Método para Adaptar una Librería de Processing a la Web
Method for Adapting a Processing Library to the Web

Cesar Colorado y Jean Pierre Charalambos

***Resumen*— Se presenta un método para la adaptación de libre- rías aportadas por usuarios del lenguaje de gráficos Processing, basado en Java, al lenguaje de gráficos para la web Processing.js, basado en HTML5, WebGL y JavaScript. Se revisan diversos métodos para hacer adaptaciones a la web. En nuestro enfoque, proponemos crear una arquitectura que permite que la librería aportada por el usuario, se compile de Java a Javascript, usando la tecnología Google Web Toolkit, evitando modificar la librería del usuario y haciendo la adaptación en un solo trunk de desarrollo. La arquitectura tiene tres capas: la librería del usuario, una capa que simula el comportamiento de Processing y una para utilizar la librería en la web. Se exponen dos prototipos de librerías adaptadas.**

***Palabras clave*— Google Web Toolkit, GWT, HTML5, JavaScript, Processing, Processing.js, WebGL, Web Graphics.**

***Abstract*— A method for the adaptation of libraries done by users for the Processing graphics language, based on Java, to the graphics engine for the web processing.js, based on WebGL and JavaScript is presented. Various methods to make adaptations to the web are reviewed. In our approach, we propose to create a architecture that is compiled to JavaScript, using Google Web Toolkit technology, in order to maintain the user library without modifications and making the adaptation in a single trunk of development. The architecture has three-tiers: the user library, a layer that simulates the Processing behavior and a layer to use the user library in the web. It is presented two prototypes of adapted libraries.**

***Index Terms*—Google Web Toolkit, GWT, HTML5, JavaScript, Processing, Processing.js, WebGL, Web Graphics.**

I. INTRODUCTION

PROCESSING es un lenguaje y entorno de programación de código abierto basado en Java , para la creación de imágenes, animaciones e interacciones; está orientado a diseñadores, artistas digitales, estudiantes e investigadores[1]. Buena parte de la funcionalidad del lenguaje es extendida mediante librerías contribuidas por miembros de la comunidad.

Los programas de Processing (denominados sketches) en su versión estable 1.5.*, se pueden ejecutar dentro de un navegador web mediante la tecnología de Applets de Java [2], lo que supone el inconveniente de tener que instalar en el navegador una extensión propietaria y no estándar. En el pasado un enfoque similar ha sido empleado por otras tecnologías como Adobe Flash [3], Microsoft Silverlight [4] y X3D [5].

Processing.js [6], es una adaptación de Processing en JavaScript (JS en adelante) para la web, basado en el elemento canvas de HTML5 y WebGL [7]. Este enfoque fue recientemente adoptado en la versión actual de desarrollo de Processing (2.2.1 en el momento de este escrito)[8], para la ejecución de un sketch cualquiera dentro del navegador, dejando en desuso la tecnología de Applets de Java. Processing.js soporta la mayoría de las funciones de Processing-1.5.*; pero su versión actual adolece (v-1.4.1 al momento de este escrito) de dos inconvenientes: 1. No soporta varias de las funciones introducidas en Processing-2.0b8, particularmente las concernientes al manejo de shaders; y 2. No cuenta con un mecanismo para incluir librerías de terceros (mecanismo que estaba presente en la tecnología de Applets de Java), limitando de manera considerable su uso.

Nuestro objetivo es atender el segundo de los problemas, proponiendo un método con el cual se puedan adaptar librerías de terceros a Processing.js, poniéndolas a disposición del programador final dentro de un contexto web, con una mínima intervención de su parte (ver Sección III y IV). Además, el método se puede adaptar fácilmente para desarrolladores Java con poco conocimiento en JS (ver Sección V). Para demostrar la validez de nuestro enfoque, dos librerías Processing escritas en Java, han sido portadas a JS (ver Sección VI). Finalmente, discutiremos las limitaciones y alcances futuros del método (ver Sección VII).

II. TRABAJOS PREVIOS

Revisando los sketches y librerías para Processing.js en [9], podemos distinguir dos tipos de métodos para realizar una adaptación de un sketch de Processing a la web: automático y

Cesar Colorado, Departamento de Ingeniería de Sistemas e Industrial, Facultad de Ingeniería, Universidad Nacional de Colombia, e-mail: cacolorador@unal.edu.co

Jean Pierre Charalambos, Departamento de Ingeniería de SIstemas e Industrial, Facultad de Ingeniería, Universidad Nacional de Colombia, e-mail: jpcharalambosh@unal.edu.co

manual. En el tipo método automático se utiliza un programa que traduzca el sketch de Java a JS, tal como lo hace el modo JS de Processing 2.0. El tipo de método manual consiste en tomar el algoritmo del sketch y escribirlo en JS.

A. *Adaptación manual*

Existen algunas librerías de Processing adaptadas a JS, por ejemplo Toxiclibs.js [8], [10], que es un ejemplo donde se reescribe clase por clase el código de Java a JS, conservando la estructura de los paquetes, las firmas de los métodos y los algoritmos originales [10].

B. *Processing 2.0 Modo Javascript*

En Processing 2.0 es posible ejecutar un sketch en un navegador web con el MODO JAVASCRIPT [8], que utiliza la función parseProcessing de Processing.js. El modo JS puede adaptar sketches simples; pero no puede realizar la adaptación de librerías más grandes o complejas, debido a que no es posible depurar ni soporta las utilidades y ventajas de Java, como POO, interfaces, colecciones, etcétera.

Sea usando el modo JS Mode o realizando la adaptación manual, es requerido abrir una rama nueva de desarrollo y la directa intervención del desarrollador, haciendo la adaptación altamente susceptible a errores.

III. ENFOQUE PROPUESTO

Como se mencionó, el objetivo de nuestro método es adaptar las librerías de Processing a la web modificando lo menos posible el código fuente original y sin requerir abrir una rama nueva de desarrollo. Proponemos una arquitectura de tres capas para mantener el código fuente original intacto, donde se traducen las librerías de Java a JS automáticamente utilizando el compilador Google Web Toolkit (GWT) [11]. La adaptación está limitada a usar las clases del lenguaje Java definidas en el GWT Java Runtime Environment Emulation Reference (JRE ER)[12], sin embargo podemos utilizar muchas de las utilidades del lenguaje como colecciones, programación orientada a objetos, enums, tipos genéricos, además del depurador.

A. *Alcance y objetivos de la adaptación*

El esquema general del método se ve en la figura 1 y cumple dos objetivos:

1. La librería portada a JS debe tener una API muy similar o idéntica a la de la librería en Java.
2. La librería portada a JS debe poder realizar los mismos llamados a la API de Processing que a la librería en Java.

B. *Arquitectura del método*

El código compilado a JS de GWT es ofuscado y usa una sintaxis propia [11]. Para que este pueda ser utilizado por el usuario, es necesario crear un "wrapper" de JS sobre la librería en Java. En nuestra implementación, se construyen tres capas jerárquicas donde irán el wrapper y el acceso a la API de Processing de esta manera:

- Core: El código original de la librería.
- Facade: El wrapper que pública a la capa Core en un objeto JS. Cumple el objetivo 1.
- Back: Empleando la terminología de Java RMI [13], esta capa tiene clases Stub que simulan el comportamiento de Processing y llaman en su lugar a Processing JS. Cumpliendo el objetivo 2.

Esta arquitectura se ve en la Figura2.

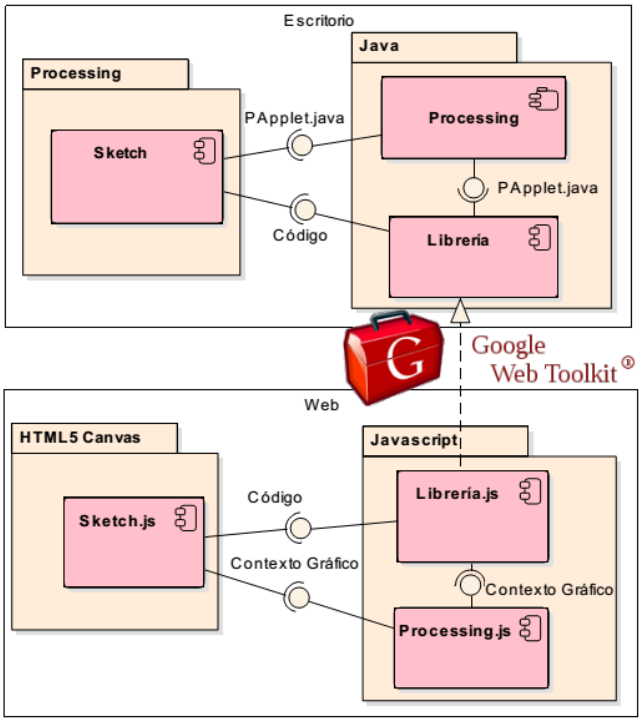

Figura 1. Adaptación de una librería de Processing

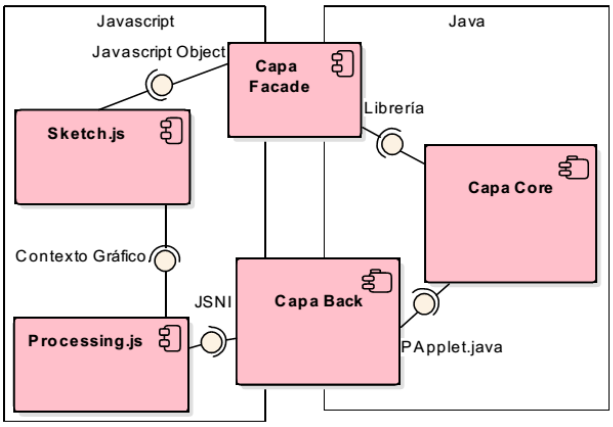

Figura 2. Arquitectura de tres capas

IV. IMPLEMENTACIÓN

GWT funciona por módulos identificados por un archivo xml con nombre único, estos módulos son conjuntos de paquetes Java seleccionados para compilar. En nuestro caso

cada módulo es una capa de la adaptación.

Las tres capas se organizan en tres proyectos de este modo:

- Web Publish Project (ver Sección IV-C): Aplicación GWT Web dedicada a generar el archivo final, tiene la clase Entry point que es la primera en ser ejecutada cuando la página web cargue y donde se publicara la librería portada en el objeto JS window de la página web.
- Core-Facade Project (ver Sección IV-B): Aplicación GWT Java que contiene las capas Core y Facade.
- Back Project (ver Sección IV-A): Aplicación GWT Java que contiene la capa Back de la arquitectura.

El objetivo de la capa Facade es servir de interfaz a la capa Core, es decir que estarán fuertemente acopladas y por eso se dejan en el mismo proyecto. La capa Back con el Processing Stub puede ser reutilizada por otras adaptaciones, por eso se deja en un proyecto separado. La organización de los proyectos se ve en la Figura 3.

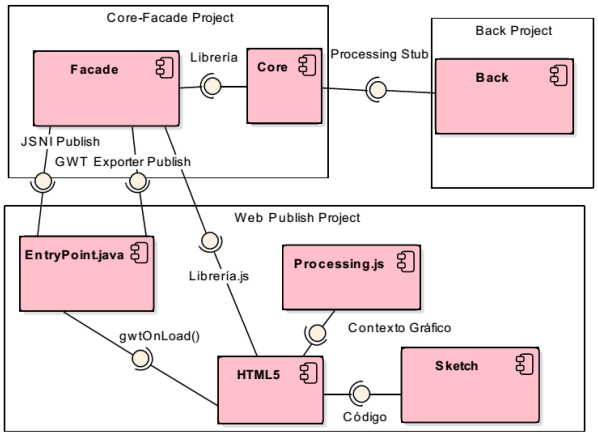

Figura 3. Implementación

A. *Proyecto Back: Processing Stub*

El propósito de esta capa es evitar la modificación del código original de la librería cuando realice llamados a Processing. Siendo PApplet la clase principal de Processing, crearemos un módulo GWT llamado Processing con una clase Stub que será llamada PApplet también. Esta encapsula llamados a atributos, métodos y funciones de Processing.js; emulando los métodos y el comportamiento de la clase PApplet original.

En esta clase Processing Stub se mantiene una referencia GWT JavaScriptObject al contexto gráfico de Processing.js (CGPJS en adelante); los atributos, métodos y funciones son accedidos desde Java a JS por medio de la JavaScript Native Interface (JSNI) [14]. Ver algoritmo 1 y la Figura 4.

Algoritmo 1 Clase Stub

```
public class PApplet {                                              1
 private JavaScriptObject context;                                  2
 public object atribute;                                            3
                                                                    4
 public PApplet (JavaScriptObject object){                          5
  init(object);                                                     6
 }                                                                  7
 private native void init(JavaScriptObject object)/*-{              8
  this.@processing.core.PApplet::context = object;                  9
  this.@processing.core.PApplet::atribute = object.atribute         10
      ; }-*/;
                                                                    11
 public final native void method (object o)/*-{                     12
  var context = this.@processing.core.PApplet::context;            13
  context.method(o); }-*/;                                          14
}                                                                   15
```

Línea2 Atributo context de tipo GWT JavaScriptObject, es una referencia al CGPJS, con el cual podemos acceder a objetos JS desde Java y viceversa usando JSNI.

Línea8 Método JSNI que asigna el CGPJS a la clase, conociendo el nombre de los atributos en JS, pueden ser asignados a

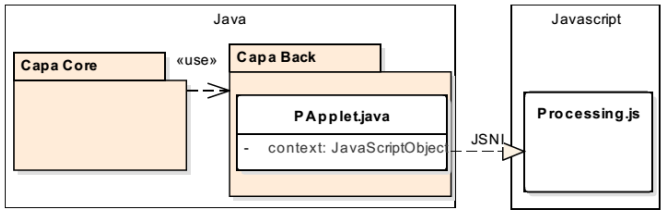

Figura 4. Estructura del Processing Stub

B. *Proyecto Core-Facade*

La capa Core contiene la librería original sin modificaciones. La capa Facade contiene la clase ExporterFacade.java, que es el wrapper JS de la librería compilada, en esta clase los objetos Java de la capa Core son recreados y publicados dentro de objetos JS utilizando JSNI. Estos objetos JS tienen el mismo nombre y comportamientos que sus originales en Java. Cada objeto JS mantiene una referencia a su objeto Java correspondiente para acceder a él desde la página web. Estos objetos JS pueden tener una referencia al CGPJS, de manera que puede ser usado luego en la clase Processing Stub en la capa Back de jerarquía inferior. Ver algoritmo 2.

Este tipo de publicación manual JSNI, dependiendo de la complejidad de la librería puede resultar largo y difícil, sin embargo existe una herramienta que realiza este proceso automáticamente: GWT EXPORTER [15].

Gwt Exporter: Esta librería GWT publica objetos Java en objetos JS igual que la publicación manual JSNI, pero usando anotaciones e interfaces. En este artículo, se mostrara como usar ambas opciones, ver Figura 3. Para que la librería original no sea modificada se utiliza la interfaz ExportOverlay. Ver algoritmo 3.

Algoritmo 2 Clase JsniFacade

```
1  public class ExporterFacade {
2
3   public static LibraryClass CreateLibraryClass(
        JavaScriptObject context,object param)
4       {return new LibraryClass(new PApplet(context),param);}
5
6   public  native void JsniPublish() /*-{
7      $wnd.LibraryClass  = function(context,param)
8    {var JSClass = @facade.client.JsniFacade::LibraryClass(
         Lcom/google/gwt/corepis /client/JavaScriptObject;FFF)
         (context,param);
9     JSClass.method = JSClass.@LibraryPackage.LibraryClass::
          method(F);
10    return JSClass;
11      }
12  }-*/;
13 }
```

Línea4 Instancia de la librería original, con algún parámetro requerido o el CGPJS para ser usando en la clase PApplet de la capa Back.

Línea7 Un objeto facade con el mismo nombre de la librería original es creado en el objeto $wnd, que es una constante de GWT para referirse al objeto window en JS, objeto principal JS de cada página html.

Línea8 Una vez el objeto JSClass JS ha sido instanciado, siendo JSClass una referencia a la clase Java, se pueden crear apuntadores JS a los métodos Java como en la línea 9.

C. *Web Publish Project*

Este es el proyecto principal que utiliza el usuario, aquí está contenida la página web html donde se muestra el sketch y se compila la capa Facade a JS. Todo proyecto web GWT debe tener una clase que implemente la interfaz GWT EntryPoint, en este caso Web.java. Esta interfaz GWT EntryPoint obliga a implementar el método onModuleLoad (), este método compilara a JS la clase ExporterFacade como se ve en la Figura 5

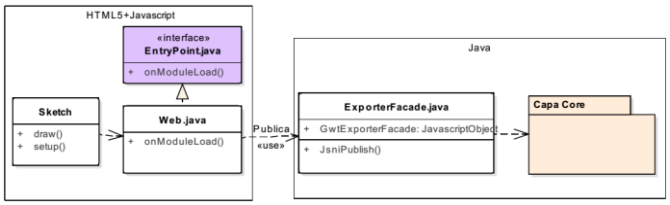

Figura 5. Proyecto Web Publish

El método onModuleLoad() es el primero en ejecutarse cuando se carga la página web. Processing.js debe iniciarse después de que publique la clase JsniFacade con GWT Exporter o manualmente con JSNI, ver el algoritmo 4.
Finalmente la librería es llamada en la página web como se ve en el algoritmo 5.

V. DESARROLLADORES JAVA

Hasta ahora el método propuesto está dirigido principalmente a los desarrolladores web JS; sin embargo nuestro método permite fácilmente adaptarse a los desarrolladores Java sin requerir casi ningún conocimiento JS; en lugar de exportar cada clase (y sus métodos), podemos crear una única clase Sketch.java (Ver Algoritmo 6), que contendrá todo el código del sketch. Esta clase Sketch.java debe heredar de la clase PApplet.java todos los métodos necesarios para funcionar y puede iniciarse con el CGPJS. Al final solo la clase Sketch.java y los métodos utilizados de Processing.js serán exportados usando GWT Exporter (Ver algoritmo 7), así todo el código será ejecutado de manera trasparente dentro de la clase portada Sketch.java en JS.

Algoritmo 3 Exportar con Gwt Exporter

```
public  class ExporterFacade {                                  1
 @ExportPackage("")                                             2
 @Export("LibraryClass")                                        3
 public static abstract  class GwtExporterFacade implements     4
       ExportOverlay<LibraryClass>
 {                                                              5
  @ExportConstructor                                            6
  public static LibraryClass constructor( JavaScriptObject     7
       context,object param){
   return new LibraryClass(new PApplet(context),param); }       8
  public abstract OtherLibraryClass method();                   9
  public abstract void method2(OtherLibraryClass other);        10
 },,                                                            11
 @ExportPackage("")                                             12
 @Export("OtherLibraryClass")                                   13
 @ExportClosure()                                               14
 public abstract class OtherLibraryClass implements             15
      ExportOverlay<OtherLibraryClass>   {
  public abstract int attribute();                              16
 }                                                              17
}                                                               18
```

Línea2 @ExportPackage: Da nombre al objeto JS que contendrá todas las clases exportadas, el objeto window JS es la selección por defecto.

Línea3 @Export: Marca una clase y métodos públicos para ser exportados. Define el nombre final de la clase o método en JS.

Línea6 @ExportConstructor: Marca un método que hará de constructor de la clase a exportar, para que esta etiqueta funcione, debe existir un constructor sin parámetros en la clase a exportar.

Línea14@ExportClosure: Marca una clase para ser un argumento de algún método exportado a JS.

Algoritmo 4 Implementando Gwt EntryPoint

```
public class Web implements EntryPoint {                       1
 public void onModuleLoad() {                                   2
  JsniFacade f = new JsniFacade(); //JSNI Export                3
  f.publish();                                                  4
  ExporterUtil.exportAll(); // Gwt Export                       5
  onLoadImpl();                                                 6
 }                                                              7
 private native void onLoadImpl() /*-{                          8
  if ($wnd.gwtOnLoad && typeof $wnd.gwtOnLoad == 'function      9
      ')
   $wnd.gwtOnLoad(); }-*/;                                      10
}                                                               11
```

Línea3 Publicando con JSNI.
Línea5 Publicando con GWT Exporter.
Línea9 Método JS gwtOnLoad.

VI. RESULTADOS

Para ilustrar nuestra propuesta, estas librerías fueron adaptadas exitosamente a la web manteniendo sus comportamientos:

- Obsessive Camera Direction (OCD) , [16] (ver Sección VI-A): Librería para el control de cámara de una escena, consta únicamente de la clase Camera.java, se adaptó con el propósito de probar la publicación JSNI y la capa Back, ya que internamente llama a la clase PApplet.java de Processing. Se adaptó el siguiente ejemplo: OCD Reference Zoom.

Algoritmo 5 Llamado de la librería desde la página web

```
1  <script type="text/JS" language="JS" src="webpublish/
       webpublish.nocache.js"></script>
2  <script src="processing.js"></script>
3  <script language='javascript'>
4   function gwtOnLoad()
5   { var p = new Processing("processing-canvas", sketchProc);
6     function sketchProc(processing) {
7     var c;
8     processing.setup = function() {
9      c = new LibraryClass(this, 1);
10    };
11    processing.draw = function()
12    {c.method(1);};
13   }
14  }
15 </script>
```

Línea9 Llamado de la librería. Se puede pasarle el CGPJS al constructor.
Línea12Llamado un método de la librería.

Algoritmo 6 Case Java con el código del *Sketch*

```
1  public class Sketch extends PApplet {
2   public Sketch(){}
3   public Sketch(JavaScriptObject ctx) {
4    super(ctx);
5   }
6   public void setup() {
7    size(100, 100);
8   }
9   public void draw() {
10   background(0);
11  }
12 }
```

Line2 Constructor vacío para que @ExportConstructor funcione.
Line3 Constructor donde la superclase PApplet.java es instanciada con el CGPJS.

- Traer's physics Port [17] (ver Sección VI-B): Librería de física que calcula y aplica colisiones, sistemas de partículas, etcétera, a objetos manipulables con el mouse.
  Se adaptó con el propósito de probar la publicación con GWT de una librería con varias clases; pero no realizan llamados a Processing.

Algoritmo 7 Usando la clase portada Sketch.java

```
1  <script type="text/JS" language="JS" src="webpublish/
       webpublish.nocache.js"></script>
2  <script src="processing.js"></script>
3  <script language='javascript'>
4   function gwtOnLoad()
5   { var p = new Processing("processing-canvas", sketchProc);
6     function sketchProc(processing) {
7     var c;
8     processing.setup = function() {
9      c = new Sketch( this );
10     c.setup();
11    };
12    processing.draw = function()
13    {c.draw();};
14   }
15  }
16 </script>
```

Línea9 Instancia la super clase Sketch.java y ejecuta setup().
Línea13Ejecuta draw().

Para demostrar la flexibilidad del método, las dos librerías fueron adaptadas utilizando el archivo de código .PDE de Processing y también con código JS embebido dentro de la página web, pueden verse corriendo junto con su código fuente en [18].

A. Web ocd

Empleando nuestro método, se crearon tres proyectos: la capa back Processing Stub, la capa core OcdJs y la capa facadeWebOcd que contienen tres módulos GWT, processing.gwt.xml, ocd.gwt.xml y facade.gwt.xml.

Se identifican los métodos que OCD utiliza de Processing son: PApplet.perspective y PApplet.camera. Así que los métodos Stub correspondientes fueron creados en la capa Back.

En la capa Core se coloca la librería OCD sin modificar. En la capa Facade se realiza la publicación en JS usando JSNI manualmente, exportando la clase Camera.java y sus métodos.

B. Web Traer Physics y GWT Exporter

Traer Physics 3.0 [19], no realiza llamados a Processing; pero las clases son complejas e implementan interfaces. Siguiendo el método, se crearon dos proyectos: la capa Core TraerPhy- sicsJs y la capa Facade WebTraerPhysics; contienen los dos módulos GWT: traerphysics.gwt.xml y facade.gwt.xml.

La capa Core contendrá la librería Traer Physics intacta y las clases principales para exportar son ParticleSystem.java, Vector3D.java y Particle.java.

La publicación a JS se realiza en la capa facade con la clase ExporterFacade.java usando GWT Exporter, allí se utilizó la anotación @ExporterPackge ("TraerPhysics"), lo cual significa que las clases están contenidas en el objeto JS TraerPhysics. Se utilizó la anotación @ExporterConstructor en la clase Particle.java, y se creó un constructor sin parámetros para ella.

C. Ejemplos para desarrolladores Java y JavaScript

Algunos sketches simples de Processing con interacción teclado y mouse han sido adaptados para probar los modos de desarrollo para programadores Java o JS; los cambios al código original fueron mínimos y también pueden verse corriendo juto a su código fuente en [18].

- Action Driven Callback: Sketch para desarrolladores Java donde se pueden arrastrar círculos con el Mouse. Se reutiliza la capa Back del ejemplo VI-A. El código del Sketch se mantiene intacto en la capa Core, sin embargo en la capa Facade, no se exportan los componentes del sketch si no únicamente la clase principal llamada ActionDrivenCallback.java y los métodos setup() y draw(); de manera que el sketch se mantiene completamente en Java; pero para modificar el código del sketch se requiere compilar el código de Java a JS.
- Boring clic And Drag: Sketch simple para desarrolladores JS donde se puede cambiar al azar la ubicación de círculos con un clic. Se reutiliza de nuevo la capa Back del ejemplo VI-A; pero la capa Core contiene únicamente los componentes del sketch: MouseAgent.java, TerseHandler.java y GrabbableCircle. En la capa Facade los componentes son exportados cada uno a un objeto JS conrrespondiente usando GWT Exporter. El código del sketch es escrito en JS y embebido en la página html, donde puede ser modificado sin necesidad de compilar a JS.

D. Comparación y ventajas

Para el desarrollador de librerías, las principales ventajas que ofrece el método son dos: permite tener un solo trunk de desarrollo y no se requiere tener conocimientos en JS, sin embargo es flexible para quien quiera usar JS o Java.

El enfoque por módulos permite la reutilización de código, por ejemplo la capa Back que contiene el Processing Stub, puede adaptarse para ser utilizado por otras librerías que lo requieran, incluso esta capa puede contener otros motores gráficos. El enfoque permite también crear diferentes módulos para varias utilidades que pueden ser recurrentes en futuras adaptaciones como hilos, tablas Hash, parsers de arreglos entre Java y JS, utilidades matemáticas, acceso a las Apis de HTML5, etcétera.

VII. Conclusiones

Se ha presentado un método para adaptar librerías de Processing a Processing.js, sin conocimientos de JS y manteniendo un solo trunk de desarrollo. Las librerías adaptadas en este artículo son solo pruebas de concepto que demuestran que desarrollando más el método podremos en el futuro adaptar librerías con mucha complejidad como Proscene [20].

Como trabajo futuro, se podría automatizar este método en una herramienta para el IDE de Processing o para Eclipse IDE; sin embargo las limitaciones del método están dadas por el uso de GWT, diferencias irreconciliables entre los lenguajes Java y JS como, hilos, reflection y demás; pero pueden ser abordados con nuevas tecnologías como HTML5 Web Workers and extensions.

Referencias

[1] C. Reas and B. Fry, Getting Started with Processing. O'Reilly, 2010.

[2] L. Burdy, A. Requet, and J.-L. Lanet, "Java applet correctness: A developer-oriented approach," in FME 2003: Formal Methods. Springer, 2003, pp. 422–439.

[3] Adobe, "Adobe flash platform," http://www.adobe.com/flashplatform/, 2012. [Online]. Available: \protect\unhbox\voidb@x\penalty\@M\http://www.adobe.com/flashplatform/

[4] Microsoft, "Microsoft silverlight perspective 3d graphics," 2012. [Online]. Available: http://www.microsoft.com/silverlight/perspective-3d-graphics/

[5] D. Brutzman and L. Daly, X3D: extensible 3D graphics for Web authors. Morgan Kaufmann, 2010.

[6] J. Resig, B. Fry, and C. Reas, "Processing. js," 2012.

[7] T. Parisi, WebGL: Up and Running. O'Reilly Media, 2012. [Online]. Available: http://shop.oreilly.com/product/0636920024729.do

[8] J. Vantomme, Processing 2: Creative Programming Cookbook: Over 90 Highly-effective Recipes to Unleash Your Creativity with Interactive Art, Graphics, Computer Vision, 3D, and More. Packt Publishing, 2012, chapter 9: Exploring JavaScript Mode.

[9] P. team, "Processingjs exhibition," 2015. [Online]. Available: http://processingjs.org/exhibition/

[10] K. Phillips, "Toxiclibs.js open source computational design," http://haptic-data.com/toxiclibsjs/, 2011. [Online]. Available: http://labs.hapticdata.com/2011/01/toxiclibs-js-open-source-computational-design/

[11] Google, "Understanding the gwt compiler," https://developers.google.com/web-toolkit/doc/latest/DevGuideCompilingAndDebugging, October 2012. [Online]. Available: https://developers.google.com/web-toolkit/doc/latest/DevGuideCompilingAndDebugging#DevGuideJavaToJavaScriptCompiler

[12] ——, "Jre emulation reference," https://developers.google.com/web-toolkit/doc/latest/RefJreEmulation, October 2012. [Online]. Available: https://developers.google.com/web-toolkit/doc/latest/RefJreEmulation

[13] Oracle, "Stubs and skeletons," 2010. [Online]. Available: http://docs.oracle.com/javase/7/docs/platform/rmi/spec/rmi-arch2.html

[14] Google, "Coding basics - javascript native interface (jsni)," https://developers.google.com/web-toolkit/doc/latest/DevGuideCodingBasicsJSNI, 2012. [Online]. Available: https://developers.google.com/web-toolkit/doc/latest/DevGuideCodingBasicsJSNI

[15] M. C. M. Ray Cromwell, "gwt-exporter," http://code.google.com/p/gwt-exporter/, 2012. [Online]. Available: http://code.google.com/p/gwt-exporter/

[16] K. L. Damkjer, "Obsessive camera direction (ocd) reference," http://www.gdsstudios.com/processing/libraries/ocd/reference/, 2009. [Online]. Available: http://www.gdsstudios.com/processing/libraries/ocd/reference/

[17] M. Niemi, "Traer's physics library to processing.js - notes," http://svbreakaway.info/tp.php, 2011. [Online]. Available: http://svbreakaway.info/tp.php#tpjs

[18] X, "processing adapted libraries," http://goo.gl/KzvxH1, 2013. [Online]. Available: http://goo.gl/KzvxH1

[19] J. T. Bernstein, "Traer.physics 3.0," http://murderandcreate.com/physics/, 2010. [Online]. Available: http://murderandcreate.com/physics/

[20] J. P. Charalambos, "Proscene description," http://code.google.com/p/proscene/, 2011. [Online]. Available: http://code.google.com/p/proscene/

Cesar Colorado nació en la ciudad de Bogotá en 1988. Se gradúa con el título de Ingeniero de Sistemas en la Universidad Nacional de Colombia - Sede Bogotá en 2011.
De 2010 a 2015, ha ejercido desarrollando aplicaciones web para el sector financiero. Desde 2010, ha estado en el grupo de investigación sobre computación gráfica RemixLab, perteneciente a la Facultad de Ingeniería de la Universidad Nacional de Colombia. En esta misma institución, es candidato actualmente al título de "Magister en Ingeniaría – Ingeniería de Sistemas y computación". Sus temas de interés para investigación incluyen gráficos para la web, realidad aumentada y gráficos 3d.

Jean Pierre Charalambos recibe en 1994 el título de Ingeniero Industrial de la Pontificia Universidad Javeriana Sede Bogotá. Recibe el título de "Magister en Ingeniería de Sistemas" en la Universidad Nacional de Colombia - Sede Bogotá y luego realiza un Doctorado en Software en la Universidad Politécnica de Cataluña que termina en 2008.
Ha ejercido como docente en la Pontificia Universidad Javeriana y la Universidad Nacional de Colombia , donde funda y dirige el grupo de investigación sobre computación gráfica RemixLab. Ha publicado varios artículos, asistidito a varios eventos científicos y ha sido director de varias tesis de postgrado, además de desarrollar el software para control de cámara Proscene. Sus temas de interés para investigación incluyen rendering en tiempo real, visualización científica, grádicos 3d y la interacción hombre-máquina.
El doctor Charalambos recibió un reconocimiento por parte de Qtopia Worldwide Developer Contest,Trolltech y Sharp Co en 2002 y ha sido miembro del grupo de investigación Bioengenium así como del Computer graphics Institute de la Vienna University of Technology.