Contenido

[CARGAR DATOS EN UN SELECT WIDGET 2](#_Toc405208193)

[1. Cargar datos desde un script groovy: 2](#_Toc405208194)

[1.1 Escribir directamente los campos del select. 2](#_Toc405208195)

[1.2 Conectar directamente con BBDD 3](#_Toc405208196)

[1.3 Conectar con rest desde groovy 6](#_Toc405208197)

[2. Modificar los valores dinámicamente 8](#_Toc405208198)

[2.1 Actualizar datos a través de JQUERY+REST+AJAX 8](#_Toc405208199)

[2.2 Generar los componentes desde un html widget. 14](#_Toc405208200)

# CARGAR DATOS EN UN SELECT WIDGET

Hay varias alternativas:

* **Cargar datos desde un script groovy:**

Esta solución es buena cuando los datos no van a cambiar durante todo el uso del select.

Aquí a su vez hay varias alternativas:

1. Escribir directamente los campos y valores del select.
2. Directamente desde BBDD
3. A través de un servicio rest.

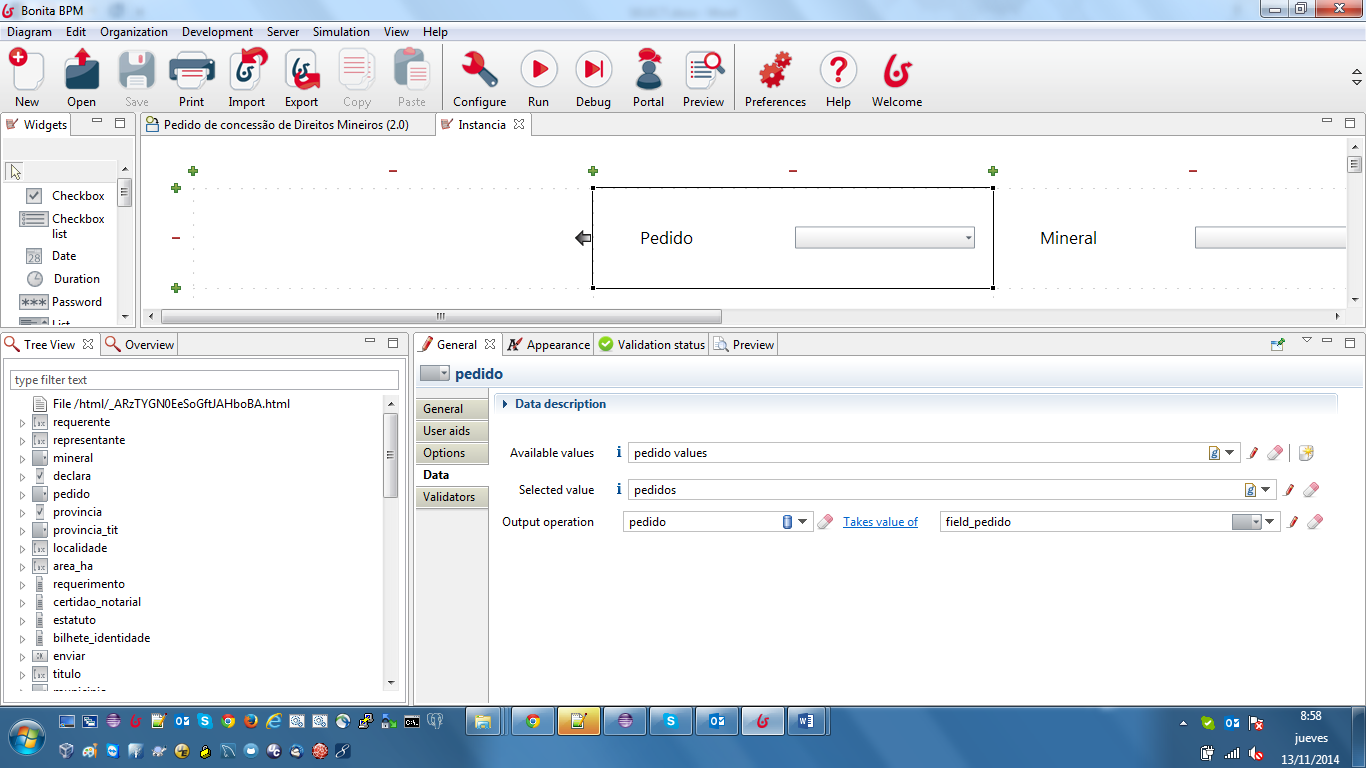
* **Modificar los valores dinámicamente:**

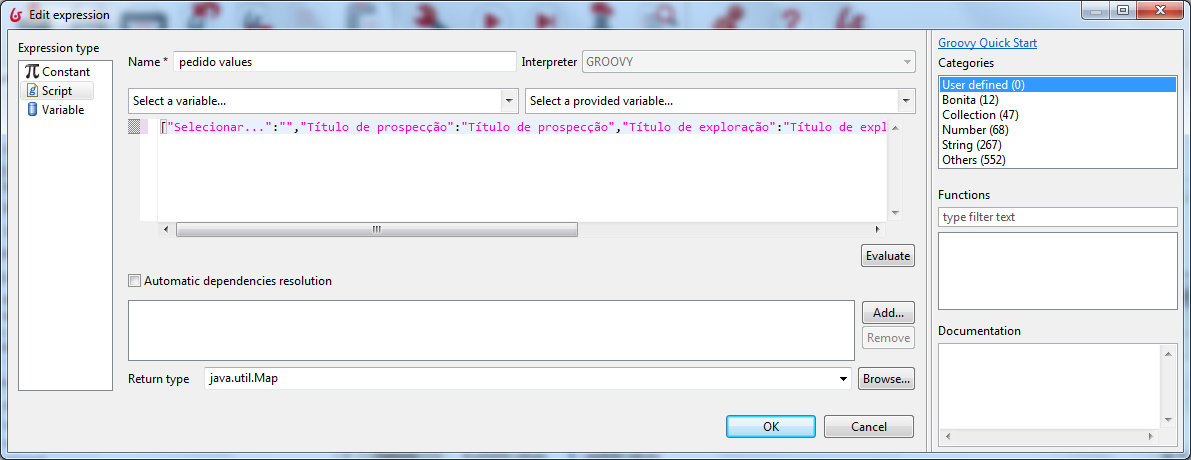
Esta solución es buena cuando el select afectado, va a cambiar su valor dependiendo del valor seleccionado en otro campo.

## Cargar datos desde un script groovy:

### Escribir directamente los campos del select.

Editamos los datos disponibles del select





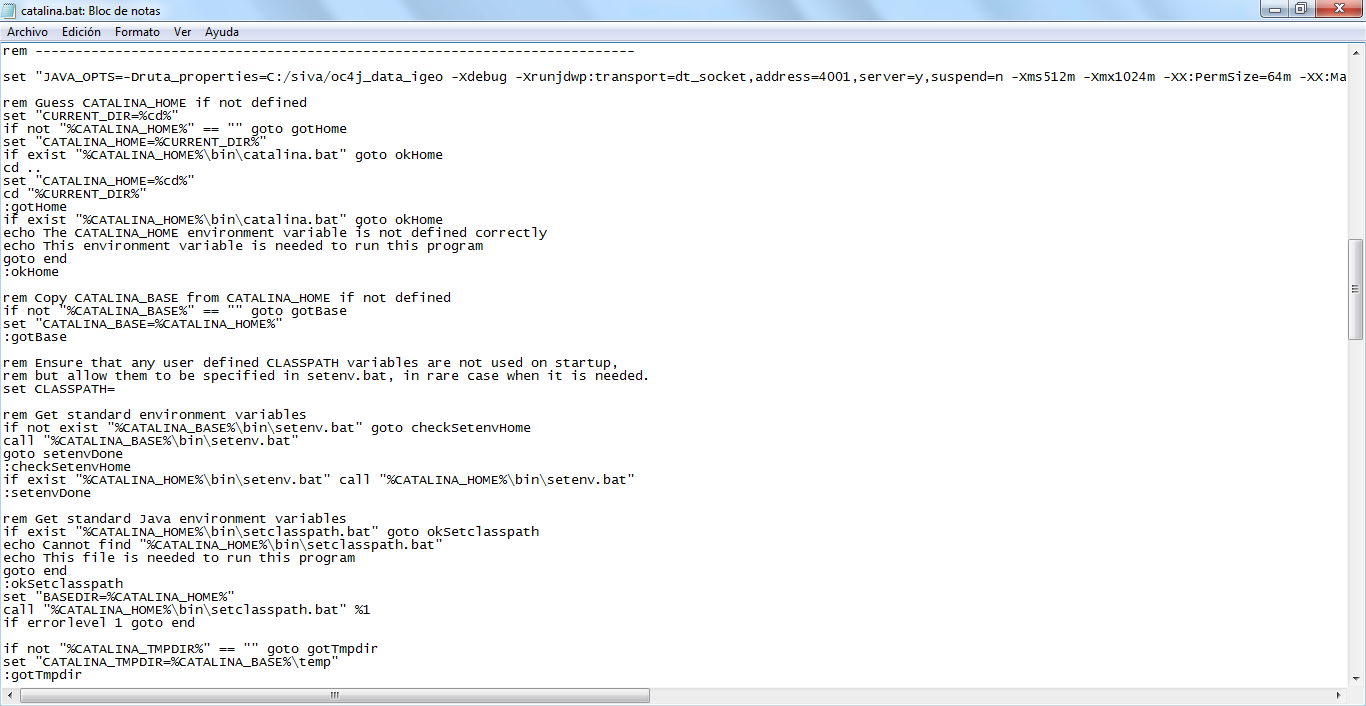
["Selecionar...":"","Título de prospecção":"Título de prospecção","Título de exploração":"Título de exploração","Alvará mineiro":"Alvará mineiro","Senha mineira":"Senha mineira"]

**El tipo de retorno debe ser Map, si fuera list, se debería mapear un list y los valores y los textos coincidirian**

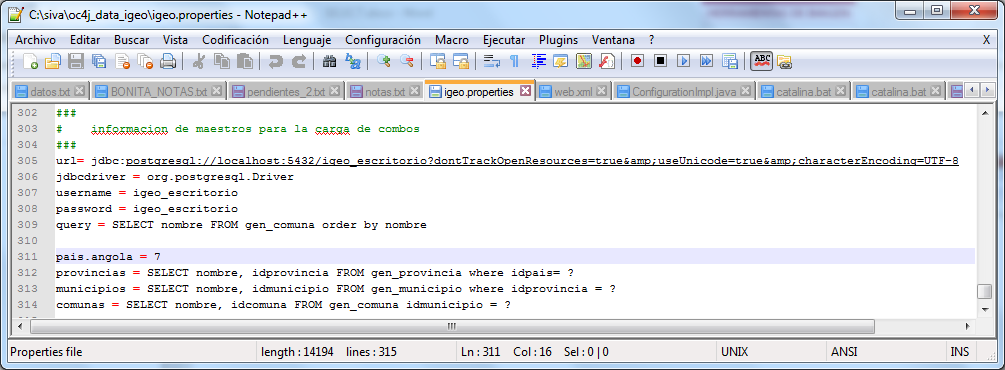
### Conectar directamente con BBDD

EN PRE NO VA A FUNCIONAR, NO SE PERMITE ACCEDER A UNA BBDD DESDE UNA MAQUINA EXTERNA.

* En el servidor de bonita ponemos una variable de path del properties para poder acceder a él como una variable del system desde java.



* En el properties apuntamos los datos de acceso a BBDDs y las urls de servicios rest que necesitemos.



###

# informacion de maestros para la carga de combos

###

url= jdbc:postgresql://localhost:5432/igeo\_escritorio?dontTrackOpenResources=true&amp;useUnicode=true&amp;characterEncoding=UTF-8

jdbcdriver = org.postgresql.Driver

username = igeo\_escritorio

password = igeo\_escritorio

query = SELECT nombre FROM gen\_comuna order by nombre

pais.angola = 7

provincias = SELECT nombre, idprovincia FROM gen\_provincia where idpais= ?

municipios = SELECT nombre, idmunicipio FROM gen\_municipio where idprovincia = ?

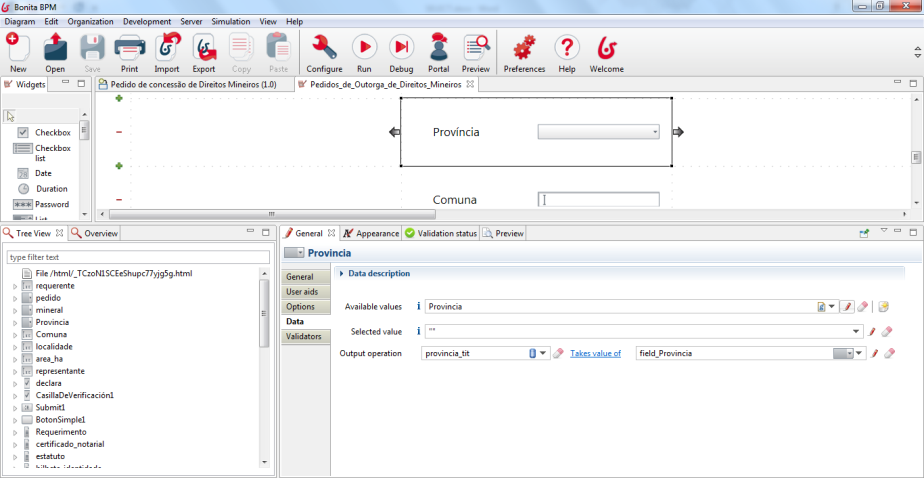
comunas = SELECT nombre, idcomuna FROM gen\_comuna idmunicipio = ?

url\_maestros = <http://localhost:8080/silcam-core-ws>

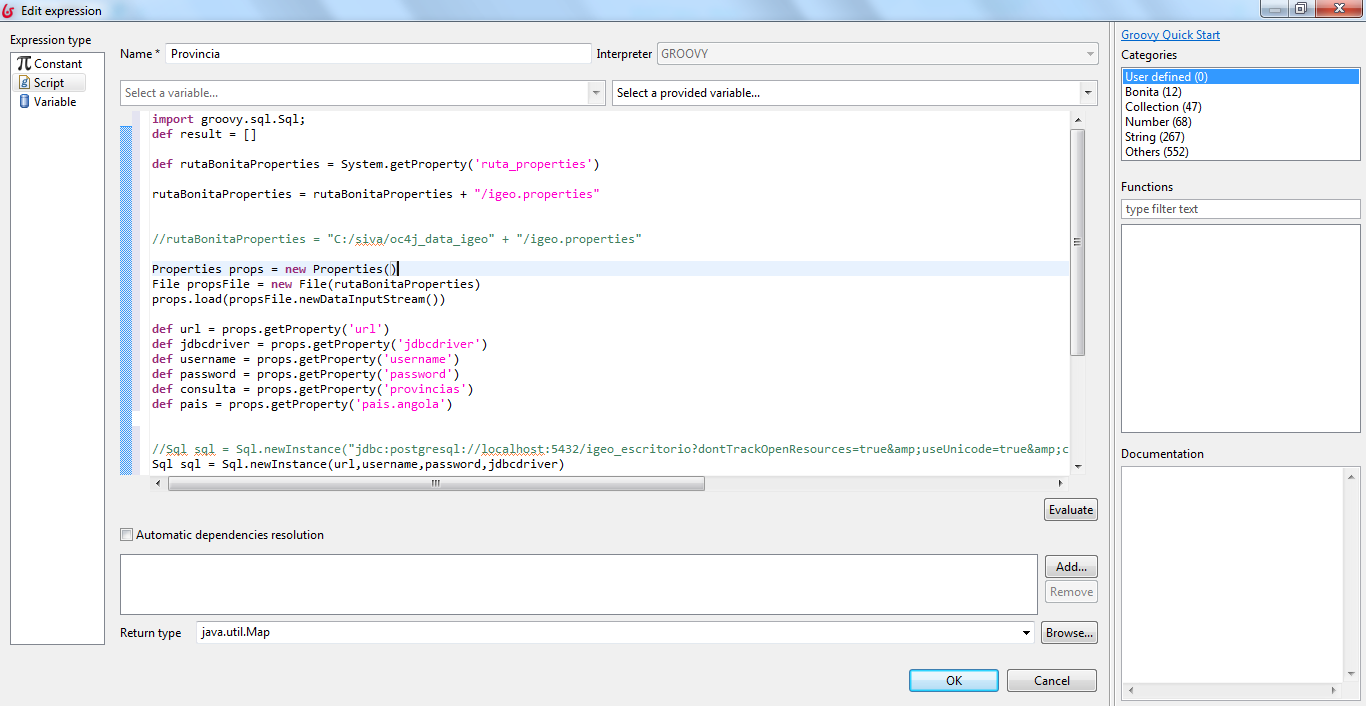
Incluir librerías de BBDD en el servidor

* El servidor tiene que tener las librerías necesarias para poder conectar con el driver de BBDD correspondiente.

Ahora informamos el campo con los datos desde BBDD



Editamos los posibles valores y con el script groovy



**import** groovy.sql.Sql;

**def** result = []

**def** rutaBonitaProperties = System.getProperty('ruta\_properties')

rutaBonitaProperties = rutaBonitaProperties + "/igeo.properties"

//rutaBonitaProperties = "C:/siva/oc4j\_data\_igeo" + "/igeo.properties"

Properties props = **new** Properties()

File propsFile = **new** File(rutaBonitaProperties)

props.load(propsFile.newDataInputStream())

**def** url = props.getProperty('url')

**def** jdbcdriver = props.getProperty('jdbcdriver')

**def** username = props.getProperty('username')

**def** password = props.getProperty('password')

**def** consulta = props.getProperty('provincias')

**def** pais = props.getProperty('pais.angola')

//Sql sql = Sql.newInstance("jdbc:postgresql://localhost:5432/igeo\_escritorio?dontTrackOpenResources=true&amp;useUnicode=true&amp;characterEncoding=UTF-8","igeo\_escritorio","igeo\_escritorio","org.postgresql.Driver")

Sql sql = Sql.newInstance(url,username,password,jdbcdriver)

**def** myMap = [:]

myMap.put("Seleccionar uno...", "")

sql.rows(consulta, Integer.parseInt(pais)).each {

Map row -> myMap.put(row.nombre, row.idprovincia)

}

sql.close()

myMap

El tipo de retorno debe ser MAP, para poder tener opciones que mostrar y valores que seleccionar.

**Referencias:**

<https://www.youtube.com/watch?v=is6vpnQFq6A>

<http://community.bonitasoft.com/blog/how-populate-select-box-field-database-using-groovy>

Dependencias de librerías según el caso:

postgresql-9.3-1101.jdbc41.jar

oracle-jdbc-jdk5-11.2.0.2.0.jar

mysql-connector-java-5.1.30-bin.jar

### Conectar con rest desde groovy

Similar al anterior pero obteniendo los datos por rest.

**import** groovyx.net.http.RESTClient

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

Logger logger = LoggerFactory.getLogger("org.bonitasoft.groovy.script.Provincia")

logger.info("Set initial value of process variable using Groovy script")

**def** client = **new** RESTClient( 'http://ih-vm-stccatastro.c.mad.interhost.com:8080/silcam-core-ws/rest/' )

**def** provincias = client.get( path : 'maestros/provincias/7' )

**def** myMap = [:]

myMap.put("Seleccionar uno...", "")

provincias.data.provincia.each() {

provincia -> myMap.put(provincia.nombre, provincia.clave)

}

myMap

Dependencias de librerías:

java-property-utils-1.9.1.jar

http-builder-0.7.jar

appengine-api-1.0-sdk-1.3.8.jar

commons-beanutils-1.8.0.jar

commons-codec-1.7.jar

commons-collections-4-4.0.jar

commons-lang-3-3.1.jar

commons-logging-1.1.1.jar

ezmorph-1.0.6.jar

httpclient-4.3.5.jar

httpcore-4.3.2.jar

json-lib-2.3-jdk15.jar

nekohtml-1.9.16.jar

signpost-commonshttp4-1.2.1.1.jar

signpost-core-1.2.1.1.jar

xercesImpl-2.9.1.jar

xml-apis-1.4.01.jar

xml-resolver-1.2.jar

**Referencias:**

[**http://groovy.codehaus.org/modules/http-builder/index.html**](http://groovy.codehaus.org/modules/http-builder/index.html)

<http://groovy.codehaus.org/modules/http-builder/download.html>

<http://extensions.xwiki.org/xwiki/bin/view/Extension/REST+HTTP+Client>

<http://extensions.xwiki.org/xwiki/bin/view/Extension/GitHub+Integration>

http://rest.elkstein.org/2008/02/using-rest-in-groovy.html

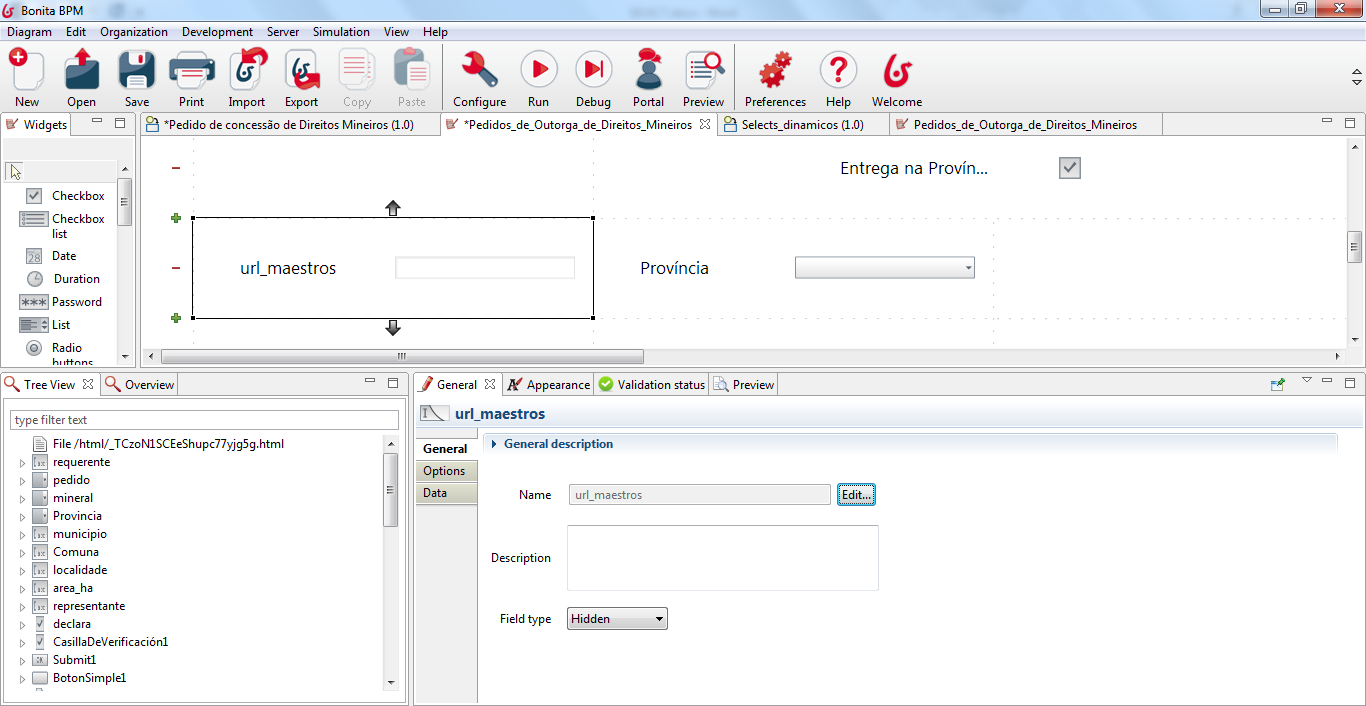
<http://community.bonitasoft.com/groups/contributions-development-6x/creating-restful-web-service-connector-groovyhttpbuilder>

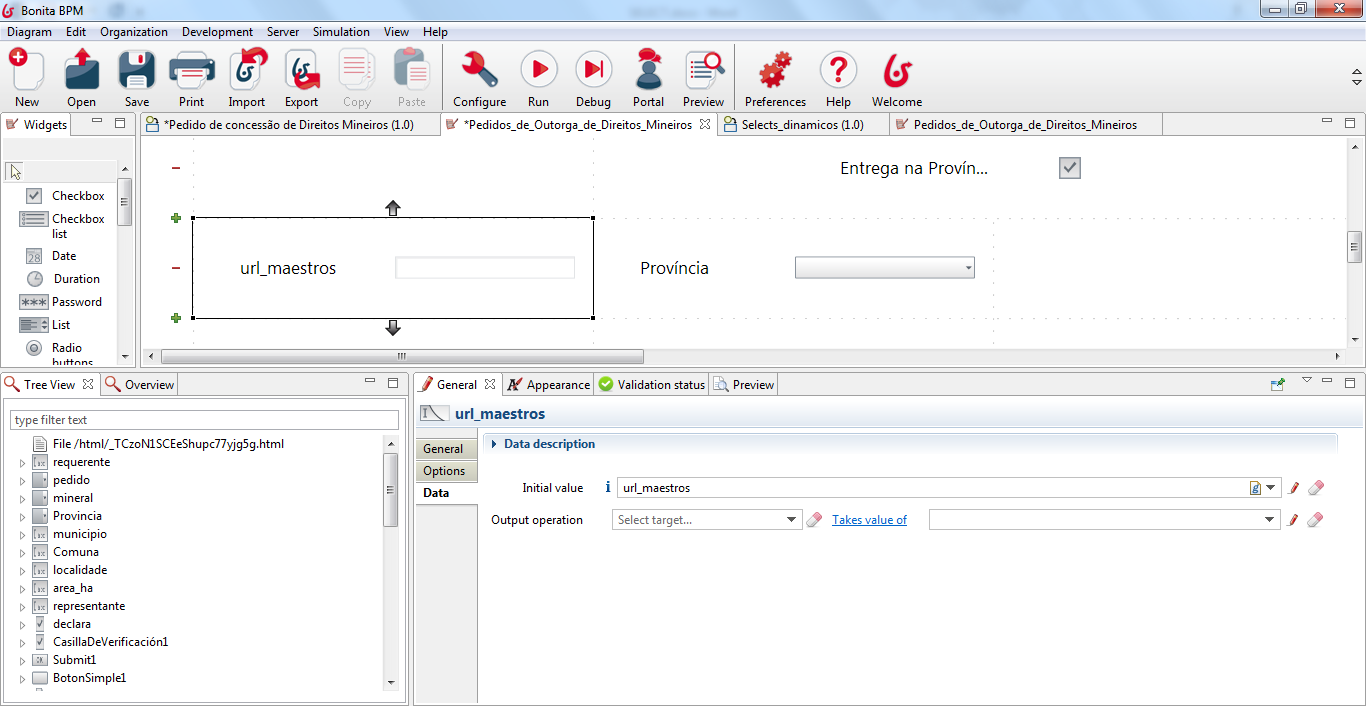
## Modificar los valores dinámicamente

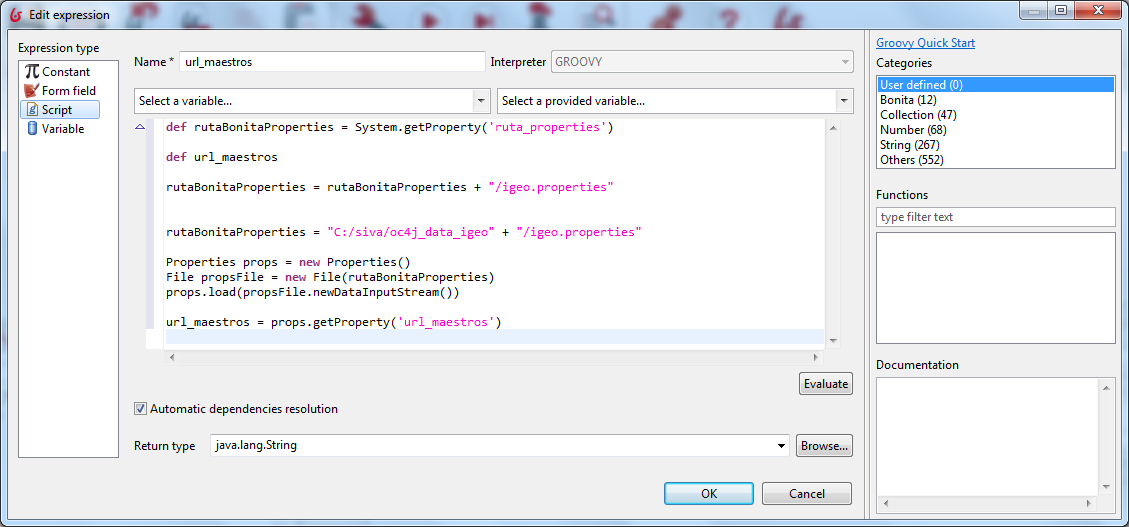
### Actualizar datos a través de JQUERY+REST+AJAX

Esta solución es ideal para Selects dinámicos dependientes

* Lo primero montamos un campo oculto que lea del properties accesible por la variable del servidor, la ruta de los servicios rest:







**def** rutaBonitaProperties = System.getProperty('ruta\_properties')

**def** url\_maestros

rutaBonitaProperties = rutaBonitaProperties + "/igeo.properties"

rutaBonitaProperties = "C:/siva/oc4j\_data\_igeo" + "/igeo.properties"

Properties props = **new** Properties()

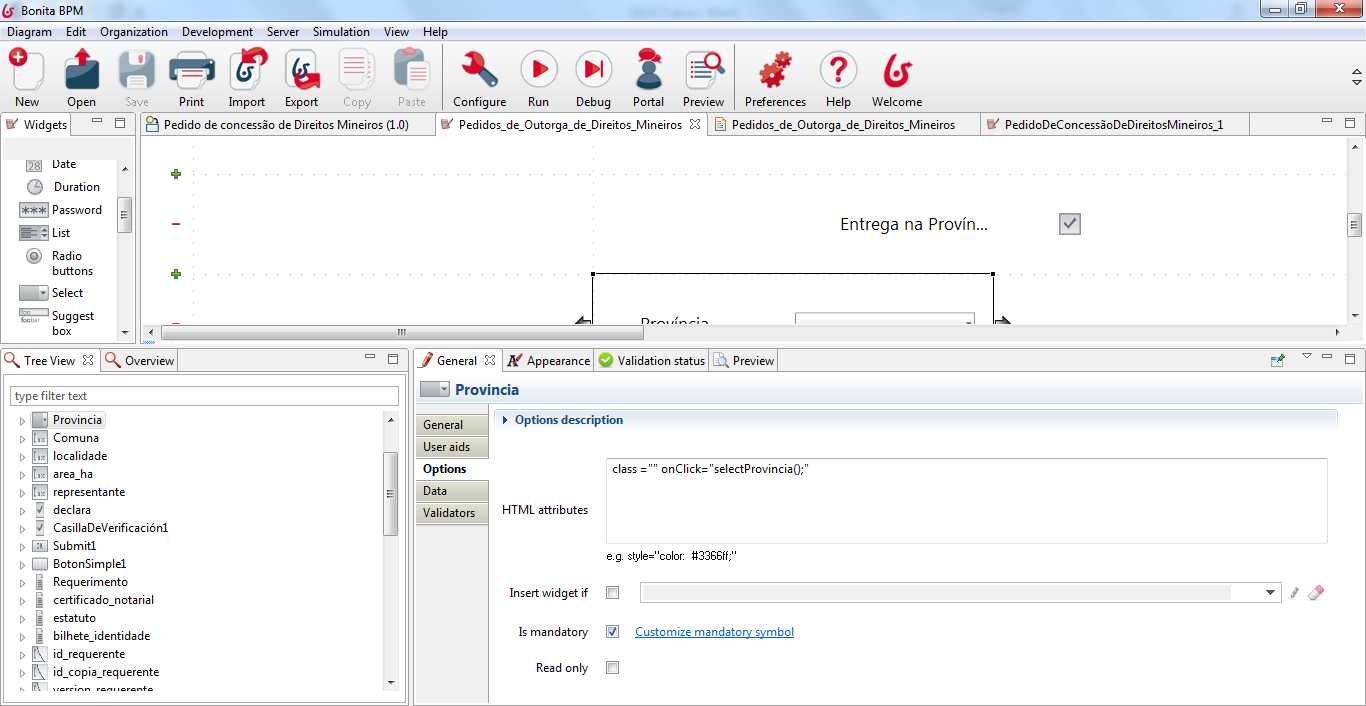
File propsFile = **new** File(rutaBonitaProperties)

props.load(propsFile.newDataInputStream())

url\_maestros = props.getProperty('url\_maestros')

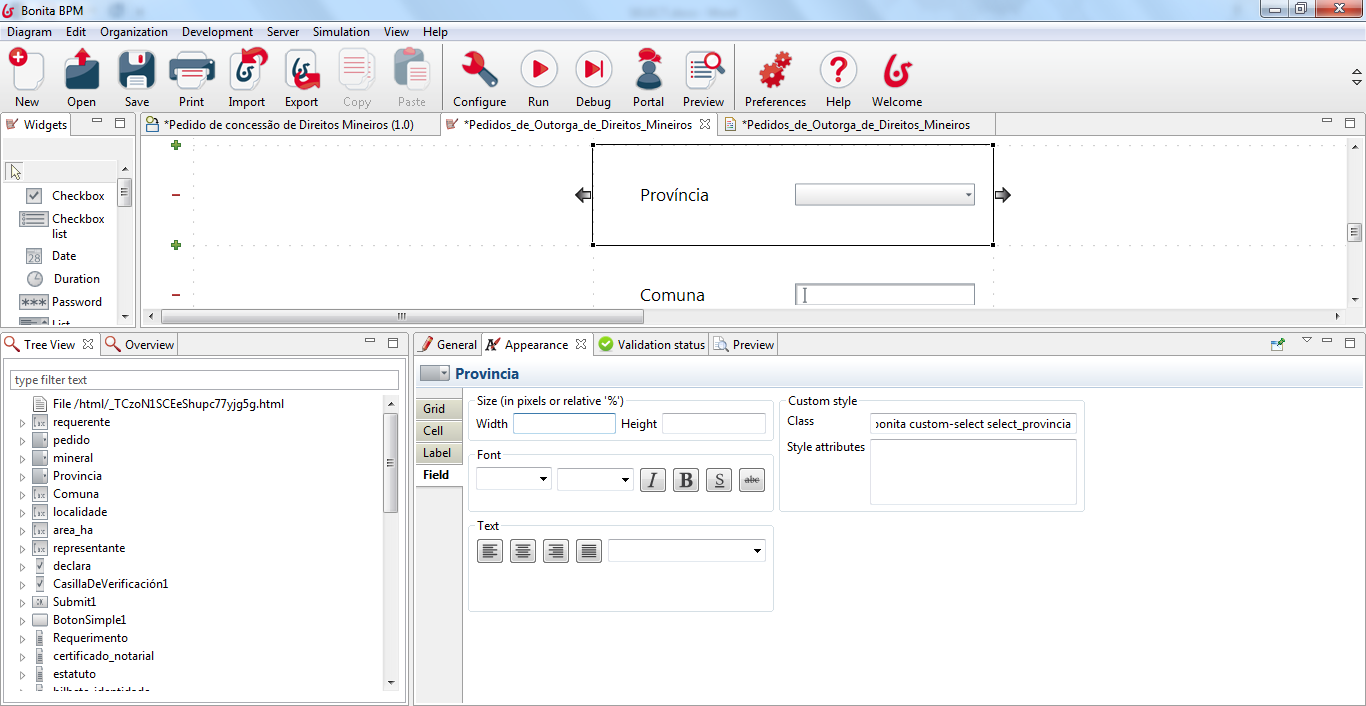
Ahora desde JavaScript esta url estará accesible asi:

**var** url\_maestros\_rest = $('#url\_maestros').find('.bonita\_form\_field').val();

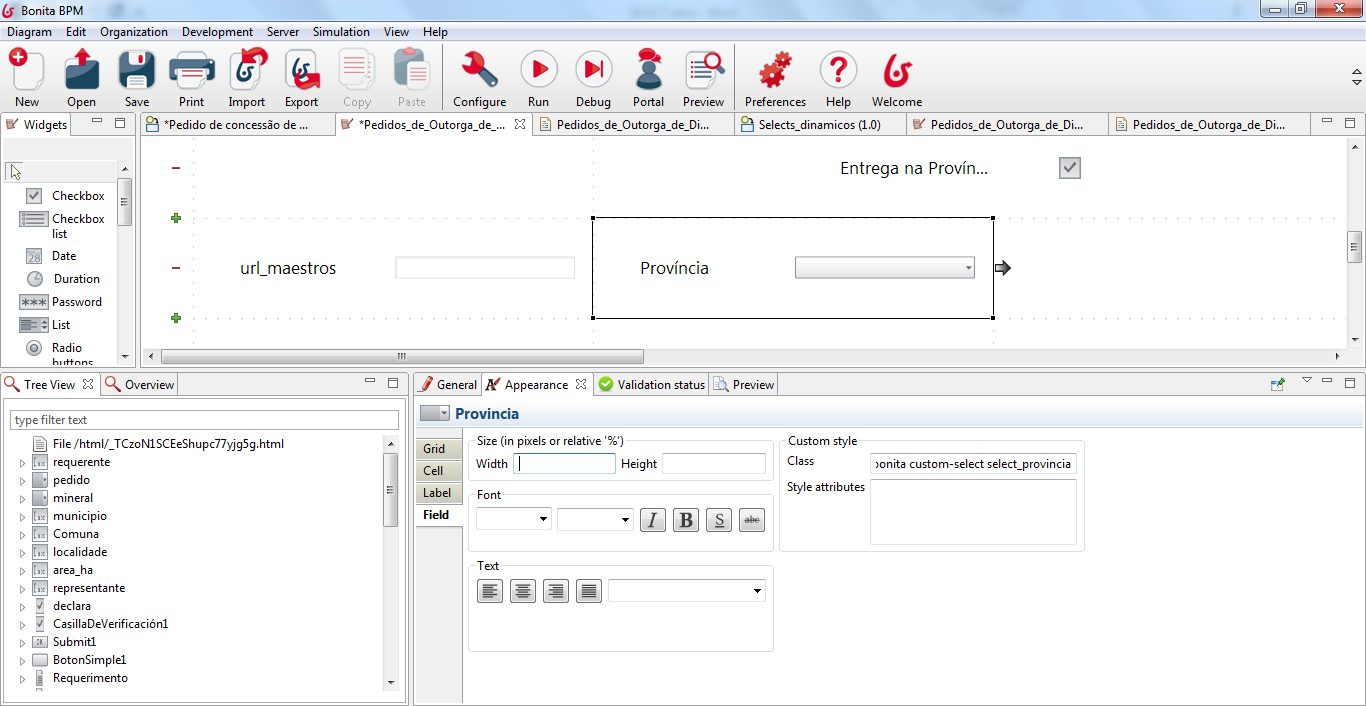


* Tenemos que conseguir que los selects ejecuten jquery al cambiar sus valores:

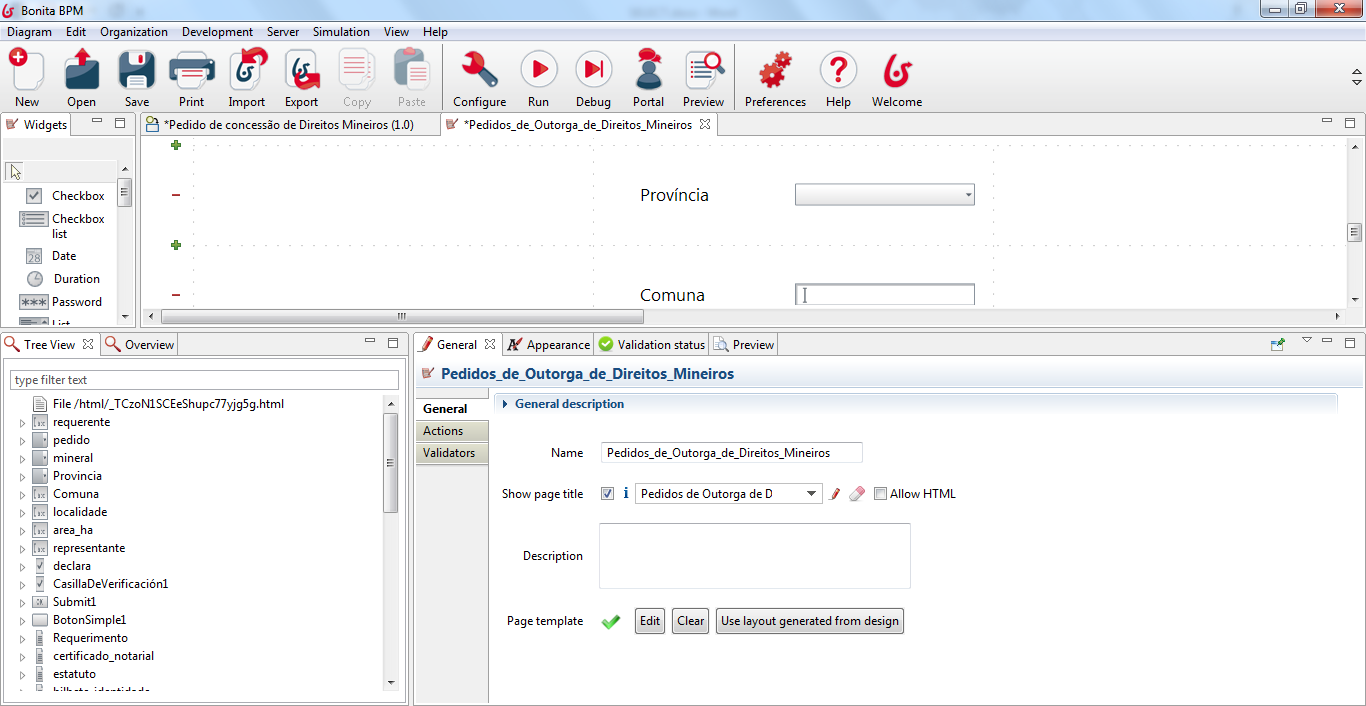
Le ponemos un identificador único por class al campo select

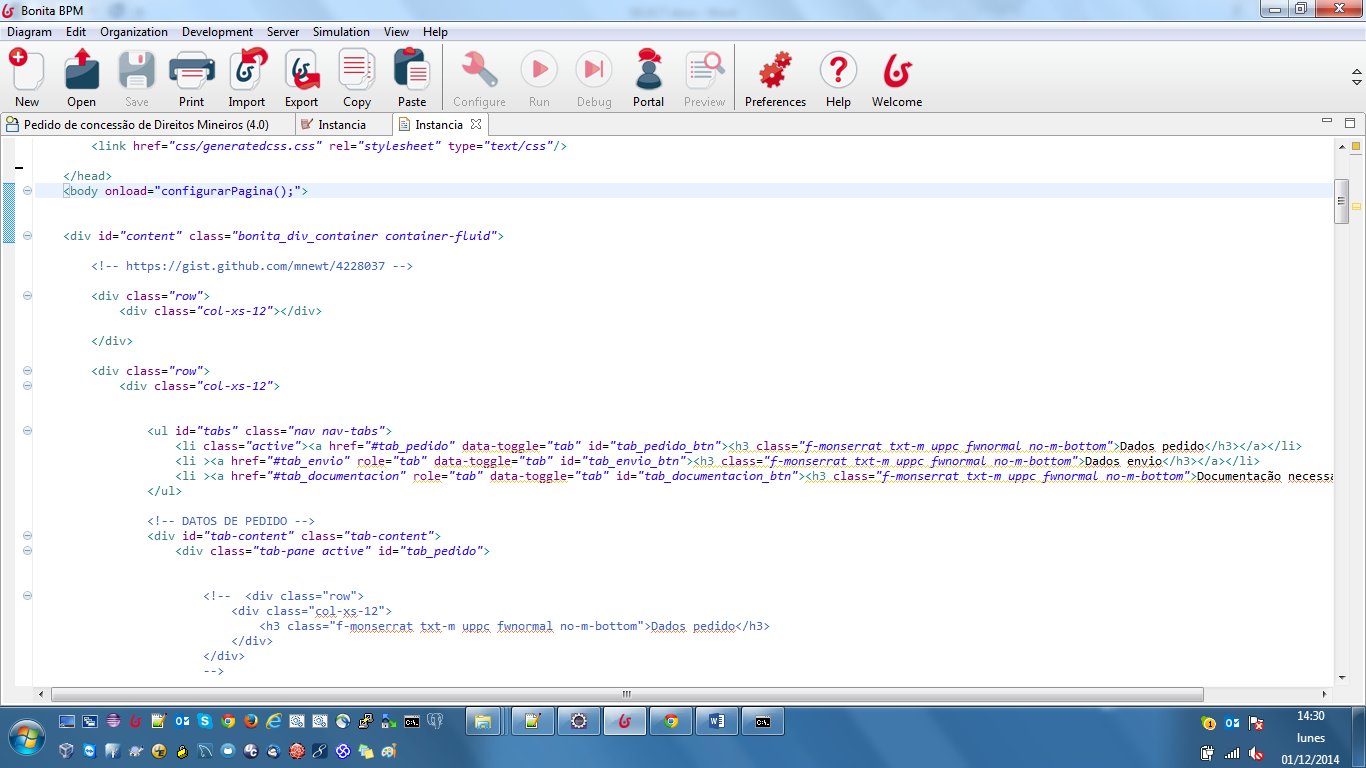


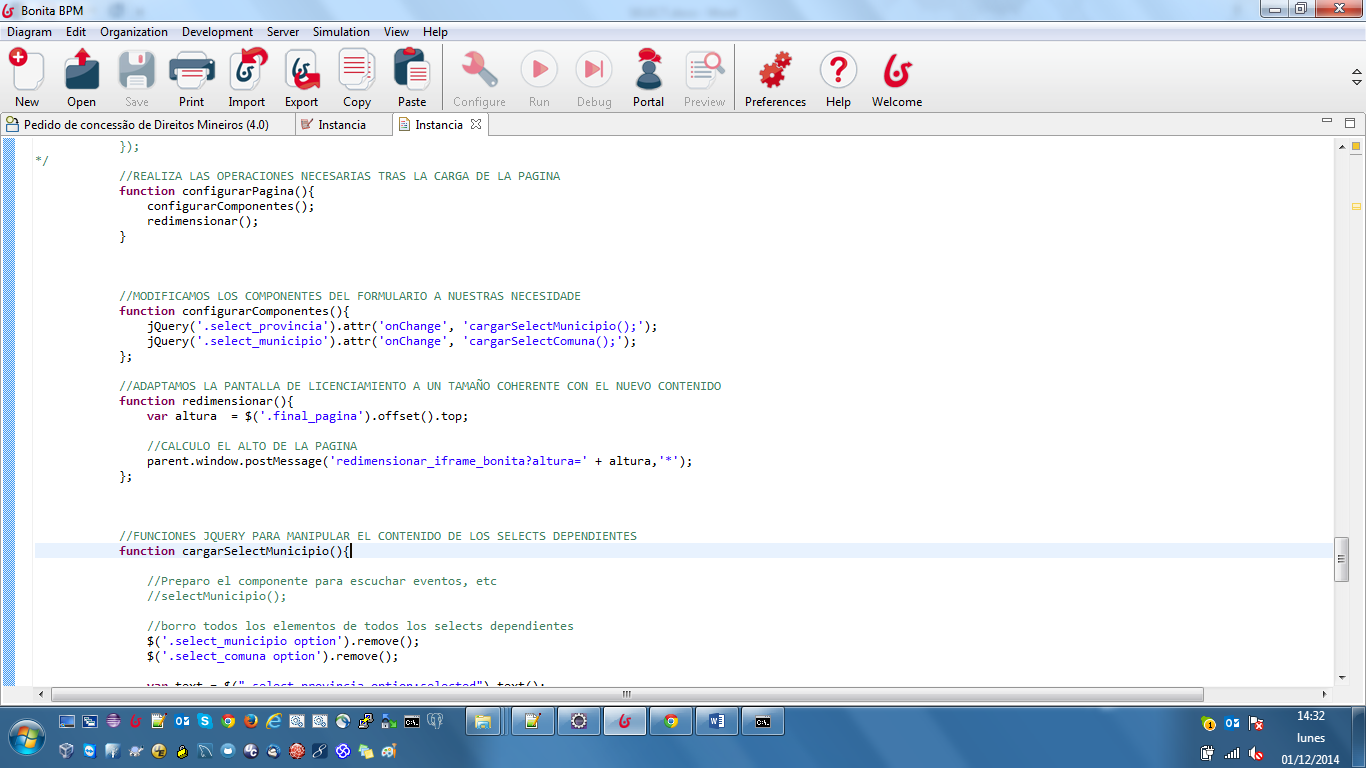
Añadimos un estilo al class del campo del select, para poder identificarlo unívocamente por jquery.



* En el Layout de la página indicamos una función javascript que se debe ejecutar cuando la página esté completamente cargada:







//REALIZA LAS OPERACIONES NECESARIAS TRAS LA CARGA DE LA PAGINA

**function** configurarPagina(){

configurarComponentes();

redimensionar();

}

//MODIFICAMOS LOS COMPONENTES DEL FORMULARIO A NUESTRAS NECESIDADE

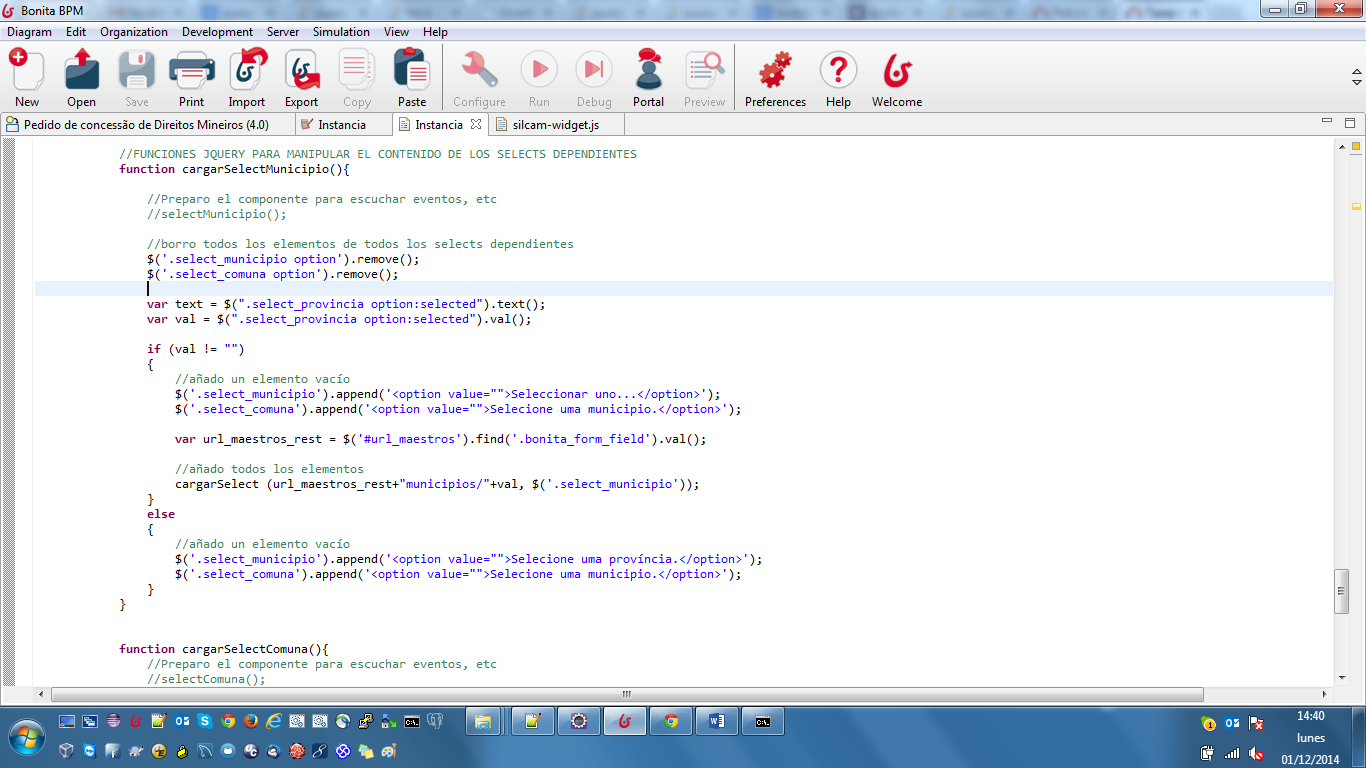
**function** configurarComponentes(){

jQuery('.select\_provincia').attr('onChange', 'cargarSelectMunicipio();');

jQuery('.select\_municipio').attr('onChange', 'cargarSelectComuna();');

};

* Incluimos el javascript para controlar los eventos de cambio de los selects:



<script type=*"text/javascript"*>

//FUNCIONES JQUERY PARA MANIPULAR EL CONTENIDO DE LOS SELECTS DEPENDIENTES

**function** cargarSelectMunicipio(){

//Preparo el componente para escuchar eventos, etc

selectMunicipio();

//borro todos los elementos de todos los selects dependientes

$('.select\_municipio option').remove();

$('.select\_comuna option').remove();

$('.select\_localidade option').remove();

**var** text = $(".select\_provincia option:selected").text();

**var** val = $(".select\_provincia option:selected").val();

**if** (val != "")

{

//añado un elemento vacío

$('.select\_municipio').append('<option value="">Seleccionar uno...</option>');

$('.select\_comuna').append('<option value="">Selecione uma municipio.</option>');

$('.select\_localidade').append('<option value="">Selecione uma comuna.</option>');

**var** url\_maestros\_rest = $('#url\_maestros').find('.bonita\_form\_field').val();

//añado todos los elementos

cargarSelect (url\_maestros\_rest+"/rest/maestros/municipios/"+val, $('.select\_municipio'));

}

**else**

{

//añado un elemento vacío

$('.select\_municipio').append('<option value="">Selecione uma província.</option>');

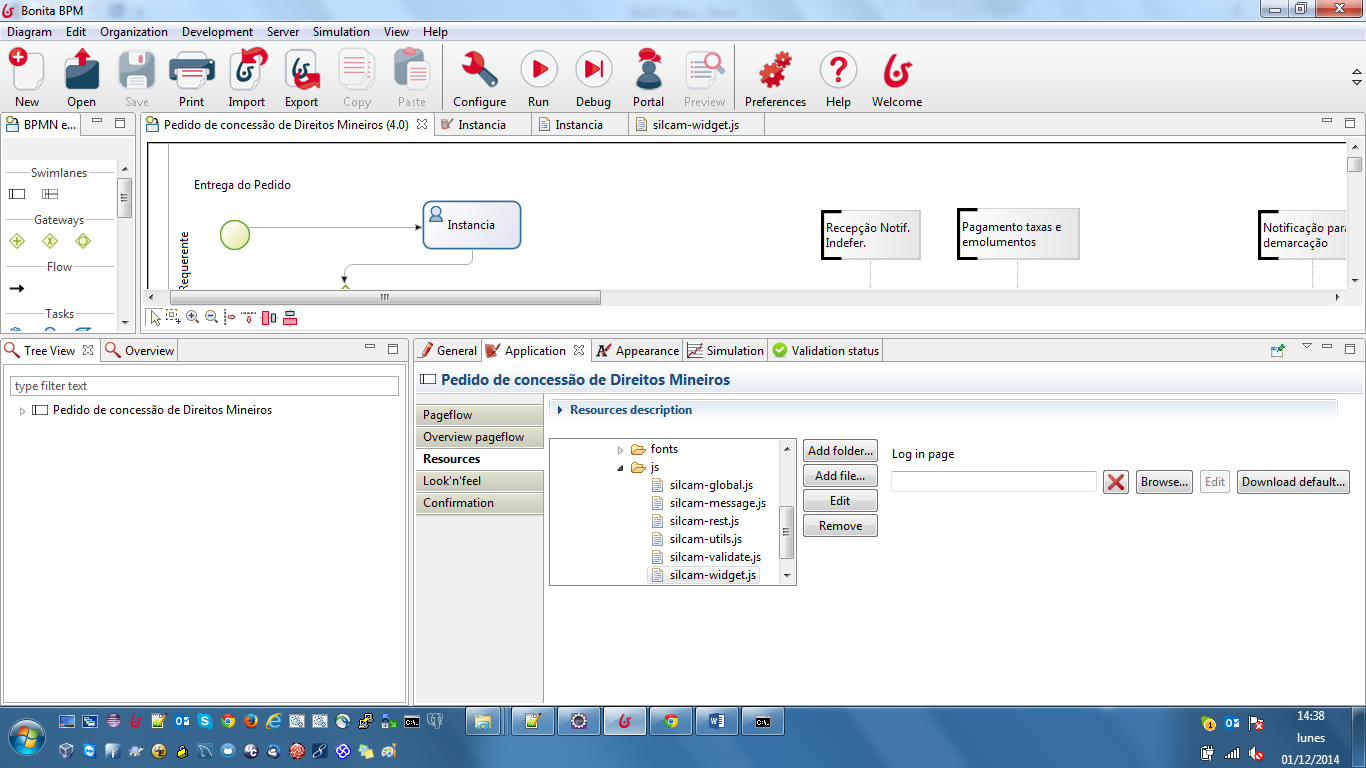
$('.select\_comuna').append('<option value="">Selecione uma municipio.</option>');

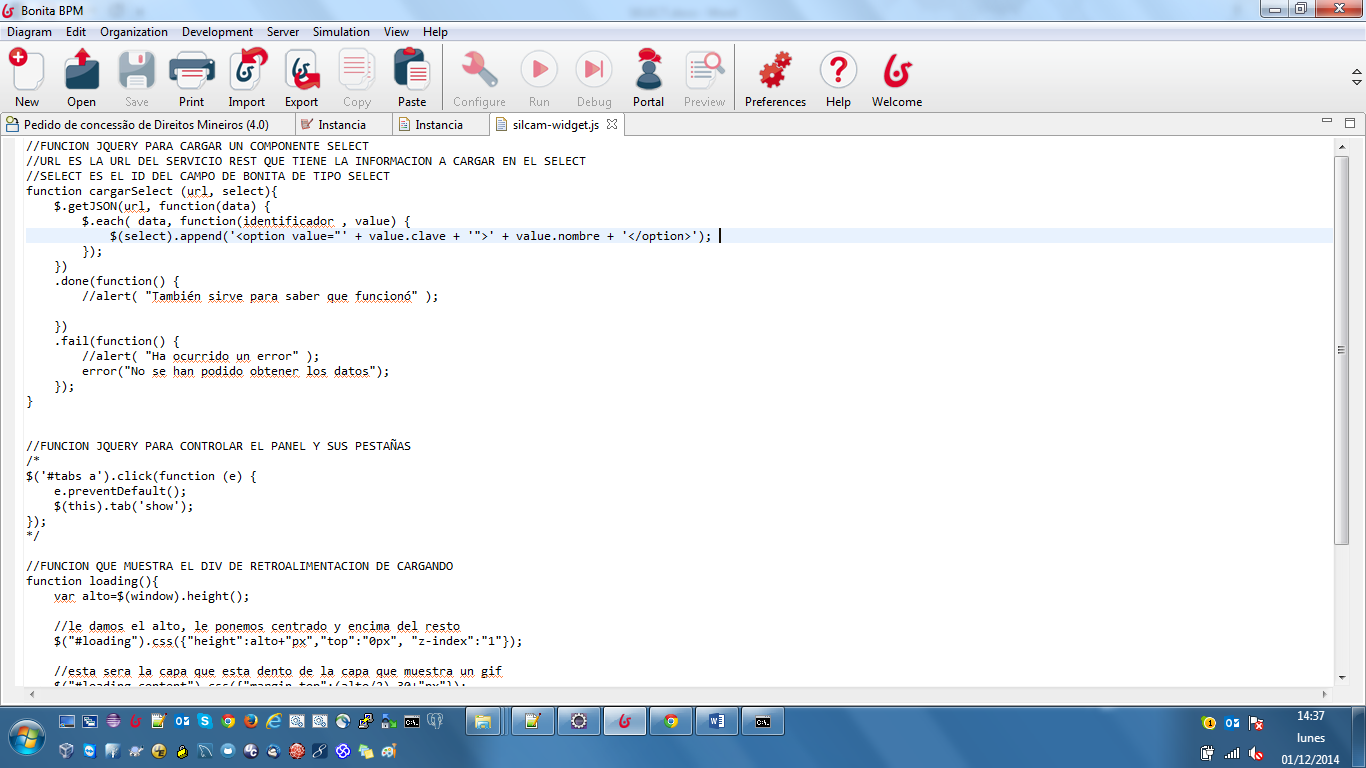
$('.select\_localidade').append('<option value="">Selecione uma comuna.</option>');

}

}

Dentro de los recursos del Proyecto en silcam-widget.js tenemos la función genérica que carga con datos el select





**function** cargarSelect (url, select){

$.getJSON(url, **function**(data) {

$.each( data, **function**(identificador , value) {

$(select).append('<option value="' + value.clave + '">' + value.nombre + '</option>');

});

})

.done(**function**() {

//alert( "También sirve para saber que funcionó" );

})

.fail(**function**() {

//alert( "Ha ocurrido un error" );

});

}

El truco está en que los components no se renderizan por pantalla hasta que se ha cargado completamente la página y por tanto no están accesibles por javascript. Una vez cargada la página por jQuery le añadimos los eventos, atributos, valores etc al componente identificado por su estilo dentro del div.

Usar js para obtener los datos por rest/BBDD y modificar los selects con estos datos.

**Referencias:**

<http://www.lawebdelprogramador.com/codigo/JQuery/2308-Ejemplo_de_utilizar_jquery_con_un_select.html>

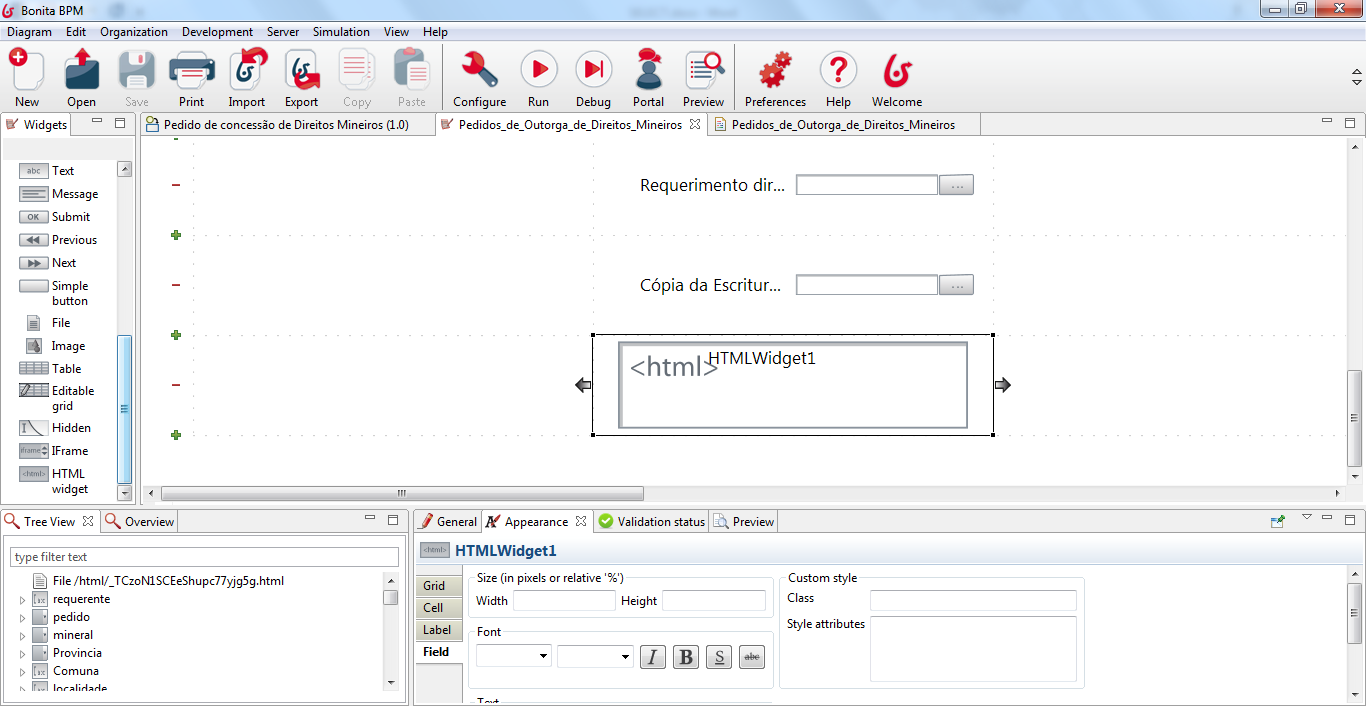
<http://www.actualidadjquery.es/2011/12/24/agregar-eliminar-elementos-dinamicamente-en-un-campo-select-con-jquery/>

### Generar los componentes desde un html widget.

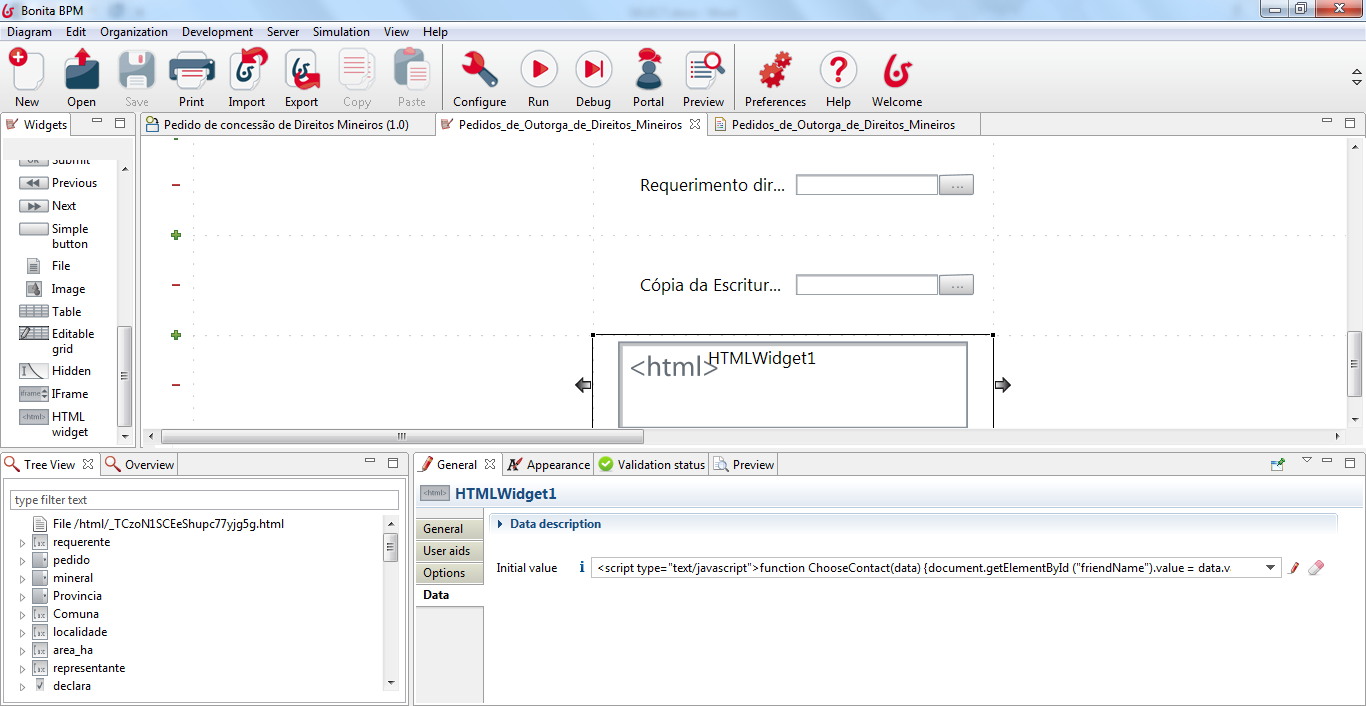
Las pegas que tiene esta solución es que al no ser componentes creados por bonita, el motor no controla su validación, mapeos de datos etc, automáticamente, tendríamos que programarla.

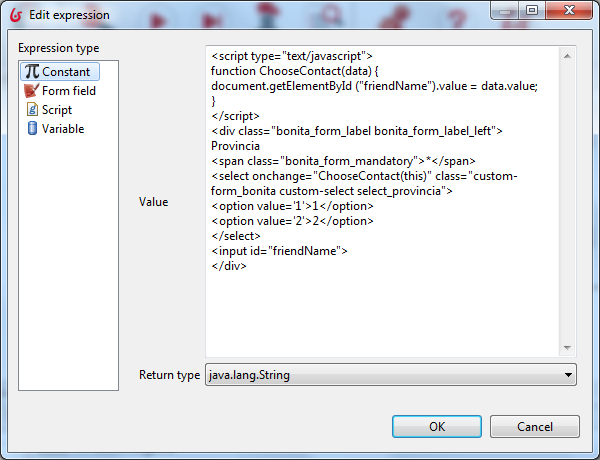
Las ventajas es que al programarlo nosotros, tenemos control completo sobre el componente.

Se incluye un componente HTMLWidget



En data en valor inicial, incluimos el html y JavaScript necesario





<script type="text/javascript">

function ChooseContact(data) {

document.getElementById ("friendName").value = data.value;

}

</script>

<div class="bonita\_form\_label bonita\_form\_label\_left">

Provincia

<span class="bonita\_form\_mandatory">\*</span>

<select onchange="ChooseContact(this)" class="custom-form\_bonita custom-select select\_provincia">

<option value='1'>1</option>

<option value='2'>2</option>

</select>

<input id="friendName">

</div>