que el usuario ya haya dado la orden de emitir (por ejemplo presionando el botón "play" en el Web Plugin) se comienza con la transmisión inmediatamente después de procesar los parámetros enviados por la Suite. En este caso dichos parámetros son:

  - Transcode
  - Bitrate
  - Muxer
  - URL del GoalBit Access
  - URL del Tracker
  - Bit que indica si debe cifrarse la información a transmitir
  - Bit que indica si debe firmarse la información a transmitir
  - Claves pública y privada de firmado en caso de que el bit anterior tenga valor 1

10.1.2. Media Player

El módulo Media Player (src/control/media_player.c) brinda una API que permite interactuar con el reproductor. Un reproductor (representado por una estructura de tipo libvlc_media_player_t) nuevo es creado para cada medio que se desea reproducir. Dicha API permite actuar sobre el reproductor realizando operaciones como play, stop, seek, etc.

En la arquitectura de GoalBit Media Player este módulo funciona como interfaz entre la capa de presentación y la capa lógica. Al mismo se le agregaron métodos que permiten la comunicación con el Web Professional Broadcaster Manager, el cual es un módulo que se agruparía dentro de la capa lógica. Estos métodos son:

- int libvlc_media_player_activate_web_broadcaster(libvlc_instance):
  Esta función activa el Web Proffesional Broadcaster Manager poniendo en 1 la variable global *web-broadcast-active* tal como ha sido descrito en la Sección anterior.

- int libvlc_media_player_deactivate_web_broadcaster(libvlc_instance):
  Esta función es análoga a la anterior. Desactiva el WPBM poniendo en 0 la variable global *web-broadcast-active*.

- void libvlc_media_player_broadcast_play (media_player):
  Esta función comienza a reproducir el ítem que se encuentra en el Media Player. Es similar a la función libvlc_media_player_play del mismo módulo con la única diferencia que la primera notifica al WPBM de que se ha comenzado la reproducción. Esta notificación se traduce en el WPBM como un evento provocado por el cambio de estado de la variable *web-broadcast-state*. En el caso de libvlc_media_player_broadcast_play dicha variable pasa a tener el valor WB_PLAY.

- void libvlc_media_player_broadcast_stop (media_player):
  Al igual que void libvlc_media_player_broadcast_play, esta función se asemeja a libvlc_media_player_stop. Se diferencia de la misma en que la primera notifica al WPBM de que se desea terminar la transmisión. Esto se hace de forma análoga al caso anterior. En este caso el valor de la variable *web-broadcast-state* pasa a ser WB_STOP.

10.1.3. Broadcaster

El módulo Broadcaster es quien genera los chunks de video, los persiste en el disco duro y notifica a los peers para que éstos se encarguen de distribuir dichas piezas.

El Broadcaster se inicia cuando el GMP procesa la cadena *sout,* en la cual se especifica mediante el parámetro duplicate que la salida del *streaming* será procesada por el mismo. Esto se produce al invocar la función input_Start(input) donde input es el medio que se desea transmitir. Una vez iniciado el Broadcaster se ingresa en loop principal del módulo. La primera vez que se

entra en el loop se ejecuta una función de inicialización. Entre otras cosas esta función "busca" un Broadcaster Manager para registrarse con él. La búsqueda del WPBM se realiza mediante la siguiente llamada:

```
vlc_object_find_name(
    p_access,
    "goalbit_web_prof_broadcaster_manager",
    FIND_ANYWHERE
    )
```

donde p_access es un struct que representa los datos privados del módulo. La función anterior devuelve una referencia a la estructura (struct) que representa al hilo del Web Profesional Broadcaster Manager. Una vez obtenida esta referencia es posible comunicarse con el WPBM a través de intf_Control. Para que el WPBM obtenga una referencia del Broadcaster y pueda acceder al mismo se invoca:

```
intf_Control(
    p_broad_manager,
    INTF_BROAD_SET_ACCESS,
    p_access
    )
```

Mediante esta referencia el WPBM puede comunicarse con el Broadcaster de forma análoga, utilizando la función similar a intf_Control (sout_AccessOutControl). Esto es necesario cuando el WPBM debe verificar si el *streaming* se encuentra atascado (véase la Sub-sección 10.1.1). Para esto realiza la siguiente invocación al Broadcaster:

```
sout_AccessOutControl(
    p_sys->p_broadcaster,
    ACCESS_OUT_LAST_CHUNK_TIME,
    &i_last_chunk_time
    )
```

El firmado de las piezas se realiza obteniendo la instancia de la interfaz *Goalbit Crypt* y realizando la siguiente invocación:

```
intf_Control(
    instancia_goalbit_crypt,
    EVENT_CRYPTO_SIGN_DATA,
    channel_name,
    chunk_id,
    data
    );
```

11. Portal GoalBit-Ceibal

Parte importante de este proyecto consiste en brindar un medio por el cual los usuarios puedan acceder a los nuevos servicios. Particularmente, es necesario disponer de una aplicación que permita al menos realizar una transmisión utilizando el Web plugin, poder localizar esta transmisión de alguna forma y poder consumirla. Dicha aplicación es una aplicación Web y corresponde al Branch en la arquitectura de GoalBit.

Debido a que el producto en este caso está orientado a usuarios en edad escolar, el diseño del Branch debe contemplar diversos factores de usabilidad, además de tener en cuenta las restricciones planteadas por el uso de la XO. Es por eso que se definió al comienzo del proyecto la creación de un Branch específico para Ceibal al cual se le llamó Portal GoalBit Ceibal (PGC).

Las funcionalidades principales del portal se definieron luego de una investigación acerca de las capacidades de la XO en cuanto a realizar una transmisión de distintos medios a través del Web plugin y el análisis de los requerimientos del cliente (Ceibal) referidos a usabilidad y seguridad de la aplicación. Dichas funcionalidades y lineamientos generales para el diseño del portal son:

- Mediante el portal debe poderse transmitir la cámara Web de la XO
- El portal debe ser capaz de buscar canales y reproducirlos mediante el Web plugin de GoalBit
- Debe poder contar con canales auspiciados por Ceibal y destacarlos de alguna forma
- Se debe contar con todas las funcionalidades de un reproductor multimedia de archivos locales
- Debe ser posible denunciar canales con contenido ofensivo y deshabilitar dichos canales desde un panel de control para administradores.
- El producto debe contar con medidas de seguridad que aseguren la autenticidad del contenido reproducido por los usuarios.
- Es deseable tener una interfaz sencilla, minimalista, que tenga en cuenta el tamaño de la pantalla de la XO y no distraiga al usuario con demasiados elementos accesorios.
- Debe minimizarse la información que debe ser manejada y recordada por los usuarios. En especial evitar el manejo de nombres de usuario y contraseñas por parte de los escolares.

El portal ha sido construido teniendo en cuenta los requerimientos anteriores. Desde la perspectiva del usuario el mismo está dividido en dos vistas principales. Una vista se encuentra disponible para todos los usuarios y es la que permite acceder a la mayor parte de la funcionalidad. La otra es un panel de control desde el cual los administradores pueden controlar diversos aspectos de los canales pertenecientes al *branch*.

Herramientas Se utilizó el framework CodeIgniter [3] y PHP5 [19].

CodeIgniter es un framework simple que implementa el modelo MVC (Model View Controller) [58] y reduce el trabajo que implica escribir una aplicación de mediano porte en PHP desde cero.

Como manejador de bases de datos se utilizó MySQL.

Arquitectura El *Portal GoalBit-Ceibal* es una aplicación Web construida en PHP. La misma se desarrolló utilizando el modelo MVC como puede apreciarse en la Figura 43. En dicha imagen se aprecian claramente estos elementos: las clases con prefijo "C_" corresponden a los controladores, los cuales dependen de los modelos cuyo prefijo es "M_". Las vistas no se incluyen en el diagrama ya que las mismas corresponden a la capa de presentación y no son siquiera clases PHP.

La mayor parte de la capa lógica de la aplicación está distribuida en seis controladores principales, a saber:

- *C_Login*: se encarga de gestionar el login de los usuarios al sistema, redirigiendo al usuario a la vista correcta dependiendo de si se trata de un usuario común o un administrador, así como también si se accede desde un PC o una XO. Se encarga también de la creación del usuario en el caso adecuado, tal como será explicado más adelante.

- *C_Admin*: es el encargado de obtener la información que se muestra en el panel de control para administradores y el que permite comunicarse con la Suite para habilitar y deshabilitar canales del *branch*.

- *C_Player*: es el controlador más extenso ya que es el encargado de todo lo referente a cargar la vista principal del portal hacia los usuarios. Como será detallado en secciones subsiguientes, esta vista incluye las funcionalidades de reproductor de señales, reproductor de archivos locales, transmisor y buscador de canales. En cada momento solo una de dichas funcionalidades se encuentra activa mientras que las demás están ocultas. Sin embargo, todas ellas se comunican con el mismo controlador. Así, C_Player se encarga de obtener los datos del canal desde la Suite, crear canales nuevos para aquellos usuarios que no cuentan con uno, realizar la búsqueda de canales, modificar la metadata de los mismos mantenida por el *branch* y administrar las preferencias de los usuarios.

- C_Cron: este controlador se encarga de la sincronización de la información que se encuentra en la Suite con aquella que se encuentra en el *branch*. Contiene una función que es ejecutada periódicamente y consulta acerca de eventos que hayan ocurrido en la Suite, como por ejemplo un cambio en la información de un canal en vivo o la generación de una miniatura nueva para un canal. Luego de obtenida esa información la procesa mediante una operación abstracta que debe ser implementada por sus clases hijas.

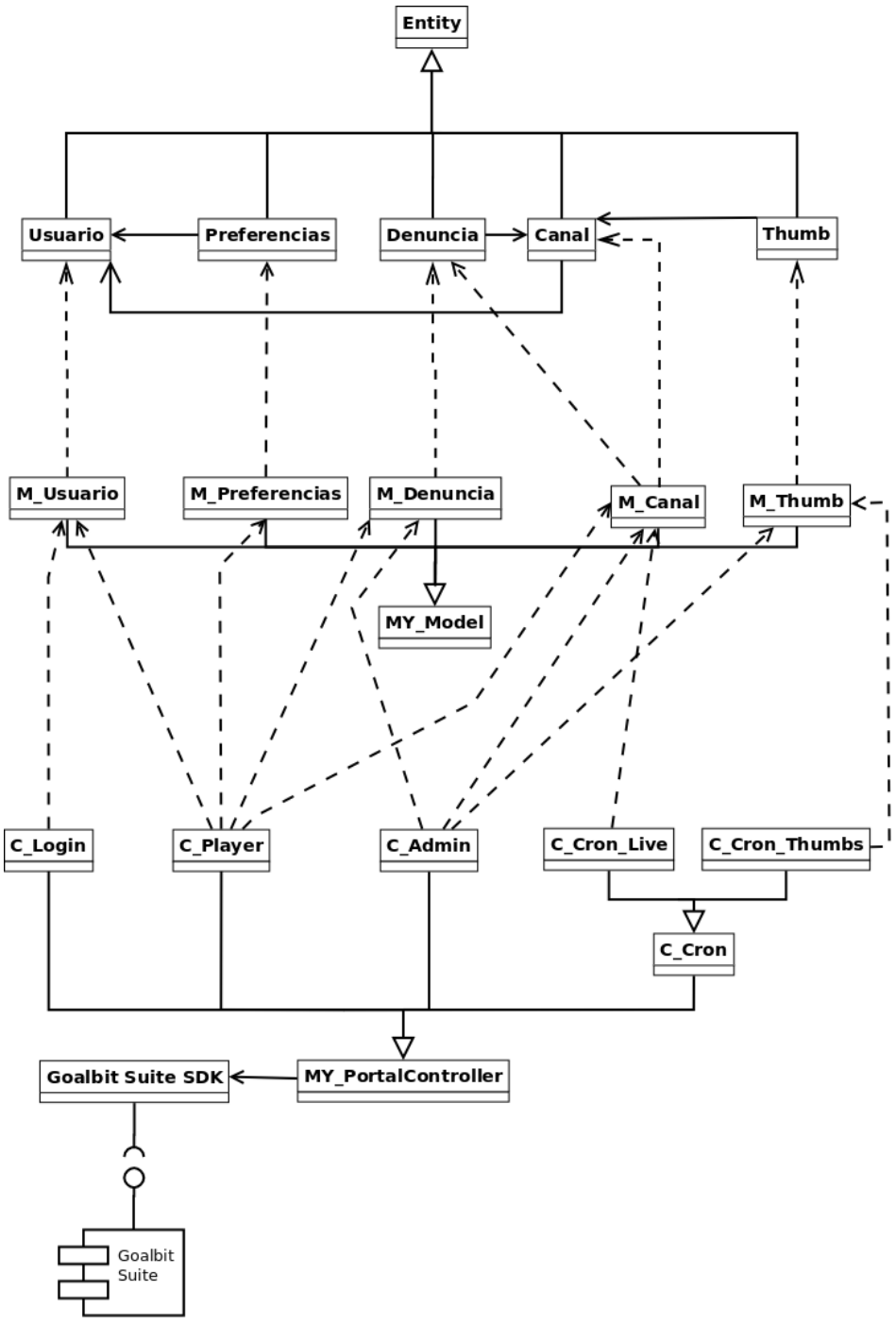

Figura 43: Diagrama de clases del Portal GoalBit-Ceibal

- *C_Cron_Live*: extiende a C_Cron. Contiene una función encargada de procesar la información correspondiente a los canales en vivo que es obtenida de la Suite.
- *C_Cron_Thumbs*: también extiende a C_Cron. Contiene una función encargada de procesar la información correspondiente a las miniaturas de los canales en vivo que es obtenida de la Suite.

Todos los controladores anteriores acceden a la base de datos del portal por algún motivo. Como indica el patrón MVC, los Modelos son quienes implementan el acceso a los datos (consultas) y transforman estos datos en las correspondientes "entidades" pertenecientes al modelo de dominio.

En este caso el dominio está formado por cinco tipos de entidad los cuales son:

- *Usuario*:
  Representa a un usuario del portal. Del mismo se conocen los siguientes atributos:
  - id: Identificador numérico único del usuario. Es autogenerado en la base de datos.
  - login: El nombre de usuario con el que ingresa al portal.
  - password: Contraseña del usuario.
  - create_date: Fecha de creación del usuario
  - is_admin: Bit que indica si el usuario cuenta con privilegios de administrador en el *branch*.
  - Suite_cert_cn: Es el "Common Name" del certificado del usuario. Este es el identificador único del usuario dentro de la Suite.
- *Canal*:
  Representa un canal en vivo. Sus atributos son:
  - Suite_id: Identificador del canal en la Suite.
  - id: Clave candidata del canal en la Suite, correspondiente al campo "XMLTVId" del Canal. Generado desde el portal utilizando la función "uniqid" de PHP.
  - user_id: Identificador del usuario asociado al canal.
  - name: Nombre del canal.
  - description: Descripción del contenido del canal.
  - active: Indica si el canal está habilitado por la Suite para transmitir.
  - broadcasting: Indica si el canal se encuentra en el aire.
  - ceibal: Indica si el canal es un canal auspiciado por Ceibal. En este caso el canal no tiene un usuario Web asociado, por lo cual se ignora el atributo user_id.

- last_update: Fecha y hora en la cual la entidad fue sincronizada con la información de la Suite.

- *Preferencia*:
  Representa una entrada en la base de datos de tipo *atributo:valor* en la tabla de preferencias para un usuario específico. Sus atributos son:

  - id: Identificador de la preferencia. Clave primaria en la tabla preferencias de base de datos del branch.
  - user_id: Identificador del usuario al cual corresponde la preferencia.
  - name: Nombre de la preferencia.
  - valor: Valor de la preferencia.

- *Thumb*:
  Representa una miniatura del canal (pequeña imagen con la captura del canal en cierto instante). Sus atributos son:

  - thumb_id: Identificador numérico único, autogenerado en la base de datos.
  - channel_Suite_id: El identificador del canal al cual corresponde la miniatura.
  - content: El contenido de la imagen. Este contenido es mantenido por la entidad codificado en base64.
  - date: Fecha y hora de generación de la miniatura. Generalmente existen cuatro o cinco miniaturas por canal las cuales se van re-generando a medida que avanza el *streaming*. Cuando llega una nueva en caso de que ya exista un máximo de miniaturas, se remplaza la más antigua.

- *Denuncia*:
  Representa la denuncia de un usuario sobre un canal en caso de que considere su contenido inapropiado por algún motivo. De la denuncia se mantienen los siguientes atributos:

  - id: Clave primaria en la base de datos del branch.
  - user_id: Identificador del usuario que realiza la denuncia.
  - Suite_channel_id: Identificador del canal que es denunciado.
  - date: Fecha de la denuncia.
  - obs: Texto que debe ser utilizado por el usuario para describir los motivos de la denuncia.
  - active: Indica si la denuncia se encuentra activa.

Como se puede observar en la Figura 43, en la mayoría de los casos existe una relación de dependencia "uno a uno" entre un modelo y un tipo de entidad. Únicamente el modelo M_Canal conoce más de una entidad, ya que debe conocer tanto a los canales como a las denuncias a los efectos de poder devolver la cantidad de denuncias que tiene determinado canal. El resto de los modelos se han concebido de forma tal que cada uno se especializa sobre una tabla, conoce la clave primaria de la misma y puede realizar operaciones genéricas conociendo esta información.

Cada modelo es una subclase de My_Model, tal como se aprecia en el diagrama de la Figura 43. A su vez, My_Model es una subclase de Model, quien pertenece al framework CodeIgniter y que por esto ha sido omitida del diagrama. La clase My_Model cuenta con dos atributos que son inicializados al instanciar la misma. Estos atributos son el nombre de la tabla sobre la cual actúa el modelo y el nombre de su clave primaria. Luego se implementaron en My_Model varias operaciones básicas sobre la tabla. Esto permite que los controladores no hagan consultas directas sobre las tablas, abstrayéndose de la estructura interna de la base de datos y manteniendo un buen diseño. Dichas operaciones son:

- protected function afromEntityToDb(Entity $e):
  Operación abstracta que convierte una Entidad en un arreglo asociativo. Debe ser implementada por la subclase.

- protected function afromDbToEntity(array $p_e):
  Operación abstracta que recibe un arreglo asociativo y devuelve una entidad. Debe ser implementada por la subclase, devolviendo el tipo de entidad al cual está asociado el modelo.

- protected function fromArrayDbToArrayEntity(array $resultset):
  Convierte un arreglo de arreglos asociativos en un arreglo de entidades.

- protected function getDateToDb(DateTime $date):
  Convierte una fecha de tipo DateTime al formato utilizado en la base de datos.

- protected function getDateFromDb($dbdate):
  Devuelve una fecha de tipo DateTime a partir de una fecha en el formato utilizado en la base de datos.

- protected function getByWhat($key,$val):
  Esta operación obtiene de la tabla correspondiente al modelo aquellos registros cuyo atributo $key tenga el valor $value. Es decir, si t fuera la tabla del modelo, la consulta sobre la base de datos sería similar a "select * from t where $key = $value". Devuelve un arreglo de entidades del tipo de entidad asociado al modelo.

- public function deleteById($id):
  Permite eliminar registros de la tabla correspondiente al modelo especificando su valor de clave primaria.

- public function create(Entity $e):
  Crea un registro en la tabla del modelo a partir de una entidad.
- public function update(Entity $e):
  Actualiza los valores correspondientes a una entidad en la base de datos.

Las operaciones create($e) y update($e) reciben entidades como parámetro y deben convertirlas a información manejable por la base de datos. A priori, es imposible saber en la clase MyModel que tipo de entidad manejará la clase cuando sea instanciada. Para resolver el problema de forma elegante se ha aplicado el patrón de diseño "template method". Dichas funciones invocan a la operación abstracta afromEntityToDb de la que se ha hablado recientemente. Cada subclase de MyModel operará de forma distinta según el tipo de entidad que se corresponda con el modelo (es decir, según su implementación de la operación afromEntityToDb). Sin embargo, la estructura de las operaciones update y create es siempre la misma y está definida en la superclase. Las operaciones deleteById() y getByWhat() también utilizan este patrón de diseño.

11.1. Vista hacia el usuario

A lo largo de esta Sub-sección se hablará de la parte del portal que es visible para los usuarios "comunes" (distinguiéndolos de aquellos con privilegios de administrador).

Las funcionalidades brindadas al usuario son:

- Login.
- Reproductor multimedia de archivos locales.
- Buscador de canales.
- Reproductor de una señal en vivo.
- Trasmisión de señal en vivo.

Debido a las diferencias sustanciales entre los tamaños y resoluciones de pantalla de la XO respecto a los habituales, cada pantalla cuenta con dos estilos distintos, uno para una PC convencional y otra para una XO.

Personalizando la vista según el tipo de usuario Como fue mencionado en la introducción, es deseable que el portal se muestre de forma distinta en una XO y en un PC convencional, considerando las diferencias en el tamaño de la pantalla de los anteriores. Para esto, primero debe ser posible distinguir un usuario de otro.

Para cada tipo de usuario se tiene una URL diferente. La URL de ingreso al portal para un usuario de una PC convencional es de la forma "http://dominio/subdominio/main/pc", mientras que para una XO es de la forma "http://dominio/subdominio/main/xo". No se utilizó el campo *User-Agent* del cabezal HTTP con el que se envía el pedido para ingresar al portal, ya que el campo *User-Agent*

que envía el Navegador Web de la XO y la Actividad GoalBit (ver Sección 8) no es distinguible del de un navegador Mozilla Firefox convencional de la misma versión. Los usuarios Ceibal entran automáticamente al portal con la URL correcta por medio de la Actividad Goalbit 8. [14]. La diferencia básica que tiene una vista y otra es que para las PCs convencionales se les agrega un hoja de estilo extra (".../css/pc/style.css") que sobreescribe algunas entradas de la hoja de estilo de la XO.

Una sola instancia del plugin GoalBit en el portal Al cargar una página que usa al plugin de GoalBit se crea una instancia del GMP. Dado que esto es costoso para el cliente, se decidió mantener la misma instancia del plugin para todas las pantallas de usuario. A excepción del Login, todas las funcionalidades del portal pueden ser accedidas sin hacer un *GET* de otra página Web. Para lograr esto se hizo una gran pantalla que contempla a todas estas funcionalidades (Reproductor multimedia de archivos locales, Reproductor de una señal en vivo, Trasmisión de una señal en vivo y el Buscador de Canales), a estas funcionalidades les llamaremos "Funcionalidades del Reproductor". Cuando el usuario va navegando entre las distintas funcionalidades, se le ocultan ciertas secciones de la pantalla, se le muestran otras y para hacer pedidos al servidor Web se usa Ajax[67]. De esta manera se pudo lograr tener una página Web dinámica sin necesidad de refrescarla y tener que estar creando y destruyendo instancias del GMP.

11.1.1. Login

Esta es la pantalla inicial del portal, en la que se le pide al usuario que ingrese su usuario y contraseña para ingresar al portal. Esta pantalla no está orientada a ser usada por el escolar que utiliza la XO puesto que la actividad GoalBit autentica al usuario con el portal automáticamente, como se explico en la Sección 8.

Actualmente cuando el usuario que se ingresa no existe, al mismo se lo crea con la contraseña especificada. En un futuro, en el caso de que el usuario especificado no exista, se debería validar si la IP origen corresponde a una IP de Ceibal, y en ese caso crearlo como en este momento y de lo contrario mostrar un mensaje de error. Adicionalmente se tendría una forma de crear nuevos usuarios.

Una vez que el usuario fué autenticado se lo redirige al Reproductor Local.

Implementación Para la implementación de esta funcionalidad se utilizó la vista *Login* (login.php), el controlador *C_Login* (login.php) y el modelo *M_Usuario* (m_user.php).

Como podemos ver en la Figura 45, cuando el usuario no existe, se lo crea. Luego para el usuario recién creado o para el que ya se tenía, se controla que tenga un certificado de usuario (campo "Suite_cert_cn" de la entidad *Usuario*)

[14] El cabezal HTTP *User-Agent* que envía la actividad GoalBit es "Mozilla/5.0 (X11; U; Linux i586; en-US; rv:1.9.1.9) Gecko/20080724 Firefox/3.0".

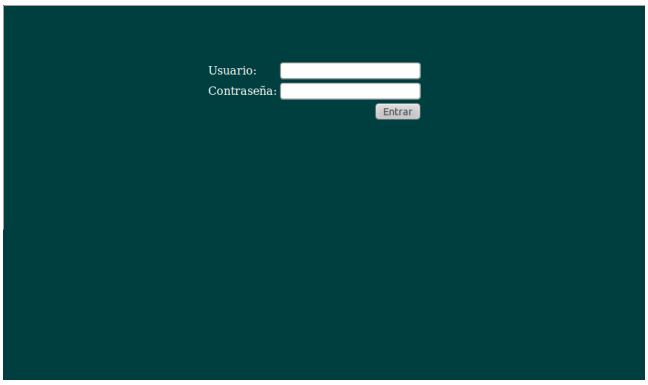

Figura 44: Página de ingreso al portal GoalBit-Ceibal

creado en la GoalBit Suite. Si no se tiene dicho certificado porque éste es nuevo o fue eliminado, se lo crea en la Suite y se guarda el "cert-cn" en el campo "Suite_cert_cn" del *Usuario*.

Una vez que el usuario es autenticado se graba en la sesión Web la entidad *Usuario* correspondiente y se lo redirige al reproductor local.

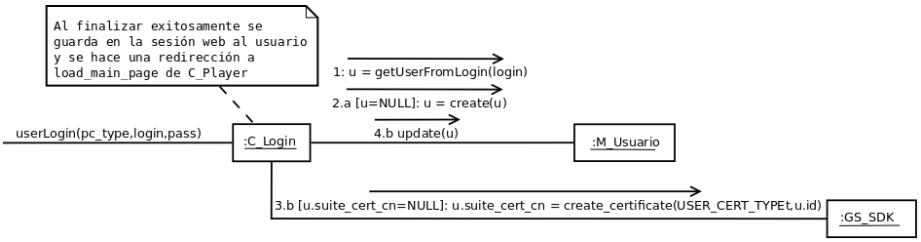

Figura 45: Diagrama de comunicación del login y creación de usuarios en el portal GoalBit-Ceibal.

La función del controlador *M_Player* a la que se invoca luego del login es "load_main_page". Esta función recibe por parámetro el tipo de interfaz deseada "xo" o "pc", la cual por defecto es "pc". En ésta se valida que el usuario este "logueado" a partir de la sesión Web y luego se controla que éste tenga un canal creado en la GoalBit *Suite*, y si no lo tuviera se le crea uno.

Para cada entidad *Canal* del portal se tiene su correspondiente canal en la GoalBit *Suite*. Luego si es que el canal del usuario está habilitado por la Suite (campo "active" de la entidad *Canal*) entonces se le pide a la GoalBit *Suite* los datos necesarios ("controller_url" y "certificates_url") para la vista "main" por medio de la función de la SDK "get_web_broadcast_data". En el caso de que el usuario no tenga permitido realizar *streaming* multimedia no se le muestra la opción "Emitir" en el menú de la vista principal. Luego se obtiene desde las preferencias del usuario la ruta al último directorio seleccionado como lista de

reproducción. Finalmente se muestra la vista "main.php" (.../views/main.php).

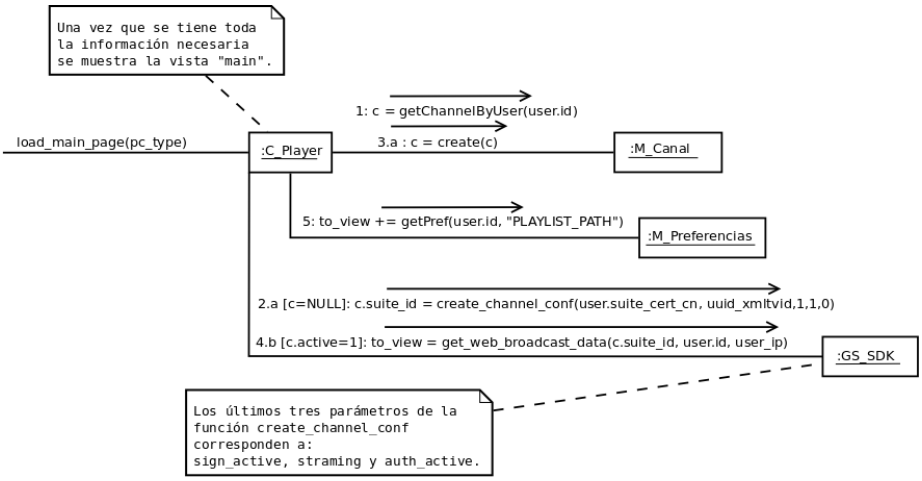

Figura 46: Diagrama de comunicación que representa la creación de un nuevo canal y obtención de datos para la pantalla principal del portal.

Creación de un canal en la GoalBit Suite Para la creación de un nuevo canal por medio de la función "create_channel_conf" de la SDK de la *Goalbit Suite* se deben especificar los siguientes parámetros:

- *certificate_cn*, correspondiente al certificado de usuario que se tiene para el usuario que esta logueado.
- *xmltv_id*, es un identificador único en la *GoalBit Suite* para cada Canal. Para este parámetro se utiliza la función nativa de PHP "uniqid", que genera un UUId. Éste es un dato que se conserva en la entidad *Canal* del portal.
- *sign_active*, este parámetro fué agregado en este proyecto junto con la implementación del firmado como se explica en la Sección de Seguridad, Sub-sección 9.3 de la solución. Este tiene un valor por defecto '0', que indica que el firmado esta desactivado, para ganar compatibilidad con la versión del SDK actual. Para este portal todos los canales son con firmado activo y por esto es que siempre lo especificamos en '1'.
- *streaming*, este parámetro fué agregado en este proyecto y luego es grabado directo en la tabla de Canales. El mismo indica que se permite hacer streaming multimedia por medio de este canal, y por defecto es '0' por la misma razón que el parámetro anterior. Para este portal todo canal que se crea tiene al principio permiso para hacer streaming multimedia por lo que su valor es siempre especificado como '1'.

- *auth_ active*, este parámetro fué agregado en este proyecto y luego es grabado directo en la tabla de Canales. El mismo indica que la URL de acceso a los ".goalbit" de una señal tiene un token diferente por cada usuario distinto que lo quiere ver, haciendo esta validación se intenta hacer canales públicos y privados. Para el caso de este portal todos los canales son públicos y por defecto se creaban privados. Para asegurar compatibilidad hacia atrás como el caso del parámetro "sign_ active" se lo agrego con un valor por defecto '1' y desde el portal se especifica siempre un '0'.

Al agregar estos nuevos parámetros se tuvo que modificar además de la SDK, a la *GoalBit Suite.*

Se modificó al controlador "live_ content" (.../application/controller/api/-controllers/live_ content.php), para que procesara los nuevos parámetros y para que adicionalmente retorne el identificador del canal creado, en la función correspondiente. También se modificó el modelo "live_ content_ model" (.../application/controller/api/models/live_ content_ model.php) en la función "create_ signal".

11.1.2. Reproductor multimedia de archivos locales

Esta funcionalidad es la misma que se le brinda a los usuarios que usan la actividad GoalBit en modo "sin conexión", pero con la capacidad de recordar el último directorio seleccionado del selector de archivos para la lista de reproducción. Este directorio seleccionado es recordado como una preferencia de usuario.

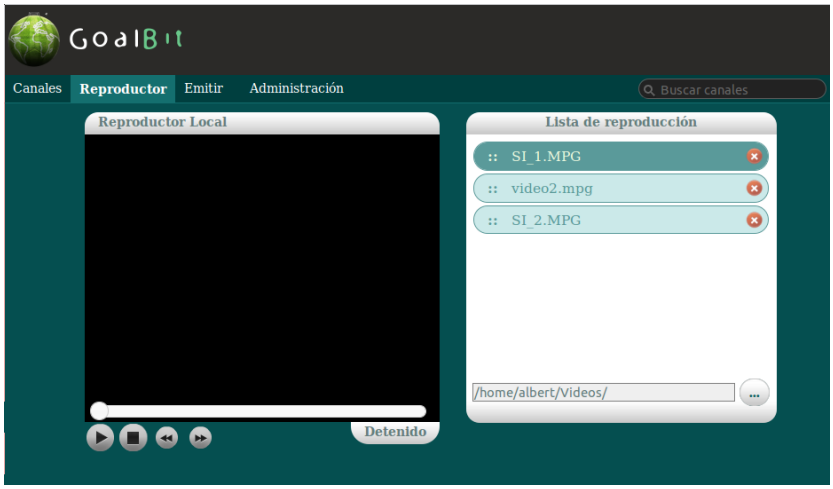

Figura 47: Página del reproductor multimedia de archivos locales del portal GoalBit-Ceibal

Implementación Como se explicó en la Sub-sección anterior esta vista se muestra luego del Login. Para recordar el último directorio seleccionado como lista de reproducción, cada vez que se selecciona un directorio como lista de reproducción se hace un llamado Ajax con método POST al servidor Web desde javascript. Se invoca una función expuesta por el portal llamada "setUserPreferences" con los parámetros "name" (nombre de la preferencia) y "value" (valor), que en este caso corresponden a "PLAYLIST_PATH" y el directorio seleccionado respectivamente.

11.1.3. Buscador de canales

Por medio de esta funcionalidad se permite buscar canales activos, es decir a los que se les tiene permitida la emisión de contenido multimedia, se encuentran haciendo una transmisión en ese momento y que además cumplan con el filtro por nombre. Como se puede apreciar en la Figura 48, se tienen dos tipos bien distinguidos de canales, los canales Ceibal y los canales de usuario llamados "XOs". Para buscar por nombre se digita donde dice "Buscar canales" y se presiona la tecla <*Enter*>, esto hace que se actualicen ambas listas de canales aplicándoles el mismo filtro por nombre. Ambas listas están páginadas, la lista de canales Ceibal solo permite mostrar cuatro canales por página, mientras que las XOs se muestran ocho por página. La información que se muestra para cada canal corresponde a una miniatura (Thumb[15]) y debajo su nombre. Finalmente cuando se desea ver el canal correspondiente se da un "click" sobre el *thumb* correspondiente del canal deseado y se pasa la pantalla del "Reproductor de una señal en vivo" Sub-sección 11.1.4.

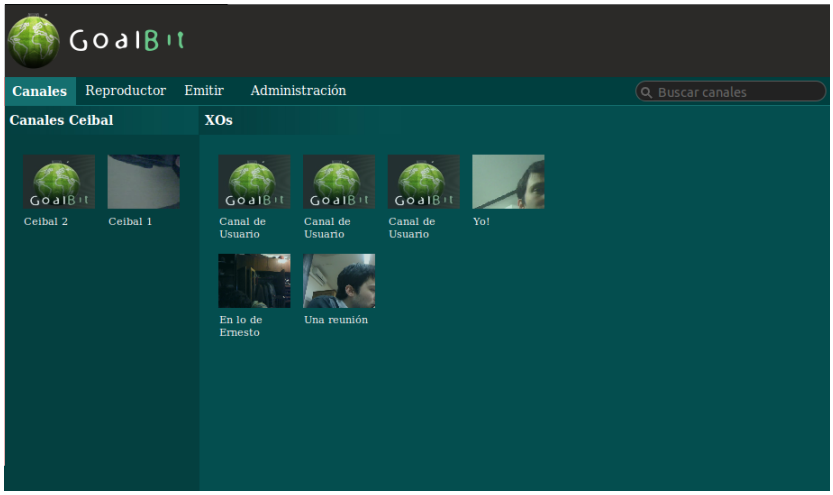

Figura 48: Página del buscador de canales del portal GoalBit-Ceibal

[15] El Thumb del canal que se muestra puede ser una foto tomada de lo que se esta emitiendo por el GMP o un ícono por defecto para los canales que no tienen contenido emitido.

Implementación Para la implementación de esta funcionalidad se utilizó la vista *Main* (main.php), así como lo hacen todas las Funcionalidades del Reproductor y adicionalmente a la vista *Search_ Channels* (search_channels.php) que es la que muestra el resultado de la búsqueda. El controlador *C_ Player* (player.php) y los modelos *M_ Canal* (m_channel.php) y *M_ Thumb* (m_channel_thumb.php). Cuando el usuario decide hacer la búsqueda de canales, puede ir a la pestaña "Canales" que se puede ver en la Figura 48 o digitar el nombre del canal en la entrada de texto que se encuentra en la esquina superior derecha. Para cualquiera de los dos casos hace un llamado Ajax con método POST al servidor Web desde javascript. Invocando a la función expuesta por el portal llamada "find_channels" con los parámetros "busqueda", "page" y "page_ceibal", que en este caso corresponden a la busqueda que se dígitó en el campo de texto, la página deseada de los canales de XOs que al inicio es "1" y la página de canales Ceibal que al inicio es "1" también.

Como se muestra en la Figura 49, por cada canal de la página se obtiene un *Thumb* al azar entre los 5 que se mantienen por canal (ver Sub-sección 11.3). Este *Thumb* se mantiene almacenado en base de datos, entidad *Thumb* y campo *content*, como texto en formato base-64 (ref Base-64). Las imágenes que se muestran en la vista *Search_ Channels* son directamente en base-64, y es por esto que el atributo *src* de la tag HTML <img> es de la forma "..." donde "..." es donde va el contenido de la imagen. De esta forma en un solo pedido POST se trae todo el contenido HTML sin necesidad por lo tanto de hacer un *GET* por cada imagen miniatura que se quiere mostrar. Finalmente una vez que se tiene el HTML asociado a la vista *Search_ Channels* se la muestra.

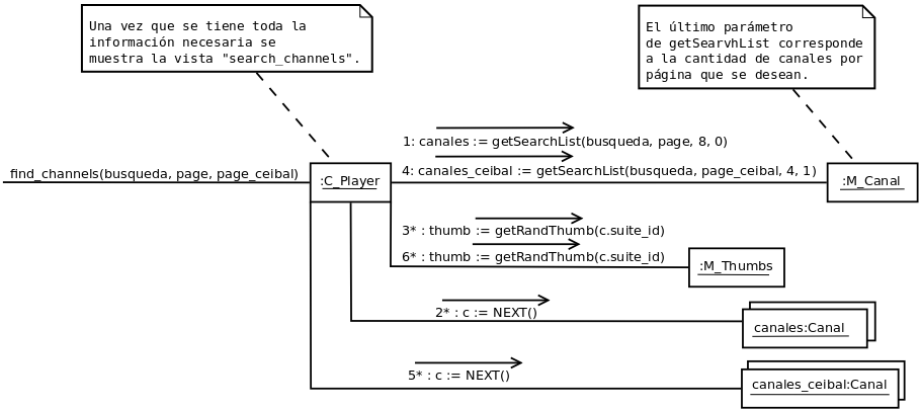

Figura 49: Diagrama de comunicación que representa la búsqueda de canales en el portal.

11.1.4. Reproductor de una señal en vivo

Para la reproducción de una señal en vivo se debe previamente buscar el canal mediante la funcionalidad anterior, y luego presionar click sobre la imagen que

representa este canal. Una vez que se seleccionó el canal se lo puede reproducir o parar y adicionalmente indicar si el mismo es ofensivo, indicando el motivo, para que luego desde un panel de administrador se decida que hacer con éste.

Figura 50: Reproductor de una señal en vivo del portal GoalBit-Ceibal.

Implementación Para la implementación de esta funcionalidad se utilizó la vista *Main* (main.php), así como lo hacen todas las Funcionalidades del Reproductor, el controlador *C_Player* (player.php) y el modelo *M_Canal* (m_channel.php). Una vez que se hace click sobre la imagen correspondiente al canal que se desea ver, se hace por javascript una invocación Ajax a la función "streaming_url" que luego en el servidor se mapea a la función "get_streaming_url" del controlador *C_Player* del portal, de la cual se ilustra en la Figura 51. Una vez que se obtiene la *URL* asociada al ".goalbit" se lo agrega a la lista de reproducción:

```
goalbit.playlist.add("gbtp://"+response.url);
```

Para que luego al hacer "play" sobre ésta (la lista de reproducción) se reproduzca el canal que se encuentra siendo trasmitido.

11.1.5. Trasmisión de señal en vivo

Por medio de esta funcionalidad se le permite al usuario trasmitir la cámara Web de la PC. Una imagen de esta funcionalidad puede apreciarse en la Figura 52. Solo se podrá trasmitir por medio de este canal si el mismo se encuentra autorizado (ver Sub-sección 11.2). Si el canal no estuviera autorizado a transmitir el usuario correspondiente no tendrá la opción "Emitir" en el menú, si al

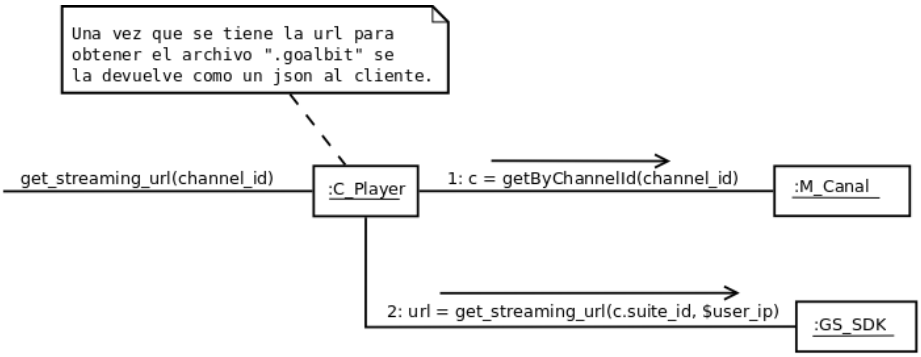

Figura 51: Diagrama de comunicación que representa la obtención de una URL válida para obtener el ".gaolbit" asociado al canal deseado.

mismo se lo desautoriza en medio de una transmisión ésta será suspendida automáticamente en a lo sumo lo que demore el cliente GMP tarde en reportarse al *Controller Server* de la *Goalbit Suite*.

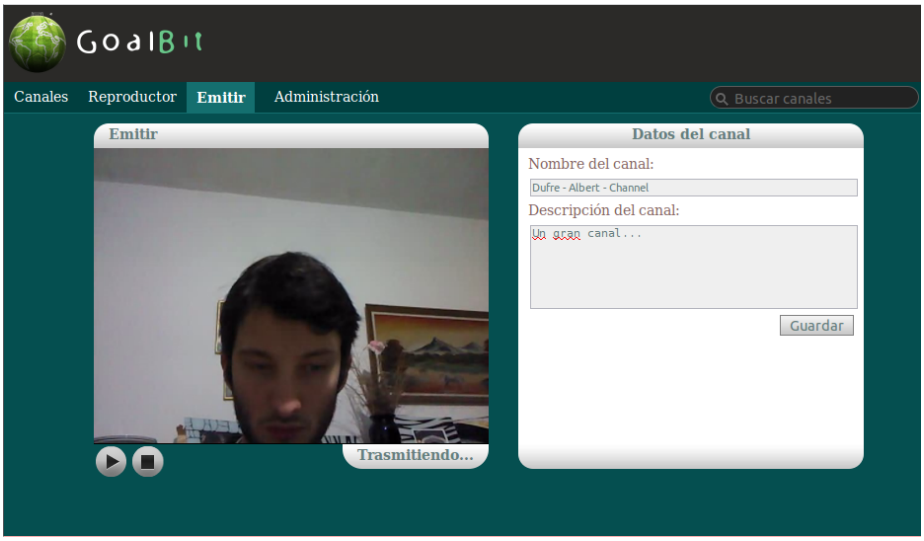

Figura 52: Transmisión de una señal en vivo del portal GoalBit-Ceibal.

Implementación Para la implementación de esta funcionalidad se utilizó la vista *Main* (main.php), así como otras funcionalidades, el controlador *C_ Player* (player.php) y el modelo *M_ Canal* (m_channel.php).

Una vez que se selecciona esta funcionalidad se agrega a la lista de reproducción la cámara Web. Para agregar esta fuente multimedia se procede de la

misma forma que se muestra en la Sub-sección 7.3.1, referente a la especificación de la fuente multimedia que se desea transmitir en el GMP edición Web Plugin.

```
goalbit.playlist.add(
    “v4l2://”,null,
    ":sout=#transcode{...}
    :duplicate{dst=display, dst=qoe_injector_duplicate{ dst=std{
    access=goalbit_broadcaster{...}}}"
);
```

Posteriormente se activa al *Web Professional Broadcasting* invocando a la función del plugin “activateWebBroadcast”.

```
goalbit.activateWebBroadcast();
```

Esto implica que el módulo correspondiente al *Web Profesional Broadcasting* comience a anunciarse con el *Controller Server* de la *Goalbit Suite* como se especifica en la Sección 10.

De esta forma cuando se de “play” y además se haya autorizado por parte del *Controller Server* al *GMP*, se comenzara a trasmitir la señal en vivo.

Además cada vez que se deja esta funcionalidad se le invoca la función “deactivateWebBroadcast” al plugin web, para que el *GMP* deje de anunciarse con el *Controller Server*.

```
goalbit.deactivateWebBroadcast();
```

11.2. Panel de control para administradores

Debido a que era una necesidad del cliente Ceibal que se tuviera un panel de administración del portal que contara al menos con la posibilidad de ver las denuncias de los canales que fueron considerados ofensivos por algún motivo y adicionalmente permitir desactivar a los mismos o cualquiera que se desee; se creo este panel para administradores.

Este panel esta compuesto básicamente por una única pantalla, que le permite ver todos lo canales del portal, estén activos o no y estén trasmitiendo o no, pudiendo filtrarlos por nombre de canal.

Desde este panel es posible:

- Ver las denuncias de cada canal.
- Eliminar las denuncias que se desee de un canal.
- Activar y Desactivar el canal.

Para ver las denuncias de un canal se debe hacer “click” sobre la imagen que aparece bajo la columna “Denuncias” de cada canal como se puede apreciar en la Figura 53.

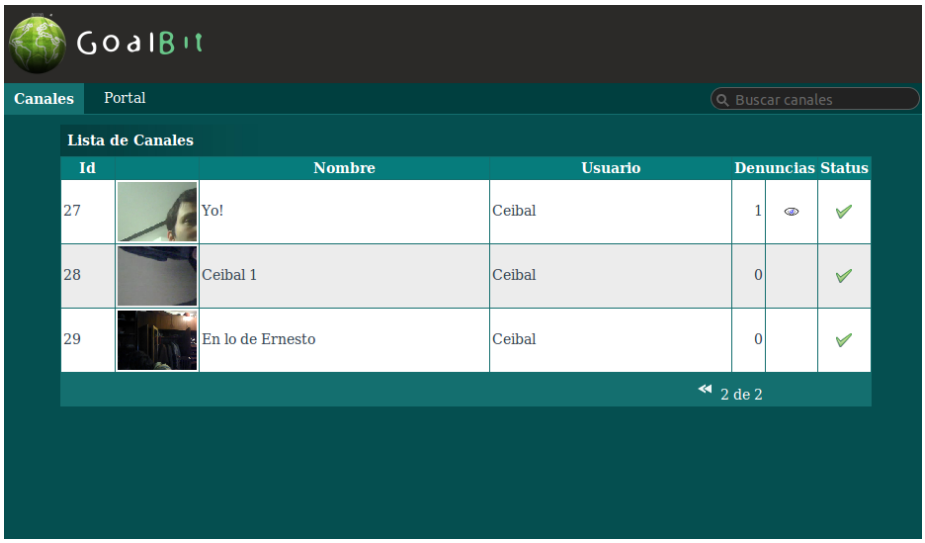

Figura 53: Panel de administración del portal GoalBit-Ceibal.

Para activar o desactiva un canal se debe hacer "click" sobre la imagen que esta bajo la columna "Status" de cada canal. Esto implicará que cuando se desactive el canal se le comunique a la *Goalbit Suite* que dicho canal no esta más autorizado para trasmitir una señal en vivo y será marcado como desactivado en el portal. De esta manera si el GMP asociado al canal correspondiente estuviera trasmitiendo se detendrá, quien lo estuviera consumiendo perderá la señal y además el mismo no será mostrado en las búsquedas de canales desde el Portal hacia los usuarios.

11.3. Sincronización de Información con la Goalbit Suite

Debido a que la comunicación entre el la *Goalbit Suite* y el branch es costosa (debe utilizarse ssl para la comunicación y puede existir una única *Suite* para varios branches) es necesario mantener información de la *Goalbit Suite* en el *branch* con el fin de responder las consultas acerca de los canales en un tiempo razonable. En esta Sub-sección se explica como se lleva a cabo la sincronización de la información entre la *GS* y el *Branch*.

La información que es sincronizada de la *Goalbit Suite* para el portal es:

- Datos del canal, principalmente los campos "active" ("broadcaster_streaming" en la *GS*) y "broadcasting" ("broadcaster_active" en la *GS*) de la entidad *Canal*, que indican si a un canal se le permite trasmitir y el otro si un GMP esta trasmitiendo en este canal respectivamente.

- *Thumbs* o Fotos miniatura de los canales.

La sincronización de la información se hace por medio de tareas programadas que ejecutan periódicamente. Para cada tipo de dato diferente que se sincroniza se tiene una tarea diferente, de esta forma es posible darle diferente importancia a la sincronización de un tipo de datos respecto al otro. En el caso del portal es más importante sincronizar los datos del canal que los thumbs asociados al mismo, dado que es más importante poner el canal cuanto antes en la lista de canales trasmitiendo que ver su miniatura actualizada.

11.3.1. Eventos del Backend en la GoalBit Suite

La *GoalBit Suite,* como se dijo en la Sección 1.3.3, se encarga de los aspectos relacionados con la transmisión de contenido. En la realidad que modelaba la *GoalBit Suite*, en donde se consideraban una cantidad acotada y controlada de señales en vivo, no era necesaria una forma eficiente de sincronización de la información de las señales hacia los branches por lo que la estrategia consistía simplemente en sincronizar la información referente a todos los canales cada cierto tiempo. La mayor cantidad de señales en vivo que se espera se manejen en este *branch* hace necesaria la implementación de un método para hacer más eficiente la sincronización de la información entre el *branch* y la *Goalbit Suite*.

En este proyecto se diseñó e implementó una solución por medio de la cual los *branches* pueden solicitar el tipo de información deseado sin recibir información redundante, devolviéndose sólo los datos que hayan cambiado desde la última consulta. De esta forma baja la cantidad de tráfico de datos entre ambos sistemas y por ende aumenta el rendimiento del branch en la actualización.

Solución La solución en este caso se encuentra dividida en dos partes. Primero debe detectarse y procesarse adecuadamente la información que ha de sincronizarse con el *branch*. Esta parte de la solución es implementada en la *Goalbit Suite*. Luego el *branch* debe ser capaz de recuperar esta información desde la *Suite* cada cierto período de tiempo para lo cual la *Suite* debe exponer un servicio que debe ser consumido por el *branch*.

Generación de información asociada a eventos ocurridos Se agrega la tabla "backend_event" en una de la base de datos que maneja la *GoalBit Suite*. La estructura de esta tabla es:

- *branch_id*, identificador numérico del branch, de esta manera se clasifica la información por branch.
- *ref_id*, identificador numérico de recurso según tipo de evento.
- *ref_type*, identificador de tipo texto, correspondiente al tipo de recurso o evento sucedido. Los tipos de recursos que se usan en este proyecto son: "live" y "live_thumb". Los tipos de recurso estan abiertos a ser del tipo deseado sin restricciones.
- *ticket*, valor numérico que que se incrementa para indicar el orden en que se dieron los eventos.

- *action*, valor de tipo texto que representa la acción que se hizo sobre el recurso. Posibles valores para tipos de recursos "live" y "live_thumb" son ["add", "del", "mod"]. Estas acciones son dependientes del tipo de recurso o evento sucedido.

La clave primaria de esta tabla es la tripleta ["branch_id", "ref_id", "ref_type"].

Todo evento que sea de interés para el branch se encuentra registrado en esta tabla. Para mantener actualizada la misma, en el caso de los tipos de eventos "live" y "live_thumb" se utilizaron *Triggers* (Disparadores) [17]. Para cada par "ref_type" y "action" se crea un nuevo Trigger sobre la tabla correspondiente. Como ejemplo se presenta un pseudo código del trigger activado después de una modificación en la tabla goalbit_channel_conf:

```
Para cada registro modificado
   si se modificó el campo ''broadcaster_active''
           o ''broadcaster_streaming'' o ''xmltv_channel_id''

       nuevo_ticket = siguiente_ticket()
       INSERT INTO backend_event


    (branch_id, ref_id, ref_type, ticket, action)
           SELECT
              branch_live_content.branch_id,
              NEW.goalbit_channel_conf_id,
              'live',
              nuevo_ticket,
              'mod'
           FROM
              branch_live_content
           WHERE
              branch_live_content.goalbit_channel_conf_id
                 =
              NEW.goalbit_channel_conf_id
        ON DUPLICATE KEY UPDATE
           ticket = nuevo_ticket;


    fin si
fin para
```

Para el caso del *Trigger* anterior, solo se insertan nuevos eventos en la tabla "backend_event" si se modifica el campo "broadcaster_active", "broadcaster_streaming" o "xmltv_channel_id" de un registro en la tabla "goalbit_channel_conf" (tabla de canales de la *GoalBit Suite*). Se inserta un evento por cada branch que tenga asignado dicho canal en la tabla "branch_live_content" (tabla que vincula canales con los branches).

Como puede apreciarse el campo "ticket" en la tabla no es autonumerado y sustituye a valores viejos sobre la tabla cuando una tupla con la misma clave primaria esta presente, de esta forma no se entrega un historial de eventos a los branches cada vez que solicitan información actualizada de los recursos sino que se devuelve solo la información que interesa, que es el estado actual de cada recurso.

Consulta por los eventos ocurridos Se agrega la función "get_backend_events" en la *API* de la *GoalBit Suite* y el *SDK*:

```
get_backend_events( ticket, cant_max = 0, event_types = array() ) : Result;
```

El retorno de esta función es un arreglo asociativo con la información asociada a los eventos ocurridos posteriormente a la última consulta; y recibe por parámetro a:

- *ticket*, un valor numérico mayor o igual a cero, que indica cual fue la última notificación que se ha recibido de la *Goalbit Suite*. De esta manera es posible que sean retornados solo los eventos ocurridos posteriormente a la última notificación. Dentro del retorno de esta función esta el próximo "ticket" con el que se debe consultar Result["next_ticket"], para no obtener información antes consultada.
- *cant_max*, un valor numérico indicando la cantidad máxima de eventos nuevos que se desea recibir para procesar cada vez.
- *event_types*, este corresponde a un arreglo de "string", correspondiente a los eventos que se desea consultar. Los eventos considerados para este proyecto son: "live" y "live_thumb", como se dijo anteriormente.

Algo que se puede destacar es que si el *branch* perdiera la información consultada, puede actualizarse consultando con el mismo "ticket", dado que la información una vez consultada por el *branch* no se elimina. La información consultada con el mismo *ticket* puede no ser la misma debido a que ésta fue actualizada. Esta solución no intenta mantener un historial de lo ocurrido sino que tiene por objetivo mantener actualizados a los distintos *branches* con la información que tiene la *Goalbit Suite*.

11.3.2. Tareas programadas para la actualización del portal

Para mantener actualizado al portal con la información asociada a los Canales, se utilizó *Cron* [66], dado que se implementó para un ambiente Unix.

Cron es un manejador de tareas programadas basadas en tiempo, que existe en sistemas operativos basados en Unix. Permite a los usuarios programar tareas para que sean ejecutadas periódicamente o en una fecha particular, típicamente utilizados para automatizar el mantenimiento de sistemas o administración.

Los scripts que son ejecutados por los crons estan bajo el directorio "crons" del portal Web y corresponden a "run_cron_live.sh" y "run_cron_live_thumb.sh". Un ejemplo de como configurarlos para un sistema operativo Ubuntu es el siguiente:

```
*/1 * * * * [portal]/crons/run_cron_live.sh
*/2 * * * * [portal]/crons/run_cron_live_thumb.sh
```

En el ejemplo anterior se ejecuta cada un minuto la actualización de la información de los canales y cada dos minutos a los *thumbs*.

El script "run_cron_live.sh" simplemente ejecuta, por medio de la herramienta PHP, al script "update_channel_states.php" y éste invoca a la función "index" del controlador *C_Cron_Live*. El script "run_cron_live_thumb.sh" es análogo e invoca a la función "index" del controlador *C_Cron_Thumbs*.

11.3.3. El controlador C_Cron

El controlador *C_Cron* implementa el patrón de diseño *OO Template Method* [40] en donde el "AbstractClass" es *C_Cron*, las "ConreteClass" son *C_Cron_Live* y *C_Cron_Thumbs* y el "TemplateMethod" es "run".

La firma de la función run tiene la siguiente firma:

```
void run(ticket_name, event_types).
```

La función "run" recibe por parámetro una arreglo de "string" correspondiente a los tipos de eventos que se desea consultar a la *Goalbit Suite* y el nombre del parámetro de configuración correspondiente al "ticket" de esta consulta. Un pseudo código de esta función se detalla a continuación:

```
run (ticket_name, event_types)

    ticket_val = obtener_param_conf(ticket_name);
    response = get_backend_events(ticket_val, CANT_MAX_RESULTS, event_types);
    process_response( response[“resultset”] );
    actualizar_param_conf( ticket_name, result[“next_ticket”]);
    si response[“num_rows”] >= CANT_MAX_RESULTS
       //existen más eventos que consultar
        run (ticket_name, event_types)
    fin_si

fin
```

La función "process_response" es una función abstracta de la clase *C_Cron* y es la encargada de procesar la información de los tipos de eventos que corresponda.

C_Cron_Live Esta clase hereda de *C_Cron*, posee la función "index" que es la que se ejecuta periódicamente gracias a los crons como se ha explicado al comienzo de esta Sub-sección. La función "index" simplemente invoca a la función "Template" "run":

```
run ("SUITE_TICKET_LIVE", array("live"))
```

Implementa la función "process_response" procesando la información retornada al consultar por los eventos de tipo "live", correspondientes a modificaciones de canales en la *GoalBit Suite*. Los posibles eventos pueden indicar que se creo un nuevo canal en la *GoalBit Suite*, que se eliminó o se modificó algún atributo de este. Tanto para el caso de eliminación de canales en la *GoalBit Suite* o modificaciones se procede a eliminar o modificar los canales de la base de datos del portal según corresponda.

Un caso particular se da ante la notificación de eventos de nuevos canales, cuando se recibe un evento de este tipo, y el canal en el portal no existe significa que el mismo fue creado utilizando la *Goalbit Suite*, y a estos se los cataloga como "Canal Ceibal", creándolos de este tipo. La clasificación de canales puede verse en Sub-Sección 11.1.3 correspondiente a la funcionalidad del portal que permite buscar canales en el portal para los usuarios.

C_Cron_Thumb Esta clase hereda de *C_* Cron, análogamente a *C_Cron_Live*. La invocación a la función "run" es la siguiente:

```
run ("SUITE_TICKET_LIVE_THUMB", array("live_thumb"))
```

En la función "process_response", se procesa la información asociada al tipo de evento "live_thumb", correspondiente a nuevas miniaturas de canales.

Las imágenes recibidas de la *Goalbit Suite* son en formato base-64 y de un tamaño mucho mayor al que se usa en el portal, por esta razón se reducen a un tamaño más acorde al uso que se les da en el portal. De esta forma además se ahorra espacio en la base de datos y mejoran los tiempos de transmisión de información y renderización de las imágenes en el navegador Web del cliente.

Dado que las distintas vistas que muestran las imágenes lo hacen con un base-64 embebido directamente en el HTML, éstas se almacenan también en base-64, optimizando así el procesamiento de datos si estas se guardaran como un campo de tipo *BLOB*.

12. Evaluación de la solución

En las siguientes secciones, se detallarán las prueen este *branch*bas realizadas sobre la solución.

Primeramente se presentan las pruebas funcionales realizadas sobre el Portal Goalbit-Ceibal y la Actividad Goalbit. Luego se presentan pruebas de *performance*, en donde se evaluará el uso del sistema en distintos escenarios, generando

un *streaming* de video con firmado activo, por medio de la Actividad y el Portal Web y consumiéndolo con las XO de una red local.

También se evaluará la eficacia del sistema de firmado, mostrando cómo un ataque que era exitoso ante un *streaming* sin firmado deja de serlo al activar esta característica.

Las pruebas que evalúan la solución implementada fueron realizadas en el ambiente de desarrollo. Para las mismas se contó con dos computadoras de escritorio con las siguientes características:

- Procesador Intel Core i3-330M (2.13GHz)
- 3MB de Cache L2
- Bus de 1066MHz
- 4GB RAM DDR3.

Y con tres XO, prestadas por el cliente Ceibal [20] para su uso durante este proyecto, las cuales tienen instalada la actividad GoalBit. Como servidor web se utilizó una de las PCs de escritorio y como clientes a las cinco PCs.

A las PC de escritorio nos referiremos más adelante como PC-1 y PC-2, mientras que a las XO les llamaremos XO-1, XO-2 y XO-3.

12.1. Pruebas funcionales

12.1.1. Prueba de conectividad

En la siguiente prueba se busca validar que cuando una XO no tenga conectividad para poder hacer uso del portal Web, se le presente el reproductor local con el que cuenta la actividad Goalbit.

Procedimiento Dada una XO, se verifica que no tenga acceso a Intenet y luego se inicia la actividad GoalBit.

Resultado esperado La actividad se inicia, se muestra el reproductor local.

Resultado obtenido El resultado de la prueba fue exitoso.

12.1.2. Pruebas sobre reproductor multimedia

En la siguiente prueba se busca probar la funcionalidad referente al reproductor multimedia, tanto el local como el del portal Web.

Procedimiento Dada una XO, se inicia la actividad GoalBit. Una vez cargada la página Web del portal GoalBit-Ceibal:

1. Se escogen los elementos que van a componer la lista de reproducción, por medio del selector de archivos.
2. Se presiona el botón de reproducir (*play*).
3. Se espera un minuto.
4. Se presiona el botón de parar (*stop*).

Resultado esperado La Actividad se inicia, automáticamente se accede al Portal GoalBit-Ceibal y se muestra el reproductor multimedia del portal.

Son agregados los elementos seleccionados a la lista de reproducción. Luego cuando se presiona el botón *play* comienza a reproducirse el primer elemento de la lista de reproducción.

Una vez que se presiona el botón *stop* se detiene la reproducción.

Resultado obtenido El resultado de la prueba fue exitoso.

12.1.3. Pruebas del buscador de canales

Procedimiento Dada una XO, se inicia la actividad GoalBit. Una vez cargada la página web del portal GoalBit-Ceibal:

1. Se accede al buscador de canales.
2. Se navega entre las páginas de canales encontradas.
3. Se filtran los canales con la palabra "canal".
4. Se navega entre las páginas de canales filtrados.

Resultado esperado Dados los canales con los que cuenta el portal para realizar pruebas se esperan los siguientes resultados:

- Una vez que se accede a la funcionalidad del buscador de canales y antes de filtrarlos, se espera encontrar 3 canales ceibal y 3 páginas de canales “XOs” totalizando 19 canales de tipo “XOs”.
- Al filtrar los canales con la palabra "canal" se espera que no hayan canales ceibal que cumplan con el filtro y 2 páginas de canales “XOs” totalizando 9 canales de tipo “XOs” que contienen dicha palabra en su nombre.

Resultado obtenido El resultado de la prueba fue exitoso.

12.2. Pruebas de performance

El objetivo de estas pruebas es evaluar el rendimiento del sistema en distintos escenarios. El objetivo del sistema es poder realizar una transmisión que sea recibida por cierto número de computadoras XO. Por esta razón, interesa comprobar si el *streaming* se recibe correctamente en todos los clientes.

Se registrará el tiempo que demora cada cliente en comenzar a visualizar el *streaming*. En caso de cortarse el video debido a un *buffering* se registrará este evento, junto con cuanto tiempo tarda en estabilizarse nuevamente (tiempo de recuperación).

Para evaluar la performance de la solución se considerará el tiempo que transcurre hasta comenzar a visualizarse el video *(latencia)*, la cantidad de *bufferings* ocurridos durante la prueba y el tiempo de visualización de video sobre el tiempo total de la prueba (% visualización).

Los resultados obtenidos por las computadoras XO se compararán con los resultados obtenidos por las PC de escritorio.

12.2.1. Escenario 1a

Objetivo: Transmisión del contenido capturado por la cámara Web de la XO dentro de una red local.

Descripción

Broadcaster: XO-1

Peers: XO-2, XO-3

Topología: Todos los pc se encuentran dentro de una red local inalámbrica

Resultados:

	XO-1	XO-2
latencia (s)	15	21
bufferings	0	0
% visualización	100	100
tiempo de rec. promedio (s)	-	-
duración de la prueba (s)	600	

12.2.2. Escenario 1b

Descripción

Broadcaster: XO-1

Peers: PC-1, PC-2

Topología: Todos los pc se encuentran dentro de una red local inalámbrica

Resultados:

	PC-1	PC-2
latencia (s)	11	(*)
bufferings	0	0
% visualización	100	100
tiempo de rec. promedio (s)	-	-
duración de la prueba (s)	600	

(*) Por la PC-2 se esperó por unos 30 segundos, luego se hizo "stop" y "play" inmediatamente, y en un segundo comenzo a reproducir normalmente.

12.2.3. Escenario 2a

Objetivo: Consumir un *streaming* Goalbit desde Internet dentro de una red local.

Descripción

Broadcaster: TNU

Peers: XO-1, XO-2, XO-3

Topología: Todos los clientes se encuentran dentro de una red local inalámbrica mientras que el broadcaster se encuentra en Internet.

Resultados:

	XO-1	XO-2	XO-3
latencia (s)	32	37	43
bufferings	0	0	0
% visualización	100	100	100
tiempo de rec. promedio (s)	-	-	-
duración de la prueba (s)	900		

12.2.4. Escenario 2b

Descripción

Broadcaster: TNU

Peers: PC-1, PC-2, PC-3

Topología: Todos los clientes se encuentran dentro de una red local inalámbrica mientras que el broadcaster se encuentra en Internet.

Resultados:

	PC-1	PC-2	PC-3
latencia (s)	35	32	40
bufferings	0	0	0
% visualización	100	100	100
tiempo de rec. promedio (s)	-	-	-
duración de la prueba (s)	600		

12.2.5. Escenario 3a

Objetivo: Consumir un *streaming* generado por la cámara Web de una XO desde Internet, por los clientes dentro de una red local.

Descripción

Broadcaster: XO-1

Peers: XO-2, XO-3

Topología: XO-1 dispone de una IP pública en Internet y el resto de los PC se encuentra en una red local inalámbrica.

Resultados:

	XO-1	XO-2
latencia (s)	19	17
bufferings	0	0
% visualización	100	100
tiempo de rec. promedio (s)	-	-
duración de la prueba (s)	600	

12.2.6. Escenario 3b

Descripción

Broadcaster: XO-1

Peers: PC-2, PC-1

Topología: XO-1 dispone de una IP pública en Internet y el resto de los PC se encuentra en una red local inalámbrica.

Resultados:

	PC-1	PC-2
latencia (s)	16	15
bufferings	0	0
% visualización	100	100
tiempo de rec. promedio (s)	-	-
duración de la prueba (s)	600	

12.3. Pruebas del firmado de piezas

Procedimiento

Primera parte (sin firmado activo)

1. Se comienza la transmisión del contenido que se desea atacar.
2. Se inicia un cliente que consuma el *streaming.*

3. Se inicia la transmisión del *streaming* malicioso con un ABI superior al del primer *streaming*. Para lograr esto sin alterar el código del Goalbit Media Player atacante, se comienza previamente una transmisión en el mismo canal y se deja pasar un tiempo para que el ABI aumente.

4. Se inicia un nuevo cliente o se reinicia el primero (presionando play y luego stop).

Segunda parte

1. Se activa el firmado para el *streaming* legítimo.

2. Se inicia un cliente que consuma el *streaming* o se reinicia uno (presionando play y luego stop).

Resultado esperado

Primera parte Se espera que el cliente del paso 2 consuma el *streaming* legítimo sin firmado. Cuando el cliente se reinicia debería comenzar a consumir el *streaming* malicioso ya que el mismo tiene un ABI superior al del primero.

Segunda parte Se espera que el cliente no pueda consumir el *streaming* malicioso ya que el mismo no se encuentra firmado.

Resultado obtenido

Primera parte El resultado obtenido es el esperado, el cliente en el paso 2 consume el *streaming* legítimo y al reiniciarse comienza a reproducir el *streaming* generado por el atacante.

Segunda parte El resultado es el esperado, dado que no se consume el *streaming* del atacante. Sin embargo, se observó que el cliente tampoco reproduce el *streaming* original. Esto se debe a que el *Tracker*, a la hora de indicarle al nuevo peer en qué ABI debe comenzar a descargar piezas, le retorna el mayor ID de *chunk* registrado para ese canal. De esta manera el ID del *chunk* con el cual el nuevo peer comienza es el correspondiente al ABI del atacante.

Parte IV
Conclusiones Generales y Trabajo a futuro

13. Conclusiones Generales

El objetivo general del proyecto era construir un producto utilizando GoalBit que permitiera a un escolar reproducir archivos multimedia, consumir y generar *streaming* de video en tiempo real y en forma segura, desde su XO. Se puede concluir que estos objetivos fueron cumplidos con éxito.

Para alcanzar los diferentes objetivos planteados se trabajó en cuatro productos de software:

- *GoalBit Media Player.*
- Actividad GoalBit.
- *GoalBit Suite.*
- Portal Web o Branch.

A la *GoalBit Media Player* se le extendió, creando un nuevo módulo encargado de manejar el "Web Professional Broadcasting" que es quien interactúa con la *Goalbit Suite* en forma segura. Y para poder comandar al *GoalBit Media Player* desde la Web se extendió al Web plugin de éste, agregando la capacidad de *broadcasting*. Adicionalmente, para poder autenticar las piezas de video que son compartidas en la red, evitando ataques de imitación (impersonation attack) y la inyección de contenido indebido, se agregó un esquema de firmas digitales utilizando el algoritmo digital de firmado *DSA* [45].

El sistema de firmado evita los ataques mencionados. Sin embargo el sistema es vulnerable a ataques de tipo DoS (Denial of Service) a una señal, inhabilitando que los usuarios finales puedan consumirla. Un atacante podría inyectar tráfico ingresando previamente al tracker, y anunciándose como broadcaster de esa señal y con un ABI superior al original. De esta manera, la víctima que entra encuentra al ABI más alto e intenta consumir ese *streaming*, gracias al firmado digital ninguna de las piezas recibidas es reproducida porque no son válidas, pero de todas formas la víctima queda imposibilitada de ver la señal requerida porque sigue considerando que el ABI del atacante es el correcto.

Para lograr que los escolares tuvieran en su XO un reproductor multimedia, que además les permitiera interactuar en portales Web en donde se utilice el plugin Web de Goalbit se creó la Actividad GoalBit. Esta actividad esta basada en el navegador Web con el que cuentan actualmente las XO. La actividad incluye una versión del *GMP* compilado para la arquitectura de las XO, con

el plugin Web de GoalBit (el cual se instala automaticamente al iniciar por primera vez la actividad) y cuenta además con una interfaz Web que agrega valor a este producto. Por medio de esta interfaz Web se les brida la funcionalidad de un reproductor multimedia de archivos locales y un medio para ingresar sin necesidad de especificar credenciales de tipo "usuario" y "contraseña" al portal GoalBit-Ceibal creado especialmente para este proyecto.

Al sub-sistema *Goalbit Suite* se le realizaron ciertas modificaciones para adaptarlo a la nueva realidad que a partir de ahora soporta. La realidad anterior consistía en tener cierta cantidad controlada de señales en vivo, con *broadcasters* configurados en forma manual emitiendo casi constantemente, mientras que ahora se manejan potencialmente miles de canales creándose a diario, en donde los *broadcasters* aparecen de forma esporádica y dejan la red a demanda de cada usuario final. Entre otras modificaciones se agrego el manejo de los certificados asociados a la firma digital de las piezas de video de cada señal y una mejora muy importante en cuanto a la forma de sincronización de datos entre éste subsistema y el componente "Branch" considerado en la arquitectura de *GoalBit*.

Se diseñó e implementó un Portal Web, que representa al *Branch* en la arquitectura de *GoalBit*, en donde los escolares pueden buscar contenidos, reproducirlos y transmitirse a ellos mismos utilizando la cámara Web de la XO. Incluye un sistema que permite denunciar y dar de baja canales con contenido indebido desde un Panel de Control para administradores.

14. Trabajo a futuro

En esta sección se expondrán los lineamientos básicos por los cuales se podría continuar el proyecto para mejorar la solución que se ha alcanzado.

Actividad Goalbit Si bien se ha logrado una versión final de la actividad Goalbit, la cual cumple con los requerimientos acordados con el usuario responsable Ceibal representado por el Sr. Daniel Castelo, queda pendiente incluir en el compilado del *GMP* la biblioteca libx264 para poder emitir utilizando el codec de video x264[18] que permite manejar streams en formato H.264/MPEG-4 AVC.

Descarga de las credenciales del broadcaster profesional por SSL. Actualmente las credenciales correspondientes al broadcaster se descargan de la *GoalBit Suite* mediante el protocolo HTTP. Esto significa una vulnerabilidad en el sistema ya que dichas credenciales pueden ser interceptadas y utilizadas al ser transmitidas en texto plano. Esto se resolvería utilizando HTTPS y un certificado público autorizado por las CAs en que confían los principales navegadores.

Evitar ataque de DoS (Denial of Service) Como se ha visto en este documento, es posible efectuar un ataque de negación de servicio a un canal. Esto se da en el caso de que el tracker sea público y permita ingresar a un otro par del mismo canal que ofrezca piezas de video maliciosas (chunks) con un ABI más alto que las correspondientes al flujo real.

Para evitar esto es necesario aplicar modificaciones en el protocolo *GBTP* (GoalBit Transport Protocol) que permitan detectar estos casos y penalizar al peer malicioso permitiendo a los otro peers volver a descargar piezas con el ABI correcto.

Portal Web Si bien el portal cumple con las funcionalidades acordadas podrían agregarse nuevas características hacia los usuarios, así como también mejorar la parte administrativa del portal.

Maximizar la eficiencia del sistema en el escenario del Plan Ceibal Trabajar en conjunto con el grupo de seguridad del Plan Ceibal para definir una configuración de topología de red que maximice la eficiencia del sistema. Entendiéndose por eficiencia que los estudiantes de una misma escuela descarguen la mayor cantidad de *straming* entre ellos y no desde los servidores centrales, ahorrando ancho de banda de la escuela.

Parte V
Apéndices

A. Detalles de implementación del firmado

Aquí se especifican con más detalle las modificaciones hechas sobre cada módulo para utilizar el sistema de firmas digitales.

A.1. Módulo encryption.cpp

newPublicPrivateKeyDSA Esta función genera un par de claves DSA (pública y privada). Recibe como parámetro el largo de la clave, el cual es uno de los parámetros que dan una idea acerca de la seguridad del algoritmo. En este caso el largo de la clave utilizada es 1024 bytes. En el estándar original era el máximo largo permitido aunque ahora se recomiendan largos mayores para claves que no cambiarán en mucho tiempo.

El par de claves es generado a través de la función gcry_pk_genkey de la biblioteca gcrypt. Esta función recibe una s-expression de la forma (genkey (dsa (nbits n:l)(transient-key))) donde n es la cantidad de cifras de l y l es el largo de la clave. El parámetro transient-key permite utilizar un generador de números aleatorios más rápido (y de alguna forma menos seguro).

La s-expression es generada a partir de un string similar al descrito anteriormente mediante la función gcry_sexp_build. La s-expression generada se pasa como parámetro a gcry_pk_genkey, quien devuelve otra s-expression que contiene el par de claves DSA, las cuales luego se extraen de la s-expression y se convierten a un string mediante gcry_sexp_find_token y gcry_sexp_sprint.

signChunk La función se llama signChunk, siguiendo la nomenclatura de funciones ya existentes como por ejemplo encryptChunk y encryptChunkSymmetric, pero su propósito es generar una firma digital para ciertos datos cuya longitud se proporciona como parámetro de la función al igual que la clave privada que se empleará en el algoritmo de firmado. La función devuelve una referencia a un buffer que contiene la firma digital la cual consiste en dos números (r y s) en formato hexadecimal representado como string concatenados.

La función consta de seis pasos. Primero debe generarse una s-expression con la clave que es pasada por parámetro en formato string. Luego se genera un hash sha1 de los datos pasados como parámetro. Ese hash es convertido posteriormente a un formato numérico utilizado por libgcrypt (y muchas otras aplicaciones que trabajan con números grandes) llamado MPI (multi-precision-integer). Una vez hecho esto se genera otra s-expression con este número (que representa los datos a firmar). Una vez que se cuenta con ambas s-expressions se llama a la función gcry_pk_sign, la cual devuelve otra s-expression con la firma digital de los datos. A los efectos de manipular la firma cómodamente, se extraen los números r y s que forman la firma, se convierten a un string y

se concatenan para formar un único string de 80 caracteres (representando una firma de 40 bytes en hexadecimal). Este string es el que se copiará dentro del header del chunk en un espacio de 80 bytes reservado a tales efectos.

verifyChunk El proceso previo al verificado de la firma es similar al descrito en el caso de la función signChunk. En este caso, la función recibe por parámetro un string con la clave pública en formato de s-expression, los datos y la firma tal como es extraída del header del chunk. Se debe, entonces, recuperar los valores r y s de la firma, computar un hash sha1 de los datos, convertir este hash a un MPI, elaborar dos s-expressions con el MPI y la clave pública y, por último, llamar a la función gcry_pk_verify, quien devolverá un error en caso de que la firma sea inválida para los datos proporcionados.

B. Procedimiento de instalación del branch Goalbit-Ceibal

1. Instalar CodeIgniter y MySQL
   Es necesario instalar el framework PHP CodeIgniter y el manejador de bases de datos MySQL.
   CodeIgniter se encuentra disponible en: http://codeigniter.com/download.php
   MySQL se puede obtener desde http://www.mysql.com/downloads/mysql/

2. Instalar la base de datos del branch

   a) Modificar el script "database/dump.sql" con el nombre del esquema deseado.

   b) Debe correrse el script en database/dmp.sql indicando el nombre del esquema. Este script crea la base de datos y las tablas correspondientes en MySQL.

   ```
   mysql -u<username> -p<password> < dump.sql
   ```

3. Copiar el branch al servidor web Apache (por ejemplo /var/www/).

4. Modificar .htaccess con el directorio del branch.

5. Agregar el directorio virtual del branch al Apache.

6. Reiniciar el Apache.

7. Modificar configuración de CodeIgniter en el portal

   a) En _system_config.php definir el nombre del branch de la siguiente forma:

   ```
   DEFINE( 'BRANCH_ID','nombre_branch')
   ```

b) En config.php

1) $config['base_url'] = "/url_protal/";
2) Modificar configuración referente al acceso a los servicios de la *Goalbit Suite* y el portal:

```
DEFINE('RUTA_PORTAL','ruta_portal');
DEFINE('URL_SUITE','url_Suite');
```

c) En index.php

1) Modificar la ruta al directorio "system" de CodeIgniter en la variable $system_folder. Ejemplo:

```
$system_folder = "/var/www/codeigniter/system";
```

2) Modificar la ruta a la aplicación en la variable $application_folder. Ejemplo:

```
$application_folder = "application";
```

d) Asegurarse de que en "config/database.php" se apunta a la base de datos correcta.

```
//Portal
$db['pceibal_goalbit']['hostname'] = <server>;
$db['pceibal_goalbit']['username'] = <username_mysql>;
$db['pceibal_goalbit']['password'] = <password_mysql>;
$db['pceibal_goalbit']['database'] = <nombre_esquema>;
...
```

8. Crear un branch en la Suite desde el panel de control:

a) Crear un certificado para el branch en Access->Certificates. El tipo de certificado debe ser "Branch certificate".

b) Crear un branch en Access->branches que utilice los certificados creados en el paso anterior.

c) Descargar los certificado del branch y colocarlos en el directorio "goalbit/ssl/services/control".

9. Copiar el certificado de la CA de Goalbit al directorio "goalbit/ssl/access/-ca"

10. Instalar los crons
Para el caso particular de Ubuntu Linux se procede de la siguiente forma:

```
crontab -e
# m h  dom mon dow   command
*/1 * * * * <ruta_portal>/crons/run_cron_live.sh
*/1 * * * * <ruta_portal>/crons/run_cron_live_thumb.sh
```

C. Especificación del plugin GoalBit

El plugin de GoalBit maneja el el tipo de contenido (content-type) "application/x-goalbit-plugin". A continuación se detallan aspectos de la incialización del plugin asi como también los diferentes atributos y métodos que éste expone hacia javascript.

C.1. Inicialización

Al objeto del DOM (Domain Object Model) *EMBED* se le pueden especificar los siguientes parámetros/atributos:

- *target/mrl/filename/src* cadena de caracteres (string) que especifica la fuente multimedia que se desea reproducir.
- *autoplay/autostart* booleano que indica que el reproductor comenzara la reproducción de la fuente multimedia automáticamente, una vez inicializado el plugin.
- *fullscreen* booleano para tener el plugin en modo pantalla completa.
- *mute* booleano para que el plugin inicie en silencio.
- *loop/autoloop* booleano que indica que se desea comenzar a reproducir desde el comienzo la lista de reproducción, una vez esta haya sido finalizada.
- *toolbar* booleano que indica que se desea mostrar una barra de controles para manejar el reproductor multimedia.
- *b_yourself* booleano, cuando este tiene el valor "true" indica que el plugin será utilizado en modo *Broadcast Yourself* .
- *auto_broad_type*, numérico que indica el tipo de broadcaster que se desea levantar, en donde 1 es un broadcaster común y 4 es un broadcaster-super-peer.
- *broadcast_prof* booleano, cuando este tiene el valor "true" indica que el plugin será utilizado en modo *Web Profesional Broadcasting*.
- *controller-url* cadena de caracteres (string) que se agrega para dar soporte al *Web Profesional Broadcasting*, indica la URL del *Controller* en la GoalBit *Suite*.
- *cn-suffix* cadena de caracteres (string) que se agrega para dar soporte al *Web Profesional Broadcasting*, es un identificador de canal para la GoalBit *Suite*, más particularmente es el campo *xmltv_id* del canal.
- *certs-url* cadena de caracteres (string) que se agrega para dar soporte al *Web Profesional Broadcasting*, indica la URL base para obtener los certificados necesarios para poder comunicarse con la GoalBit *Suite* en forma segura.

- *goalbitvod* booleano que indica si el GoalBit Media Player se utilizara en modo *VOD* (Video On Demand).

C.2. Propiedades y Métodos expuestos

Los métodos expuestos por el objeto raíz del plugin se listan a continuación:

- *versionInfo*, devuelve la versión del plugin, ejemplo: "0.7.6 Closer to the lighthouse".
- *activateWebBroadcast*, por medio de ésta se activa el módulo correspondiente al Web profesional Broadcasting, Sección 10, lo que implica que éste comience a anunciarse con el *Controller Server* de la *Goalbit Suite*.
- *deactivateWebBroadcast*, por medio de ésta se desactiva al módulo correspondiente al Web profesional Broadcasting, Sección 10, lo que implica que éste deje de anunciarse con el *Controller Server* de la *Goalbit Suite*.

En las Sub-secciones siguientes se detallarán las propiedades del Web Plugin, que se corresponden a nuevos objetos con propiedades y funciones propias.

C.2.1. Goalbit.audio

Este objeto agrupa las funcionalidades referentes al audio. Corresponde a la clase *LibvlcAudioNPObject*.

Propiedades de solo lectura:

No tiene.

Propiedades de lectura y escritura:

- *mute*, booleano e indica si el reproductor esta en silencio.
- *volume*, numérico correspondiente al volumen del reproductor. El rango posible es [0-200].
- *track*, numérico que indica el track de audio que se desea reproducir o se esta reproduciendo. El rango posible es de [0-65535].
- *channel*, numérico correspondiente a un enumerado en donde los valores posible son "1=stereo","2=reverse stereo", "3=left", "4=right", "5=dolby".

Métodos:

- *toggleMute()*, cambia de valor a la propiedad mute.

C.2.2. Goalbit.input

Este objeto brida las funcionalidades relacionadas con el elemento que se encuentra en reproducción. Corresponde a la clase *LibvlcInputNPObject.*

Propiedades de solo lectura:

- *length*, numérico corresponde al largo del archivo multimedia medido en milisegundos.
- *fps*, numérico de punto flotante correspondiente a los frames por segundo.
- *hasVout*, booleano, indica cuando el video esta siendo mostrado.

Propiedades de lectura y escritura:

- *position*, numérico de punto flotante correspondiente a la posición normalizada en el archivo multimedia de entrada y su rango válido es de [0-1].
- *time*, numérico correspondiente al tiempo transcurrido del archivo multimedia medido en milisegundos.
- *state*, numérico correspondiente a un enumerado que indica el estado actual del reproductor en donde los valores posibles son: "0=IDLE/CLOSE", "1=OPENING", "2=BUFFERING", "3=PLAYING", "4=PAUSED", "5=STOPPING", "6=ERROR".
- *rate*, numérico de punto flotante correspondiente a la velocidad de reproducción, 1.0 es la velocidad norma, 0.5 la mitad de la velocidad y 2.0 sería el doble de velocidad.

Métodos:

No tiene.

C.2.3. Goalbit.playlist

Es una lista de reproducción que permite manejar múltiples entradas multimedia, cada elemento de ésta se referencia por su índice que comienza con 0. Corresponde a la clase *LibvlcPlaylistItemsNPObject.*

Propiedades de solo lectura:

- *itemCount*, numérico que indica la cantidad de elementos en la lista de reproducción.
- *isPlaying*, booleano que indica si la lista de reproducción esta en reproducción.

Propiedades de lectura y escritura:

No tiene.

Métodos:

- *add(mrl)*, agrega un nuevo elemento a la lista de reproducción, especificando el nuevo ítem con el MRL (Multimedia Resource Locator) como cadena de caracteres.
- *add(mrl,name,options)*, agrega un nuevo elemento a la lista de reproducción, especificando el MRL, un nombre y ciertas opciones, todos los parametros son de tipo cadena de caracteres.
- *play*(), comienza a reproducir el actual elemento de la lista de reproducción.
- playItem(index), comienza a reproducir el elemento especificado por su índice en la lista de reproducción.
- *togglePause()*, si se encontraba reproduciendo, pausa la reproducción y si estaba en pause reanuda la reproducción.
- *stop()*.
- *next()*, comienza a reproducir el siguiente elemento de la lista de reproducción.
- *prev()*, comienza a reproducir el anterior elemento de la lista de reproducción.
- *clear()*, limpia la lista de reproducción.
- *removeItem(index)*, saca el elemento especificado por su índice de la lista de reproducción.

C.2.4. Goalbit.video

Este objeto brida operaciones y propiedades relacionadas con el video. Corresponde a la clase *LibvlcVideoNPObject*.

Propiedades de solo lectura:

- *width*, numérico que indica el tamaño horizontal del video .
- *height*, numérico que indica el tamaño vertical del video.

Propiedades de lectura y escritura:

- *fullScreen*, booleano que indica si el video se encuentra en modo pantalla completa.
- *aspectRatio*, cadena de caracteres que indica la relación de aspecto del video y los valores posibles son: "1:1", "4:3", "16:9", "16:10", "221:100" y "5:4".

Métodos:

- *toggleFullscreen()*, cambiar a pantalla completa o volver a visualizar normal.

C.2.5. Goalbit.fileManager

Este objeto fué agregado en este proyecto y brinda ciertas operaciones relacionadas con el manejo del sistema de archivos. Corresponde a la clase LibgoalbitFileManagerNPObject.

Propiedades de solo lectura:

No tiene.

Propiedades de lectura y escritura:

No tiene.

Métodos:

- *ls(path)*, dada la ruta a un directorio, devuelve en formato JSON (JavaScript Object Notation) el contenido de éste, un ejemplo de salida es: [{}, {is_dir:1, is_hidden:1, file_name:'.config'}], como se puede ver el primer elemento es auxiliar y el resto especifican si es un directorio con el atributo is_dir en 1, si es un archivo oculto con el atributo is_hidden en 1 y por último el nombre del archivo.
- *getRootPath()*, devuelve la ruta al directorio raíz, actualmente para el sistema operativo basado en Linux devuelve el valor "/" pero se actualizará para que devuelva según configuración de usuario el directorio a partir del cual se le permite al plugin explorar el sistema de archivos del cliente Web.
- *getTmpPath()*, devuelve la ruta al directorio temporal ejemplo "/tmp".

Bibliografía

[1] Adobe homepage. URL www.adobe.com.

[2] Bsd license definition. URL http://www.linfo.org/bsdlicense.html.

[3] Codeigniter. URL http://codeigniter.com/.

[4] Jquery filetree. URL http://abeautifulsite.net/blog/2008/03/jquery-file-tree.

[5] Jquery homepage. URL http://jquery.com/.

[6] Olpc project. URL http://laptop.org/.

[7] Openvpn. URL http://openvpn.net/.

[8] Plan ceibal. URL http://ceibal.edu.uy/.

[9] Microsoft windows. URL www.microsoft.com.

[10] Youtube. URL http://www.youtube.com/.

[11] Apple homepage. URL www.apple.com.

[12] Mozilla foundation. URL www.mozilla.org.

[13] Opera browser. URL http://www.opera.com/.

[14] Veetle. URL http://www.veetle.com/.

[15] Jffs2: The journalling flash file system, version 2, 07 2003. URL http://sourceware.org/jffs2/.

[16] Mozilla firefox, 2011. URL http://www.mozilla.com/en-US/firefox/new/.

[17] Mysql 5.0 reference manual, disparadores (triggers), 06 2011. URL http://dev.mysql.com/doc/refman/5.0/es/triggers.html.

[18] x264, 5 2011. URL http://developers.videolan.org/x264.html.

[19] Mehdi Achour et al. *PHP Manual*, 1997.

[20] Karina Acosta, Reneé Albornoz, María F. Argenti, Mónica Báez, Marcela Brener, Ana J. Caro, Fernando da Rosa, Roberto Elissalde, Ricardo Garay, Mario Gonzales, Andrés Morales, Graciela Rabajoli, Mauro D. Ríos, Shirley Siri, et al. *CEIBAL en la sociedad del siglo XXI*. UNESCO, 2008. ISBN 978-92-9089-123-9.

[21] Matías Barrios, Andrés Barrios, Juan José Comas, and Pablo Perdomo. Analisis y evaluacion del rendimiento de la plataforma p2p goalbit. Master's thesis, Facultad de Ingeniería, Universidad de La República, 1999.

[22] M. E. Bertinat, D. D. Vera, D. Padula, F. Robledo, P. Rodríguez-Bocca, P. Romero, and G. Rubino. Goalbit: The first free and open source peer-to-peer streaming network. In *LANC '09: Proceedings of the 5th international IFIP/ACM Latin American conference on Networking*, 2009.

[23] Pablo Rodríguez Bocca. *Quality-centric design of Peer-to-Peer systems for live-video broadcasting*. PhD thesis, University of Rennes 1, 2008. URL http://goalbit.sourceforge.net/publications/these.pdf.

[24] Andres Chadarevián, Daniel De Vera, and José Luis Fernandez. Monitoreo y estadísticas en una red de distribucion de multimedia. Master's thesis, Facultad de Ingeniería - Universidad de La República, 2006.

[25] Inc. Cisco Systems. Hyperconectivity and the approaching zettabyte era, Junio 2010.

[26] Josh Coalson. Introduction - what is flac? URL http://flac.sourceforge.net/features.html.

[27] Bram Cohen. The bittorrent protocol specification, Enero 2008. URL http://www.bittorrent.org/beps/bep_0003.html.

[28] Netscape Communication. *Gecko Plugin API Reference*.

[29] Facultad de Ingenieria Universidad de La República GSI. Material del curso "fundamentos de seguridad informática", 2010.

[30] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.1, Abril 2006. URL http://www.ietf.org/rfc/rfc4346.txt.

[31] Fedora.org. Fedora project homepage. URL http://fedoraproject.org/.

[32] Free Software Foundation. Fsf homepage. URL http://www.fsf.org/.

[33] Xiph.Org Foundation. *Vorbis I specification*, February 2010.

[34] Xiph.Org Foundation. *Theora Specification*, Marzo 2011.

[35] Gnome. Pygtk: Gtk+ for python, . URL http://www.pygtk.org/.

[36] Gnome. Gnome, . URL www.gnome.org.

[37] Derk-Jan Hartman, Christophe Massiot, Samuel Hocevar, Geoffroy Couprie, and Jean-Baptiste Kempf. Vlc documentation: Hacker's guide. URL http://wiki.videolan.org/Documentation:Hacker%27s_Guide.

[38] James F. Kurouse Keith W. Ross. *Computer Networking*. Pearson Addison-Wesley, 2010.

[39] laptop.org laptop.org laptop.org laptop.org. Xo laptop. URL http://wiki.laptop.org/go/Hardware.

[40] Craig Larman. *UML y Patrones*. Prentice Hall, primera edición edition, 1999.

[41] Moving Pictures Experts Group (MPEG). Overview of the mpeg-4 standard, Marzo 2002.

[42] MPlayer. Funcionalidades de mplayer. URL http://www.mplayerhq.hu/design7/info.html.

[43] Mozilla Developer Network. Applications based on gecko. URL https://developer.mozilla.org/en/Gecko.

[44] OLPCWiki. Sugar. URL http://wiki.laptop.org/go/Sugar.

[45] Arati Prabhakar. Digital signature standard, Mayo 1994. URL http://www.itl.nist.gov/fipspubs/fip186.htm.

[46] Claudia Rostagnol. Goalbit platform overview. Technical report, Goalbit Solutions, 2011.

[47] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp: A transport protocol for real-time applications, 1996.

[48] SugarLabs. Human interface guidelines. URL http://wiki.sugarlabs.org/go/Human_Interface_Guidelines.

[49] Jean-Marc Valin. *The Speex Codec Manual Version 1.2 Beta 2*, 2002.

[50] Veetle. Publisher's description. URL http://download.cnet.com/Veetle-TV/3000-12512_4-75153877.html.

[51] Daniel De Vera. Goalbit: la primer red p2p de distribucion de video en vivo de codigo abierto y gratuita. Master's thesis, Facultad de Ingeniería - Universidad de La República, 2010.

[52] VideoLan.org. Vlc hacker's guide: Module writers guide. URL http://wiki.videolan.org/Documentation:Hacker%27s_Guide/Module_Writers_Guide.

[53] W3C. Svg (scalable vectors graphics). URL http://www.w3.org/Graphics/SVG/.

[54] Wikipedia. Certificate authority, . URL http://en.wikipedia.org/wiki/Certificate_authority.

[55] Wikipedia. Discrete cosine transform, . URL http://en.wikipedia.org/wiki/Discrete_cosine_transform.

[56] Wikipedia. Gestión de derechos digitales, . URL http://es.wikipedia.org/wiki/Gesti%C3%B3n_de_derechos_digitales.

[57] Wikipedia. Mpeg-2, . URL http://es.wikipedia.org/wiki/MPEG-2.

[58] Wikipedia. Model view controller, . URL http://es.wikipedia.org/wiki/Modelo_Vista_Controlador.

[59] Wikipedia. Npapi, . URL http://en.wikipedia.org/wiki/NPAPI.

[60] Wikipedia. Teorema de muestreo de nyquist-shannon, . URL http://es.wikipedia.org/wiki/Teorema_de_muestreo_de_Nyquist-Shannon.

[61] Wikipedia. Transport layer security, . URL http://es.wikipedia.org/wiki/Transport_Layer_Security.

[62] Wikipedia. Ycbcr, . URL http://en.wikipedia.org/wiki/YCbCr.

[63] Wikipedia. Ext 3, . URL es.wikipedia.org/wiki/Ext3.

[64] Wikipedia. Sun microsystems, . URL http://es.wikipedia.org/wiki/Sun_Microsystems.

[65] Wikipedia. Tivo, . URL http://es.wikipedia.org/wiki/TiVo.

[66] Wikipedia. cron (unix), 06 2011. URL http://es.wikipedia.org/wiki/Cron_(Unix).

[67] Wikipedia. Ajax, 08 2011. URL http://es.wikipedia.org/wiki/AJAX.

[68] Kevin Wixson. *Inkscape User Manual*.

[69] David Woodhouse. Jffs2: The journalling flash file system. Technical report, RedHat Inc., 2008.

Figuras

Índice de figuras