GTW (Google Web Toolkit)

Jesús David Calle Calle [1] and Javier Trujillo Hernández [1]

[1] GMV Innovating solutions
jdcalle@gmvsolutions.es
[2] INDRA
jtrujillo@indra.com

Resumen.En las ultimas décadas internet ha experimentado un crecimiento exponencial lo que ha permitido que las tecnologías que se usan en internet crezcan al mismo ritmo. Una de esas tecnologías es el kit de herramientas que proporciona Google a los desarrolladores para que optimicen el diseño de sus webs según los estandares web. Diversas tecnologías han dominado el mercado del desarrollo web, aunque actualmente, Google propone el uso del lenguaje de programación Ajax o herramientas Ajax, que engloba el uso de los principales lenguajes de programación web (HTML, CSS3, JavaScript) además del objeto XMLHttpRequest. El Google web toolkit viene para solucionar las peculiaridades de cada navegador lo que va a facilitar a los desarrolladores la tarea de escribir y depurar el código de las paginas web.

Palabras clave: Google web toolkit; Desarrollo paginas web

Abstract. In the last decades the Internet has experienced an exponential growth which has allowed the technologies used in the Internet to grow at the same rate. One of these technologies is the toolkit provided by Google to developers to optimize the design of their websites according to web standards. Various technologies have dominated the web development market, although currently, Google proposes the use of Ajax programming language or Ajax tools, which includes the use of the main web programming languages (HTML, CSS3, JavaScript) in addition to the XMLHttpRequest object. The Google web toolkit comes to solve the peculiarities of each browser which will make it easier for developers to write and debug the code of web pages.

Keywords: Google web toolkit; Website development

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238

1 Introducción

1.1 Contexto

Para entender el importante papel que aporta la tecnología de GWT al desarrollo profesional de aplicaciones web, podemos analizar la evolución que ha seguido la propia Red.
Hace aproximadamente una década, nos encontrábamos con un escenario en que la Web había evolucionado de simples páginas estáticas hacia conjuntos de páginas dinámicas cuya lógica se ejecutaba en el lado del servidor. Este hecho había aportado interactividad a la relación de los usuarios con internet pero implicaba que, en resumidas cuentas, cada operación del usuario (cada clic) desembocaba en la carga de una nueva página web, con el consecuente tráfico de ficheros, carga de procesamiento del servidor, tiempos de espera del cliente, etc.
Paralelamente, las interfaces venían siguiendo dos tendencias principales: las páginas formadas por html puro y, en mayor o menor medida, estilos css; y páginas que además incluían toques de javascript o elementos escritos en Flash [1-5].
Con el paso del tiempo, la evolución en el dinamismo de las interfaces gráficas supuso el nacimiento del termino RIA (Rich Internet Application, aplicaciones de internet enriquecidas), aplicado a desarrollos con las plataformas Adobe Flash, Microsoft Silverlight y Oracle JavaFX, principalmente. El problema que presentan estas herramientas es la necesidad de que el usuario instale en su navegador un plugin que sea capaz de interpretar el contenido correspondiente a cada plataforma.
Como respuesta a estas herramientas se acuñó el término Ajax para denominar una forma de desarrollo que se nutría, fundamentalmente, de las tecnologías html, css, javascript, además de la piedra angular, el objeto XMLHttpRequest, para manejar la transferencia asíncrona de información entre la aplicación cliente y el servidor. Idealmente se resolvía el problema de la transferencia de páginas completas a cada clic, pero la diferente implementación de javascript por parte de cada fabricante trajo un problema mayor para los desarrolladores: por cada navegador, el desarrollador tenía que escribir una versión del código, lo que aumentaba la complejidad de los desarrollos, etapa de pruebas, depuración, etc.
En este contexto, Google decidió presentar en 2006 GWT (Google Web Toolkit).

1.2 ¿Qué es GWT?

Diversos autores, considerando javascript como el lenguaje ensamblador de la Web, definen GWT como un compilador cruzado (un traductor) de Java a javascript. Y en el fondo, eso es. Pero además, viene a solucionar las peculiaridades de la implementación propia de cada navegador, haciendo transparente esta problemática para el desarrollador, el cual escribe una única versión de su código Java, y obtiene una versión javascript adaptada a cada navegador.
Además, GWT nos proporciona un conjunto de elementos gráficos para el desarrollo de la interfaz, un sistema de llamada a funciones remotas para la comunicación entre la capa del cliente y el servidor, y una serie de herramientas que facilitan al desarrollador la labor de escribir y depurar el código.
Una ventaja importante de GWT es que el desarrollador, en principio, va a programar una aplicación final en javascript y puede que desconozca por completo dicho lenguaje, ya que el código que él escriba será Java.

1.3 ¿Qué no es GWT?

GWT es definido por algunos como un framework, pero no es exactamente un framework. Técnicamente, como indican sus propias siglas, es un toolkit, un conjunto de herramientas que se

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

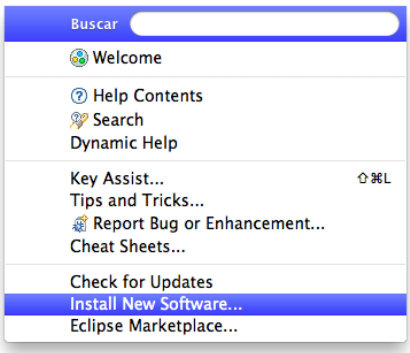
verán en profundidad durante este curso (aunque algunas, como el compilador, ya han sido nombradas).
De hecho es común ver aplicaciones empresariales que utilizan por un lado un framework, como Spring, y además hacen uso de GWT para, por ejemplo, la capa web y la transferencia asíncrona de información con el servidor.

2 Primeros pasos con GWT

Para familiarizarnos con el entorno de GWT vamos a estudiar un ejemplo sencillo de aplicación, la que Google nos construye cada vez que creamos un nuevo proyecto en Eclipse. Aunque no es obligatorio el uso de Eclipse (podemos utilizar GWT desde la línea de comandos) es conveniente utilizarlo, ya que nos aporta las ventajas de un entorno de desarrollo integrado (uno de los más populares en el mundo empresarial). Además, el plugin de GWT para Eclipse se adapta perfectamente al entorno.

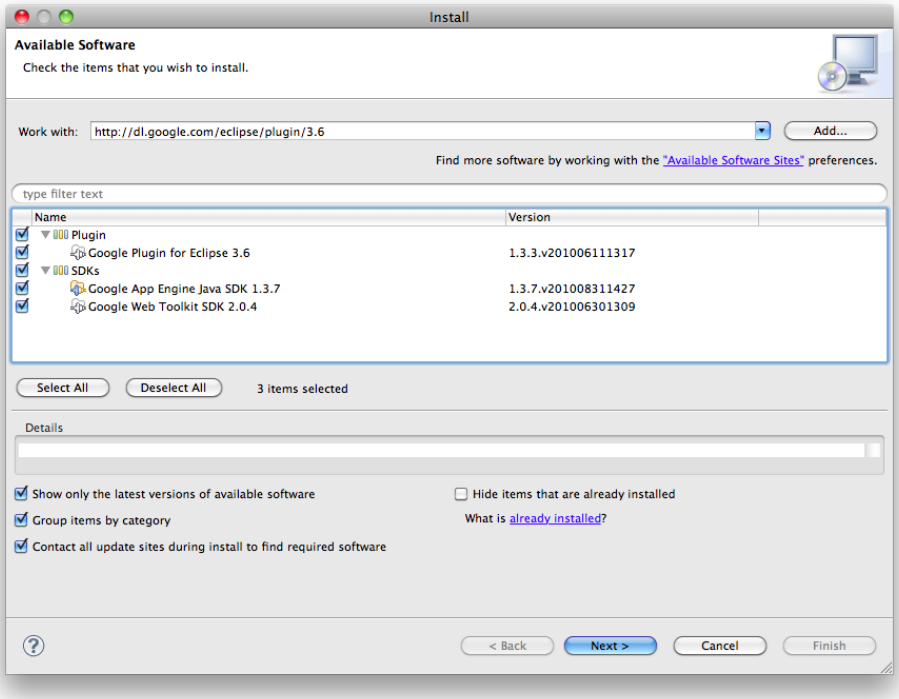
2.1 Instalación del plugin para Eclipse

El primer paso consiste en arrancar Eclipse e instalar el plugin de Google. Para ello, accederemos al menú “Help”, “Install new software...”.



En la ventana que aparece, escribiremos la URL donde se encuentra el plugin: http://dl.google.com/eclipse/plugin/3.6 (Esta dirección corresponde a la versión 3.6 de Eclipse, para otras versiones conviene consultar la dirección http://code.google.com/intl/en/eclipse/docs/download.html en la que aparecerán los enlaces correspondientes a cada versión)

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

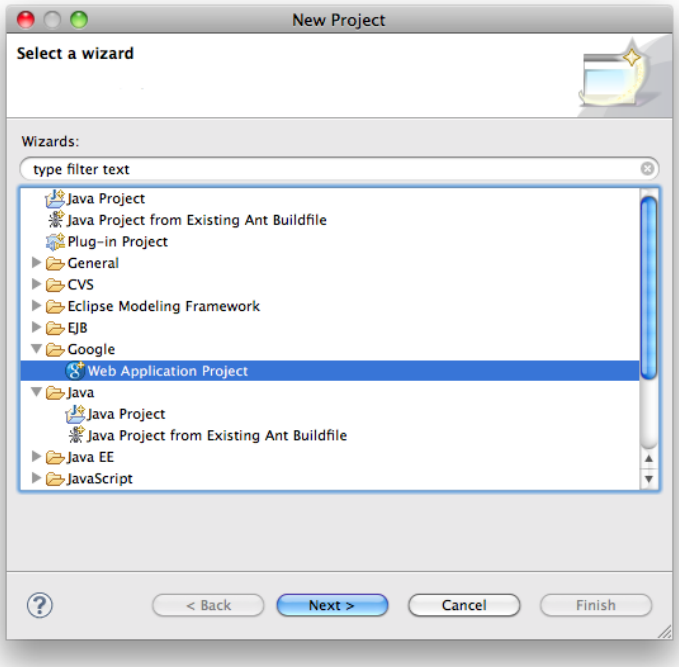


Cuando escribamos la URL, aparecerá el contenido del repositorio de Google. Seleccionamos todo lo que nos aparezca y vamos pulsando “Next” hasta que nos aparezca la licencia del software. Si estamos de acuerdo, seleccionamos la opción correspondiente y pulsamos “Finish”. Tras la instalación, Eclipse nos solicitará permiso para reiniciarse, momento tras el cual ya tendremos el IDE preparado para trabajar con GWT [6-10].

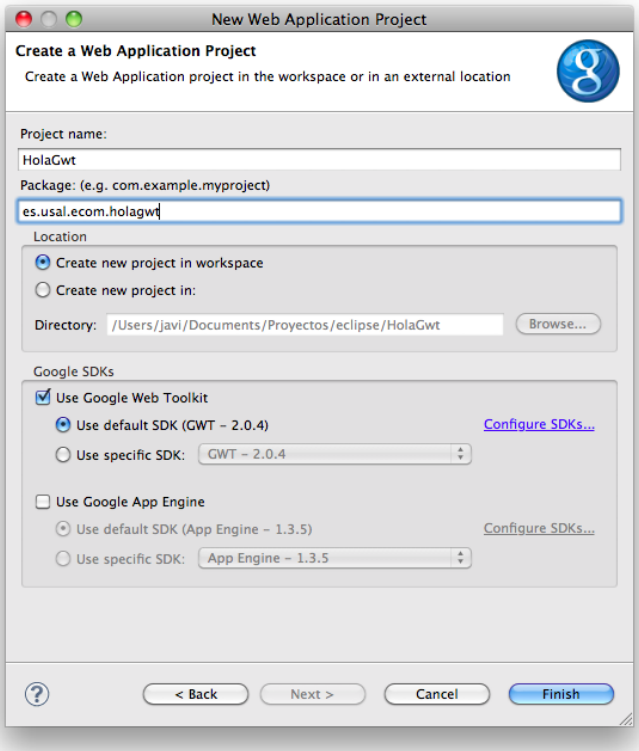
2.2 Creación del proyecto de ejemplo

Ya estamos listos para crear un nuevo proyecto. Seleccionaremos pues “File”, “New”, “Project...”. En la ventana que aparece, hay que desplegar la categoría “Google” y elegir el asistente para un nuevo “Web Application Project”.

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

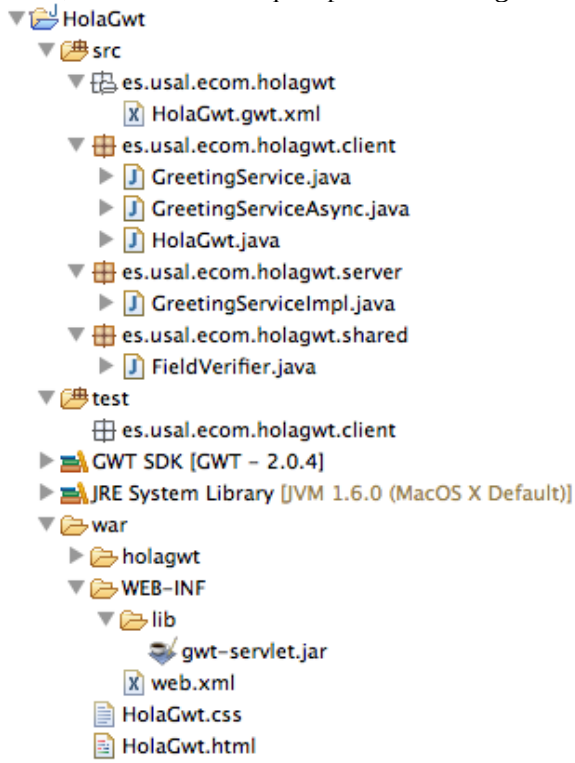


Hacemos clic en “Next” y a continuación configuramos los parámetros que deseemos del proyecto.



ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

Como se puede ver en la imagen, hemos nombrado al proyecto “HolaGwt”, y el paquete asimismo se llamará es.usal.ecom.holagwt
La opción “Use Google App Engine” vamos a dejarla desactivada. Dicha opción nos permitiría instalar y ejecutar la aplicación que creemos desde los servidores de Google, de forma que puede estar accesible desde cualquier ordenador conectado a internet.
Cuando pulsemos en “Finish”, tendremos en nuestro workspace de Eclipse un proyecto que se compone, fundamentalmente, de la estructura que aparece en la siguiente imagen.



Vamos a ver qué responsabilidad cumplen los distintos ficheros y directorios.
La división inicial se da en tres directorios principales:

- src: contiene el archivo de configuración del módulo (nombre_del_proyecto.gwt.xml), así como el código del proyecto, dividido en tres subpaquetes (client, shared y server)
- test: contiene el código que opcionalmente escribiremos para automatizar las pruebas de los distintos escenarios de nuestro proyecto
- war: contiene la aplicación web final, compuesta por los recursos estáticos y el código del servidor compilado

Analicemos con más detalle el contenido del directorio “src”. Debido a que hemos especificado como paquete base “es.usal.ecom.holagwt”, bajo el directorio “src” encontraremos el subdirectorio “es”; dentro, el subdirectorio “usal”... así hasta “holagwt”. Aquí encontramos el fichero HolaGwt.gwt.xml
Éste es el fichero de configuración base de todo proyecto GWT. No vamos a explicar aquí la totalidad del fichero, pero vamos a ver qué significan algunas líneas:

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
<entry-point class='es.usal.ecom.holagwt.client.HolaGwt'/>
```

Ésta define el punto de entrada de la aplicación, es decir, la clase ~.client.HolaGwt

```
<source path='client'/>
<source path='shared'/>
```

Éstas dos definen los directorios que contienen código java que será traducido a código javascript.
Si volvemos al árbol de directorios, vemos que aparte de "client" y "shared", tenemos también el directorio "server". El contenido de "server" no será traducido a javascript, sino que será compilado como código java y se ejecutará en el lado del servidor.
Vamos a empezar viendo el contenido del paquete "client". Aquí tenemos una clase (HolaGwt.java) y dos interfaces (GreetingService y GreetingServiceAsync).
HolaGwt.java es la clase principal del proyecto y, como hemos visto en el fichero de configuración, será el punto de entrada de la aplicación, a través de su método onModuleLoad(), cuya responsabilidad es la de crear la interfaz gráfica con la que se comunicará el usuario. También se encarga de crear un proxy para dialogar con el servidor:

```
private final GreetingServiceAsync greetingService =
              GWT.create(GreetingService.class);
```

Como se puede ver, el tipo de dato del objeto proxy corresponde con una de las interfaces que tenemos en el directorio, pero el argumento que se le pasa a la clase GWT a través del método create es la otra interfaz.
Esto es parte del mecanismo RPC (Remote Procedure Call, llamada a procedimiento remoto) de GWT: si queremos proporcionar un método remoto, que se ejecutará en el servidor, tenemos que crear dos interfaces de este modo:
La interfaz GreetingService.java es la que implementará la clase en el lado del servidor. Su contenido, en este caso, es el siguiente:

```
public interface GreetingService extends RemoteService {
       String greetServer(String name)
                     throws IllegalArgumentException;
}
```

La interfaz GreetingServiceAsync.java se llama igual que la anterior, utilizando el sufijo "Async"

```
public interface GreetingServiceAsync {
       void greetServer(String input, AsyncCallback<String> callback)
                     throws IllegalArgumentException;
}
```

Además, es bastante parecida a la anterior, salvo en los siguientes aspectos:

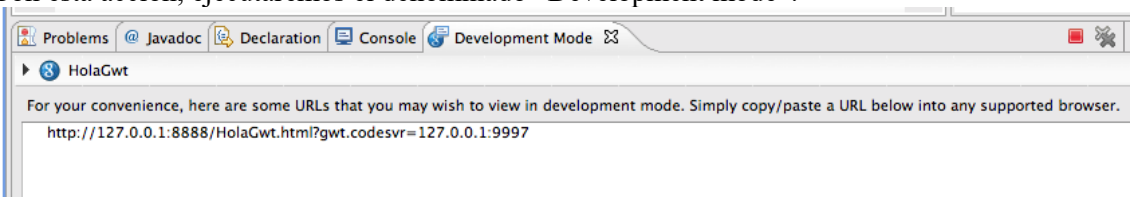
- no hereda de "RemoteService"
- el tipo de retorno es "void"
- el método tiene un parámetro adicional, "AsyncCallback<String> callback"

Esta peculiar estructura se debe al funcionamiento del mecanismo RPC. La llamada al método realmente es asíncrona, de forma que el cliente se comunica con la interfaz "GreetingServiceAsync", pasándole los parámetros deseados, además de una serie de funciones (que veremos en el siguiente ejemplo) a las que GWT llamará cuando el método termine su ejecución y decida respondernos.
La implementación del servicio remoto correspondiente a estas interfaces la hace el fichero "GreetingServiceImpl.java", del paquete "server".
En el siguiente apartado veremos detenidamente el código de cada uno de estos ficheros a través de un ejemplo concreto. Ahora vamos a probar a compilar y ejecutar la aplicación.

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

Para ello, en primer lugar, pulsaremos sobre el botón con forma de caja de herramientas (el logo de GWT) que aparece en la barra de herramientas de Eclipse. Cuando termine el proceso, podemos proceder a lanzar la aplicación, para lo que pulsaremos el botón "Run":

Con esta acción, ejecutaremos el denominado "Development mode":



En el que se nos muestra una URL. Si la seleccionamos, copiamos y abrimos en un navegador, veremos la aplicación de ejemplo.

Ésta es una de las formas de ejecución que tenemos. Alternativamente, al haber compilado la aplicación previamente, tendremos en la carpeta "war" la aplicación lista para que la llevemos a nuestro servidor de aplicaciones. En Tomcat, por ejemplo, podemos pegar la carpeta "war" dentro de "webapps", renombrarla a nuestro gusto y acceder a ella de la forma habitual.

Una tercera opción para ejecutar nuestra aplicación es hacerlo a través del Google App Engine (el icono con forma de "avión" que aparece junto al botón de la caja de herramientas). Como se ha mencionado previamente, ésta es una opción que nos brinda Google para desplegar la aplicación en sus servidores.

3 Caso práctico 1: Ecuaciones

Vamos a construir una aplicación GWT que resuelve ecuaciones de segundo grado. En el lado del cliente, construiremos una interfaz en la que pediremos los coeficientes de la ecuación al usuario, y le mostraremos el resultado. En el lado del servidor recibiremos los datos, los procesaremos y enviaremos la doble solución al cliente.

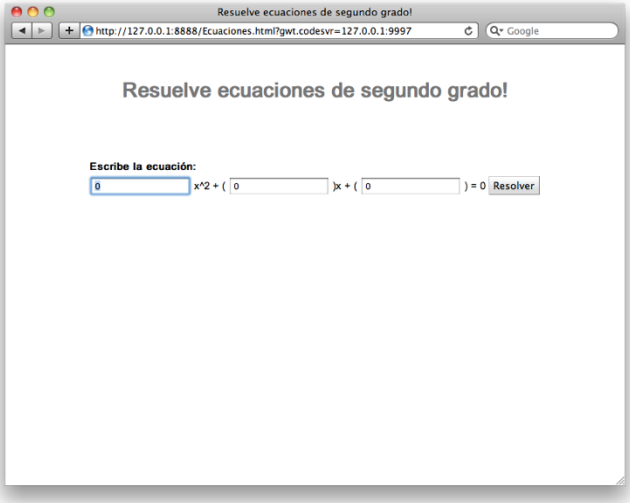
La base del proyecto será la misma que en el ejemplo anterior, es decir, el código que nos genera GWT cada vez que creamos un nuevo proyecto en Eclipse, por lo que los pasos iniciales serán los mismos.

En este caso vamos a nombrar al proyecto "Ecuaciones" y el nombre del paquete base será "es.usal.ecom.ecuaciones". Una vez creado el proyecto, lo seleccionaremos en el "Package explorer", haremos clic en el menú "Project", y a continuación, "Properties", "Resource", "Text file encoding". Seleccionaremos "Other" y "UTF-8". Esto es importante, ya que de lo contrario no se mostrarán de forma correcta las tildes ni algunos caracteres especiales en la aplicación final.

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

La aplicación final va a tener un aspecto similar a la siguiente figura:



Por tanto, vamos a ponernos manos a la obra con el código. En primer lugar, modificaremos el fichero host "Ecuaciones.html" que se encuentra bajo el directorio "war". Aquí podemos editar el título de la página, y lo que nos parezca oportuno, pero lo importante está en la tabla que aparece al final del fichero, que tendrá que tener un aspecto final similar a:

```
<table align="center">
  <tr>
    <td colspan="2" style="font-weight:bold;">
      Escribe la ecuaci&oacute;n:
    </td>
  </tr>
  <tr>
    <td id="AContainer"></td>
    <td id="x2Container"></td>
    <td id="BContainer"></td>
    <td id="xContainer"></td>
    <td id="CContainer"></td>
    <td id="ceroContainer"></td>
    <td id="resolverContainer"></td>
  </tr>
  <tr>
    <td colspan="2" style="color:red;" id="errorContainer"></td>
  </tr>
</table>
```

Como se puede ver, hemos modificado el código de forma que ahora tenemos una serie de celdas con un identificador que utilizaremos más adelante.

Ya hemos terminado con este fichero, y con lo que teníamos que modificar en el directorio "war". Vamos ahora con "src" y, en primer lugar, vamos con el lado del cliente (paquete "~.client"). Abrimos el fichero "Ecuaciones.java" y eliminamos todo el código de la función "onModuleLoad()". En esta función escribiremos los siguientes bloques de código, que vamos a ir explicando:

```
// Creamos los widgets que necesitamos:
final TextBox A = new TextBox();
A.setText("0");
final Label x2 = new Label();
x2.setText("x^2 + (");
```


ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
final TextBox B = new TextBox();
B.setText("0");
final Label x = new Label();
x.setText(")x + (");

final TextBox C = new TextBox();
C.setText("0");
final Label cero = new Label();
cero.setText(") = 0");
final Button resolver = new Button("Resolver");
final Label errorLabel = new Label();
```

Para el primer coeficiente, creamos una caja de texto donde el usuario introducirá el valor de dicho coeficiente. Lo iniciamos a 0. Después creamos una etiqueta que acompañará a la caja anterior. Esto lo haremos para los tres coeficientes de la ecuación. Después creamos un botón que, cuando sea pulsado, enviará los datos al servidor para su procesamiento. Por último creamos una etiqueta vacía que utilizaremos en caso de que el usuario introduzca valores incorrectos (como letras, cuando esperamos números).

Seguimos con más código:

```
// Los insertamos en la web:
RootPanel.get("AContainer").add(A);
RootPanel.get("x2Container").add(x2);
RootPanel.get("BContainer").add(B);
RootPanel.get("xContainer").add(x);
RootPanel.get("CContainer").add(C);
RootPanel.get("ceroContainer").add(cero);
RootPanel.get("resolverContainer").add(resolver);

RootPanel.get("errorContainer").add(errorLabel);
```

Ahora lo que hemos hecho es insertar los objetos que acabamos de crear en los contenedores que definimos previamente en el fichero “Ecuaciones.html”, de forma que GWT los pinte en el sitio indicado en ese fichero.

```
// Ponemos el foco de la aplicación sobre el primer campo, A:
A.setFocus(true);
A.selectAll();
```

Con estas líneas hacemos que el foco de la aplicación (donde se sitúa el cursor inicialmente, cuando la aplicación carga) recaiga en el primer coeficiente de la ecuación. Además, seleccionamos todo el contenido inicial del campo (para que cuando el usuario escriba, se elimine el 0 que pusimos por defecto).

```
// Creamos el popup que muestra la solución:
final DialogBox solucion = new DialogBox();
solucion.setText("Solución");
solucion.setAnimationEnabled(true);
final Button cerrarSolucion = new Button("Cerrar");

// Etiquetas del popup, la ecuación y la solución:
final Label ecuacionLabel = new Label();
final HTML  solucionLabel = new HTML();

// Panel vertical que va a albergar los items anteriores:
VerticalPanel panel = new VerticalPanel();
panel.add(new HTML("<strong>Ecuaci&oacute;n</strong>"));
panel.add(ecuacionLabel);
panel.add(new HTML("<br />"));
panel.add(new HTML("<strong>Soluci&oacute;n</strong>"));
```

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
panel.add(solucionLabel);
panel.add(cerrarSolucion);
solucion.setWidget(panel);
```

La solución la vamos a mostrar a través de una ventana ("popup") que aparecerá cuando el usuario pulse sobre "Resolver". Por tanto en las primeras líneas empezamos creando la ventana, poniendo como título "Solución", haciendo que su aparición sea animada, y añadiendo a la misma un botón para cerrarla [11-15].

Después, creamos dos etiquetas, una que presentará la ecuación mediante una cadena de caracteres, y otra que mostrará la solución. Como la solución la enviaremos como código html, la etiqueta tendrá ese formato especial.

Por último creamos un panel vertical, objeto que nos permite apilar elemento verticalmente cada uno debajo del anterior. Le añadiremos una serie de etiquetas y finalmente, el botón de cerrar. Por último, configuraremos el panel como contenido de la ventana que muestra la solución.

A continuación tenemos que crear las funciones que manejen el comportamiento de los botones que hemos creado.

```
// Manejador del botón cerrarSolucion:
cerrarSolucion.addClickHandler(new ClickHandler() {
    public void onClick(ClickEvent event) {
        solucion.hide();
        resolver.setEnabled(true);
        resolver.setFocus(true);
    }
});
```

El botón "Cerrar" de la ventana de la solución es muy sencillo, puesto que lo único que hace es ocultar la ventana, habilitar el botón de "Resolver" (que deshabilitaremos más adelante, cuando sea pulsado y se muestre ésta ventana) y devolver el foco a dicho botón.

Para el botón "Resolver" es más complicado, puesto que tenemos que recoger los datos que ha escrito el usuario, validarlos, enviarlos al servidor y recoger la respuesta, considerando que además pueda haber fallos de comunicación, entradas incorrectas, etc.

La forma de hacerlo será definiendo una clase "inline", esto es, dentro de otra clase (nuestra clase principal), que sirva de respuesta tanto para el clic en "Resolver" como para cuando el usuario, dentro de una caja de texto, pulse intro.

A continuación vamos a ir viendo el código de esta clase.

```
class ResolucionHandler implements ClickHandler, KeyUpHandler {
    // Método manejador de los clics sobre el botón Resolver
    public void onClick(ClickEvent event) {
        resolver();
    }

    // Método manejador de los "intros" del usuario
    public void onKeyUp(KeyUpEvent event) {
        if (event.getNativeKeyCode() == KeyCodes.KEY_ENTER) {
            resolver();
        }
    }
```

Hasta aquí hacemos que la clase implemente las interfaces necesarias para responder al clic sobre "Resolver" y al intro sobre las cajas de texto. En ambos casos vamos a llamar a la función que sigue.

```
    // Método encargado de la resolución:
    private void resolver() {
        // Validamos las entradas
        errorLabel.setText("");
        String a_ = A.getText();
```


ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
            String b_ = B.getText();
            String c_ = C.getText();
            if (!FieldVerifier.isValidFloat(a_)) {
                errorLabel.setText("El primer campo no contiene un número");
                return;
            }
            if (!FieldVerifier.isValidFloat(b_)) {
                errorLabel.setText("El segundo campo no contiene un número");
                return;
            }
            if (!FieldVerifier.isValidFloat(c_)) {
                errorLabel.setText("El tercer campo no contiene un número");
                return;
            }
```

Ahora vaciamos la etiqueta reservada para los errores, por si tenemos que utilizarla. Recogemos como cadenas las entradas de los usuarios y utilizamos la clase "FieldVerifier" (que comparten tanto el lado cliente como el lado servidor) para comprobar que las entradas son correctas. Más adelante veremos el código de esta clase.

```
            // Nos comunicamos con el servidor:
            resolver.setEnabled(false);
            ecuacionLabel.setText(
              "(" + a_ + ")x^2 + (" + b_ + ")x + (" + c_ + ") = 0");
            solucionLabel.setText("");
```

Con estas líneas hemos desactivado el botón "Resolver" para que el usuario no lo pulse mientras se muestra la ventana con la solución. Además, hemos creado una cadena con la ecuación que se va a resolver y hemos vaciado la cadena de respuesta, para poner en ella lo que nos responda el servidor.

```
            greetingService.resolver(a_, b_, c_,
                    new AsyncCallback<String>() {
                        public void onFailure(Throwable caught) {
                            // Si hay un error de comunicación, lo mostramos:
                            solucion.setText("Fallo en la comunicación");
                            solucion.setHTML(SERVER_ERROR);
                            solucion.center();
                            cerrarSolucion.setFocus(true);
                        }
                        public void onSuccess(String result) {
                            solucion.setText("Solución");
                            solucionLabel.setHTML(result);
                            solucion.center();
                            cerrarSolucion.setFocus(true);
                        }
                    });
        }
    }
```

Ahora llevamos a cabo la comunicación con el servidor. Para ello, llamamos al método resolver que veremos más adelante, pasándole los tres coeficientes y un objeto "AsyncCallback" que implementa el comportamiento correspondiente tanto al funcionamiento correcto, como al incorrecto debido a un fallo en la comunicación con el servidor. En ambos casos añadimos una serie de etiquetas a la ventana. Concretamente, cuando la comunicación es correcta, añadimos la solución al cálculo.

Por último, con el siguiente código, creamos un objeto de la clase que acabamos de escribir, y hacemos que sea el manejador de "Resolver" y de las cajas de texto:

```
    ResolucionHandler resolucion = new ResolucionHandler();
    resolver.addClickHandler(resolucion);
```

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
A.addKeyUpHandler(resolucion);
B.addKeyUpHandler(resolucion);
C.addKeyUpHandler(resolucion);
```

Ya tenemos los cambios correspondientes al fichero Ecuaciones.java, vamos ahora a ver qué hemos modificado en las interfaces correspondientes al método remoto. Empecemos con GreetingService.java, en el que vamos a modificar el prototipo del método que declara para que figure así:

```
String resolver(String A, String B, String C) throws IllegalArgumentException;
```

Como vimos en el primer ejemplo, tenemos que cambiar también la interfaz GreetingServiceAsync.java; el prototipo en este caso quedará así:

```
void resolver(String A, String B, String C, AsyncCallback<String> callback)
                        throws IllegalArgumentException;
```

Al cambiar las interfaces, tenemos que cambiar también el código de la clase que implementa el servicio. Por tanto, en el paquete ~.server, en la clase GreetingServiceImpl.java, dejaremos un único método, el siguiente:

```
public String resolver(String A, String B, String C)
  throws IllegalArgumentException {
    if (!FieldVerifier.isValidFloat(A)) {
        throw new IllegalArgumentException(
                "El primer campo no contiene un número");
    }
    if (!FieldVerifier.isValidFloat(B)) {
        throw new IllegalArgumentException(
                "El segundo campo no contiene un número");
    }
    if (!FieldVerifier.isValidFloat(C)) {
        throw new IllegalArgumentException(
                "El tercer campo no contiene un número");
    }

    Float a = 0f;
    Float b = 0f;
    Float c = 0f;

    try {
        a = Float.parseFloat(A);
        b = Float.parseFloat(B);
        c = Float.parseFloat(C);
    }
    catch (Exception e) {}

    Double x1 = ((-b)+(Math.sqrt((b*b)-(4*a*c)))) / 2*a;
    Double x2 = ((-b)-(Math.sqrt((b*b)-(4*a*c)))) / 2*a;

    return "<strong>" + x1 + "</strong> y <strong>" + x2 + "</strong>";
}
```

El código, en resumen, vuelve a comprobar la validez de los campos que nos llegan a través de la clase que comparte con el cliente (FieldVerifier.java, que veremos a continuación), calcula las dos soluciones a la ecuación de segundo grado, y construye una cadena html con la solución, que será la que le llegue al cliente.

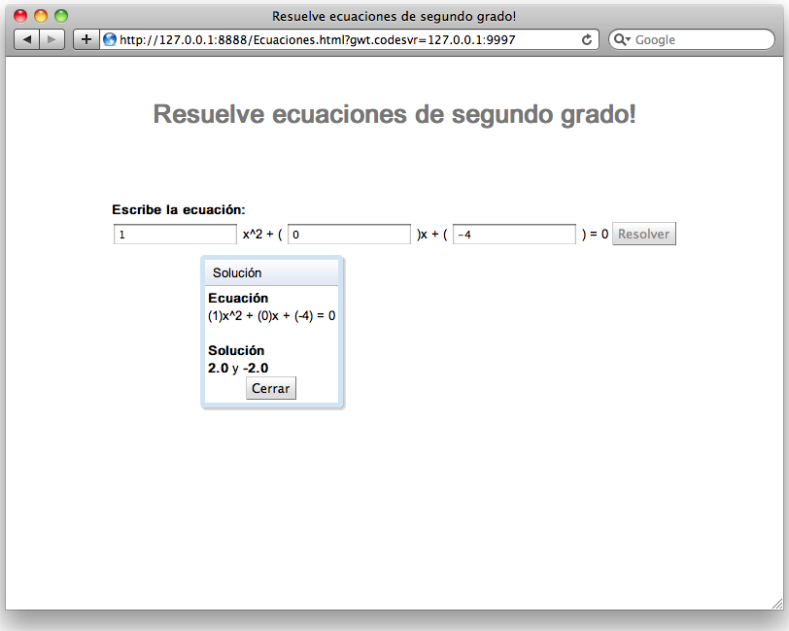
Para terminar, explicaremos el código de FieldVerifier.java, clase que sólo contendrá un método:

```
public static boolean isValidFloat(String campo) {
    if (campo == null) {
        return false;
    }
    try {
```


ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
            Float.parseFloat(campo);
        }
        catch (Exception e) {
            return false;
        }
        return true;
    }
```

Esta clase intenta convertir cada entrada del usuario en un número real. Si encuentra algún fallo en el proceso, devuelve "false". En caso contrario, "true", permitiendo continuar con el cálculo.
La razón por la que la verificación se realiza en ambos lados es por seguridad en el servidor y por comodidad en el cliente. Del lado del cliente, permite ahorrarnos una llamada al servidor si el usuario se ha saltado las restricciones en las entradas, pero del lado del servidor obliga a que se vuelvan a comprobar, por si el usuario se las ha saltado intencionadamente [16-20].
Ahora podemos proceder con la compilación, utilizando el icono de la caja de herramientas de Google. Si todo ha ido bien, pulsamos el botón "Run" y nos aparecerá en la parte inferior de la pantalla la URL en la que tenemos accesible la aplicación (en la ventana "Development mode" de Eclipse). Abrimos un navegador, accedemos a dicha URL, y ya tendremos nuestra aplicación funcionando:



4 Bibliotecas para GWT

Google API libraries for Google Web Toolkit es una colección de bibliotecas que Google nos proporciona de forma gratuita, su cometido es encapsular la complejidad del código Javascript de los principales APIs de Google, superponiendo una fachada para su uso en Java como si de una biblioteca más se tratase. Estas bibliotecas permiten a los desarrolladores embeber rápida y fácilmente los contenidos de los típicos APIs de Google en sus aplicaciones GWT.
Los APIs para los que disponemos estas bibliotecas son Gears, Search, Maps, Chart Tools, Language, Gadgets y AjaxLoader.

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

A continuación describiremos estos APIs, lo que nos ofrece cada uno y realizaremos una breve la explicación de un ejemplo de los APIs más relevantes, estos ejemplos se adjuntan en la documentación del curso.

4.1 Search

El API de Google para búsquedas permite añadir la funcionalidad estrella de Google, el buscador, a un sitio web. Puedes insertar un cuadro de búsqueda dinámico y sencillo y mostrar los resultados de la búsqueda en tus propias páginas web o bien utilizar los resultados de una forma programática e innovadora.
Ver ejemplo: SearchAPIExample

Editar el fichero SearchAPIExample.xml, para que el proyecto utilice la biblioteca Search. Para utilizar esta biblioteca en un servidor remoto (distinto de localhost) es necesario obtener una key que es gratuita, la key es única para cada url.

```
<inherits name='com.google.gwt.search.Search' />
<!--
        If you want to deploy this application outside of localhost,
        you must obtain a Google AJAX Search API key at:
        http://code.google.com/apis/search/signup.html
        append &amp;key=ABC to the string below, replacing ABC with the key
        obtained from the site above.
        -->
<script src="http://www.google.com/uds/api?file=uds.js&amp;v=1.0&amp;gwt=1"/>
```

El programa que hemos creado como ejemplo crea un buscador de webs e imágenes. El procedimiento es crear los elementos que queremos incluir en nuestro buscador, en este caso un WebSearch y un ImageSearch, y añadirlos a un objeto SearchControlOptions, a partir del cual podemos crear el SearchControl que es el widget que deseábamos crear. Éste lo podemos añadir a nuestra interfaz final.

```
public void onModuleLoad() {

//Creamos los elementos que queremos incluir
    WebSearch webSearch = new WebSearch();
    webSearch.setResultSetSize(ResultSetSize.LARGE);
    ImageSearch imageSearch = new ImageSearch();
```


ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

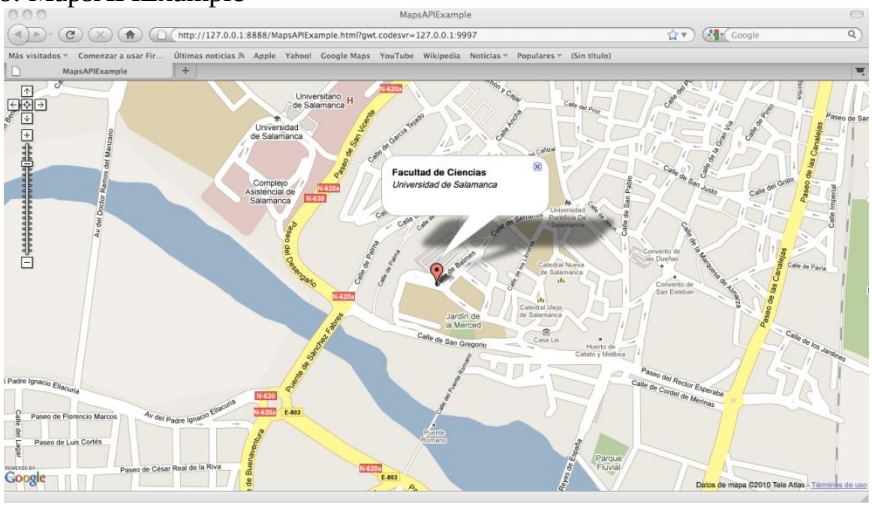
```
//Creamos un SearchControlOptions, al que añadimos
//los objetos creados anteriormente
    SearchControlOptions options = new SearchControlOptions();
    options.add(webSearch);
    options.add(imageSearch, ExpandMode.OPEN);

//Utilizando el SearchControlOptions podemos crear el control que queríamos
    final SearchControl control = new SearchControl(options);
//Podemos hacer una búsqueda
    control.execute("bisite");
//Y finalmente lo añadimos a un panel visible por el usuario
    RootPanel.get().add(control);
  }
```

Se pueden utilizar más tipos de búsquedas, como videos, libros o noticias. La biblioteca también permite recoger los resultados para procesarlos y presentarlos al usuario de distintas formas.
Para más información consultar el API completo de la biblioteca en http://gwt-google-apis.googlecode.com/svn/javadoc/search/1.1/index.html

4.2 Maps

Google Maps API, permite añadir el archiconocido widget de Google Maps y personalizar infinidad de sus funcionalidades o crear otras nuevas. Probablemente, es uno de los más interesantes de cara al desarrollo de herramientas atractivas al usuario final, tanto por su usabilidad, como por la imagen que genera una aplicación con tanta aceptación en nuestra aplicación.
Ver ejemplo: MapsAPIExample



Editar el fichero MapsAPIExample.xml, para que el proyecto utilice la biblioteca Maps.

```
<inherits name='com.google.gwt.maps.GoogleMaps' />
```

Para utilizar esta biblioteca en un servidor remoto (distinto de localhost) es necesario obtener una key que es gratuita, al igual que para la biblioteca Search.
Hay dos formas de utilizar la key, la primera es utilizarla en la llamada al API de Maps de la siguiente forma:

```
// “KEY” sería la key que se nos ha proporcionado para nuestra URL.
Maps.loadMapsApi(“KEY”, "2", false, new Runnable() {
public void run() {
            //Crear el map
      }
});
```

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

La segunda posibilidad es similar a la utilizada en el ejemplo del API Search. Luego, añadiríamos a nuestro fichero de configuración xml el siguiente código

```
	<!-- “KEY” sería la key que se nos ha proporcionado para nuestra URL. -->
	<script src="http://maps.google.com/maps?file=api&amp;v=2&amp;sen-
sor=true_or_false&amp;key=”KEY”" type="text/javascript"></script>
```

En nuestro ejemplo para este API, pretendemos crear un mapa centrado en la Facultad de Ciencias de la Universidad de Salamanca. Sobre este punto pondremos un marcador, el cual al accionarse creará una burbuja con un texto en su contenido. Lo describimos paso a paso.

```
	// Creamos un objeto LatLng con la latitud-longitud de salamanca
	LatLng latLonFac = LatLng.newInstance(40.960743,-5.670259);
	// Creamos el objeto mapa utilizando como parametros la posición a centrar y
el zoom inicial
	final MapWidget map = new MapWidget(latLonFac, 2);
	map.setSize("100%", "100%");
	// Añadimos el control del zoom y del tipo de mapa (mapa/satélite/hibrido)
	map.addControl(new LargeMapControl());
	map.addControl(new MapTypeControl());
	// Añadimos un Marker en la posición que queramos, en nuestro caso en la Fa-
cultad de Ciencias
	final Marker marker = new Marker(latLonFac);
	map.addOverlay(marker);
	// Configuramos el Marker para que al hacer click aparezca la burbuja de texto
	marker.addMarkerClickHandler(new MarkerClickHandler(){
	@Override
	public void onClick(MarkerClickEvent event) {
			map.getInfoWindow().open(marker,new InfoWindowContent("<b>Fac-
ultad de Ciencias</b><br><i>Universidad de Salamanca</i>"));
		}
	});
	// Añadimos el widget del mapa a un panel visible por el usuario
	RootLayoutPanel.get().add(map);
```

El API permite gran cantidad de posibilidades, como la utilización de Streetview, capturar eventos sobre cualquier Marker o geocodificación.

Para más información consultar el API completo de la biblioteca en http://gwt-google-apis.googlecode.com/svn/javadoc/maps/1.1/index.html

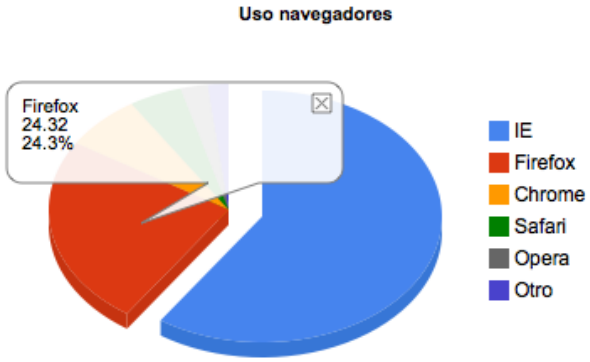
2.2. Chart tools

También conocido como Visualization, nos permite añadir distintos tipos de gráficos dinámicos a nuestra aplicación. Se trata de gráficos que podemos personalizar y adaptar a nuestras necesidades.

Este API puede resultar muy interesante de cara a la creación de herramientas enfocadas al Business Intelligence tan de moda en nuestros días. Un ejemplo de uso exitoso de este API lo encontramos en la herramienta Google Analytics.

Ver ejemplo: ChartAPIExample

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc



Editar el fichero ChartAPIExample.xml, para que el proyecto utilice la biblioteca Chart.

```
<inherits name='com.google.gwt.visualization.Visualization'/>
```

El ejemplo concreto que hemos realizado representa en un PieChart las estadísticas de uso de navegadores. Veamos una descripción del código, ya que puede resultar un poco más complejo que los dos anteriores.

Primeramente, para construir un gráfico, independientemente del tipo que sea éste, necesitamos una tabla con los datos y objeto de opciones dependiente del gráfico.

Para facilitarnos el trabajo con la tabla de datos, el API nos proporciona la clase AbstractDataTable. En el ejemplo hemos creado una función que devuelve el objeto con todo el contenido.

```
private AbstractDataTable createTable() {
        DataTable data = DataTable.create();
        data.addColumn(ColumnType.STRING, "Browser");
        data.addColumn(ColumnType.NUMBER, "% Usuarios");

        data.addRows(6);
        data.setValue(0, 0, "IE");
  ....
        data.setValue(5, 0, "Otro");
        data.setValue(5, 1, 1.89);
        return data;
    }
```

En cuanto al fichero de opciones, completamos un objeto Options (del paquete PieChart, ver los imports), con las opciones que deseemos sobre las se nos ofrecen.

```
private Options createOptions() {
        Options options = Options.create();
        options.setWidth(400);
        options.setHeight(240);
        options.set3D(true);
        options.setTitle("Uso navegadores");
        return options;
    }
```

Tanto el uso de AbastractDataTable, como de Options es bastante simple y aparece descrito en el API.

Una vez que tenemos los datos y las opciones nos falta crear el gráfico, para ello tendremos que cargar el API Chart de Google, esto se hace dinámicamente, y una vez cargado se crea el gráfico.

```
// Crear un callback para llamarlo cuando se haya
// cargado el API (ojo sólo se crea)
Runnable onLoadCallback = new Runnable() {
public void run() {
            Panel panel = RootPanel.get();
// Crear un gráfico de pastel/quesitos
```

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
                PieChart pie = new PieChart(createTable(), createOptions());
        panel.add(pie);
        }
};
// Cargar el API de visualizacion y pasarle el callback a llamar
VisualizationUtils.loadVisualizationApi(onLoadCallback,PieChart.PACKAGE);
```

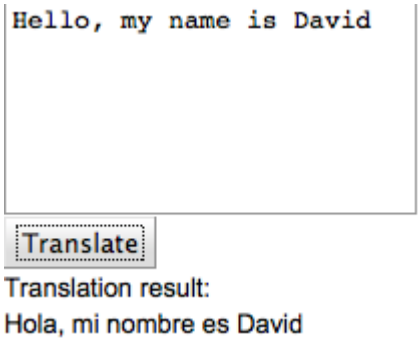
Existen distintos gráficos disponibles dentro del API, desde el típico gráfico de barras a un Geo-Map, es recomendable bucear a través de la galería porque se van añadiendo nuevos, y nuevas funcionalidades a los existentes. La dirección de la galería es http://code.google.com/intl/es-ES/apis/visualization/documentation/gallery.html

Para más información consultar el API completo de la biblioteca en http://gwt-google-apis.googlecode.com/svn/javadoc/visualization/1.1/index.html

4.3 Language

Con el API Language, puedes traducir y detectar el idioma de bloques de texto. Este API puede permitirnos dar soporte instantáneo a la Internacionalización o simplemente crear un traductor en nuestra aplicación.

Ver ejemplo: LanguageAPIExample



Editar el fichero LanguageAPIExample.xml, para que el proyecto utilice la biblioteca Language.

```
<inherits name='com.google.gwt.language.Language' />
```

El ejemplo concreto que hemos realizado es un pequeño traductor, este traduce de inglés a español. Veamos la descripción del código.

Lo primero que hacemos es cargar el API, no es recomendable hacer llamadas al API antes de hacer la llamada a loadTranslation. La forma de cargar el API es haciendo la llamada a la función y pasándole como argumento un objeto que implemente la interfaz Runnable, este caso es muy parecido al del API Charts.

```
// Cargamos el API Language Translation
LanguageUtils.loadTranslation(new Runnable() {
public void run() {
                RootPanel.get().add(createTranslationPanel());
        }
});
```

En el siguiente código vemos las líneas más destacadas de este ejemplo, donde se muestra como realizar una petición de traducción. Esto lo hacemos por medio de una llamada asíncrona y cuando recibimos la respuesta recogemos el resultado y se lo presentamos al usuario.

```
Translation.translate(transArea.getText(), Language.ENGLISH, Language.SPANISH,
        new TranslationCallback() {
@Override
        protected void onCallback(TranslationResult result) {
                if (result.getError() != null) {
```


ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
                        outputLabel.setText(result.getError().getMessage());
        } else {
                        outputLabel.setText(result.getTranslatedText());
                }
        }
        });
```

Para más información consultar el API completo de la biblioteca en http://gwt-google-apis.googlecode.com/svn/javadoc/language/1.1/index.html

4.4 Gadgets

Los Gadgets son simplemente aplicaciones HTML y JavaScript que podemos utilizar en páginas web u otras aplicaciones. Esta biblioteca trata de simplificar el desarrollo de estos mediante GWT. Una vez terminemos el desarrollo y compilemos el Gadget, éste está listo para ser publicado.
Los Gadgets son muy conocidos porque están presentes en herramientas como iGoogle y OpenSocial.

4.5 Gears

Este API nos permite crear aplicaciones web más poderosas añadiendo nuevas funcionalidades a nuestro navegador (es necesario instalar el plugin Gears en el cliente). Las principales ventajas que aporta es que permite la interacción entre el Escritorio del usuario y el navegador, almacenar datos localmente en una base de datos para el navegador y ejecutar hilos javascript en segundo plano para mejorar el rendimiento de nuestras aplicaciones.

4.6 Ajax loader

Simplemente permite a las aplicaciones controlar cuando un Google API es cargado y cambiar sus parámetros en tiempo real.
Estos APIs se pueden usar individualmente o de forma combinada, por ejemplo podemos utilizar la salida de una consulta y representarla en con Google Maps, o con un gráfico de Google Chart Tool.
Otro punto a tener en cuenta, es la continua evolución de estos, cada release liberada incluye nuevas características, y se solucionan problemas encontrados tanto por desarrolladores, como por la amplia comunidad de usuarios.
Se puede encontrar más información actualizada, así como ejemplos, preguntas frecuentes y su API en “http://code.google.com/p/gwt-google-apis/”

5 Manejar los eventos “atrás”, “adelante” y “actualizar” del navegador

Llegados a este punto nos encontramos con un problema debido a que la vista de la aplicación se crea y utiliza únicamente en el servidor, para detectarlo basta con hacer clic en atrás, adelante o actualizar en el browser y vemos como, o bien no sucede nada o bien perdemos lo poco o mucho que hubiéramos hecho.
Esto se debe a que no existe más que una forma de mantener los eventos realizados por el usuario durante el tiempo que ha estado utilizando la aplicación web, utilizando la url. En aplicaciones con continua interacción con el servidor, ésta va cambiando, pero en aplicaciones actuales que utilizan AJAX, Javascript, flash... no existe ningún histórico, ni actualización de url.
Es bastante familiar para la mayoría de nosotros encontrarnos con largas urls en las webs de Google (un ejemplo claro es GMail), esto es debido a que en ellas está contenida la información para reconstruir nuestro historial. Esta es la solución que se ha adoptado también en GWT.

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

La solución es simple, podemos mantener cualquier cadena de caracteres en la url, siempre que esté detrás de un símbolo '#', de escribir esto en la url se encarga GWT utilizando un método de clase, concretamente History.newItem("cadena_de_caracteres"). Con esto, se actualiza la url, ahora nos falta implementar un método que nos reconstruya la interfaz anterior, para ello crearemos una clase que implemente la interfaz ValueChangeHandler y por consiguiente su método no implementado, onValueChange [21-25].
Para ver un caso de uso, reutilizaremos el ejemplo anterior del API search y cambiaremos el código de tal forma que soporte el historial.

```
public class HistoryExample implements EntryPoint {

    public void onModuleLoad() {
        // Creamos los elementos que queremos incluir
        WebSearch webSearch = new WebSearch();
        webSearch.setResultSetSize(ResultSetSize.LARGE);
        ImageSearch imageSearch = new ImageSearch();

        // Creamos un SearchControlOptions,
        //al que añadimos los objetos creados
        // anteriormente
        SearchControlOptions options = new SearchControlOptions();
        options.add(webSearch);
        options.add(imageSearch, ExpandMode.OPEN);
        options.setDrawMode(DrawMode.TABBED);

        // Utilizando el SearchControlOptions podemos crear el control que
        // queríamos
        final SearchControl control = new SearchControl(options);
        // Capturamos el evento de iniciar una busqueda, y llamamos a newItem
        // el contenido de esta
        control.addSearchStartingHandler(new SearchStartingHandler() {
               @Override
               public void onSearchStarting(SearchStartingEvent event) {
                      History.newItem(event.getQuery());
               }
        });
         // Creamos un objeto que implemente ValueChangeHandler y su método,
         // en este caso recogemos la cadena y ejecutamos una busqueda
        History.addValueChangeHandler(new ValueChangeHandler<String>() {
               @Override
               public void onValueChange(ValueChangeEvent<String> event) {
                      control.execute(event.getValue());
               }
        });

        // Y finalmente lo añadimos a un panel visible por el usuario
        RootPanel.get().add(control);
    }
}
```

Este es un ejemplo simple, también se puede utilizar capturando hipervínculos, enlaces dentro de nuestra aplicación o cualquier tipo de histórico que deseemos guardar.

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

6 Interacción con un SGBD

Una de las funcionalidades de la mayor partes de las aplicaciones web es la posibilidad de tratar grandes cantidades de datos de forma rápida. Para conseguir esto de cara al cliente la aplicación normalmente utiliza un Sistema Gestor de Bases de Datos.
GWT nos permite utilizar un gestor cualquiera siempre y cuando le insertemos el correspondiente paquete que nos permita conectar con éste. El "connector" es simplemente el mismo que utilizaríamos en cualquier otra aplicación web basada en Java, recordemos que la parte de nuestra aplicación GWT que se ejecuta en el servidor no es ni más ni menos que J2EE.
Para ver como interaccionar con un SGBD, vamos a explicar como sería la creación de una aplicación GWT con el archiconocido MySQL Server.
Lo primero que necesitamos es un servidor, en este caso vamos a utilizar un servidor MySQL gratuito online y vamos a crear la siguiente tabla:

```
CREATE TABLE `gwtgwt`.`escuderia` (
`Nombre` VARCHAR( 100 ) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL ,
PRIMARY KEY ( `Nombre` )
) ENGINE = MYISAM CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

La cual rellenaremos con algunas de las escuderías que participan en Mundial de F1 y consultaremos.
Como en todos los ejemplos anteriores creamos nuestro proyecto y eliminamos el código que no nos sirve. Lo siguiente será descargar el "connector" para MySQL y añadirlo a nuestro proyecto. El connector nos lo proporciona gratuitamente MySQL y lo podemos descargar de la siguiente URL http://www.mysql.com/products/connector/
Como novedad, en este caso no nos vale con importar el paquete del connector al classpath, en este caso tendremos que copiarlo en war/WEB-INF/lib, de tal forma que este quede incluido en el war final. Luego copiamos el fichero mysql-connector-java-v.x.yy-bin.jar en el directorio indicado y lo añadimos al classpath.
Ahora crearemos el código que se ejecutará en el servidor, ya que a la base de datos accedemos desde este y no desde el cliente [26-30].
Creamos lo necesario para realizar una llamada asíncrona al servidor, en el ejemplo hemos llamado a las clases TestMySQLService, TestMySQLServiceAsync y TestMySQLServiceImpl.
En estas clases vamos a crear un método que llamaremos getScuderias() y que retornará un ArrayList de String con todas las escuderías que encuentre en la tabla de nuestra base de datos.
Nos vamos a centrar en la clase TestMySQLServiceImpl, ya que esta es la que contiene todo lo que implica el acceso al Sistema de Gestión de Base de Datos. En primer lugar, veamos como abrir una conexión. En este método es donde utilizamos todos los datos para encontrar y acceder a la base de datos (Servidor, Tabla, Usuario, Contraseña).

```
private Connection getConn() {
        Connection conn = null;
// Tenemos todos los datos
        String url = "jdbc:mysql://SQL09.FREEMYSQL.NET:3306/";
        String db = "gwtgwt";
        String driver = "com.mysql.jdbc.Driver";
        String user = "gwtgwt";
        String pass = "gwt2gwt";
        try {
                Class.forName(driver).newInstance();
        } catch (InstantiationException e) {
                e.printStackTrace();
        } catch (IllegalAccessException e) {
                e.printStackTrace();
        } catch (ClassNotFoundException e) {
```

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
                e.printStackTrace();
            }
            try {
      //Intentamos crear la nueva conexión
                conn = DriverManager.getConnection(url + db, user, pass);
            } catch (SQLException e) {
                System.err.println("Mysql Connection Error: ");
                e.printStackTrace();
            }
            return conn;
        }
```

Ahora veamos como es el método que realiza la consulta y devuelve los resultados al cliente en un ArrayList.

```
      public ArrayList getScuderias() {
      ArrayList<String> escuderias = new ArrayList<String>(); //Creamos el ArrayList
vacío
            String query = "SELECT * FROM escuderia"; //La consulta
            try {
                  Connection conn = this.getConn();
                  Statement select = conn.createStatement();
                  ResultSet result = select.executeQuery(query);
                  while (result.next()) { //Recogemos los resultados de la con-
sulta
                        String s = result.getString(1);
                        escuderias.add(s);
                  }
                  select.close();
                  result.close();
                  conn.close();
            } catch(SQLException e) {
                  System.err.println("Mysql Statement Error: " + query);
                  e.printStackTrace();
            }
            return escuderias;
        }
```

Esos dos métodos serían los necesarios para realizar la consulta en el lado del servidor. En cuanto al cliente solamente tendríamos que recoger el resultado y presentarlo por pantalla.

```
      final VerticalPanel vp = new VerticalPanel();
      RootPanel.get("main").add(vp);
      vp.add(new Label("Lista de Escuderias"));

      /* Creamos la llamada asíncrona, en  el método onResult escribimos el compor-
tamiento al recibir la respuesta */
      AsyncCallback<ArrayList<String>> callbackScuderias = new AsyncCallback<Ar-
rayList<String>>() {
                  @Override
                  public void onFailure(Throwable caught) {
                        //Something was wrong...
                  }
                  @Override
                  public void onSuccess(ArrayList<String> result) {
                        for(String s : result) {
                              vp.add(new Label(s));
                        }
                  }};
            /* Realizamos la llamada al método y como parámetro le pasamos el ob-
jeto AsyncCallback para que sepa que hacer al recibir el resultado */
```

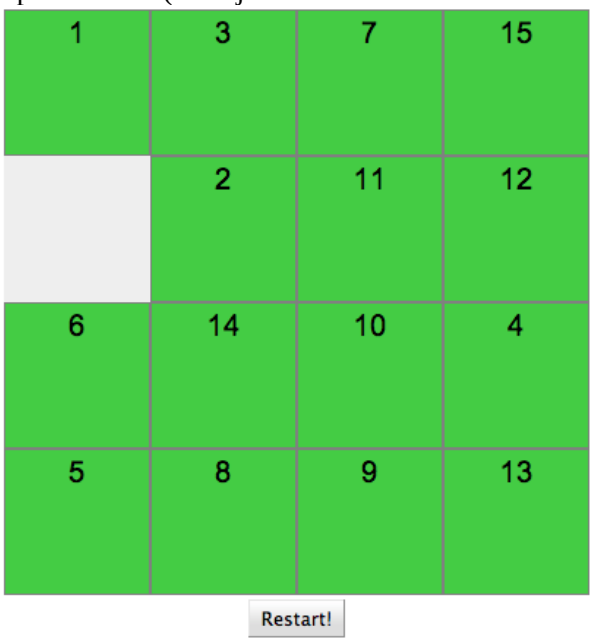

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

```
testMySQLService.getScuderias(callbackScuderias);
```

Queda como ejercicio adaptar la interfaz del cliente y añadir los métodos en la parte del servidor para insertar y actualizar la lista de escuderías. Podéis encontrar toda la información necesaria en http://code.google.com/p/gwt-examples/wiki/project_MySQLConn

7 Caso práctico 2: Quince puzzle

En el capítulo anterior, hemos visto algunas de las APIs que Google nos facilita para nuestros desarrollos, pero no sólo existen estas bibliotecas, sino que hay disponibles muchas otras desarrolladas por la comunidad de usuarios de GWT. Estas bibliotecas se pueden encontrar en el repositorio público de Google, también conocido como GoogleCode.
En este caso práctico nos vamos a centrar en la utilización de bibliotecas externas al API de Google, en concreto hemos seleccionado GWT-DnD. Esta biblioteca nos ofrece la posibilidad de añadir a nuestros widgets la funcionalidad de "arrastrar y soltar" (drag and drop), a lo largo y ancho de un panel visible.
El ejercicio propuesto consiste en construir una aplicación GWT que permita jugar a "El Quincepuzzle". El juego consiste en ordenar las piezas de un puzzle disponiendo únicamente de un hueco libre para realizar los movimientos.
En este caso, se trata de plantear los pasos y pistas a seguir para su implementación en lugar de ver directamente el ejemplo resuelto (Se adjunta en la documentación del curso). Los pasos son:



1.- Crear un proyecto GWT en Eclipse, eliminar el código innecesario y añadir al classpath la biblioteca gwt-dnd.
2.- Editar el fichero XML de configuración del proyecto GWT, añadiendo la siguiente línea para heredar la biblioteca externa:

```
<inherits name='com.allen_sauer.gwt.dnd.gwt-dnd'/>
```

3.- Crear el panel que contendrá el puzzle y la lógica de los movimientos posibles. Este punto es el más complejo, vemos algunas nociones más detalladas.
Para utilizar la biblioteca DnD necesitamos un AbsolutePanel sobre el cual arrastraremos nuestros widgets. Necesitamos crear el controlador que manejará los eventos DragAndDrop, en concreto

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

vamos a utilizar un PickupDragController. A continuación hay un ejemplo de utilización de un controlador de este tipo, en el API de la biblioteca está descrito este y otros.

```
PickupDragController dragController
  = new PickupDragController(absolutePanel, true) {
    public BoundaryDropController newBoundaryDropController(
        AbsolutePanel boundaryPanel, boolean allowDropping) {
            return new BoundaryDropController(boundaryPanel, allowDropping) {
                public void onMove(DragContext context) {
                    super.onMove(context);
                }
                @Override
                public void onDrop(DragContext context) {
                    super.onDrop(context);
                    ...
                }
        };
              }
};
absolutePanel.add(widgetDraggable);
dragController.makeDraggable(widgetDraggable);
```

Hemos de tener en cuenta que esto nos permite arrastrar y soltar un widget en cualquier punto del AbsolutePanel, la solución a este problema la podemos obtener limitando las piezas que podemos mover y limitando su destino y posición exacta. Para esto es interesante reescribir el método onDrop(DragContext context), que nos permite cambiar el comportamiento cuando soltamos el widget.

4.- Crear las clases para representar la lógica: Pieza para cada pieza del Puzzle y la clase Puzzle que contendrá la lógica de éste.

5.- A jugar y probar nuestro QuincePuzzle!

8 Desarrollo rápido de aplicaciones java con GWT y Spring Roo

Una de las noticias más importantes relativas a GWT durante 2010 fue la incorporación del mismo a Spring Roo. Veamos qué es Roo.

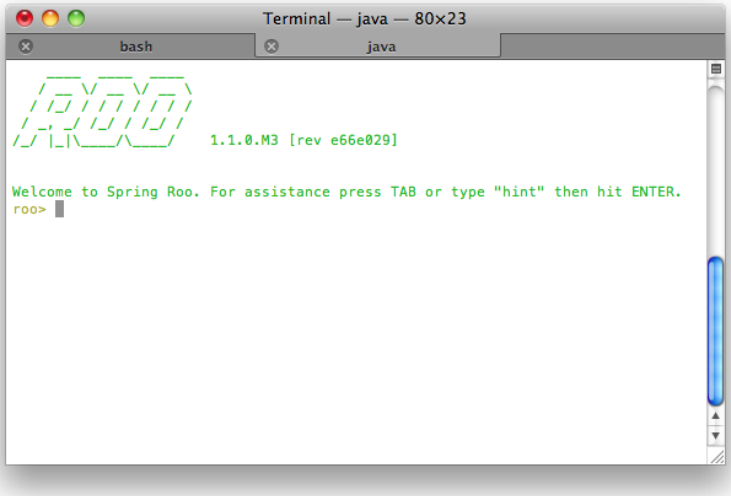
Spring Roo es una herramienta que facilita el desarrollo rápido de aplicaciones java que hacen uso del framework Spring. Más adelante veremos cómo podemos crear una aplicación web GWT completa con sólo unas cuantas instrucciones [31-35].

Inicialmente Roo fue concebido como una alternativa a Grails para desarrolladores que necesitan un framework ágil pero no desean abandonar los proyectos java nativos. Tanto Grails como Roo son desarrollos de Springsource, la empresa detrás del framework Spring.

Exactamente, Roo es una consola de comandos a través de la cual podemos ir construyendo el proyecto, las clases, atributos, configurar la persistencia, etc. Una de sus mayores ventajas, la que probablemente lo haga más atractivo para la mayoría de desarrolladores, es que cuando hayamos terminado de trabajar con Roo podemos eliminarlo del proyecto, con lo que tendremos una aplicación final java, que cumple con el framework Spring y, opcionalmente, con GWT.

Vamos a crear un proyecto muy sencillo, con una sola clase de entidad, “Alumno”, con dos atributos, “nombre” y “apellido”.

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc



El primer paso, lógicamente, es instalar Roo. Para ello, descargaremos el paquete desde la página http://www.springsource.org/roo El único requisito es que tengamos instalado Apache Maven. Descomprimiremos Roo y nos aseguraremos de que el binario principal, bin/roo.sh se encuentre en el path de nuestro sistema [36-40].

A continuación, crearemos el directorio del proyecto, al que vamos a denominar holaroo, y nos colocaremos en dicho directorio en la línea de comandos. Ejecutaremos roo, y dentro de la consola, las siguientes operaciones:

```
project --topLevelPackage es.usal.ecom.holaroo
persistence setup --provider HIBERNATE --database HYPERSONIC_IN_MEMORY
entity --class ~.dominio.Alumno
field string nombre --notNull
field string apellido --notNull
test integration
controller all --package ~.web
gwt setup
exit
```

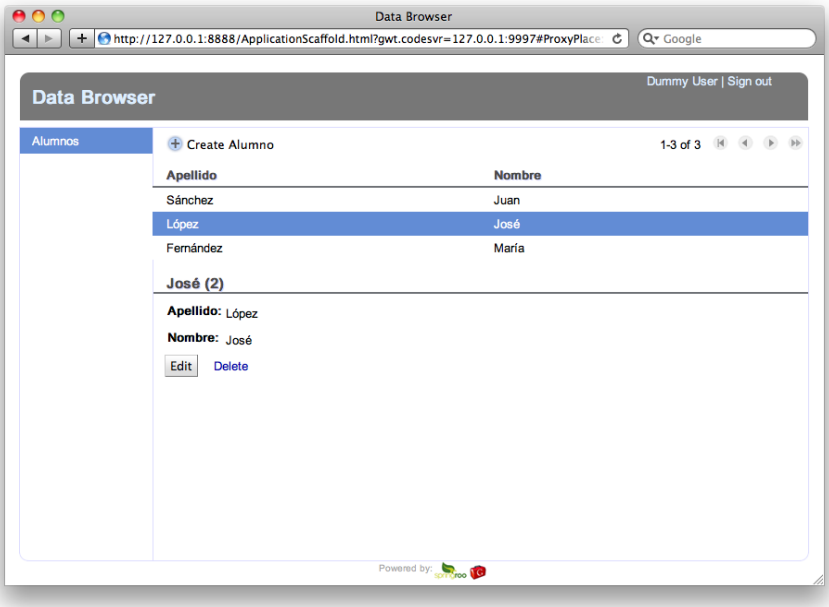
Tras esto ejecutaremos la siguiente orden, para lanzar GWT en modo desarrollador.

```
mvn gwt:run
```

Hacemos clic en “Launch default browser” y aparecerá nuestro proyecto GWT en el navegador, con la entidad “Alumno” y la posibilidad de crear nuevos alumnos, generar listados paginados, editarlos, eliminarlos, etc. Y todo ello con apenas una decena de instrucciones.

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc



De la misma forma, es posible crear proyectos Roo a través de Eclipse. El funcionamiento es muy sencillo, y los pasos a seguir son similares.

Es importante destacar que Roo nos facilita (al igual que herramientas como Ruby on Rails o Grails) la creación de los cimientos de nuestro proyecto. Tras ejecutar los scripts iniciales, somos libres de modificar lo que nos parezca oportuno (manualmente o a través de Eclipse). Cada vez que lancemos Roo, éste se encargará de analizar los ficheros que hemos modificado y aplicar los cambios necesarios al proyecto. Y finalmente, si así lo decidimos, podemos ejecutar los pasos necesarios para hacer que Roo desaparezca de la aplicación final sin dejar rastro.

Es conveniente consultar la documentación de la herramienta (disponible en la web de descarga anteriormente mencionada) pues la integración entre Roo y GWT es algo relativamente reciente y, por tanto, es un proceso en continua evolución [41-47].

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

References

1. Adrián Sánchez-Carmona, Sergi Robles, Carlos Borrego (2015). Improving Podcast Distribution on Gwanda using PrivHab: a Multiagent Secure Georouting Protocol. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 1
2. Anderson Sergio, Sidartha Carvalho, Marco Rego (2014). On the Use of Compact Approaches in Evolution Strategies. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
3. Baruque, B., Corchado, E., Mata, A., & Corchado, J. M. (2010). A forecasting solution to the oil spill problem based on a hybrid intelligent system. Information Sciences, 180(10), 2029–2043. https://doi.org/10.1016/j.ins.2009.12.032
4. Buciarelli, E., Silvestri, M., & González, S. R. (2016). Decision Economics, In Commemoration of the Birth Centennial of Herbert A. Simon 1916-2016 (Nobel Prize in Economics 1978): Distributed Computing and Artificial Intelligence, 13th International Conference. Advances in Intelligent Systems and Computing (Vol. 475). Springer.
5. Canizes, B., Pinto, T., Soares, J., Vale, Z., Chamoso, P., & Santos, D. (2017). Smart City: A GECAD-BISITE Energy Management Case Study. In 15th International Conference on Practical Applications of Agents and Multi-Agent Systems PAAMS 2017, Trends in Cyber-Physical Multi-Agent Systems (Vol. 2, pp. 92–100). https://doi.org/10.1007/978-3-319-61578-3_9
6. Casado-Vara, R., Chamoso, P., De la Prieta, F., Prieto J., & Corchado J.M. (2019). Non-linear adaptive closed-loop control system for improved efficiency in IoT-blockchain management. Information Fusion.
7. Casado-Vara, R., de la Prieta, F., Prieto, J., & Corchado, J. M. (2018, November). Blockchain framework for IoT data quality via edge computing. In Proceedings of the 1st Workshop on Blockchain-enabled Networked Sensor Systems (pp. 19-24). ACM.
8. Casado-Vara, R., Novais, P., Gil, A. B., Prieto, J., & Corchado, J. M. (2019). Distributed continuous-time fault estimation control for multiple devices in IoT networks. IEEE Access.
9. Casado-Vara, R., Vale, Z., Prieto, J., & Corchado, J. (2018). Fault-tolerant temperature control algorithm for IoT networks in smart buildings. Energies, 11(12), 3430.
10. Casado-Vara, R., Prieto-Castrillo, F., & Corchado, J. M. (2018). A game theory approach for cooperative control to improve data quality and false data detection in WSN. International Journal of Robust and Nonlinear Control, 28(16), 5087-5102.
11. Chamoso, P., de La Prieta, F., Eibenstein, A., Santos-Santos, D., Tizio, A., & Vittorini, P. (2017). A device supporting the self-management of tinnitus. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 10209 LNCS, pp. 399–410). https://doi.org/10.1007/978-3-319-56154-7_36
12. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado J.M. (2019). Social computing in currency exchange. Knowledge and Information Systems.
13. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado, J. M. (2019). Social computing in currency exchange. Knowledge and Information Systems, 1-21.
14. Chamoso, P., González-Briones, A., Rodríguez, S., & Corchado, J. M. (2018). Tendencies of technologies and platforms in smart cities: A state-of-the-art review. Wireless Communications and Mobile Computing, 2018.
15. Chamoso, P., Rodríguez, S., de la Prieta, F., & Bajo, J. (2018). Classification of retinal vessels using a collaborative agent-based architecture. AI Communications, (Preprint), 1-18.
16. Choon, Y. W., Mohamad, M. S., Deris, S., Illias, R. M., Chong, C. K., Chai, L. E., … Corchado, J. M. (2014). Differential bees flux balance analysis with OptKnock for in silico microbial strains optimization. PLoS ONE, 9(7). https://doi.org/10.1371/journal.pone.0102744
17. Constantino Martins, Ana Rita Silva, Carlos Martins, Goreti Marreiros (2014). Supporting Informed Decision Making in Prevention of Prostate Cancer. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 3
18. Costa, Â., Novais, P., Corchado, J. M., & Neves, J. (2012). Increased performance and better patient attendance in an hospital with the use of smart agendas. Logic Journal of the IGPL, 20(4), 689–698. https://doi.org/10.1093/jigpal/jzr021
19. David Griol, Jose Manuel Molina, Araceli Sanchís De Miguel (2014). Developing multimodal conversational agents for an enhanced e-learning experience. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 1

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238

20. Eva L. Iglesias, Lourdes Borrajo, R. Romero (2014). A HMM text classification model with learning capacity. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 3
21. Fernández-Riverola, F., Díaz, F., & Corchado, J. M. (2007). Reducing the memory size of a Fuzzy case-based reasoning system applying rough set techniques. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, 37(1), 138–146. https://doi.org/10.1109/TSMCC.2006.876058
22. García Coria, J. A., Castellanos-Garzón, J. A., & Corchado, J. M. (2014). Intelligent business processes composition based on multi-agent systems. Expert Systems with Applications, 41(4 PART 1), 1189–1205. https://doi.org/10.1016/j.eswa.2013.08.003
23. Glez-Peña, D., Díaz, F., Hernández, J. M., Corchado, J. M., & Fdez-Riverola, F. (2009). geneCBR: A translational tool for multiple-microarray analysis and integrative information retrieval for aiding diagnosis in cancer research. BMC Bioinformatics, 10. https://doi.org/10.1186/1471-2105-10-187
24. Gonzalez-Briones, A., Chamoso, P., De La Prieta, F., Demazeau, Y., & Corchado, J. M. (2018). Agreement Technologies for Energy Optimization at Home. Sensors (Basel), 18(5), 1633-1633. doi:10.3390/s18051633
25. González-Briones, A., Chamoso, P., Yoe, H., & Corchado, J. M. (2018). GreenVMAS: virtual organization-based platform for heating greenhouses using waste energy from power plants. Sensors, 18(3), 861.
26. González-Briones, A., De La Prieta, F., Mohamad, M., Omatu, S., & Corchado, J. (2018). Multi-agent systems applications in energy optimization problems: A state-of-the-art review. Energies, 11(8), 1928.
27. Gonzalez-Briones, A., Prieto, J., De La Prieta, F., Herrera-Viedma, E., & Corchado, J. M. (2018). Energy Optimization Using a Case-Based Reasoning Strategy. Sensors (Basel), 18(3), 865-865. doi:10.3390/s18030865
28. Hafewa Bargaoui, Olfa Belkahla Driss (2014). Multi-Agent Model based on Tabu Search for the Permutation Flow Shop Scheduling Problem. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 1
29. Jamal Ahmad Dargham, Ali Chekima, Ervin Gubin Moung, Sigeru Omatu (2014). The Effect of Training Data Selection on Face Recognition in Surveillance Application. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
30. Jorge Agüero, Miguel Rebollo, Carlos Carrascosa, Vicente Julián (2013). MDD-Approach for developing Pervasive Systems based on Service-Oriented Multi-Agent Systems. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 3
31. Sittón-Candanedo, I., Alonso, R. S., Corchado, J. M., Rodríguez-González, S., & Casado-Vara, R. (2019). A review of edge computing reference architectures and a new global edge proposal. Future Generation Computer Systems, 99, 278-294.
32. Leonor Becerra-Bonache, M. Dolores Jiménez López (2014). Linguistic Models at the Crossroads of Agents, Learning and Formal Languages. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
33. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). A particle dyeing approach for track continuity for the SMC-PHD filter. In FUSION 2014 - 17th International Conference on Information Fusion. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637583&partnerID=40&md5=709eb4815eaf544ce01a2c21aa749d8f
34. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). Random finite set-based Bayesian filters using magnitude-adaptive target birth intensity. In FUSION 2014 - 17th International Conference on Information Fusion. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637788&partnerID=40&md5=bd8602d6146b014266cf07dc35a681e0
35. Lima, A. C. E. S., De Castro, L. N., & Corchado, J. M. (2015). A polarity analysis framework for Twitter messages. Applied Mathematics and Computation, 270, 756–767. https://doi.org/10.1016/j.amc.2015.08.059
36. Margherita Brondino, Gabriella Dodero, Rosella Gennari, Alessandra Melonio, Daniela Raccanello, Santina Torello (2014). Achievement Emotions and Peer Acceptance Get Together in Game Design at School. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
37. Mata, A., & Corchado, J. M. (2009). Forecasting the probability of finding oil slicks using a CBR system. Expert Systems with Applications, 36(4), 8239–8246. https://doi.org/10.1016/j.eswa.2008.10.003
38. Miki Ueno, Naoki Mori, Keinosuke Matsumoto (2014). Picture models for 2-scene comics creating system. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 2

ISBN: 978-84-9012-858-9
Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc

39. Ming Fei Siyau, Tiancheng Li, Jonathan Loo (2014). A Novel Pilot Expansion Approach for MIMO Channel Estimation. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 3
40. Omar Jassim, Moamin Mahmoud, Mohd Sharifuddin Ahmad (2014). Research Supervision Management Via A Multi-Agent Framework. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
41. Pablo Chamoso, Henar Pérez-Ramos, Ángel García-García (2014). ALTAIR: Supervised Methodology to Obtain Retinal Vessels Caliber. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 3, n. 4
42. Paula Andrea Rodríguez Marín, Néstor Duque, Demetrio Ovalle (2015). Multi-agent system for Knowledge-based recommendation of Learning Objects. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 4, n. 1
43. Rodríguez, S., De La Prieta, F., Tapia, D. I., & Corchado, J. M. (2010). Agents and computer vision for processing stereoscopic images. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 6077 LNAI). https://doi.org/10.1007/978-3-642-13803-4_12
44. Rodríguez, S., Gil, O., De La Prieta, F., Zato, C., Corchado, J. M., Vega, P., & Francisco, M. (2010). People detection and stereoscopic analysis using MAS. In INES 2010 - 14th International Conference on Intelligent Engineering Systems, Proceedings. https://doi.org/10.1109/INES.2010.5483855
45. Román, J. A., Rodríguez, S., & de da Prieta, F. (2016). Improving the distribution of services in MAS. Communications in Computer and Information Science (Vol. 616). https://doi.org/10.1007/978-3-319-39387-2_4
46. Sigeru Omatu, Tatsuyuki Wada, Pablo Chamoso (2013). Odor Classification using Agent Technology. DCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 2, n. 4
47. Tapia, D. I., & Corchado, J. M. (2009). An ambient intelligence based multi-agent system for alzheimer health care. International Journal of Ambient Computing and Intelligence, v 1, n 1(1), 15–26. https://doi.org/10.4018/jaci.2009010102

ISBN: 978-84-9012-858-9

Colección Aquilafuente: AQ-238
Ediciones Universidad de Salamanca - cc by nc dc