ARQUITECTURA BÁSICA DE UN NAVEGADOR DVB-HTML PARA MÚLTIPLES TERMINALES

ANEXO I

José Wilmer Castillo Obando
Flavio Andrés Martínez Erazo

Director
Ing. RODRIGO ALBERTO CERÓN MARTÍNEZ
Asesores
Ing. VICTOR MANUEL MONDRAGÓN MACA
Ing. FRANCO ARTURO URBANO ORDOÑEZ

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática
Línea de investigación: Sistemas telemáticos a la tele-educación
Popayán, Junio de 2009

TABLA DE CONTENIDO

ANEXO I. DESARROLLO DE LA APLICACIÓN EDiTVBW

Este anexo expone la experiencia del desarrollo del navegador EDiTVBW basado en el estándar DVB-HTML, creado a partir de la arquitectura planteada en el capítulo 4 del documento de monografía. Presenta el montaje de aplicaciones en un flujo broadcast, además de los principales problemas presentados a lo largo del desarrollo, además, soluciones a estos inconvenientes.

El proceso de desarrollo de la aplicación del navegador estaba enfocado a la validación de la arquitectura planteada para el navegador. Antes del establecimiento claro de la arquitectura existían algunos compontes software claramente identificados a los que el navegador debería darle soporte: Parser XML, Intérprete ECMAScript, Parser CSS.

En primer lugar estuvo la exploración de la forma de construir y crear aplicaciones para ser ejecutadas en el STB, estableciendo como requisito conocer las aplicaciones DVB-J, porque no es posible acceder directamente al middleware del STB, como tampoco modificar el Gestor de Aplicaciones del Middleware del receptor.

Determinado el uso de DVB-J, son explorados ejemplos y guías para el desarrollo de aplicaciones DVB-J, estableciendo un entorno de desarrollo con el IDE Eclipse. Para trabajar con el IDE Eclipse en necesario acondicionar el JDK 1.1.8 además de *MHP stubs*, estas últimas son las APIs MHP en las que son encontradas las clases definidas por MHP adicionales a las del JDK. Con lo anterior es asumido que el IDE Eclipse cuenta con un entorno de clases igual al de un STB (fue establecido posteriormente que no es verdad totalmente). Una configuración adicional en el IDE es el estilo de compilación 1.1, debido a que inicialmente el STB no interpretó las clases de las aplicaciones creadas.

G.1 Uso del Software de Playout OpenCaster

Debido a que las aplicaciones DVB-J son aplicaciones para ser distribuidas debe tomarse en cuenta su señalización dentro del canal de emisión de archivos. Para esto fue usado el tutorial del software de *playout* Opencaster[1]. En este tutorial son encontrados ejemplos de aplicaciones y la configuración del archivo en código *phyton* para la generación de la señalización, también se encuentra la manera de generar el *Object Carrousel*, donde son transportados los archivos de la aplicación DVB-J incluido el *.class* de la aplicación.

Las aplicaciones son transportadas en el Object Carrousel por lo tanto toda la aplicación debe ser guardada en una carpeta y luego mediante el script *oc-update.sh* generar el *transport stream* de a ser multiplexado como es explicado en el manual de Opencaster. La configuración del los parámetros del canal, la formación del flujo de video y de audio, así como la integración del los flujos de audios, videos y el flujo de datos del carrusel generados este anexo no son explicados esa información esta detallada en el manual.

DSM-CC es uno de los estándares a la emisión de archivos a través de flujos de transporte. En televisión interactiva es utilizado para transportar las aplicaciones MHP. El estándar es muy complejo debido a que nació de otro ámbito.

[1] El Manual del software OpenCaster está disponible en: http://www.avalpa.com, previo registro.

Para implementar carruseles dentro de la arquitectura OpenCaster, toda la aplicación debe ser guardada en una carpeta para ser ordenado dentro de un TS y el archivo TS necesita ser creado cada vez que el directorio cambie, al igual que las tablas PSI de la aplicación. Si el carrusel es creado correctamente sólo es necesario adicionar éste como todos los otros TS (de audios y videos) a su multiplexación. Opencaster incluye secuencias de comandos para crear el carrusel con el comando *oc-update.sh* y es encontrado en el directorio de herramientas. Ejemplo:

```
oc-update.sh carrusel_1 0xb 4 2003 1 1 0 0
```

oc-update.sh generará el flujo llamado *carrusel_1.ts* que puede ser multiplexado como todos los otros archivos de *.ts*, los parámetros especifican:

- El carrusel a reunir el directorio es: ocdir1
- La etiqueta de asociación para usar la PMT y AIT es la siguiente: 0xb
- La versión de los módulos y secciones del carrusel es: 4
- El PID (identificador de paquete) del flujo es: 2003
- La identificación del carrusel para su uso en la PMT es: 1
- Finalmente: el carrusel será comprimido (1), sin relleno (0) y los archivos temporales serán borrados (0)

Luego de la generación del carrusel, es importante formalizar las tablas necesarias para que el STB reconozca la señalización de los flujos, entre las tabla necesarias para un apropiada emisión están: NIT, PAT, SDT, PMT. En la tabla PMT debe estar una descripción de los carruseles que serán enviados. Para la señalización de las aplicaciones debe editarse la tabla AIT. El software OpenCaster facilita un archivo para configurar estas tablas, denominado *mhpconfig.py*.

A continuación es presentado un fragmento del archivo *mhpconfig.py* donde es encontrada la descripción de la aplicación para la tabla de señalización AIT:

```
application_descriptor(
                  application_profile = 0x0001, # Profile MHP 1.0.2
                  version_major = 1,
                  version_minor = 0,
                  version_micro = 2,
                  service_bound_flag = 1, # 1 means the application …
                  visibility = 3, # 3 the applications is visible to the user..
                  application_priority = 1, # 1 is lowset,…
                  transport_protocol_labels = [1], # carousel Id
                   ),
                   application_name_descriptor(application_name = "Navegador"),
          dvb_j_application_descriptor(parameters = [""]), # parameter passed to the xlet
          dvb_j_application_location_descriptor(
                    base_directory = "/", # base directory
                    class_path_extension = "", # an additiona classpath ..
          initial_class = "render.MainXlet",  # the starting class implementing Xlet
interface
                            ),
```

Luego de tener los flujos .ts de los audios, de los videos, del carrusel y de las tablas de señalización PSI, se continúa con la multiplexación para la emisión. Para esto usamos tscbrmuxer para obtener un Program TS (programa de Transport Stream), con una aplicación interactiva (una mayor descripción es encontrada en el tutorial):

```
tscbrmuxer b:2300000 firstvideo.ts b:188000 firstaudio.ts b:3008
firstpat.ts b:3008 firstpmt.ts b:1500 firstsdt.ts b:1400 firstnit.ts
b:1000000 carrusel_1.ts b:2000 firstait.ts b:9772084 null.ts>
myfirstfifo.ts &
tsstamp myfirstfifo.ts 13271000 > mysecondfifo.ts &
```

```
DtPlay mysecondfifo.ts -t 110 -mt OFDM -mC QAM16 -mG 1/4 -mc 2/3 -mf
578
```

G.2 Construcción de EDiTVBW

Una vez ya desplegadas exitosamente aplicaciones de prueba en el STB, son construidas aplicaciones experimentales para probar el soporte de componentes software, inicialmente con el parser XML. Los requisitos para escoger el parser fueron:

- Intérprete de documentos XML y genere un objeto DOM nivel 2.
- Ejecución sobre el JDK 1.1.8.

Fueron explorados intérpretes XML java con sus diferentes versiones. El parser que cumplió con los requisitos fue *Xerces* versión 1.4, éste brinda soporte a la recomendación XML 1.0 y DOM nivel 2. Las versiones posteriores de *Xerces* no son compatibles con el JDK 1.1.8, aunque brindan la posibilidad de DOM nivel 2 y 3.

Fue construida una aplicación DVB-J que accedía a un documento XML, ésta obtenía un dato y lo mostraba por consola. Fue configurada exitosamente en el sistema para que el STB la ejecutara.

El estándar DVB-HTML exige que sea un parser validador, es decir, que valide el documento HTML con el DTD ofrecido por el estándar. Los problemas presentados en el desarrollo fueron los siguientes:

- El DTD presentado el anexo AA de la especificación MHP 1.2, usa XHTML *modules*, éste DTD no fue aceptado por el parser por tener errores de sintaxis.

- Al usar la recomendación de *modules*, el DTD reutiliza algunos módulos de la W3C para algunos componentes y otros módulos son agregados para el cumplimiento de lo contemplado en DVB-HTML.

- Algunos módulos resuelven URLs a un servidor de DVB, quien aloja el DTD (por ejemplo, "http://www.dvb.org/mhp/dtd/dvbhtml-1-0.dtd"), algunas URLs son recursos no disponibles (por ejemplo, http://www.dvb.org/mhp/dtd/dvb-qname-1.mod), corrigiéndose por inferencia de otras URLs (como ejemplo. http://www.dvb.org/mhp/dtd/dvbhtml-qname-1.mod).

El parser continuó presentado errores de sintaxis para muchos recursos referenciados en el DTD, por tanto, la solución fue simplificarlo. El DTD generado sólo da información al parser para que valide los atributos de una etiqueta, en especial el identificador para permitir búsquedas por ID.

Un parser CSS fue construido para la herramienta de autoría planteada en otro proyecto de grado [1]. Fue modificado para hacerlo compatible con la versión del JDK 1.1.8 y realizar las mismas pruebas que al parser XML *Xerces*.

Inicialmente no fue encontrado un intérprete ECMAScript, por tanto se decidió continuar con las pruebas de interpretación gráfica usando de la API gráfica MHP. De manera alterna continuó la búsqueda de un intérprete para ECMAScript.

Fue planteado el problema de asignarle valores a los diferentes parámetros de los compontes gráficos. Estos valores eran exactamente o al menos proporcionales a los definidos en el documento HTML o de hojas de estilo (CSS) de la aplicación DVB-HTML. Además, fue planteado que debería generarse una interpretación genérica de la información tanto de estilo como del documento HTML para hacer posible una interfaz bien definida entre la visualización y la interpretación como es establecida se establece en la arquitectura.

Tomando como referencia arquitecturas de navegadores web [2], [3], el proceso de interpretación inicialmente genera una interpretación conjunta del documento XML y de estilos, llamado *objeto de marcos* en Mozilla. Esto ayudo en la correspondencia entre los nodos del árbol DOM del documento XML con los componentes gráficos, pero estos nodos deberían estar con toda la información de presentación, por tanto, fue necesario aplicar la lógica de asignación de valores de CSS que los asigna recursivamente de nodo padre a nodo hijo.

En este punto fueron introducidas dos cotas para simplificar el trabajo y facilitar la interpretación en el sistema limitado del STB:

- Soporte sólo a posiciones fijas en los compontes y estas estarán en pixeles, esto con el fin de hacer una correspondencia más sencilla entre nodos y componentes gráficos,

- El algoritmo de hojas de estilo (CSS) es realizado con un solo nivel de profundidad, es decir, la adquisición de valores de atributos para un componente que tenga representación gráfica directa, es hecha de su nodo y de su nodo padre. Si es un nodo sin representación gráfica, la herencia de atributos no es realizada.

Con lo anterior es establecido que el objeto genérico que entrega el módulo Interpretación a Visualización es un objeto DOM enriquecido con los valores de CSS que vengan asignados al nodo, asignando primero a su tipo luego con su ID, sobrescribiéndose en ese orden. En el siguiente ejemplo el valor de `opacity` para los elementos *div* es 1.0, pero si es encontrado un valor diferente para éste atributo en un estilo asociado al id (*myId*), ese valor será sobrescrito (`opacity = 0.5`).

```
div {
        opacity: 1.0;
        background-color: rgb(255,255,255);
        border: none;
}

div#myId{
        opacity: 0.5;
        background-color: rgb(255,255,255);
        border: none;
}
```

La herencia de un nivel de profundidad es implementada en la toma de los datos por parte del componente que haga la interpretación gráfica, tomando los valores para sus atributos de nodo padre y el nodo hijo (este último el que tiene representación gráfica) sobrescribiendo atributos en este orden. Por ejemplo, para un nodo de un enlace <*a*> que está dentro de un <*div*>*,* la clase encargada de la representación *AstbJ.class* en EDiTVBW, tomará los valores para sus atributos del nodo <*div*> y posteriormente del nodo <a> realizando una sobre-escritura de valores.

Una vez determinadas las etiquetas, fueron explorados los componentes gráficos que brinda MHP, algunos fueron básicos como los campos de texto y los cuadros, en cambio la construcción de compontes de formularios exigió más desarrollo y ayuda de ejemplos de componentes gráficos de [4], donde fueron obtenidas nociones e implementaciones para componentes como el teclado y el soporte de los formularios. La determinación y explicación del soporte a determinadas etiquetas es encontrada en el anexo G.

El acceso a los flujos audiovisuales y el manejo del *player*, que es el encargado de reproducir los contenidos multimedia fueron experiencias compartidas con el desarrollo del prototipo del Proyecto EDiTV al igual que el manejo de eventos *broadcast*.

Avanzando en el proyecto, es encontrado el soporte para ECMAScript en Java con el proyecto FESI [5], éste es probado con el JDK 1.1.8 y en el software de PC *mimundoTV* que implementa MHP, funcionando adecuadamente en el IDE y en el PC luego de modificar el uso del *class-loader* de FESI. Se continúo con el desarrollo infiriendo que era idéntico al ambiente del STB, sin embargo, cuando fue realizada una prueba en él, éste registró errores (en el log) por clases no definidas, clases del manejo de *Beans*.

Se concluyó que la implementación de MHP en el STB tenía divergencias con el ambiente de desarrollo establecido con el JDK 1.1.8 y los *MHP stubs*, también con el software para PC *mimundoTV*. El soporte para FESI (ECMAScript) fue realizado para el PC, de una manera demostrativa, debido a la imposibilidad de usarlo en el dispositivo destino del prototipo (STB). De esta manera es soportado el llamado de subrutinas pero no el uso de palabras reservadas para obtener referencias. Las pruebas son encontradas en el anexo H.

BIBLIOGRAFÍA

[1] Darío F. Rojas Rosero, Erney O. Tulande Dulcey, "*Recomendaciones para la generación y distribución de contenidos educativos orientados a Televisión Digital Interactiva*" trabajo de grado, Universidad Del Cauca, Junio 2009.

[2] Ahmed E. Hassan, "*Conceptual Architecture of Mozilla Firefox (version 2.0.0.3)*" Junio 2007.

[3] Alan Grosskurth, Michael W. Godfrey, "*Architecture and evolution of the modern web browser*", University of Waterloo, Junio 2006. Disponible en: http://grosskurth.ca/papers/browser-archevol-20060619.pdf

[4] The MHP Knowledge Project, (2006, Mar), "*Analysis of the current MHP situation: Benefits and exploitation opportunities*", [En línea]. Disponible en: http://www.mhp-knowledgebase.org/publ/mhp-kdb_d1-analysis.pdf [Consulta: Enero de 2009].

[5] FESI Project, (2003, Sep), "*FESI a Free EcmaScript Interpreter*", [En línea]. Disponible en: http://www.lugrin.ch/fesi/ [Consulta: febrero de 2009].