X Congreso Ibérico de Agroingeniería
X Congresso Ibérico de Agroengenharia
Huesca, 3-6 septiembre 2019

Implantación de un Visor GIS con software libre en la Comunidad de Regantes del Trasvase Tajo-Segura de Totana, Murcia

MD Gómez-López [1], J Cánovas[2], F Costa[2], MD Carrillo[3], MP Navarro[3], P Gómez[3]

[1] Cátedra AgritechMurcia, Universidad Politécnica de Cartagena; lola.gómez@upct.es
[2] Comunidad de Regantes del Trasvase Tajo-Segura de Totana
[3] La Compañía del Agua, GIS y Energía, S.L; pgomezg@gestiontecnica.eu

Resumen: En el ámbito del desarrollo de software orientado a la implementación de sistemas de información geográfica, una de las empresas de referencia a nivel internacional es la multinacional estadounidense Esri. En particular, en el contexto de los visores cartográficos para la web, por la experiencia del usuario que aportan. No obstante, la utilización de visores, tal y como los ofrece Esri en su sitio web, queda circunscrita al consumo de servicios ArcGIS Server, que sólo se pueden gestionar mediante la aplicación de Esri ArcGIS for Server. El elevado coste de la adquisición de este software propicia, en muchos casos, la búsqueda de soluciones alternativas, que casi siempre pasan por la utilización de librerías de desarrollo de código abierto, como son OpenLayers, GeoExt y ExtJS, entre otras. Aquí se plantea una propuesta que aprovecha las características funcionales del visor de Esri, evitando recurrir a ArcGIS for Server como servidor de mapas. Es posible gracias a la generación de servicios WMS y WFS a partir de software libre (servidores de mapas MapServer o GeoServer) y, al hecho de que Esri mantenga liberado bajo la licencia Apache 2.0 el código de su visor, permitiendo la alteración del código, lo que permite implementar capacidades funcionales nuevas, como la recuperación interactiva de datos temáticos (operación GetFeatureInfo en servicios WMS) y las búsquedas multicriterio sobre el mapa (operación GetFeature en servicios WFS).

Palabras clave: sistema de información geográfica; servicio OGC; software libre; ArcGIS Server; visor Flex.

1. Introducción

En el ámbito del desarrollo de software orientado a la implementación de sistemas de información geográfica, una de las empresas de referencia a nivel internacional es la multinacional estadounidense Esri. En particular, en el contexto de los visores cartográficos para la web, son muy conocidos, por la experiencia del usuario que aportan [1], sus visores Flex, basados en la tecnología Adobe Flash [2].

No obstante, la utilización de visores Flex, tal y como los ofrece Esri en su sitio web (http://resources.arcgis.com/es/communities/flex-viewer), queda circunscrita al consumo de servicios ArcGIS Server, que sólo se pueden gestionar mediante la aplicación de Esri ArcGIS for Server [3]. La elevada inversión que para empresas y administraciones supone la adquisición de este software propicia, en muchos casos, la búsqueda de soluciones alternativas, que casi siempre pasan por la utilización de librerías de desarrollo de código abierto, como son OpenLayers, GeoExt y ExtJS, entre otras [4]. Esta alternativa, que en general puede dar

doi: 10.26754/c_agroing.2019.com.3472

resultados aceptables, necesariamente implica renunciar a las ventajas que, desde diversos puntos de vista (ergonómico y funcional, sobre todo), conlleva el uso de los visores Flex.

En esta comunicación se plantea, sin abandonar la línea del software libre, una segunda propuesta que pretende aprovechar las características funcionales del visor Flex, evitando recurrir a ArcGIS for Server como servidor de mapas. Ello es posible gracias a varios factores: en primer lugar, existe la posibilidad de generar servicios WMS y WFS a partir de software libre, como es el caso de los servidores de mapas MapServer [5] o GeoServer [6]; además, el visor Flex ya cuenta por defecto con la capacidad de mostrar servicios WMS, lo que evita la necesidad de implementar consultas GetMap; por último, el hecho de que Esri mantenga liberado bajo la licencia Apache 2.0 [7] el código ActionScript de su visor hace viable la libre alteración de dicho código, lo que, en última instancia, permite implementar las capacidades funcionales pendientes, que básicamente son la recuperación interactiva de datos temáticos (operación GetFeatureInfo en servicios WMS) y las búsquedas multicriterio sobre el mapa (operación GetFeature en servicios WFS).

En la siguiente sección se desarrolla esta idea, ilustrándola con la descripción del proceso seguido para la implementación de un visor Flex, que en lo sucesivo denominaremos "visor de Totana" (http://gis.lacompania.eu/visorcrtotana), y que muestra, entre otras capas, las fincas de los comuneros, las conducciones y los puntos de mantenimiento de la Comunidad de Regantes.

En el caso del visor de Totana, se ha seguido un protocolo compuesto por las fases siguientes:

Preparación de la cartografía
Creación de servicios OGC
Acceso a la documentación adjunta
Construcción del visor Flex

En todas las fases, salvo en la última, se ha recurrido a herramientas libres de desarrollo.

Tabla 1. Software necesario para la implementación del visor de Totana

Fase	Alternativas	Software libre o propietario (L/P)	Sistema operativo	Opción elegida
Preparación de la cartografía	QGIS, gvSIG	L	Linux / Windows	QGIS + Windows
Creación de servicios OGC	MapServer, GeoServer	L	Linux / Windows	MapServer + Apache + Linux
Acceso a documentos adjuntos	PHP, Java	L	Linux / Windows	PHP + Linux
Construcción del visor Flex	Adobe Flash Builder	P	Windows	Flash Builder + Windows

Los requisitos funcionales que, en su momento, se plantearon como condiciones *sine qua non* para el nuevo visor fueron las cuatro siguientes:

1. La interfaz de usuario y las funcionalidades básicas deben ser idénticas o muy similares a las ofrecidas por el visor Flex de Esri.
2. Ha de ser posible, mediante click de ratón sobre cada elemento, recuperar la información temática vinculada al mismo, recurriendo a algún tipo de panel o cuadro de diálogo emergente.
3. Ha de ser posible, mediante click de ratón sobre un punto cualquiera del mapa, recuperar la información pública catastral de la parcela concernida (rústica o urbana).

4. Debe permitirse la localización sobre el mapa de cualquier parcela catastral, mediante la indicación, por parte del usuario, de su provincia, municipio, número de polígono y número de parcela.

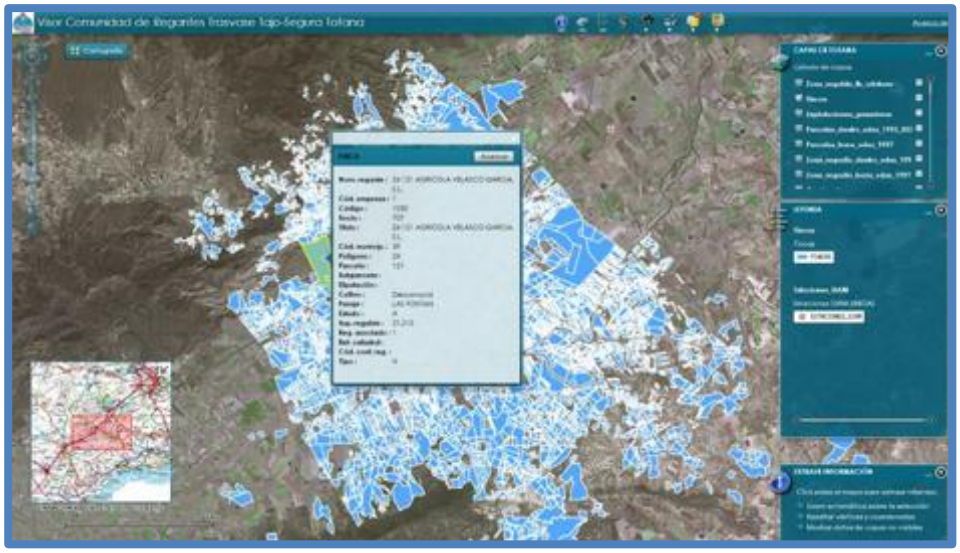

Figura 1. Captura de pantalla del visor de Totana

El requisito funcional número 1 se consigue gracias a la disponibilidad del código fuente del visor Flex original. Un desarrollador puede, de un modo relativamente sencillo, modificar o extender discrecionalmente la funcionalidad del visor Flex de Esri, siempre y cuando los cambios que introduzca en el código fuente se atengan a la jerarquía de clases ActionScript sobre la que se sustenta su estructura.

El resto de requisitos funcionales requieren, cada uno de ellos, la implementación de su correspondiente *widget*. Un widget es un módulo ejecutable con identidad y funcionalidad propias. Dispone de su propia interfaz de usuario y es integrable, como fichero SWF, en otros visores, siempre y cuando estos otros visores hayan sido compilados con la misma versión de Flex y con el mismo API de Esri que el widget. Este modo de proceder es consecuencia directa de la estructura del visor, que se basa en un diseño modular de componentes autónomos y que tiene muy en cuenta los conceptos de alta cohesión (un widget ejecuta sólo una tarea determinada) y bajo acoplamiento (los widgets descienden de una clase común, pero no mantienen entre ellos ningún tipo de vínculo). Algunos de estos componentes autónomos son troncales y otros, los widgets, son componentes adicionales que han sido concebidos e implementados para un fin concreto. En general, extender la funcionalidad de un visor Flex no implica, salvo raras excepciones, alterar el código fuente preexistente, sino sólo añadir código nuevo para implementar un widget nuevo. Se evita así o, en el peor de los casos, se atenúa el riesgo de que el visor resultante manifieste algún tipo de efecto colateral.

Entre los widgets incluidos en el visor de Totana hay tres que permiten el cumplimiento de los requisitos 2, 3 y 4. Estos tres widgets son, respectivamente, **PopupWMSWidget.swf**, **CaPPSearchWidget.swf** y **CaRCSearchWidget.swf**. En consonancia con la filosofía de desarrollo que acaba de ser expuesta, en las siguientes fases (diseño y programación) nos centraremos en únicamente el desarrollo del más destacable de estos tres widgets: **PopupWMSWidget.swf** (la implementación de los otros dos es conceptualmente similar, y metodológicamente más sencilla).

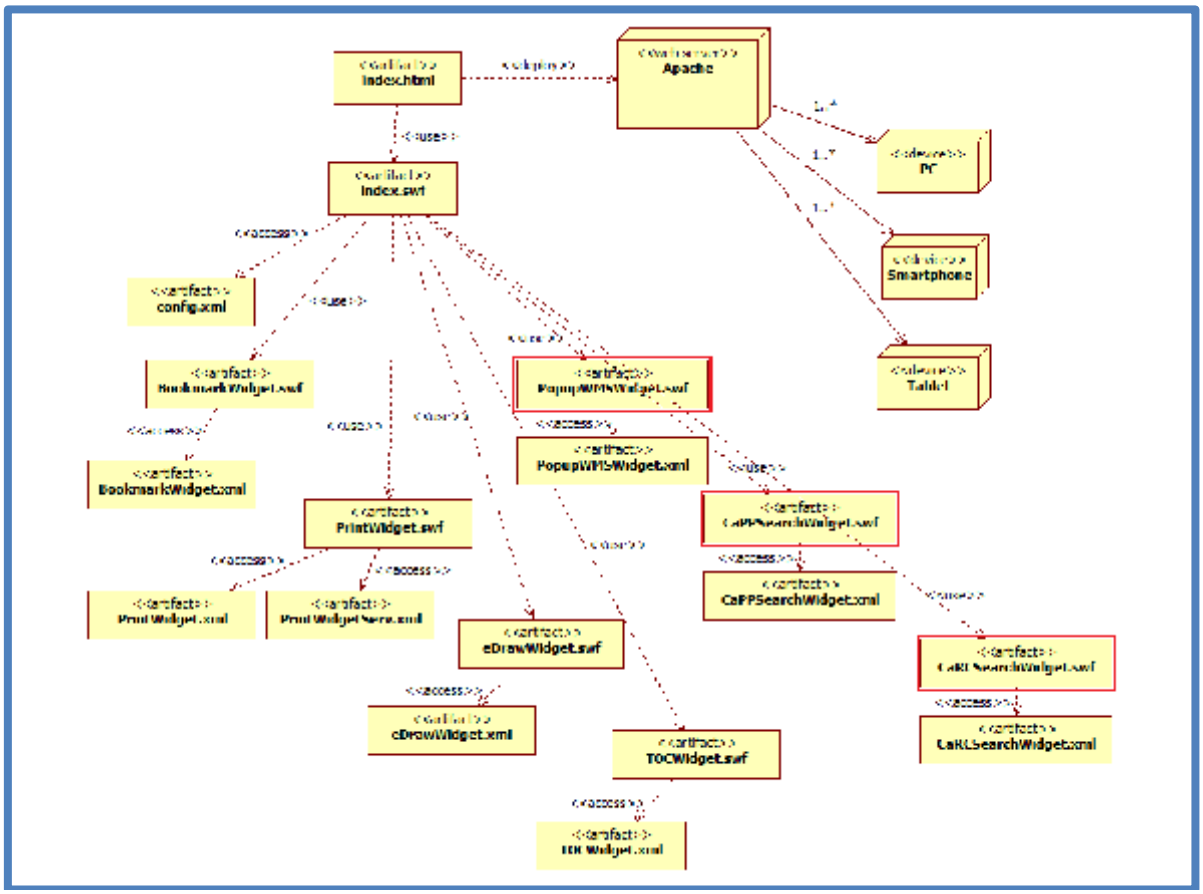

Figura 2. Diagrama de parte del despliEgue del visor de Totana (los widgets enmarcados son los de implementación propia)

2. Materiales y métodos

2.1. Preparación del entorno de desarrollo

En el caso general, la introducción de modificaciones y nuevas mejoras en un visor Flex requiere, en primer lugar, descargar del sitio web de Esri el código fuente del visor (https://github.com/Esri/arcgis-viewer-flex). También es necesaria la descarga de las librerías del API para Flex.

A continuación, siguiendo las instrucciones proporcionadas por Esri, se debe configurar apropiadamente el entorno de desarrollo de la herramienta Adobe Flash Builder. A partir de este momento, el visor ya debería ser compilable [8]. Los cambios en el código fuente han de consistir en fragmentos de código ActionScript, que, por lo común, tendrán por objeto implementar llamadas a operaciones de servicios OGC, como, por ejemplo, la operación GetFeatureInfo de un servicio WMS [9] o la operación GetFeature de un servicio WFS [10].

En el caso del visor de Totana, la funcionalidad añadida mediante la modificación del código fuente incluye, entre otras características, la invocación a la operación GetFeatureInfo del servicio WMS crtotana y la obtención de datos catastrales públicos de cualquier punto del mapa, vía peticiones HTTP a los servidores de la Dirección General de Catastro.

Del conjunto de fases que, desde el punto de vista de la ingeniería del software, requiere el desarrollo completo de una aplicación (obtención de requerimientos, análisis de requisitos, diseño arquitectónico, programación, pruebas, documentación y mantenimiento), en este caso resultan hasta cierto punto críticas, por su relativa dificultad, las fases de diseño y

programación. Procede, pues, llevar a cabo una exposición con cierto detalle de cómo se han ejecutado estas dos fases.

2.2. Diseño arquitectónico

El desarrollo de los componentes del visor que se están mencionando en este documento (se han desarrollado otros como la búsqueda de fincas y dos widgets que extraen información pública de las parcelas registradas en el catastro español que no se detallarán en este documento), PopupWMSWidget.swf, CaPPSearchWidget.swf y CaRCSearchWidget.swf, inevitablemente habrá de estar sometido al modelo de implementación de widgets preestablecido en el código fuente original.

A la hora de describir cómo implementar nuestra solución, que es el propósito de todo diseño arquitectónico, el diseñador deberá partir de dos clases básicas de la jerarquía ActionScript del visor: BaseWidget y WidgetTemplate.

La clase BaseWidget es el ancestro común a todos los widgets. Cualquier nuevo widget que se implemente obligatoriamente deberá pertenecer a una clase derivada de BaseWidget; de lo contrario, la instancia que gestiona los widgets (perteneciente a la clase WidgetManager) sería incapaz de cargar el correspondiente módulo SWF. BaseWidget proporciona, por una parte, una interfaz con el fichero de XML de configuración del widget, y por otra, una segunda interfaz con el mapa mostrado en el visor.

La clase WidgetTemplate viene a desempeñar el rol de interfaz de usuario del widget. Se trata de un panel que, en principio, cuenta con únicamente un icono, una barra de título y los botones de minimización y cierre. El resto de controles que, en cada caso, sean precisos para completar la interfaz de usuario tendrían que ser colocados por el desarrollador en el WidgetTemplate.

Resultará particularmente útil ofrecer detalles sobre las particularidades de los atributos y métodos más relevantes de las clases BaseWidget y WidgetTemplate mencionadas.

Clase BaseWidget:

Module es la clase ActionScript que se utiliza para desarrollar nuevos módulos SWF que puedan ser dinámicamente cargados desde otros módulos SWF. Es, pues, el punto de partida apropiado para *BaseWidget*.

El atributo *configXML* es la interfaz con el fichero de configuración del widget; la consulta de cualquier parámetro del widget siempre se realizará por medio de este atributo. El método más adecuado para consultar, a través de *configXML*, la parametrización del widget es cualquiera que se constituya en respuesta al evento *widgetConfigLoaded*, que es automáticamente disparado por el propio widget dentro del evento *creationComplete* de la clase padre (*Module*).

El atributo *map* representa el mapa del visor. Pertenece a *Map*, una clase que no forma parte del SDK nativo de Flex ni está en el código fuente del visor, sino que se encuentra en la librería compilada que conforma el API de Esri para Flex (**agslib-3.7-2014-11-06** -para la versión 3.7-). Cualquier representación de elementos vectoriales propios sobre el mapa (iconos, líneas, etc) deberá realizarse sobre una intancia de *GraphicsLayer*, que posteriormente se añadirá al mapa a través del método *addLayer* de la clase *Map*.

widgetTemplate, por último, es el atributo que contiene el puntero a la instancia *WidgetTemplate* que proporciona la interfaz de usuario del widget.

Clase WidgetTemplate:

Aparte de servir de panel contenedor para los controles que doten de interfaz gráfica al widget, la clase *WidgetTemplate* ofrece un conjunto de eventos que permiten asociar métodos a

instantes muy concretos del ciclo de vida del citado contenedor, como son su apertura, su cierre, su minimización y su arrastre.

Una instancia de *WidgetTemplate* puede acceder a los atributos y métodos públicos de su *BaseWidget* a través del atributo *baseWidget*.

El diagrama de clases que ilustra el diseño de **PopupWMSWidget.swf** es el mostrado en la Fig. 3.

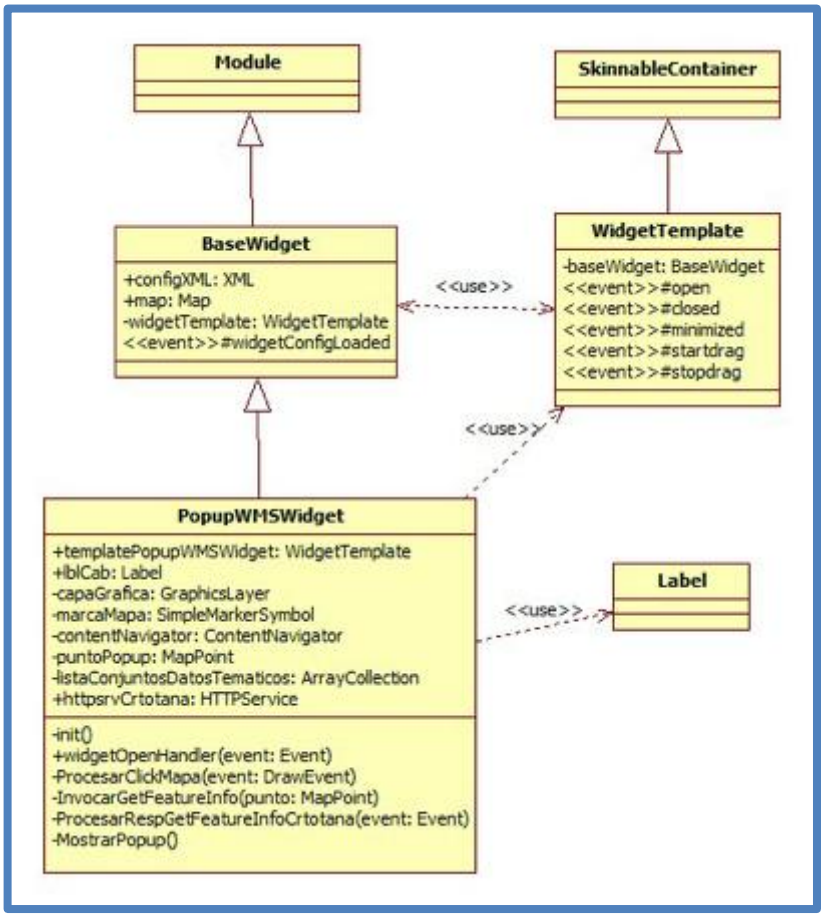

Figura 3. Diagrama de clases de PopupWMSWidget.swf

2.3. Programación del widget PopupWMSWidget.swf

La finalidad de este widget es satisfacer el requisito funcional número 2 ("ha de ser posible, mediante click de ratón sobre cada elemento, recuperar la información temática vinculada al mismo, recurriendo a algún tipo de panel o cuadro de diálogo emergente").

Para el código fuente, en este caso, la clase descendiente de BaseWidget es PopupWMSWidget, y también existe un WidgetTemplate, aunque su contenido es prácticamente irrelevante, ya que consiste en únicamente una etiqueta de ayuda. Ateniéndonos a la estructura de PopupWMSWidget, expuesta en el anterior diagrama de clases, se mostrará seguidamente los fragmentos de código ActionScript de mayor interés.

El primer método en ejecutarse es init(). En él se crean las instancias asociadas a las variables capaGrafica (capa que contiene el punto rojo sobre el que se hace click), listaConjuntosDatosTematicos (lista con los datos devueltos por la operación GetFeatureInfo) y contentNavigator (panel emergente que muestra el resultado de GetFeatureInfo).

```
<?xml version="1.0" encoding="utf-8"?>
<!--
////////////////////////////////////////////////////////////////////////////
// Copyright (c) 2010-2011 Esri.
// Copyright (c) 2017 La Compañía.
//
// PopupWMSWidget: Extracción de información temática incluida en el
//                 servicio WMS "crtotana".
////////////////////////////////////////////////////////////////////////////
-->
<viewer:BaseWidget xmlns:fx="http://ns.adobe.com/mxml/2009"
                           xmlns:s="library://ns.adobe.com/flex/spark"
                           xmlns:mx="library://ns.adobe.com/flex/mx"
                           xmlns:viewer="com.esri.viewer.*"
                           widgetConfigLoaded="init()">
    <fx:Script>
        <![CDATA[
            import com.esri.ags.Graphic;
            /* (resto de “import” y declaración de variables) */
                    private function init():void {
                           // Creación de la capa gráfica que se utiliza
                           // para pintar cada feature sobre la que se haga click.
                           capaSeleccion = new GraphicsLayer();
                           map.addLayer(capaSeleccion);

                           // Creación de la capa gráfica de sobreimpresiones.
                           capaGrafica = new GraphicsLayer();
                           map.addLayer(capaGrafica);
                           marcaMapa = new SimpleMarkerSymbol("diamond");
                           contentNavigator = new ContentNavigator();

                           // Creación de la capa gráfica que se utiliza
                           // para escribir la altura de cada punto.
                           capaAlturas = new GraphicsLayer();
                           map.addLayer(capaAlturas);
                if (configXML) {
                     /* (lectura de parámetros) */
                 }
            }
```

Una vez creado el widget, se produce la ejecución del método widgetOpenHandler(), que está asociado al evento *open* del *WidgetTemplate*. En widgetOpenHandler() se acaba llamando a dos métodos de *BaseWidget* que "preparan al terreno" para la extracción de información mediante los click de ratón sobre el mapa; estos dos métodos son SetMapNavigation() y SeMapAction(). El primero de ellos anula temporalmente el comportamiento inicial de los clicks sobre el mapa (ya no servirán, por ejemplo, para obtener información proveniente de algún servicio ArcGIS Server); el segundo establece una función, en este caso ProcesarClickMapa(), que se ejecutará cada vez que el usuario haga click sobre un punto cualquiera del mapa. ProcesarClickMapa() provocará, a su vez, la ejecución del método InvocarGetFeatureInfo().

```
// Apertura de la interfaz de usuario del widget.
private function widgetTemplateOpenHandler(event:Event):void {
    if (!widgetCargado) {
       // Se anula (provisionalmente, mientras el widget esté en ejecución)
    // el modo de navegación por defecto (el modo que, entre otras cosas, nos
       // permite extraer información a través de paneles "pop-up").
       setMapNavigation(null, null);
       // Se activa el click sobre el mapa para extraer las coordenadas.
       var value:String=DrawTool.MAPPOINT;
       var status:String="click";
       setMapAction(value, status, null, ProcesarClickMapa, null, false, true);
       widgetCargado = true;
       }
}

// Procesamiento de "clicks" sobre el mapa.
private function ProcesarClickMapa(event:DrawEvent):void {
    var geom:Geometry = event.graphic.geometry;
       var punto:MapPoint = geom as MapPoint;
       var coorX:int = (int)(punto.x);
       var coorY:int = (int)(punto.y);

       if (coorX > 0 && coorY > 0) {
              InvocarGetFeatureInfo(punto);
              if (ObtenerAlturas)
                     PonerAltura(punto);
       }
}
```

El método InvocarGetFeatureInfo() admite como único argumento el punto sobre el que el usuario haya hecho click. Este punto, como puede verse en el anterior fragmento código fuente,

es devuelto a través del parámetro DrawEvent de ProcesarClickMapa(). InvocarGetFeatureInfo() considerará el punto que se le suministra y el marco actual del mapa para construir la línea de parámetros de la petición HTTP-REST asociada a la operación GetFeatureInfo (servicio WMS http://gis.lacompania.eu/wms/crtotana).

```
// Adición de contenido a la ventana "popup".
private function InvocarGetFeatureInfo(punto:MapPoint):void {
    var hayInfoTematica:Boolean = false;
       var BBOX_x0:int = 0, BBOX_y0:int = 0, BBOX_xf:int = 0, BBOX_yf:int = 0;
                          var WIDTH:int = 0, HEIGHT:int = 0;
       var X:int = 0, Y:int = 0;
       var capaVisible:Boolean = false;

       puntoPopUp = punto;
       listaConjuntosDatosTematicos.removeAll();

    // Se resetean los flags de respuesta para todas las capas.
    averias_CapaConsultada = false;
    arquetasHidrantes_CapaConsultada = false;
    /* (resto de capas) */

    // Inicialización de los argumentos de "GetFeatureInfo" que
    // tienen que ver con la ubicación del punto.
    BBOX_x0 = map.extent.xmin; BBOX_y0 = map.extent.ymin;
    BBOX_xf = map.extent.xmax; BBOX_yf = map.extent.ymax;
    WIDTH = map.toScreenX(BBOX_xf) - map.toScreenX(BBOX_x0);
    HEIGHT = map.toScreenX(BBOX_yf) - map.toScreenX(BBOX_y0);
    X = map.toScreenX(punto.x); Y = map.toScreenY(punto.y);

    // Llamada a "GetFeatureInfo" para la capa "averias" (si la capa
    // es visible en estos momentos).
    capaVisible = LlamarAHttpServiceDeGetFeatureInfo
        (urlWMSCrtotana, NOMBRE_CAPA_AVERIAS, BBOX_x0, BBOX_y0,
            BBOX_xf, BBOX_yf, WIDTH, HEIGHT, X, Y, httpsrvCrtotanaAverias);
       if (!capaVisible) averias_CapaConsultada = true;

    // Llamada a "GetFeatureInfo" para la capa "arquetas_hidrantes" (si la
capa
    // es visible en estos momentos).
    capaVisible = LlamarAHttpServiceDeGetFeatureInfo
        (urlWMSCrtotana, NOMBRE_CAPA_ARQUETAS_HIDRANTES, BBOX_x0, BBOX_y0,
            BBOX_xf,       BBOX_yf,       WIDTH,       HEIGHT,       X,       Y,
        httpsrvCrtotanaArquetasHidrantes);
       if (!capaVisible) arquetasHidrantes_CapaConsultada = true;
```

Las llamadas HTTP-REST, httpsrvCrtotanaAverias, httpsrvCrtotanaArquetasHidrantes, etc. son asíncronas. Deben efectuarse mediante un objeto *HTTPService* que lleve asociados un método para una respuesta satisfactoria y otro para una respuesta de error. La obtención de una respuesta correcta por parte de la operación GetFeatureInfo del servicio WMS de Totana provocará, en este caso, el disparo de los métodos ProcesarRespGetFeatureInfoAverias(), ProcesarRespGetFeatureInfoArquetasHidrantes(), etc.

```
<fx:Declarations>
    <!-- Invocación a la operación "GetFeatureInfo" del WMS
    "http://gis.lacompania.eu/wms/crtotana", para recuperar los datos
       temáticos de los elementos de la capa "averias". -->
       <s:HTTPService id="httpsrvCrtotanaAverias"
                              resultFormat="e4x"
fault="ProcesarErrorAverias(event)"
                              result                                    =
"ProcesarRespGetFeatureInfoAverias(event)"/>

       <!-- Invocación al PHP que permite obtener la lista de ficheros
       contenidos en un directorio de un servidor web, e inclusión de
       dicha lista en el nodo de "listaConjuntosDatosTematicos"
       correspondiente a la capa "averias". -->
       <s:HTTPService id="httpsrvAccesoDirAverias"
                              resultFormat="e4x" fault="ProcesarError(event)"
                              result                                    =
"ProcesarRespAccesoDirAverias(event)"/>

       <s:HTTPService id="httpsrvCrtotanaArquetasHidrantes"
                              resultFormat="e4x"
fault="ProcesarErrorArquetasHidrantes(event)"
                              result                                    =
```

ProcesarRespGetFeatureInfoAverias() recibirá, a través de su parámetro ResultEvent, la respuesta XML de la operación GetFeatureInfo. Tras procesar dicha respuesta, este método la introducirá en la lista listaConjuntosDatosTematicos y, por último, invocará a MostrarPopup(), que es el método responsable de mostrar el panel de resultados.

```
// "Depuración" de la respuesta XML obtenida desde un servicio HTTP.
public static function BorrarEspacionombresEnXML(xml:XML):XML {
    var rawXMLString:String = xml.toXMLString();
       var xmlnsPattern:RegExp = new RegExp("xmlns=[^\"]*\"[^\"]*\"", "gi");
       var cleanXMLString:String = rawXMLString.replace(xmlnsPattern, "");
       return new XML(cleanXMLString);
}
```

```
// Extracción de los datos temáticos devueltos (en formato XML) por la
operación
// GetFeatureInfo ejecutada por "httpsrvCrtotanaAverias".
private function ProcesarRespGetFeatureInfoAverias(evento:ResultEvent):void
{
var                                resp:XML                                =
UtilStr.BorrarEspacionombresEnXML(XML(evento.result.valueOf()));
var hayInfoTematica:Boolean = resp.children().length() > 0;

if (hayInfoTematica) {
         // Se crea el conjunto de datos respuesta.
         var datosExtraidos:ArrayCollection = new ArrayCollection();
         datosExtraidos.addItem(NOMBRE_CAPA_AVERIAS);
         // Se añaden los campos de información.
         var               nombre:String               =               new
XMLListCollection(XMLList(resp.nombre)).text();
         datosExtraidos.addItem(nombre);
         /* (lectura e inclusión del resto de campos) */

         listaConjuntosDatosTematicos.addItem(datosExtraidos);
         if (dir_docadj == "") {
                averias_CapaConsultada = true;
                MostrarPopup();
         }
else {
                var                subdir_dir_docadj:String                =
dir_docadj.substr(urlServidor.length+1);
                if (subdir_dir_docadj != "") {
                       var patronSust:RegExp = new RegExp("/", "g");
                       subdir_dir_docadj                                    =
subdir_dir_docadj.replace(patronSust, "%2F");
                       var  patronSustPubPorPri:RegExp  =  new  RegExp("%2F"  +
nombreDirPub, "g");
                       var subdir_dir_docadj_pri:String =
                              subdir_dir_docadj.replace(patronSustPubPorPri,
"%2F" + nombreDirPri);
                              httpsrvAccesoDirAverias.url = urlPhp +
                                     "?op=leerdirbis&dir="                  +
subdir_dir_docadj +
                                     "&dir2=" + subdir_dir_docadj_pri;
                              httpsrvAccesoDirAverias.send();
                }
```

```
// Extracción de la lista de archivos devuelta por la llamada a
// "httpsrvAccesoDirAverias" y posible exhibición de la ventana popup.
private function ProcesarRespAccesoDirAverias(evento:ResultEvent):void {
    // Obtención de la respuesta generada por "accesodir.php".
       var datosExtraidos:XML =
             UtilStr.BorrarEspacionombresEnXML(XML(evento.result.valueOf()));
       // Extracción de las posibles referencias a documentos adjuntos
    // que haya en "datosExtraidos", relacionadas con la capa "averias".
       AgregarAConjuntoDatosLosDocAdj(datosExtraidos,
             NOMBRE_CAPA_AVERIAS,
             INDICE_CAMPO_DOCADJ_AVERIAS);
       // Exhibición de resultados.
       averias_CapaConsultada = true;
       MostrarPopup();
}
```

El método MostrarPopup() construye un panel en el que se van colocando, de forma ordenada, todos los nodos de la lista listaConjuntoDatosTematicos (junto con sus respectivos títulos), incluidos los documentos adjuntos. Para ello se llama al método EscribirDatosAverias(). A continuación, elimina cualquier posible punto de algún "popup" anterior y crea otro nuevo para señalar el lugar exacto del mapa del que se acaba de extraer información temática. Por último, se añade al contentNavigator adscrito a la variable map (ver método init()) el panel que acaba de construirse, y se ordena su exhibición por medio del método infoWindow() de la clase *Map*.

```
private function MostrarPopup():void {
       var conjunto:ArrayCollection = null;
       var nombreCapa:String = "";

       if (averias_CapaConsultada && arquetasHidrantes_CapaConsultada &&
    (variables del resto de capas) && listaConjuntosDatosTematicos.length > 0) {
              // Se añade el punto.
              capaGrafica.clear();

              if (MostrarPopupSiempre) {
                     capaGrafica.add(new Graphic(puntoPopUp, marcaMapa));
                     var listaItems:ArrayList = new ArrayList();
                     // Se añade a "listaItems" un panel por cada capa de la que
                     // haya que mostrar información temática.
                     for each (conjunto in listaConjuntosDatosTematicos) {
                            // Creación del panel.
                            var pnlPopUp:Panel = new Panel();
                            pnlPopUp.width = anchuraPanel;
                            pnlPopUp.height = alturaPanel;
                            pnlPopUp.setStyle("backgroundColor",
configXML.popup.backgroundcolor);
                            pnlPopUp.setStyle("color",
configXML.popup.normalcolor);
                            nombreCapa = conjunto[0];
                            var indiceCampo:int = 0;
                            switch(nombreCapa) {
                                   case NOMBRE_CAPA_AVERIAS:
                                   {
                                          indiceCampo                          =
INDICE_CAMPO_DOCADJ_AVERIAS;
                                          break;
                                   }
                     /* (igual para el resto de capas) */
                            }

                  // ¿Cuántas columnas deberá tener el panel?
                            var hayDocAdj: Boolean = conjunto[indiceCampo + 3] !=
"" ||
                                   conjunto[indiceCampo + 5] != "";
                            var hayImagenes: Boolean = conjunto[indiceCampo + 4]
!= "" ||
```

```
                        if (!hayImagenes) pnlPopUp.width = pnlPopUp.width -
                              (int)((anchuraPanel-(6*MARGEN))    /    3)    -
(2*MARGEN);
                        // Poblado del panel (en función de la capa).
                        switch(nombreCapa) {
                              case NOMBRE_CAPA_AVERIAS:
                              {
                              pnlPopUp.uid = NOMBRE_CAPA_AVERIAS + " " +
                                    NOMBRE_CAMPO_IDENTIF_AVERIAS + " " +

      conjunto[INDICE_CAMPO_IDENTIF_AVERIAS];

                  EscribirDatosAverias(pnlPopUp, conjunto, puntoPopUp);
                                    break;
                              }
           /* (igual para el resto de capas) */
               }
                  listaItems.addItem(pnlPopUp);

                  pnlPopUp.addEventListener(FlexEvent.UPDATE_COMPLETE,
   PopupActivate);
            }

            map.infoWindowContent = contentNavigator;
            contentNavigator.dataProvider = listaItems;
            map.infoWindow.show(puntoPopUp);
      }
      else {
          conjunto = listaConjuntosDatosTematicos[0];
            nombreCapa = conjunto[0];
            switch(nombreCapa) {
                  case NOMBRE_CAPA_AVERIAS:
                  {
                        IluminarFeature(nombreCapa,
                              NOMBRE_CAMPO_IDENTIF_AVERIAS,
                              conjunto[INDICE_CAMPO_IDENTIF_AVERIAS]);
                        break;
                  }
             /* (igual para el resto de capas) */
          }
```

```
// Inclusión en el panel "popup" de los datos temáticos del elemento
// de la capa "averias" recuperado en la última operación "GetFeatureInfo".
private function EscribirDatosAverias
       (panel:Panel, datos:ArrayCollection, punto:MapPoint):void {

       // Inicialización de los marcadores de coordenadas de controles.
       var posY:int = MARGEN;
       var x:int = 0, y:int = 0;
       var anchoPanelCab:int = (int)((anchuraPanel-(6*MARGEN)) / 3);
       var posXCol01:int = MARGEN;
       var posXCol02:int = (3*MARGEN) + anchoPanelCab - 2;
       var posXCol03:int = (5*MARGEN) + (2*anchoPanelCab) - 2;

       panel.title = configXML.labels.averiasname;
       x = posXCol01;
       y = posY + ALTURA_PANEL_CAB + MARGEN;

       // Nombre
       var lblnombreTit:spark.components.Label =
       new spark.components.Label();
       lblnombreTit.left = x; lblnombreTit.top = y;
       lblnombreTit.setStyle("fontWeight", "bold");
       lblnombreTit.text = configXML.labels.averias_nombre + " :";
       panel.addElement(lblnombreTit);
       var lblnombre:spark.components.Label = new spark.components.Label();
       lblnombre.left = x + MARGEN_CONTENIDO; lblnombre.top = y;
       lblnombre.text = datos[1];
       panel.addElement(lblnombre);
       y = y + SEPARA;

    /* (igual para el resto de campos y para los posibles documentos adjuntos e
imágenes) */
```

3. Resultados y discusión

El widget **PopupWMSWidget.swf** permite la exhibición de un panel emergente que muestra, adecuadamente formateada, toda la información temática vinculada con aquel elemento que el usuario haya elegido antes, a través de un click de ratón directamente ejercido sobre el mapa. Esta información, que de otro modo sólo hubiese sido accesible en formato XML o de manera estrictamente tabular, es mostrada de una forma que para el usuario resulta, como puede apreciarse en la Fig. 4 y en la Fig. 5, mucho más inteligible.

Figura 4. Resultado de una operación GetFeatureInfo sobre una arqueta hidrante

Figura 5. Resultado de una operación GetFeatureInfo sobre una parcela de consolidado provisional

El procedimiento de introducción de cambios en el código fuente del visor Flex se atiene estrictamente a las recomendaciones fijadas por Esri. Una de estas recomendaciones establece como entorno de desarrollo la herramienta Adobe Flash Builder. Respetar tal consejo facilita mucho el inicio de los trabajos, si bien tiene como inconveniente el hecho de que esta herramienta no es software libre. A este respecto, existen alternativas libres cuyo uso podría intentarse, como es el caso de Flash Develop, e incluso el propio compilador de Flex (sin IDE), que es posible ejecutar en modo línea de comandos tanto en Windows como en Linux.

4. Conclusiones

Tras considerar la jerarquía de clases en la que cabe contextualizar la implementación de cualquiera de los widgets del visor Flex de Esri, se han desarrollado otros widgets para agregar funcionalidades adicionales que no tenían cabida en la versión original; este es el caso de los dos widgets que extraen información pública de las parcelas registradas en el catastro español y de la búsqueda de fincas (la capa 'fincas' es una de las que forman parte de la base de datos de este proyecto).

En resumen, tres son las ideas básicas que, en el contexto de los visores cartográficos, hemos querido poner de manifiesto con la presente comunicación, en la línea de pensamiento propugnada por la Free Software Foundation (http://www.fsf.org). En primer lugar, rebatir el

tópico de que con software libre no es posible construir visores con las mismas funcionalidades que los visores implementados con software propietario. En segundo lugar, recalcar las ventajas que, desde el punto de vista del ahorro de costes, supone aplicar una estrategia de desarrollo basada en el uso de software libre. Por último, un tercer objetivo, no menos importante, es lograr una plena independencia tecnológica de las empresas que comercializan software propietario.

El visor de Totana que hemos expuesto a lo largo de esta comunicación pretende servir de argumento y justificación de estas tres ideas.

Referencias

1. Fleming, J. (1998). Web Navigation: Designing the User Experience. O'Reilly.
2. Cole, A., & Robison, E. (2010). *Learning Flex 4*. O'Reilly Media.
3. Esri (2009). *ArcGIS Server in Practice* Series: Best Practices for Creating an ArcGIS Server Web Mapping Application for Municipal/Local Government. Disponible en http://www.esri.com/library/whitepapers/pdfs/creating-arcgisserver-web-mapping.pdf
4. Hazzard, E. (2011). OpenLayers 2.10 Beginner's Guide. Packt Publishing.
5. MapServer (2014). MapServer Documentation. Release 6.4.1. Disponible en http://mapserver.org/MapServer.pdf
6. GeoServer (2014). GeoServer User Manual. Release 2.3.0. Disponible en http://geoserver.org/display/GEOS/GeoServer+2.3.0
7. Apache Software Foundation (2012). Licenses. Disponible en http://www.apache.org/licenses
8. Zhang, M. (2009). Create GeoWeb Applications with the Sample Flex Viewer. Developer's Guide. Disponible en http://gis.calhouncounty.org/FlexViewerDevelopersGuide_.pdf
9. Open Geospatial Consortium (2006). OpenGIS Web Map Server Implementation Specification. Disponible en http://www.opengeospatial.org/standards/wms
10. Open Geospatial Consortium (2010). OpenGIS Web Feature Service 2.0 Interface Standard. Disponible en http://www.opengeospatial.org/standards/wfs