UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de interfaces de usuario para el control de viviendas inteligentes

Proyecto Final de Carrera

[Ingeniería Informática]

Autor: [José Enrique Giménez Osmán]

Director: [Joan Fons i Cors]

[Diciembre 2012]

Tabla de contenidos

1 Tabla de Ilustraciones

2 Introducción

Este documento es la memoria de la realización del proyecto final de carrera por el alumno José Enrique Giménez Osmán de la Escuela Técnica Superior de Ingeniería Informática en la Universidad Politécnica de Valencia.

2.1 Motivación

La selección de este proyecto viene motivada por mi interés por la programación de aplicaciones para dispositivos móviles, aunque este no se centra en ninguno en concreto sí se pretende que la parte del la interfaz con el usuario pueda ser manejada desde un dispositivo móvil.

2.2 Contexto y tecnología

2.2.1 Domótica

La domótica consiste en la automatización de los procesos que suceden dentro de un hogar, como puede ser la gestión energética, el bienestar, la seguridad y comunicación y que pueden ser controlados desde dentro como desde fuera de casa. Como por ejemplo que cuando empiece a oscurecer se enciendan automáticamente las luces donde estén las personas y se bajen las persianas.

Pretende hacer más fácil y cómoda el día a día de las personas que viven en una casa.

2.2.2 Dispositivos móviles

Aunque dispositivos móviles engloba a un gran número de aparatos en nuestro contexto llamaremos dispositivos móviles a aquellos dispositivos con capacidad de comunicación a través de internet y capacidad de introducción de datos mediante una pantalla táctil, como puede ser un *smartphone* o una *tableta*.

2.3 Propósito

El propósito principal de este proyecto nace de la necesidad de tener una interfaz de usuario que sea intuitiva, fácil de manejar, usable y transparente al usuario, para la introducción de reglas de comportamiento en una casa domótica, principalmente desde un dispositivo móvil.

Hasta este momento la introducción de estas reglas se ha hecho de forma rudimentaria directamente en el archivo de configuración de la casa domótica. Estas reglas de comportamiento son las que le dicen a la casa como tiene que actuar frete a distintos estímulos provenientes tanto del interior como del exterior.

El proyecto se engloba dentro de un proyecto de casa domótica mucho más grande y complejo. La parte del proyecto dedicada al servidor se añadirá a este en forma de *servlet* que accederá a los recursos disponibles que han sido definidos en el archivo de configuración de la casa domótica. La parte de interfaz web se colgará en un servidor web para el acceso de los usuarios a través de internet.

3 Gestión Técnica

En este apartado se va a abordar la elección de la plataforma y entorno en el que se va a desarrollar técnicamente el proyecto.

3.1 Desarrollo web para dispositivos Móviles

La proliferación de dispositivos móviles en los últimos años y el aumento que va a haber en los próximos años[1], y aunque el sistema operativo con más crecimiento en estos momentos es *Android*[2], se ha optado por la realización de la interfaz de usuario mediante tecnología web ya que se pretende que cualquier dispositivo móvil pueda acceder fácilmente.

Para la realización de esta parte se optó en un primer momento por la utilización de *iWebKit*, que es un framework de libre utilización, pero tras varias pruebas se comprobó que está optimizado para el uso en el explorador del *iPhone* dando fallos de visualización en otros navegadores como por ejemplo el Mozilla Firefox.

Tras una búsqueda de distintos frameworks finalmente se optó por realizar la interfaz con la ayuda de *jQuery Mobile* que es una librería multiplataforma realizada en *javascript* con componentes visuales en HTML5 diseñados específicamente para dispositivos móviles y que además ha sido probado en la mayoría de sistemas operativos de dispositivos móviles, como Android, iOS, Windos Phone, Symbian y en sus respectivos navegadores [3], como en navegadores de escritorio como Mozilla Firefox, Chrome, Safari o incluso en el Internet Explorer.

Esta librería se tiene que utilizar conjuntamente con la librería general *jQuery* para que funcione.

La interfaz web también hace uso del lenguaje de programación web *php* para la comunicación final con el servidor.

La parte correspondiente al servidor se hizo en la misma tecnología en la que está el resto del proyecto en el que se integrará que es *Java*.

[1] http://www.ticbeat.com/sim/2016-habra-mas-dispositvos-moviles-personas/
[2] http://www.android.es/crecimiento-de-ios-y-android-en-una-infografia.html
[3] http://jquerymobile.com/gbs/

Ilustración 1 Dispositivos mostrando jQuery Mobile

3.2 Herramientas de Desarrollo

3.2.1 Interfaz Web

Para el desarrollo, implementación y depuración de la interfaz web se han utilizado varios programas que describo a continuación.

3.2.1.1 IDE Netbeans

Para gestionar y probar correctamente el desarrollo web se utilizó el entorno de desarrollo integrado (IDE) *Netbeans* en su versión 7.2 con el módulo de desarrollo de PHP.

Este entorno proporciona herramientas de control de versiones, marcado de código y ejecución paso a paso que facilita el desarrollo de aplicaciones, así como la gestión y organización de los archivos de toda la aplicación como un proyecto del IDE.

3.2.1.2 WampServer

Como el código *php* es código que se ejecuta en el servidor antes de ser enviado al navegador se optó por utilizar la aplicación *WampServer* ya que su

utilización es muy sencilla. Este programa incluye todo lo necesario para tener un servidor web instalado en un ordenador.

Tras instalarlo, en el del directorio de instalación hay que crear un directorio para nuestra aplicación web dentro del directorio llamado "*www*" y la página inicial tiene que llamarse "*index",* en nuestro caso como vamos a utilizar php se llamará "*index.php"*.

Para acceder a la web, una vez creado el directorio y el archivo principal, solo hay que iniciar la aplicación "wampmanager.exe" y escribir en la barra de dirección del navegador web "localhost", esto hará que aparezca la web de información del *WampServer* y en el apartado "Your Projects" habrá un enlace a la web que acabamos de crear que se llamará igual que el nombre que se le haya dado al directorio de la aplicación web.

La aplicación incluye varias aplicaciones ya configuradas para trabajar juntas. En la versión utilizada para este proyecto incluye el servidor web Apache 2.2.22, PHP 5.4.3, XDebug 2.1.2, Mysql 5.5.24, XDC 1.5, PhpMyadmin 3.4.10.1, SQLBuddy 1.3.3 y webGrind 1.0, de las cuales las que nos interesan son el servidor Apache y el PHP que enviarán las páginas web creadas al navegador y el XDebug, que nos permitirá depurar y probar el código paso a paso en el navegador.

3.2.1.3 Navegadores

Para probar la aplicación web se han utilizado los navegadores Mozilla Firefox con el plugin FireBug para depurar y el navegador Iron, basado en Chrome, y las herramientas para desarrolladores que lleva incorporadas.

También se ha probado en el navegador de un smartphone con sistema operativo Android, concretamente en el HTC Wildfire.

Características más importantes del smartphone para nuestro proyecto:

- Sistema operativo Android 2.2.
- Pantalla táctil capacitiva 3,2 pulgadas 16M colores.
- Resolución QVGA 240 x 320 píxeles.
- Navegador de Android 2.2 basado en WebKit.
- Comunicación a través de Internet por conexión de datos 3G (3G HSDPA 7,2 Mbps).
- Conexión wifi (WIFI 802.11 b/g).

Ilustración 2 HTC Wildfire

3.2.2 Interfaz del servidor

La interfaz con el servidor se ha desarrollado utilizando el IDE *Eclipse Modeling Tools* y se ha desarrollado con el lenguaje de programación orientado a objetos Java.

La necesidad de intercambio de gran cantidad de datos entre el cliente, aplicación web, y el servidor, aplicación java (*servlet*), ha hecho necesaria la inclusión de una librería para la conversión y manejo de estos datos de una forma sencilla y estructurada tanto desde el cliente como desde el servidor.

Para ello se ha incluido una librería para convertir los datos de intercambio en formato json. La librería elegida ha sido Json-Smart en la versión 1.1.1, "*json-smart-1.1.1.jar*". Para la aplicación web no ha hecho falta ninguna librería ya que la implementación de javascript en los navegadores modernos ya lleva incorporada la función de tratamiento de datos en formato json.

Aunque en el fondo al final el intercambio de datos sique siendo texto, este está estructurado de forma que con las funciones de json se pueden convertir en objetos para acceder a sus propiedades fácilmente.

Ejemplo de texto en formato json sacado de la wikipedia[4]:

```
{
    "menu":{
        "id":"file",
        "value":"File",
        "popup":{
            "menuitem":[
                {
                    "value":"New",
                    "onclick":"CreateNewDoc()"
                },
                {
                    "value":"Open",
                    "onclick":"OpenDoc()"
                },
                {
                    "value":"Close",
                    "onclick":"CloseDoc()"
                }
            ]
        }
    }
}
```

[4] http://es.wikipedia.org/wiki/JSON

4 Familiarización

Al ser un proyecto que se va a integrar en otro mucho mayor, que llamaré de aquí en adelante SmartHome, y va a acceder a parte de sus clases y recursos, previamente al desarrollo del proyecto ha habido un largo periodo de familiarización y estudio de este proyecto ya que su esquema de clases es muy grande y complejo.

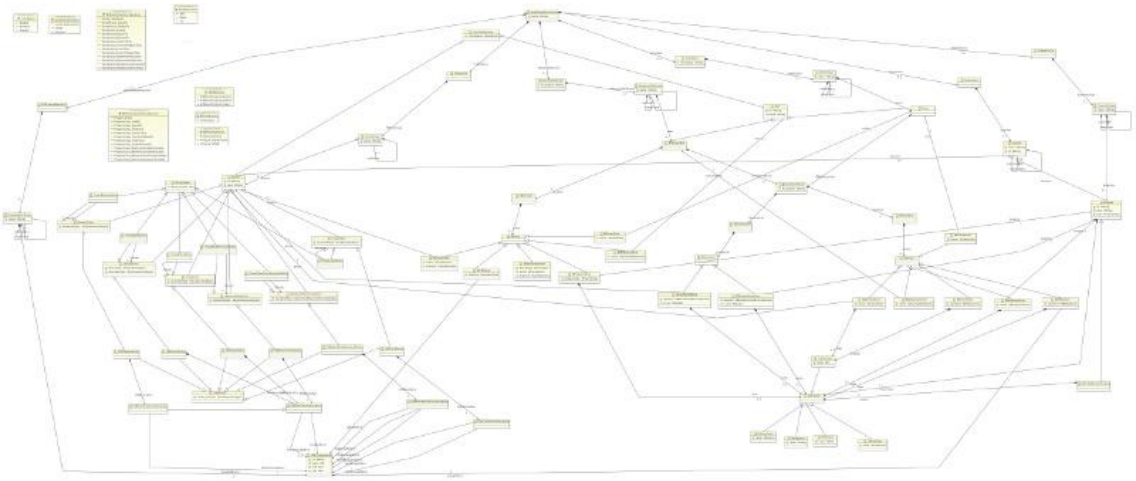

Ilustración 3 Esquema de clases del SmartHome

Durante el análisis se ha tenido que identificar las clases necesarias que se iban a emplear en el proyecto.

También ha habido un periodo de aprendizaje de las librerías *jQuery* y *jQuery Mobile* para identificar que funciones y componentes se adecuarían más a la hora de implementar la interfaz web del proyecto.

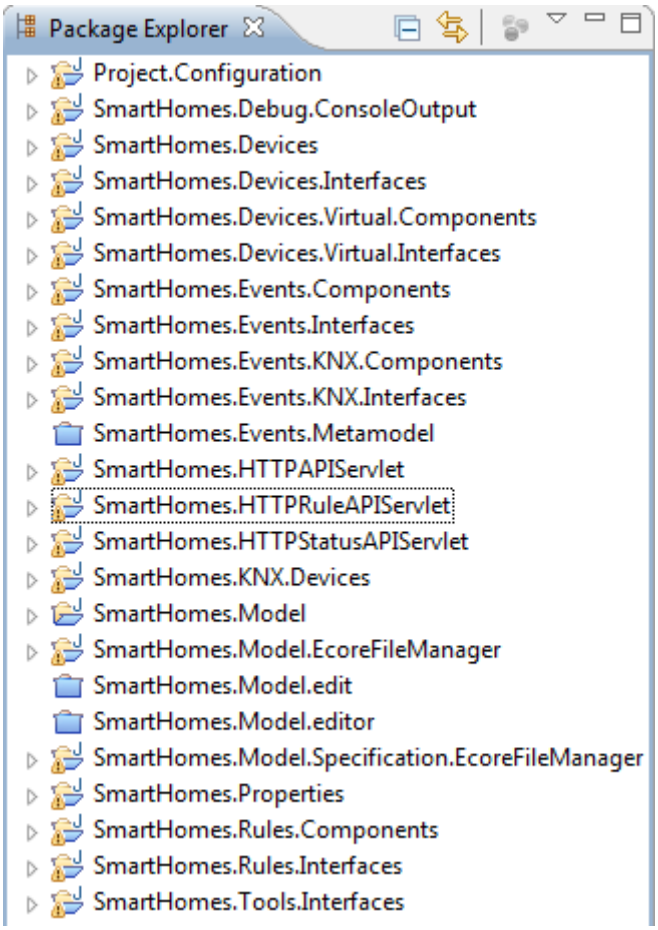

Ilustración 4 Proyectos que componen el SmartHome

5 Análisis

En este apartado se va describir las bases de lo que tiene que hacer la aplicación y su funcionalidad básica para que llegue a buen término el prototipo que se va a desarrollar.

5.1 Objetivos

El objetivo principal del desarrollo de este proyecto es la implementación de una interfaz web de comunicación entre el usuario y la casa domótica de una manera transparente, sencilla y intuitiva para que una persona que no esté familiarizada con los ordenadores pueda introducir reglas de comportamiento en el sistema de la casa domótica desde un dispositivo móvil.

5.2 Especificaciones, Requisitos y Funcionalidades

En este apartado se va a realizar una descripción de lo que tiene que cumplir la aplicación y en base a esta se realizará la implementación. El análisis inicial, como en toda aplicación, se ha ido modificando a medida que se avanzaba en la resolución del proyecto para ir ajustándose a lo que se esperaba de la aplicación.

- La aplicación deberá permitir la introducción, eliminación y edición de reglas de comportamiento de una casa domótica de una manera sencilla e intuitiva.
- La aplicación utilizará los datos del modelo de la casa que esté actualmente cargada en el servidor.
- La aplicación permitirá la introducción y eliminación de todos los tipos de *eventos* que se encuentren definidos en el modelo de la casa y cada una de sus propiedades.
- La aplicación permitirá la introducción y eliminación de todos los tipos de *condiciones* definidos en el modelo de la casa y todas sus propiedades.

- La aplicación permitirá la eliminación de todos los tipos de *acciones* definidas en el modelo de la casa y todas sus propiedades.
- La aplicación se comunicará mediante una API con el servidor para gestionar todas las peticiones de obtención y envío de datos.
- La introducción de los datos será guiada por la aplicación a modo de *wizard*, o sea, como un asistente paso por paso.
- La pantalla principal mostrará dos opciones, introducir nueva regla y seleccionar una regla ya existente.
- En la pantalla de nueva regla se podrán introducir las propiedades de la nueva regla como son el nombre y el identificador.
- Habrá una pantalla para indicar en qué grupo de los ya creados se quiere añadir la regla, ya que por restricciones del modelo solo se puede añadir una regla en un grupo de reglas. Si no hay ninguno se creará uno con un nombre predefinido.
- En la pantalla de introducción de eventos se podrán introducir y eliminar cada tipo de evento con sus propiedades y se mostrarán en una lista los ya introducidos.
- En la pantalla de introducción de condiciones se podrán introducir y eliminar cada tipo de condición con sus propiedades y se mostrarán en una lista los ya introducidos.
- En la pantalla de introducción de acciones será igual que las condiciones.

6 Diseño

En este apartado se va a explicar cómo se llevó a cabo tanto el diseño de la aplicación web como de la aplicación *servlet,* que enviará desde el servidor la información que la aplicación web demande. Se realizará una descripción de las funcionalidades de las clases implementadas y sus propiedades.

6.1 Interfaz web

Como ya he comentado anteriormente queremos implementar una interfaz web que se comunique con el modelo de la casa domótica y que sea intuitivo, transparente al usuario y fácil de manejar. Esto no siempre es fácil de conseguir y lleva un proceso de prueba y error hasta encontrar los componentes que se ajusten a las necesidades que se requieren.

Para poderse manejar bien en un dispositivo móvil, dado lo limitado del tamaño de la pantalla, la interfaz tiene que tener una serie de características fundamentales como son:

- Botones grandes, visibles y descriptivos de su funcionalidad.
- Pocos componentes a la vez en cada pantalla.
- Textos cortos y descriptivos.

En un primer diseño que se realizó de la aplicación web se especificó que desde la pantalla principal el usuario tendría acceso a todos los componentes de una regla, eventos, condiciones y acciones, pero se observó que el usuario podría llegar a liarse y en un momento dado no saber en qué parte de la aplicación se encontraba. Por ejemplo en la siguiente ilustración se muestra como en un principio iba a ser la interfaz de “Condiciones – Acciones”.

Como se puede observar en la ilustración hay demasiados componentes a la vez y cuesta de entender a simple vista. En la nueva versión se ha simplificado más ya que para la pantalla de un dispositivo móvil son muchos datos y componentes a la vez y no se verían bien además que sería difícil para el usuario poder seleccionarlos ya que los controles mostrados serian muy pequeños.

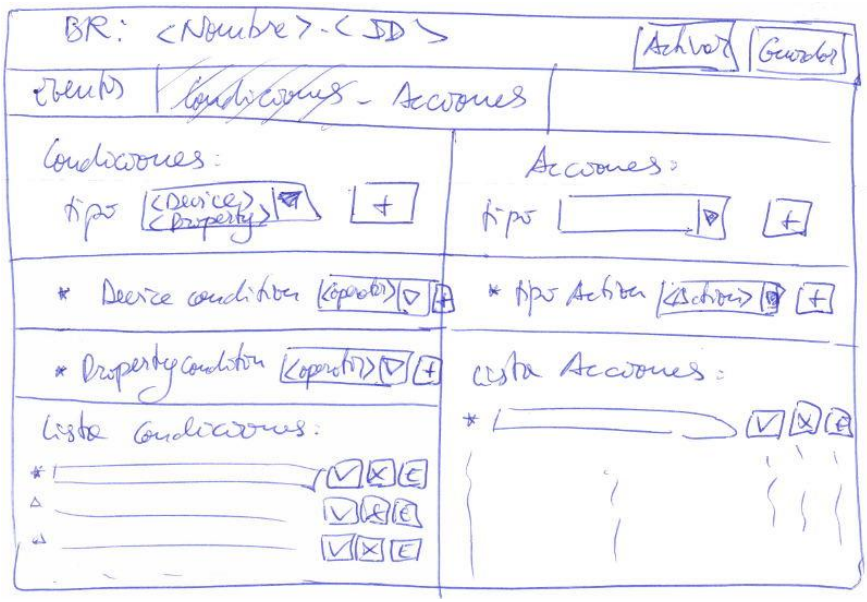

Ilustración 5 Primera versión del diseño de la interfaz de Condiciones-Acciones

Tras analizar el resultado se llegó a la conclusión que realizar la interfaz como un asistente (wizard), en el que el usuario va paso por paso rellenando cada parte de una regla, sería la mejor opción para el desarrollo de la interfaz web.

Se hizo un rediseño de la interfaz web y por ejemplo la interfaz de "Condiciones – Acciones" quedó dividida en varias partes. Por un lado la gestión de las listas de "Condiciones – Acciones", y por otro condiciones y acciones que se separó en dos pantallas distintas aunque conectadas. Además cada tipo de condición se separó en una pantalla distinta e igual se hizo con acciones.

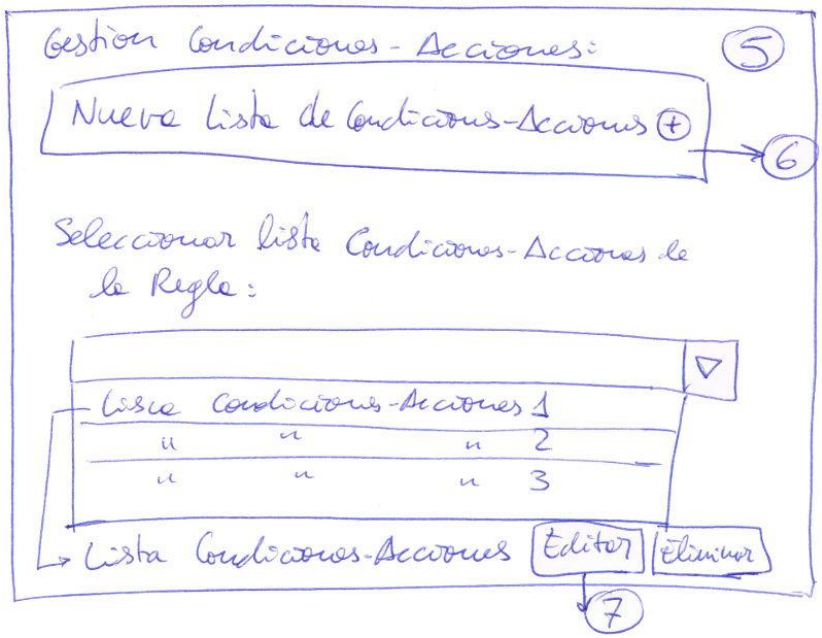

Ilustración 6 Gestión Condiciones - Acciones

6.2 Esquema de pantallas del *wizard*

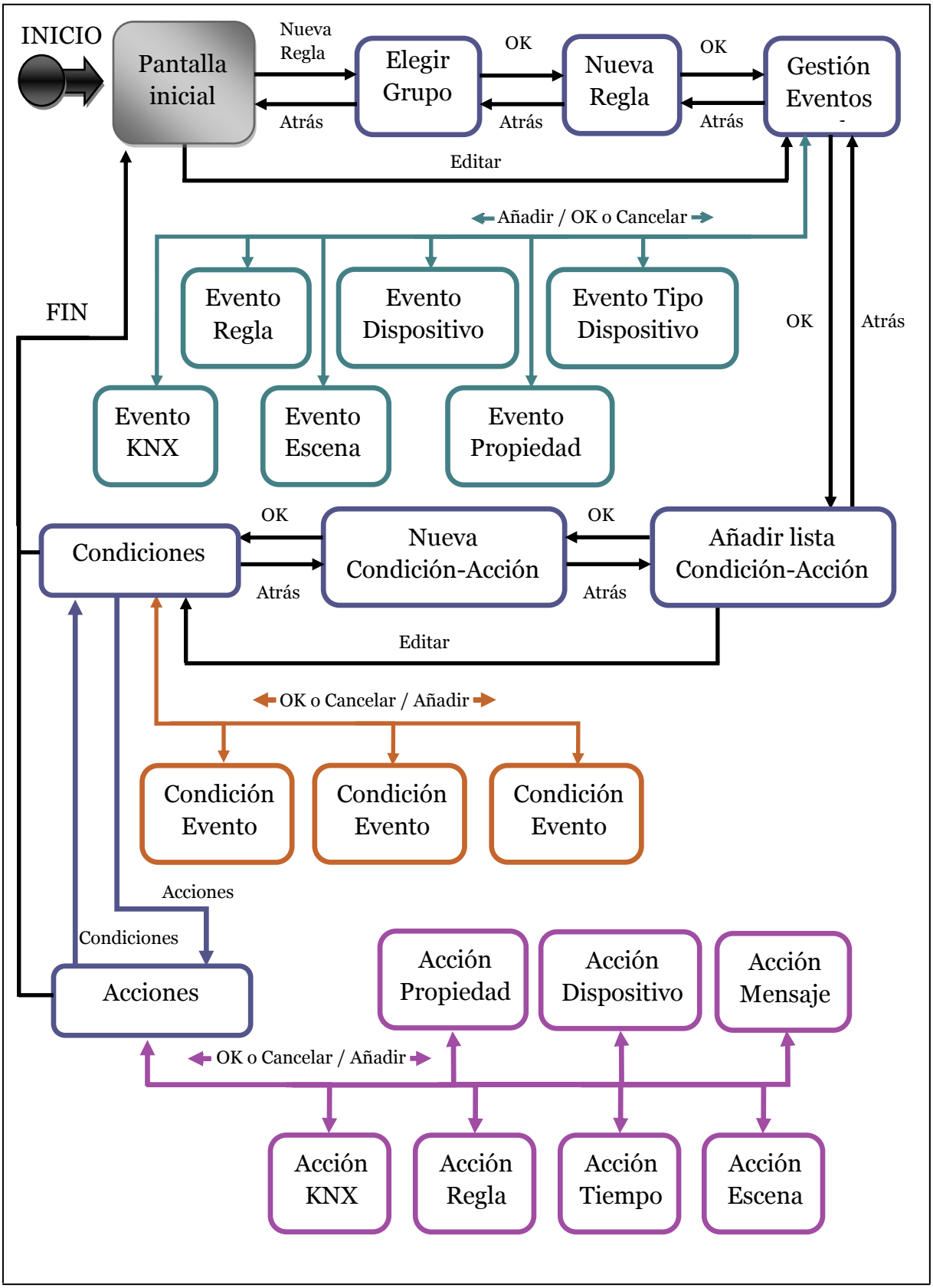

Ilustración 7 Esquema de pantallas de la interfaz web

En la ilustración anterior se expone el esquema de pantallas que va a tener el *wizard*, por las que tiene que pasar el usuario para completar la creación o edición de una regla. Las flechas indican la navegabilidad de la aplicación, hacia donde puede navegar el usuario desde una pantalla a otra. Por motivos de espacio los tipos de eventos, condiciones y acciones salen desde una misma flecha pero cada tipo no está conectado entre ellos.

6.3 Servidor de datos, *servlet*

En este apartado se va a mostrar el diseño de clases realizado para la aplicación que se ejecutará en el servidor y responderá a las peticiones de la aplicación web.

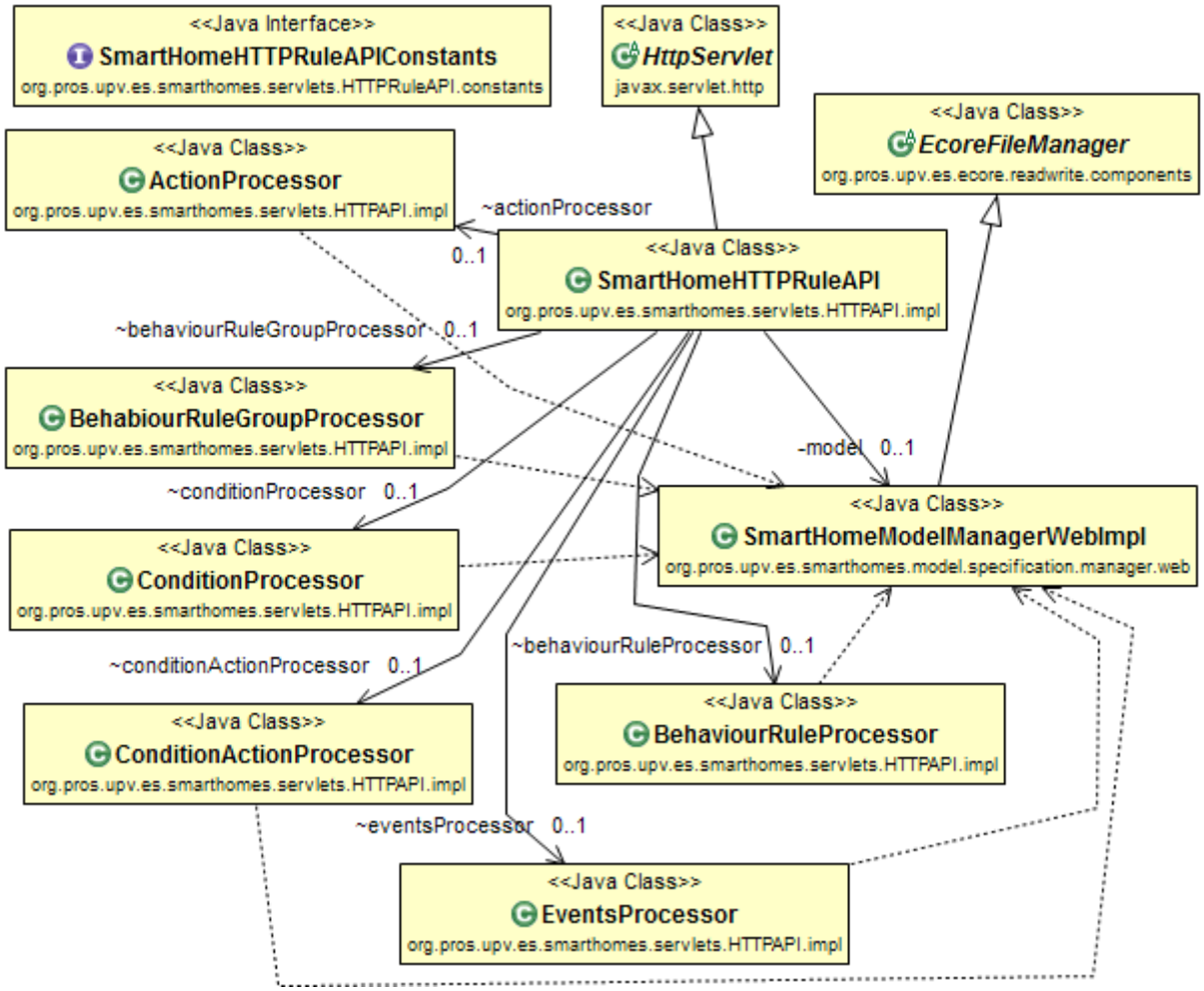

Ilustración 8 Esquema general de clases del Servlet

En la ilustración se puede observar la relación de clases que hay en el *servlet*. Únicamente se muestran las clases necesarias sin atributos y sin funciones.

A parte de estas clases en el proyecto hay otras clases utilizadas para otras funcionalidades, como mostrar por consola mensajes de texto que se van generando durante la ejecución o la clase activadora del *servlet*.

6.3.1 Descripción de clases

En este apartado se va a describir la funcionalidad que va a tener cada clase del *servlet*.

6.3.1.1 SmartHomeHTTPRuleAPI

Es la clase principal de la aplicación y todas las demás se crean a partir de esta clase. Esta clase se creará al iniciar el servidor y se quedará a la escucha, como servicio del servidor. Extenderá de hhtpServlet y se encargará de recibir las peticiones enviadas desde el navegador y distribuirlas por cada clase *processor* dependiendo del tipo de petición que se le haga.

Se creará una función de procesamiento de los datos que llegarán al servidor y que será llamada por los métodos redefinidos *doGet* y *doPost*, heredados de httpServlet, para que aunque se llamé por *get* o *post* no se pierda la información enviada. Esta función será la que se encargará de distribuir las peticiones dependiendo del tipo.

6.3.1.2 BehaviourRuleProcessor

Es la clase que se encargará de procesar las peticiones de tipo regla.

Se encargará de buscar todas las reglas presentes en el modelo y que están registradas como servicios en la SmartHome.

Procesará las órdenes de creación de nuevas reglas comprobando que no hay otra regla con el mismo identificador.

Procesará el borrado de reglas creadas y presentes en el modelo.

Seleccionará la regla que se vaya a editar.

6.3.1.3 BehaviourRuleGroupProcessor

Se encargará de procesar las peticiones de tipo grupo.

Buscará todos los grupos de reglas del modelo y los devolverá a la aplicación web.

Seleccionará el grupo que elija el usuario para añadir las nuevas reglas que se crearán.

6.3.1.4 EventsProcessor

Se encargará de las peticiones que lleguen al *servlet* de tipo evento.

Devolverá todos los eventos que contenga la regla.

Creará e insertará los distintos tipos de eventos en la regla que se esté editando o creando.

Se encargará de borrar los eventos seleccionados por el usuario.

6.3.1.5 ConditionActionProcessor

Esta clase se encargará de procesar las peticiones de tipo Condición-Acción que lleguen al *servlet*.

Devolverá todas las listas de condiciones-acciones de una regla.

Se encargará de seleccionar la lista de condición-acción que elija el usuario.

Creará nuevas listas de condiciones-acciones para la regla seleccionada en el *servlet*.

También se encargará de borrar las listas de condiciones-acciones.

6.3.1.6 ConditionProcessor

La clase se encargará de procesar las peticiones de tipo condición.

Devolverá todas las condiciones de una lista Condición-Acción.

Gestionará la creación de nuevas condiciones.

Borrará las condiciones que seleccione el usuario.

6.3.1.7 ActionProcessor

Esta clase es básicamente como la anterior solo que con las Acciones.

6.3.1.8 SmartHomeModelManagerWebImpl

Esta clase se encargará de gestionar todas las peticiones que se realizarán sobre el modelo de la casa. Actuará a modo de mánager.

Hereda de la clase EcoreFileManager que es una clase del proyecto de SmartHome que se encarga de gestionar la carga del archivo de configuración de la casa SmartHome.

Todas las clases *processor* enviarán las peticiones a esta clase.

Esta clase contendrá todas las acciones que se pueden hacer sobre las reglas y todos sus componentes y propiedades, como buscar, seleccionar, editar, insertar y borrar. Será una clase bastante extensa ya que hay muchas acciones a realizar.

6.3.1.9 SmartHomeHTTPRuleAPIConstants

En esta clase se definirán las constantes que utilizará la API para el intercambio de información entre la aplicación web y el servidor. Estas constantes se utilizarán tanto en la web como en el *servlet* para poder identificar que acción hay que ejecutar en cada momento.

6.4 Interacción Cliente - Servidor

La arquitectura que se va a emplear es una arquitectura cliente-servidor, en el que el cliente será la aplicación web y el servidor será la aplicación *servlet* que se ejecutará en el servidor a la escucha de las peticiones que le vaya realizando el cliente.

Está diseñada para que solamente haya un usuario a la vez utilizando la aplicación.

Las peticiones entre el cliente y servidor se realizaran mediante peticiones http utilizando los métodos *GET* o *POST*. Se podrán utilizar los 2 ya que el *servlet* estará diseñado para que pueda atender los dos tipos de peticiones.

El usuario mediante el navegador de su dispositivo móvil accederá a la aplicación web y esta realizará las peticiones necesarias a la aplicación *servlet* para obtener la información que el usuario pida.

Este intercambio de información se realizará de forma transparente al usuario.

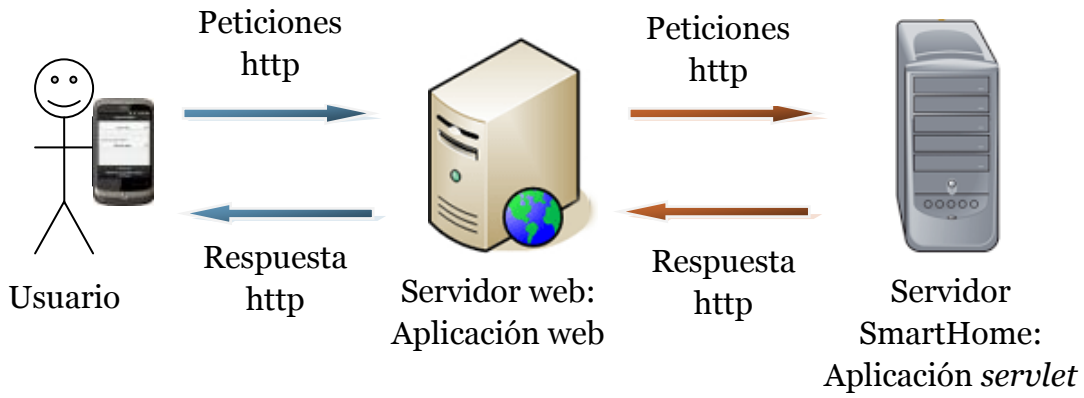

Ilustración 9 Arquitectura Cliente - Servidor

7 Implementación

7.1 Comunicación

Para la comunicación entre la aplicación web y el *servlet* se ha implementado una API para que las dos partes sepan que es lo que quiere hacer el usuario en cada momento.

Esta API se compone de varios parámetros a los que se dará un valor y se enviarán por peticiones http entre cliente y servidor.

7.1.1 Parámetros del API

Type. Este parámetro indicará de qué tipo es la petición. Reconocerá los tipos siguientes:

- Rule: Para las operaciones con reglas.
- Group: Ídem para grupos de reglas.
- Event: Ídem para los eventos de una regla.
- Ca: Ídem para las listas de condiciones acciones.
- Condition: Ídem para las condiciones en una lista de condiciones - acciones.
- Action: Ídem para las acciones en una lista de condiciones - acciones.

Action. Este parámetro indicará las acciones a realizar dependiendo del tipo al que se refiera:

- Todas las acciones que comienzan por “get” devolverán la lista de elementos del tipo al que se refiere.
- Select: Este parámetro seleccionará un elemento en el servidor y siempre irá acompañado del identificador del elemento a seleccionar, en forma de parámetro y podrán ser “id”, “name” o “index”.
- Delete: Este parámetro indicará que hay que borrar el elemento al que acompañe. Como select siempre irá acompañado de otro parámetro para indicar que elemento hay que borrar (“id”, “name” o “index”).
- New: este indicará que hay que añadir una nueva regla o un nuevo elemento en ella. Siempre irá acompañado de los datos necesarios para la creación del nuevo elemento, como por ejemplo en el caso de las reglas el identificador y su nombre.

7.2 Aplicación Web

En esta parte vamos a explicar cómo se ha llevado a cabo la implementación del la aplicación web, comentando algunas funciones características y como partiendo de los prototipos diseñados en papel se ha llegado a la implementación del código y como se ha asignado el mejor componente del framework para realizar su función.

Para la implementación se ha tenido en cuenta varios factores que influyen notablemente en la resolución del problema cuando la aplicación está destinada sobre todo a ser utilizada en un dispositivo móvil y que se han tenido en cuenta desde la etapa de diseño. Estos factores son:

- La funcionalidad es más importante que el estilo visual que se quiera dar a la aplicación.
- Es más importante la simplicidad que la belleza del acabado.
- Textos muy largos son difíciles de leer.
- Reducir en lo posible todos los campos en los que el usuario tenga que introducir texto ya que los usuarios no tiene teclados y ratones convencionales y es más difícil su introducción.
- El diseño de las páginas tienen que mantener la premisa del inglés *Short & Sweet*, o en castellano sería algo así como el refrán "Bueno, si breve, bueno dos veces".

Para la realización de la interfaz web se optó por utilizar la tecnología *jQuery Mobile* que en el fondo es programar en html y simplemente incluyendo la librería en la cabecera y añadiendo unos campos *data* a las etiquetas *jQuery Mobile* aplica su "*magia*". Esta librería dispone de componentes visuales para hacer más fácil su uso en dispositivos móviles. Todos los estilos css de estos componentes están controlados por *jQuery* y son transparentes al programador. Además incorpora navegación por *AJAX* y animaciones de transición entre páginas.

7.2.1 Estructura de la aplicación web

La aplicación web se ha estructurado en una serie de archivos y directorios para su fácil manejo y modificación que explico a continuación:

- **Css**: En este directorio van todos los archivos relativos los estilos, tanto los archivos con extensión css como las imágenes a las que hacen referencia.
- **Images**: En este directorio van las imágenes de la aplicación.
- **Include**: En este van los archivos que se van a incluir en la aplicación de forma general como son la cabecera y la configuración.
- **Js**: Aquí van todos los archivos con extensión js que son los archivos javascript que contienen toda la funcionalidad de la aplicación.
- **Xdomain**: En este directorio va el archivo php, que hace de proxy entre la interfaz web y el servidor, que se ha utilizado para solucionar el problema del *cross domain call* y que se explicará más adelante.
- **El resto de archivos**, los correspondientes a la implementación de la interfaz gráfica, irán en el directorio raíz de la aplicación.

7.2.2 Cabecera

En la implementación se decidió separa la cabecera de todas las páginas en un archivo a parte, ya que iba a ser igual para todas, e incluirla por *php* en todas ellas ahorrándonos así el engorro que supone hacer modificaciones en la cabecera, porque si no habría que hacer la modificación en todas las páginas y de este modo solo hay que modificar en un sitio.

La cabecera para una aplicación web tiene que tener una etiqueta **`meta`** de nombre `viewport` y con el contenido `width=device-width, initial-scale=1` para su correcta visualización. En esta etiqueta además se ha añadido una opción más para mejorar la apariencia de la aplicación, `user-scalable=no`, que deshabilita la opción de poder hacer zoom en las páginas web de la aplicación.

También se han añadido dos etiquetas más para mejorar la visualización en móviles iPhone:

```
<meta name="apple-mobile-web-app-capable" content="yes">
```

Que hace que las aplicaciones web se muestren a pantalla completa.

```
<meta name="apple-mobile-web-app-status-bar-style"content="black">
```

Que solo tiene efecto si está activo el modo a pantalla completa y hace que la barra de estado tenga color de fondo negro.

Después se incluye la referencia a los estilos que se van a aplicar a la web. Aquí se han incluido el archivo necesario para la librería *jQuery Mobile* en su versión comprimida (jquery.mobile-1.2.0.min.css), un archivo que amplia los iconos por defecto de *jQuery Mobile* con más iconos y un archivo de estilos creados para la aplicación (gestionReglas.css).

Parte del código de la cabecera quedará como se muestra a continuación:

```
<?php

include_once "config.php";

$PAGE_TITLE="Gestión de Reglas";
?>

<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1,
user-scalable=no" />
    <!-- Fullscreen docs in iOS
(http://jquerymobile.com/test/docs/config/index.html) -->
    <meta name="apple-mobile-web-app-capable" content="yes">
    <meta name="apple-mobile-web-app-status-bar-style" content="black">

    <title><?php echo $PAGE_TITLE ?></title>
    <link rel="stylesheet" href="css/jquery.mobile-1.2.0.min.css" />
    <link rel="stylesheet" href="css/jqm-icon-pack-2.0-original.css" />
<!-- More icons -->
    <link rel="stylesheet" href="css/gestionReglas.css" /><!-- Estilos
para la app -->
```

Después hay que incluir el fichero de la librería *jQuery*, en mi caso la versión 1.7.1, después el script de configuración global de la aplicación y luego la librería de *jQuery Mobile*. Este orden es importante porque cuando se inicia *jQuery Mobile* el primer evento en ser lazado en cuanto se carga la librería es **mobileinit**, que es donde hay que poner la configuración que se necesite al iniciar la aplicación, si no se hace así la configuración de inicialización personalizada no tendrá ningún efecto. La configuración inicial simplemente ha sido cambiar varios textos al castellano.

```
<script src="js/lib/jquery-1.7.1.min.js"></script>
    <script>
        $(document).bind("mobileinit", function() {
            $.mobile.dialog.prototype.options.closeBtnText = "Cerrar";
            $.mobile.pageLoadErrorMessage = "Error al cagar la página.";
        });
    </script>
    <script src="js/lib/jquery.mobile-1.2.0.min.js"></script>
```

A partir de ahí se ha añadido el código de las funciones que se van a usar en la aplicación y detrás los scripts correspondientes a la funcionalidad de cada página.

A continuación explicaré una muestra de funciones ya que el código empleado es muy parecido, solo cambia la gestión de los datos de un componente a otro de la interfaz gráfica.

Se ha creado una variable global para la aplicación llamada BehaviourRule en la que se irá guardando los valores de identificación que se vayan seleccionando.

Se ha implementado una función *parseJSON(datos)* que comprueba que el servidor está activo y si el modelo de la SmartHome está cargado en él. Esto lo hace comprobando los datos devueltos por el *servlet* y con la función *parse* de JSON. Si al pasar los datos por el parser da error es que los datos devueltos no tienen estructura de json por lo que el servidor no está activo. Si los datos pasan por el parser es que los datos están en estructura json por lo que ahora se comprobará que ha devuelto datos. Si no devuelve nada es que el modelo no está cargado porque para esta petición el *servlet* siempre devuelve al menos el nombre del modelo cargado en el *servlet*.

```
function parseJSON(datos){
    var r = "";
    try {
        r = JSON.parse( '{"code":0, "content":'+datos+'}' );
    } catch(e) {
        return JSON.parse( '{"code":1, "content":"<ul><li>Error, el
servidor no esta activo.</li><br><li>Active el servidor y recargue la
aplicación.</li></ul>"}' );
    }
    if (r.content == ""){
        return JSON.parse( '{"code":2, "content":"<ul><li>El modelo no
esta cargado en el servidor.</li>'+
            '<br><li>Comprueba que el modelo esté cargado.</li></ul>"}' );
    }
    return r;
}
```

Esta función es llamada solamente por la función que pide al *servlet* todas las reglas que hay en la SmartHome, que se lanza en cuanto se entra en la aplicación.

Y ahora el código de la función que devuelve todos los identificadores y nombres de las reglas:

```
function httpGetBRules(){
    var r = "";
    $.ajax({
        type: "GET",
        url: "xdomain/httpRuleAPI.php",
        async: false,
        cache: false,
        ContentType: "text/html",
        data: "type=rule&action=getrules",
        success: function(data){
            r = parseJSON(data);
        }
    });
    return r;
}
```

Como podemos observar en la propiedad `url` está la dirección de un script php de nuestro dominio "*httpRuleAPI.php*" en vez de la dirección donde se encuentra el *servlet* que responde a la petición. Esto la hacemos así para poder saltarnos una medida de seguridad (*cross domain call*) incluida en la implementación del javascript en los navegadores modernos que provoca que no se pueda llamar mediante javascript a aplicaciones que están en otro dominio distinto al del script en cuestión. Esto se va a explicar más a fondo en un apartado posterior de la memoria.

Todas las funciones que necesiten de llamadas al *servlet* lo harán mediante ajax de *jQuery* a través de este script.

La propiedad `type` indica porque método se va a enviar la información, en nuestro caso es *GET*. La propiedad `async` a falso indica que la petición al *servlet* será síncrona, o sea que se esperará hasta que le llegue la información.

Otras propiedades interesantes de la función son `data` y `success`. En `data` se ponen los parámetros que se quieren enviar al *servlet*, en nuestro caso pedimos todas las reglas. Estos parámetros son los implementados en la API. En `success` aparece una función que será la que se ejecute cuando la llamada ajax haya terminado con éxito. En el parámetro data será devuelto el resultado de la llamada, en este caso el *servlet* devuelve una cadena de texto en formato json con los nombres e identificadores de todas las reglas, además del nombre del archivo donde están definidas, y se pasa a la función *parseJSON* que decodificará la cadena en un objeto de javascript desde el cual es muy fácil acceder a sus datos.

Ejemplo de json que puede devolver el *servlet*:

```
[{"modelname":"Massamagrell"},
{"id":"BR.CUINA.SP01.TECLA1.CURTA","name":"ON CUINA (SP01)
TECLA1 CURTA"},
{"id":"BR.DIST.SP01.TECLA5.CURTA","name":"ON DISTRIBUIDOR (SP01)
TECLA5 CURTA"},
...
{"id":"id3","name":"id3"},
{"id":"uy2","name":"tengo un pollito"},
{"id":"dd1","name":"dd1"}]
```

Función de creación de nuevas reglas. Todas las funciones de creación serán básicamente como esta. Como he comentado la configuración del ajax será en todas igual excepto la propiedad data y success. En este caso la propiedad data contiene el tipo regla, la acción que es new, nueva, y dos parámetros más que son el identificador y el nombre. El *servlet* en este caso devuelve una cadena json en el que indica el identificador de la regla que se quiere crear y si se ha creado satisfactoriamente.

Ejemplo de json que puede devolver el *servlet* en este caso:

```
{"id":"dd3","done":true}
```

O en caso de que falle:

```
{"id":"dd3","done":false}
```

```
function httpNewRule(ruleid, rulename){
    var ret = null;
    $.ajax({
        type: "GET",
        url: "xdomain/httpRuleAPI.php",
        async: false,
        cache: false,
        ContentType: "text/html",
        data: "type=rule&action=new&id="+ruleid+"&name="+rulename,
        success: function(data){
            var js = JSON.parse(data);
            ret = js.done;
        }
    });
    return ret;
}
```

Para el borrado la función es muy parecida a la anterior en la que solamente es la propiedad data la que cambia. En el caso de una regla por ejemplo el data tendrá el type igual a rule, el action igual a delete y el id igual al identificado de la regla a borrar. La respuesta del *servlet* será la misma que en la anterior función, true si se ha realizado con éxito la acción y false en caso contrario.

```
function httpDeleteRule(ruleid){
    var ret = null;
    $.ajax({
        type: "GET",
        url: "xdomain/httpRuleAPI.php",
        async: false,
        cache: false,
        ContentType: "text/html",
        data: "type=rule&action=delete&id="+ruleid,
        success: function(data){
            var js = JSON.parse(data);
            ret = js.done;
        }
    });
    return ret;
}
```

La función de devolver las opciones, como el resultado es el mismo y también el componente gráfico en el que se va a mostrar, se ha creado solo una función con dos parámetros, uno para el identificador del elemento gráfico en el que se va a insertar la información recibida y otro para el tipo de datos que se va a pedir al *servlet*.

En este caso las opciones posibles son: getdevactions, getdevtypes, getruleactions, getsceneactions, getdevstatus.

Y el *servlet* devuelve un json del estilo en el que se utilizará la componente literal como el nombre a mostrar y la componente value con el valor de esta:

```
[{"value":0,"instance":"NONE","literal":""},{"value":1
,"instance":"ENABLE","literal":"Device: Enable"},...]
```

```
function httpGetEventOptionsSelect(htmlelementid, option){
    $.ajax({
        type: "GET",
        url: "xdomain/httpRuleAPI.php",
        async: false,
        ContentType: "text/html",
        data: "type=event&action="+option,
        success: function(data){
            var c = "";
            var d = JSON.parse(data);
            for(var i=0, s = d.length; i < s; i++){
                c += '<option
value="'+d[i].value+'">'+d[i].literal+'</option>';
            }
            $(htmlelementid).append(c);
        }
    });
}
```

7.2.3 Script del php proxy solución para el "*cross domain call*"

En los navegadores modernos se han implementado muchas medidas de seguridad para hacer más segura la navegación web. Una de estas medidas de seguridad es la que nos impedía que nuestra aplicación web accediera directamente a nuestra aplicación *servlet* en el servidor del SmartHome. Este problema es llamado el "*cross domain call*".

Esta medida de seguridad ayuda a que los sitios maliciosos no puedan robar información o suplantar al usuario. Pero en nuestro caso no queremos hacer tales cosas, sino simplemente acceder a otra aplicación que está en un dominio diferente al del servidor web para intercambiar información entre las aplicaciones.

Los programadores tenemos que ingeniárnoslas para poder realizar estas consultas entre dominios de confianza para hacer que nuestras aplicaciones funcionen correctamente.

Hay varias formas de poder realizar estas peticiones. De entre ellas la forma elegida es la de utilizar un script php a modo de proxy entre nuestra aplicación web y el *servlet* en el servidor SmartHome.

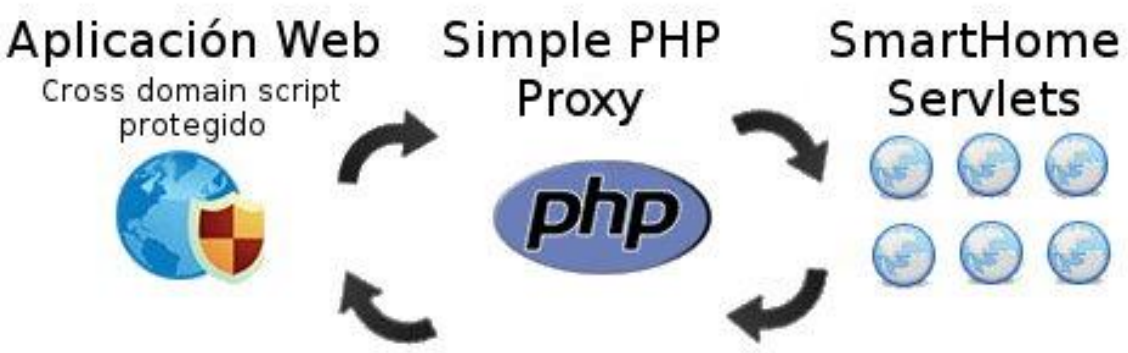

Ilustración 10 Esquema de comunicación entre la aplicación web y servlet.

La aplicación web llamará a este script php, que estará alojado en el servidor web, para pedir la información que requiere al *servlet* en SmartHome y el *servlet* se la devolverá al script php y este a la aplicación web.

Por ello todas las funciones implementadas en la aplicación web harán uso de este script en vez de llamar al *servlet* directamente para un correcto funcionamiento.

7.2.3.1 Funcionamiento del Script

Hay muchas formas de implementar este script proxy en php, en este caso se ha optado por basar todo el script en la utilización de la función php "*file_get_contents*"[5].

Esta función devuelve la información leída de un archivo a una cadena de texto o falso en caso de error. Puede ser un archivo o una dirección url y esta será la funcionalidad que requeriremos ya que la url será la dirección del *servlet* más los parámetros del API implementada.

Utilizando la función *echo* podremos devolver lo que ha leído esta función del *servlet* al javascript. Hay que puntualizar que se hará que el *servlet* siempre devuelva cadenas de texto en formato json.

[5] http://php.net/manual/es/function.file-get-contents.php

7.2.3.2 Código del script

El script lo primero que hace es incluir el archivo "*config.php*" que es donde se encuentran declararas la constantes que se van a utilizar en este script, principalmente la constante `$HTTPRULEAPI`, que es la dirección del servlet.

```
$WEBSERVER4SERVLETS="localhost";
$SERVLETPORT="8081";
$WEBROOT4SERVLETS="";
$SERVLETAPP_ROOT="http://".$WEBSERVER4SERVLETS.":".$SERVLETPORT.$WEBROOT4
SERVLETS;
$HTTPRULEAPI=$SERVLETAPP_ROOT."/httpRuleAPI?";
```

Lo siguiente es extraer con $_REQUEST (su contenido viene dado por las variables $_GET, $_POST y $_COOKIE) los dos parámetros principales de la API que son *type* y *action*.

Una vez que los tenemos filtramos por *type* y dependiendo de su valor entrará en una parte del código o en otra.

En el código se muestra la parte de gestión del tipo regla. Dentro de esta dependiendo de la *action* se realizará una acción u otra y finalmente se hará el eco de la función *file_get_contents* con la dirección correcta dependiendo del tipo y la acción.

```
<?php
include_once "../include/config.php";
$type=$_REQUEST['type'];
$action=$_REQUEST['action'];
// RULES
if($type == "rule"){
    // Return Rules
    if ($action == "" || $action == null || $action == "getrules"){
        echo
file_get_contents($HTTPRULEAPI."type=".$type."&action=getrules");
    }
    // New Rule
    else if ($action == "new"){
        $ruleid = urlencode($_REQUEST['id']);
        $rulename = urlencode($_REQUEST['name']);
        echo
file_get_contents($HTTPRULEAPI."type=".$type."&action=".$action."&id=".$r
uleid."&name=".$rulename);
    }
    // Delete/Select Rule
    else if ($action == "delete" || $action == "select"){
        $ruleid = urlencode($_REQUEST['id']);
        echo
file_get_contents($HTTPRULEAPI."type=".$type."&action=".$action."&id=".$r
uleid);
    }
//GROUPS
} else if ($type == "group"){
....
```

Hay que mencionar que en la parte de las condiciones se va a procesar la cadena completa ya que los valores que llevará asociado tendrán este aspecto {\"tipo\":\"valor\"} y no se pueden procesar como datos válidos de json.

```
else if ($type == "condition"){
    else if($action == "new"){
        $typecond = $_REQUEST['typecondition'];
        $jsonForm = json_encode($_REQUEST);
        $jsonForm = str_replace("\\\"", "\"", $jsonForm);
        $jsonForm = str_replace("\"{", "{", $jsonForm);
        $jsonForm = str_replace("}\"", "}", $jsonForm);
        $jsonForm = str_replace("}\"", "}", $jsonForm);
        $jsonForm = str_replace(" ", "+", $jsonForm);
        echo
file_get_contents($HTTPRULEAPI."type=".$type."&action=".$action."&typecon
dition=".$typecond."&form=".$jsonForm);
    }
}
```

7.2.4 Interfaz gráfica

Como ya se ha comentado en un apartado anterior de la memoria, en un primer momento la interfaz se iba a realizar utilizando *iWebKit* por que ya había desarrollos anteriores con ese framework y quedaba visualmente bien, pero como se comprobó más tarde solo estaba optimizado para su uso con iPhones y por ello se cambió a *jQuery Mobile* que el rango de navegadores en los que se puede ejecutar correctamente es mucho mayor[6]. Y como se puede comprobar en las siguientes ilustraciones el desarrollo visualmente se ha resuelto satisfactoriamente.

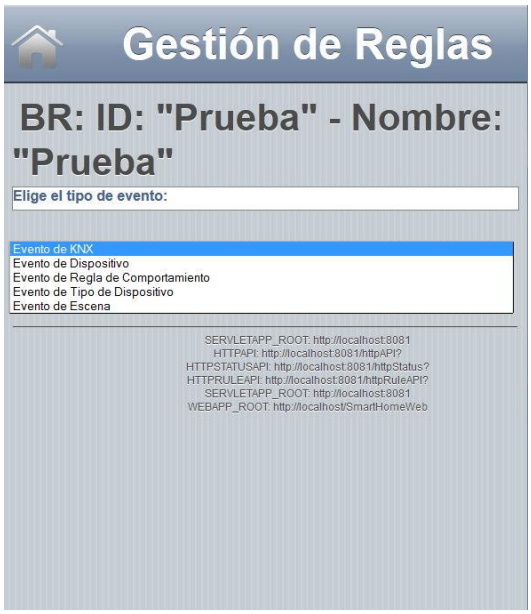

Ilustración 11 Interfaz inicial con iWebKit

Ilustración 12 Interfaz con jQuery Mobile

[6] http://jquerymobile.com/gbs/

La interfaz gráfica se ha implementado utilizando el leguaje de programación web html, javascript y con ayuda de las librerías javascript *jQuery* y *jQuery Mobile*. En el momento de empezar a realizar la implementación de esta parte de la aplicación se empezó a trabajar con la versión de *jQuery Mobile* 1.1.1 que requiere la utilización de la versión 1.7.1 de *jQuery*, pero poco después salió la versión 1.2.0 de *jQuery Mobile* con grandes mejoras y la incorporación de un nuevo widget, el popup.

Tras considerar que el cambio de la librería no afectaba a lo ya desarrollado y que se iba a hacer uso del nuevo componente se cambió a la nueva versión. Respecto a *jQuery* se mantuvo la misma versión porque aunque *jQuery Mobile* sí que soporta la última versión, en el momento del desarrollo la 1.8.2, si se actualizaba en los navegadores de BlackBerry 5 e iOS3 los componentes loader, popup y slide no se ejecutarían bien y como se va ha hacer uso de popup y se busca la máxima compatibilidad con los navegadores se dejó la versión 1.7.1.

Para parte del desarrollo se utilizo una aplicación web para generar rápidamente prototipos visuales ya que con ella es muy fácil añadir o eliminar componentes y comprobar que tal va a quedar cada pantalla.

Cabe destacar que gracias a esta aplicación el desarrollo de la interfaz web, en lo que se refiere a código html fue muy útil, aunque todo el código generado por esta fue modificado y adaptado a la aplicación final.

Se puede acceder a este editor[7] desde la web principal de *jQuery Mobile*.

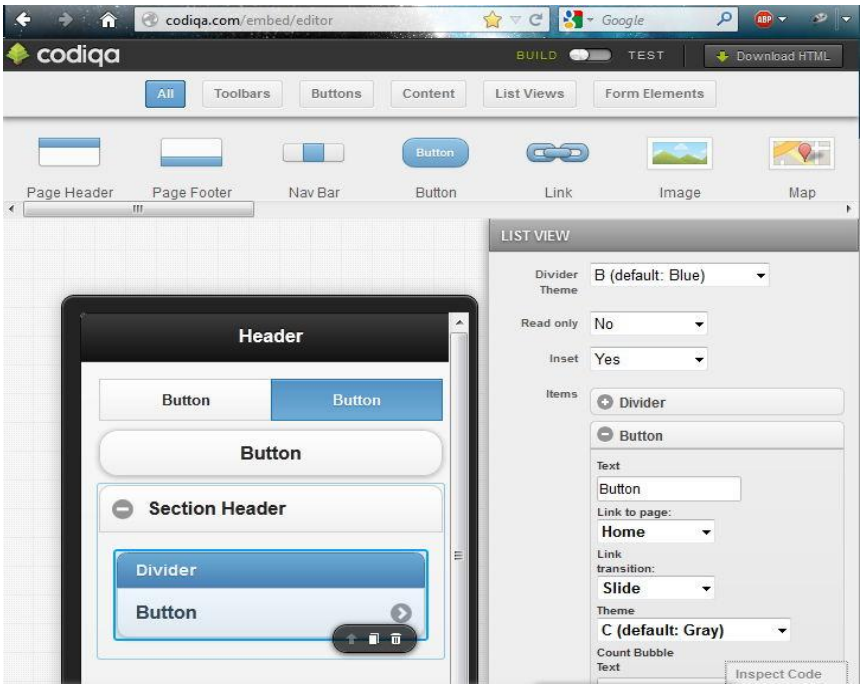

Ilustración 13 Editor web de prototipos

[7] http://codiqa.com/embed/editor

7.2.4.1 Del diseño al código

Como ya se ha comentado a lo largo de la memoria la elección de los componentes visuales es muy importante por el limitado tamaño de las pantallas de los dispositivos móviles, una buena elección hará que la aplicación sea útil y el usuario entienda mejor que es lo que está haciendo en cada momento.

Primero veamos cómo se estructura el código para la creación de las pantallas usando *jQuery Mobile*. Ya se ha visto la cabecera de la página, ahora pasamos al cuerpo "<body>".

```
<div data-role="page" id="pageHome">

    <div data-theme="a" data-role="header">
        <h3>
            Gestión de Reglas
        </h3>
        <a href="infoDialog.php" data-rel="dialog" data-icon="info"
class="ui-btn-right" data-iconpos="notext">Información</a>
    </div><!-- /header -->

    <div data-role="content">
        <p>Contenido de la página va aquí.</p>
    </div><!-- /content -->

    <div data-role="footer">
        <h4>Pie de pagina</h4>
    </div><!-- /footer -->

</div><!-- /page -->
```

Dentro de body hay un contenedor div al que se le asigna un función con la propiedad data, `data-role="page"`, esto hace que todo lo que contiene ese contenedor sea tratado como una página. Dentro de "page" ya se puede utilizar cualquier etiqueta html válida, pero para una mejor distribución del contenido es mejor utilizar las tres partes que se han utilizado en la aplicación y que se recomiendan desde la web *de jQuery Mobile*: "header", "content" y "footer".

El contenedor con el rol de "header" se va a usar para identificar la parte en la que estamos de la aplicación y el "footer" para dar información o mostrar botones de acción como cancelar o atrás.

El contenedor div "content" es la parte central de la página y es donde situaremos los controles, botones, listas, etc de la aplicación.

En esta memoria no se va a explicar todo el código html de la aplicación ya que es muy extenso, pero sí los componentes principales utilizados y como se han implementado y algunas curiosidades, ya que el resto se haría de la misma manera.

7.2.4.2 Pantalla inicial

Esta es la pantalla inicial de la aplicación web, cuando se inicie esta es la pantalla que se mostrará al usuario.

Traslado de componentes desde el diseño a la implementación:

Ilustración 14 Diseño de Gestión de Reglas

Ilustración 16 Gestión de reglas

Ilustración 15 Elección de reglas existentes con filtro

Como observamos en las ilustraciones anteriores, al trasladar el diseño en papel a la interfaz, la lista de reglas se ha convertido en una lista emergente con el componente popup y dada la gran cantidad de reglas se le ha implementado un filtro para reducir el número de reglas que se muestran a la vez. Es entonces al seleccionar una regla de la lista cuando se muestran las opciones de editar o eliminar. Se puede observar que el tamaño de los botones es considerablemente más grande para facilitar el uso al usuario.

El código utilizado para realizar la selección de reglas junto a su código javascript es el siguiente:

```
<div data-role="fieldcontain">
    <label>
        O elige una regla existente:
    </label>
    <a href="#homeSelectRulesFilter" data-rel="popup" data-role="button"
data-icon="arrow-d" data-iconpos="right" data-transition="pop">Elige una
regla...</a>
    <div data-role="popup" id="homeSelectRulesFilter" style="top: 8px;">
        <ul id="homeRuleList" data-role="listview" data-filter="true"
data-filter-placeholder="Filtrar reglas..." data-inset="true" data-
filter-theme="e" style="padding-top: 8px;"></ul>
    </div>
    <input id="homeRuleId" type="hidden" value="">
</div>
```

Código javascript:

```
//Funcionalidad de elección de reglas
$('#homeRuleList').on('click','li a', function(event){
    var id = $(this).attr('data-value');
    var name = $(this).text();
    // Compruebo que hay una regla y la selecciono en el servidor
    if(id != "" && httpSetRule(id)){
        BehabiourRule.ruleid = id;
        $('#homeRuleId').val(id);
        $('#homeRuleName').text(name);
        $('#homeRuleOpts').show();
    }else {
        $('#homeRuleOpts').hide();
    }
    $(this).parents('#homeSelectRulesFilter').popup('close');
})
```

Lo más interesante es que se puede colocar un componente “listwiew” dentro de un “popup”.

El elemento “a”, el botón de “Elige una regla...”, hace referencia al elemento popup, por lo que si el usuario hace clic en él se abrirá este popup que mostrará el contenido de otro componente el “listview” con una lista con todas las reglas. En este componente se ha puesto la propiedad `data-filter="true"` y

con ello se activa la opción de filtrar todas las entradas de la lista. Por ejemplo en una ilustración anterior se ha filtrado por “dd”.

En el código javascript lo que se hace en el evento “*on*” es que en lugar de asignar el código del evento a todos los elementos “*a*” de la lista, con la gran carga de trabajo que eso supone teniendo en cuenta el elevado número de reglas que puede haber en el modelo de la casa, se lo añadimos a un solo elemento, al componente lista, y le indicamos que cuando se haga clic en su elemento hijo “*a*” que está dentro del elemento *“li”* lance la función asociada. Esto hace que el código se ejecute mucho más rápido cuando hay muchos elementos.

```
$('#homeRuleList').on('click','li a', function(event){});
```

Dentro de la función obtenemos los valores de nombre e identificador y con este último se selecciona la regla en el servidor con la función *httpSetRule(id)* y mostramos los botones correspondientes a editar y borrar y cerramos el popup con la lista.

7.2.4.3 Pantalla de elección de grupo

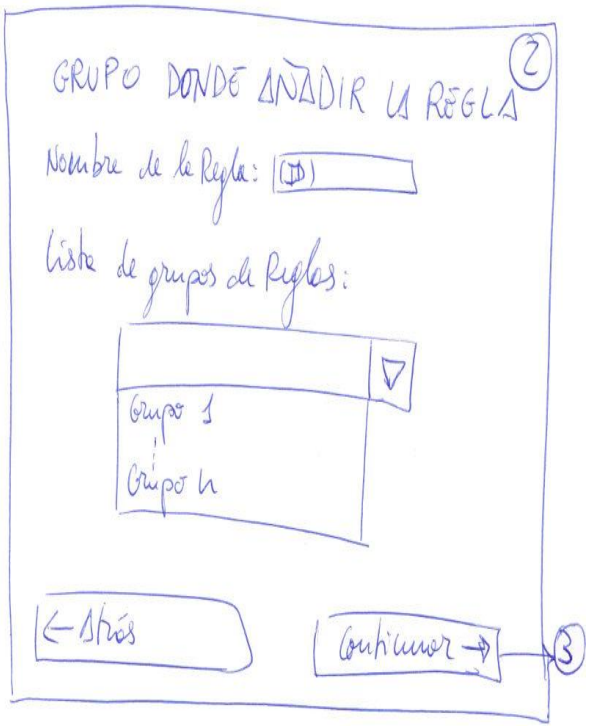

Ilustración 17 Diseño añadir regla al grupo

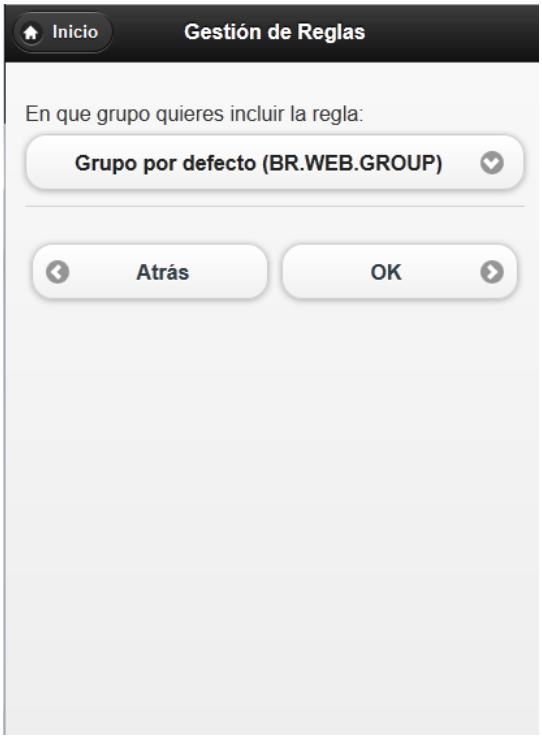

Ilustración 18 Añadir regla al grupo

En esta pantalla se mostrará por defecto el grupo al que se asignarán las reglas introducidas por la aplicación web, pero si el usuario prefiere elegir otro podrá hacerlo.

Aquí se ha implementado la elección de grupo con un elemento "*select*". El javascript ejecuta la busque de todos los grupos que hay en el modelo de la casa con la función *httpGetGroups()* y al dar al botón *OK* selecciona el grupo en el servidor con la función *httpSetGroup(id)*.

7.2.4.4 Pantalla de nueva regla

Esta es la pantalla en la que el usuario introducirá el identificador y el nombre para la nueva regla que está creando.

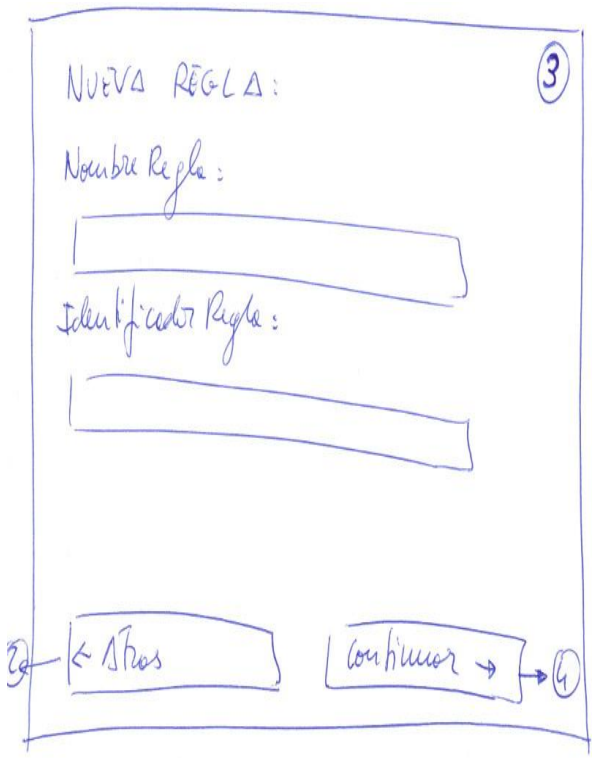

Ilustración 19 Diseño de nueva regla

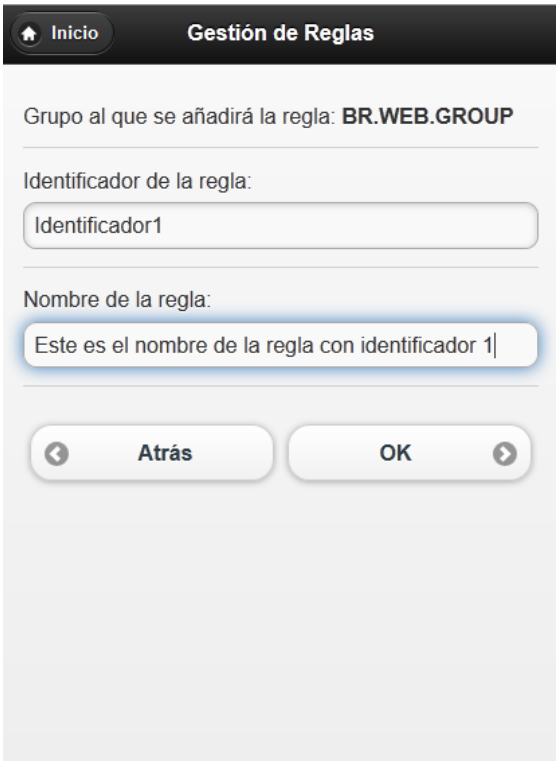

Ilustración 20 Nueva regla

Aquí se ha utilizado el elemento "*input*" para la introducción del texto del identificador y del nombre. También indicamos con un texto encima de los elementos "*input*" en que grupo se va a añadir la regla para mantener al usuario informado.

En el javascript se ejecuta la función *httpNewRule(id, nombre)* al dar al botón *OK* y esta función comprueba si ya existe una regla con el identificador introducido y si la hay no deja crear la nueva regla. También se comprueba que hay texto en el campo de identificador mostrándose un error en caso de no haberlo.

7.2.4.5 Pantalla de Gestión de eventos

La gestión de eventos se ha implementado con dos elementos colapsables, esto es, que se pueden plegar cuando no se necesiten y ocupan poco espació en la pantalla. Dentro de uno estará la lista de botones para añadir cada tipo de evento y en el otro estarán los eventos ya añadidos con su botón para eliminar.

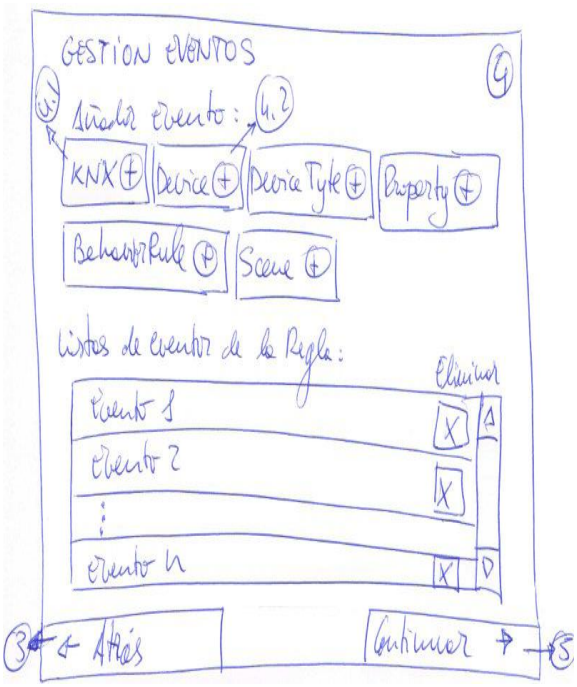

Ilustración 21 Diseño gestión de eventos

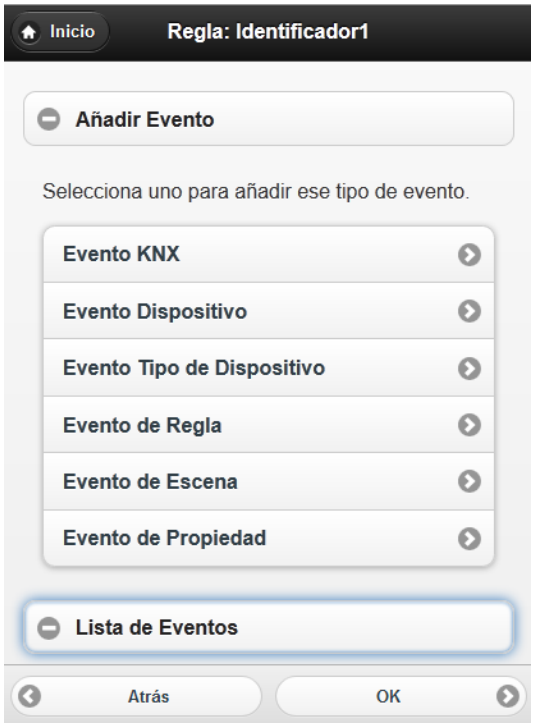

Ilustración 22 Gestión de eventos

El código implementado para la lista *listview* es como en el resto de listas. El elemento colapsable es un *div* con el rol de "*collapsible*", luego se añade una cabecera para el título del elemento y ya tenemos un elemento colapsable con una lista de botones.

```
<div data-role="collapsible">
    <h3>Añadir Evento</h3>
    <p>Selecciona uno para añadir ese tipo de evento.</p>
    <ul data-role="listview" data-inset="true" data-transition="pop">
        <li><a href="eventKNX.php" rel="external">Evento KNX</a></li>
        <li><a href="eventDevice.php" rel="external">Evento
Dispositivo</a></li>
        <li><a href="eventDeviceType.php" rel="external">Evento Tipo de
Dispositivo</a></li>
        <li><a href="eventRule.php" rel="external">Evento de
Regla</a></li>
        <li><a href="eventScene.php" rel="external">Evento de
Escena</a></li>
        <li><a href="eventProperty.php" rel="external">Evento de
Propiedad</a></li>
    </ul>
</div>
```

En los elementos "*a*" se han añadido la opción `rel="external"` para que el framework no utilice ajax para llamar a los enlaces y así poder acceder a la página anterior, gestión de eventos, desde la página llamada.

En el siguiente elemento colapsable se van a mostrar todos los eventos que ya se han añadido. En el código html está vacía porque esta lista de elementos se va a rellenar con la información que llegue del servidor.

```
<div data-role="collapsible">
    <h3>Lista de Eventos</h3>
    <ul id="eventList" data-role="listview" data-inset="true"></ul>
</div>
```

Código javascript:

```
function fillListEvents(listView, listItems){
    $(listView).empty(); // Vaciamos la lista primero.
    var js = JSON.parse(listItems);
    for (var i=0, s=js.length; i < s; i++){
        var ind = js[i].index;
        var type = js[i].type;
        var item = '<li>'+
                   '<h3>'+type+' ('+ind+')</h3>'+
                   '<p>Activo: '+js[i].enabled+'</p>'+
                   '<p>'+JSON.stringify(js[i].event)+'</p>'+
                   '<div data-role="controlgroup" data-type="horizontal"
class="btnRight">'+

'<a href="#" value="'+ind+'" name="delete" data-role="button" data-
icon="delete" data-iconpos="notext">Borrar</a>'+
                   '</div>'+
                   '</li>';
        $(listView).append(item);
    }
    $(listView).listview('refresh').trigger('create');
}
```

La función `fillListEvents` es la función que se va a encargar de rellenar la lista de eventos. A esta función se le pasa como parámetro el identificador de la lista donde queremos añadir los elementos, en nuestro caso `eventList` y los datos que queremos añadir en formato json, los cuales se obtienen llamando a la función *httpGetEventList()*.

Lo primero que hacemos es vaciar la lista, luego parseamos los datos json en un objeto jacascript y los vamos añadiendo uno por uno a la lista como se muestra en `$(listView).append(item)`. Al finalizar tenemos que refrescar la lista para que se muestren los elementos `listview('refresh')` y como hemos incluido un botón, eliminar, para que este se visualice como botón y no como enlace hay que añadir `trigger('create')` a la lista para forzar su creación.

En la siguiente ilustración se muestra el resultado de añadir eventos a la lista.

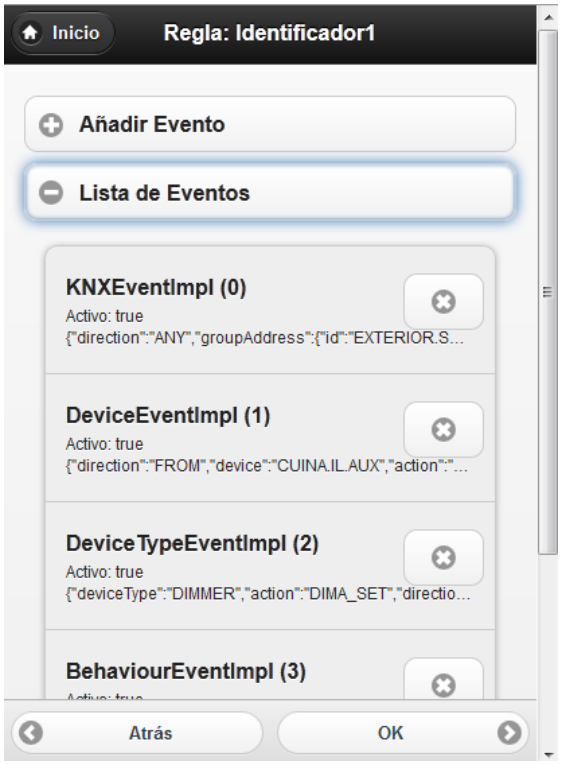

Ilustración 23 Gestión eventos - Lista de eventos

7.2.4.6 Pantalla de Evento KNX

En esta pantalla se mostrará cómo se realiza una inserción de un evento de tipo KNX a una regla.

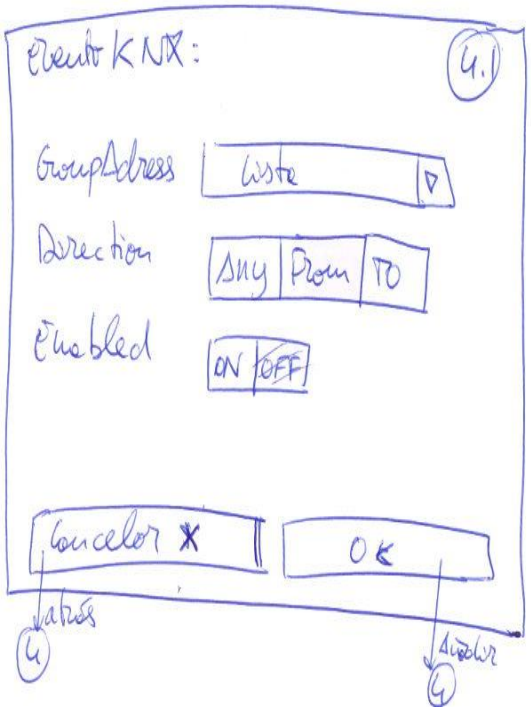

Ilustración 24 Diseño Evento KNX

Ilustración 25 Evento KNX

En esta pantalla aparecen dos nuevos componentes que vamos a detallar a continuación.

El primero es el llamado interruptor, que tiene solo dos posibles estados que se pueden configura. Como por ejemplo para usar las opciones Si / No o Verdadero / Falso. El código para este componente es el siguiente:

```
<select name="enabled" id="eventenabled" data-role="slider">
    <option value="false">
        No
    </option>
    <option value="true" selected="selected">
        Si
    </option>
</select>
```

Se puede observar que el código es un *select* que tiene como rol `slider`, con dos opciones con su valor y su texto, y además en este caso por defecto estará elegida la opción "Si".

El otro componente nuevo que aparece es el grupo de controles. Estos son varios *inputs* del tipo *radio* agrupados en un contenedor con rol `controlgroup` y añadiendo el `data-type="horizontal"` hace que se muestren como botones en los que solo se puede elegir uno.

```
<fieldset data-role="controlgroup" data-type="horizontal">
    <legend>
        Elige la dirección:
    </legend>
    <input id="any" name="direction" value="any" type="radio"
checked="checked">
    <label for="any">
        ANY
    </label>
    <input id="from" name="direction" value="from" type="radio">
    <label for="from">
        FROM
    </label>
    <input id="to" name="direction" value="to" type="radio">
    <label for="to">
        TO
    </label>
</fieldset>
```

El código javascript para añadir el nuevo evento es el que se muestra a continuación. En el evento clic del botón de OK se comprobará que los valores que se van a añadir no están vacios y después se creará una cadena de texto con el tipo, la acción y el tipo de evento correspondiente y a esta se le añadirán los valores del formulario `$('form').serialize()` y esta cadena será la que se envíe al servidor con la función `httpNewEvent(str)`.

```
$("#btnOk_eKNX").on('click', function ( e ){
    if($('#groupaddress').val() != ""){
        var str = "type=event&action=new&typevent=knx&";
        str += $('form').serialize();
        var ret = httpNewEvent(str);
    }
    else {
        $('#popupEventKNX').popup('open');
        return false;
    }
});
```

Cabe destacar la función `serialize()` de jQuery hace que los componentes del formulario se serialicen en una cadena de texto con forma de url estándar, cogiendo el nombre del componente y su valor y los concatena. Por ejemplo este formulario quedaría así:

```
enabled=true&address=EXTERIOR.SE.ESTMETEO.PLUTJA&direction=any
```

7.2.4.7 Pantalla de Evento de Dispositivo

En esta pantalla no nos encontramos ningún elemento visual nuevo y el código javascript es parecido a la pantalla anterior, para más detalle recurrir al código fuente.

Ilustración 26 Evento de Dispositivo

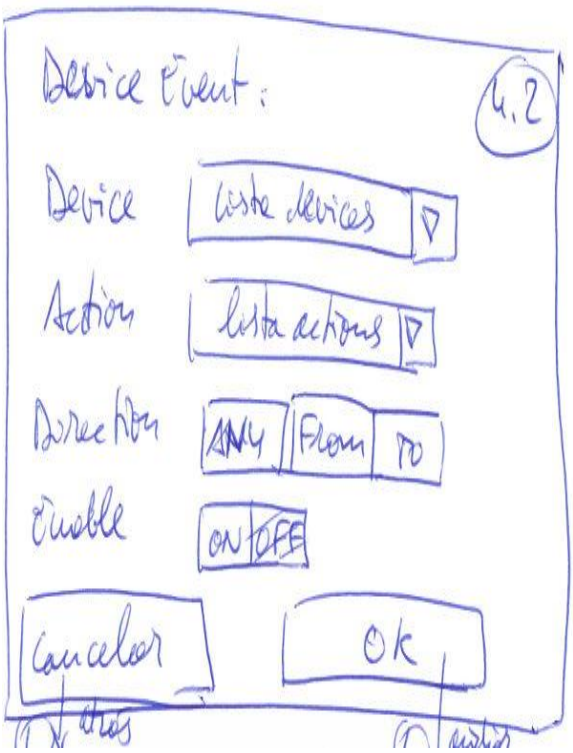

Ilustración 27 Diseño Evento de Dispositivo

7.2.4.8 Pantalla de Evento de Propiedad

En esta pantalla también aparece un componente de tipo `controlgroup` pero esta vez con `data-type="vertical"` para que los botones estén uno encima del otro y se verá como se muestra en la ilustración.

Cada vez que se selecciona una entrada aparecerá al final de la página un *input* para poder introducir el valor del tipo seleccionado.

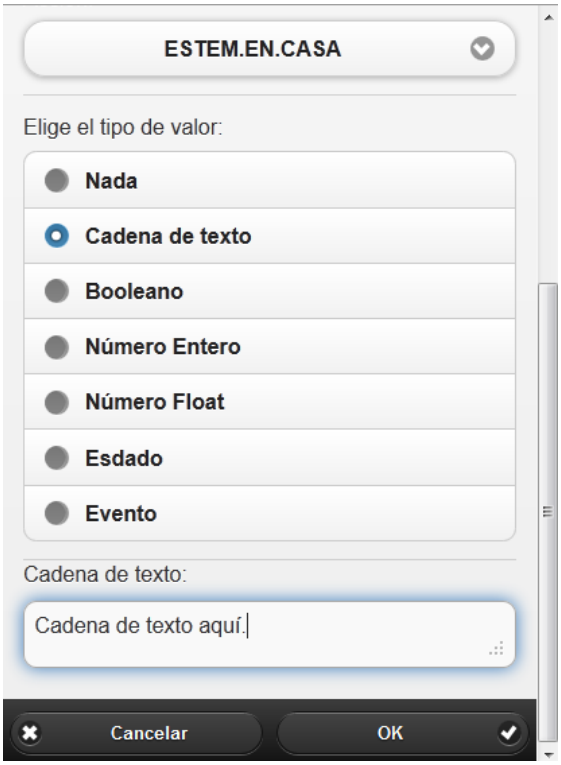

Ilustración 29 Evento Propiedad - Texto

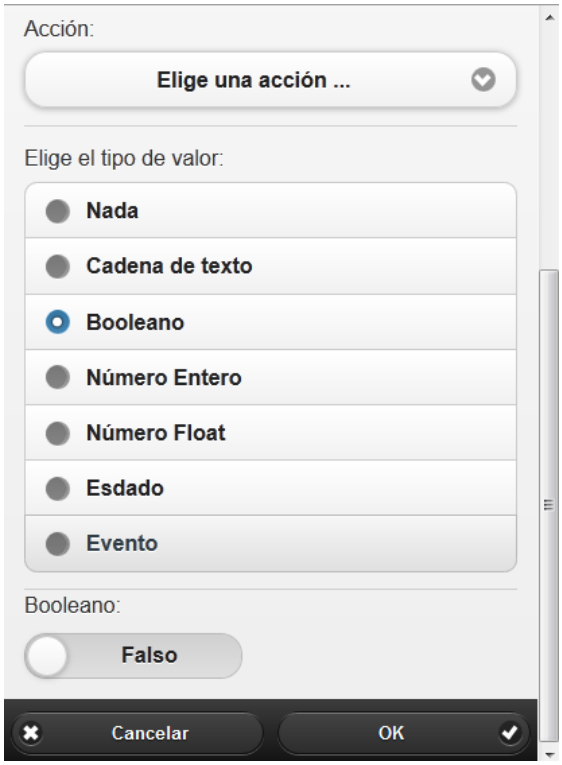

Ilustración 28 Evento Propiedad - Booleano

Por ejemplo si se selecciona “Nada” no aparecerá nada, si se selecciona “Cadena de texto” aparecerá un *textarea* para poder introducir texto, si se selecciona “Boleano” aparecerá un interruptor con los valores de verdadero y falso, con “Número entero” y “Número Float” saldrá un *input* de tipo texto para poder introducir los número y con estado se mostrará un elemento *select* con las opciones correspondientes.

Al hacer clic en OK se comprobarán que los valores introducidos corresponden al tipo seleccionado. Por ejemplo si se introduce texto en los campos asignados para números saldrá un mensaje de error.

7.2.4.9 Pantalla de Evento de Tipo de Dispositivo

La implementación es muy parecida al resto de eventos.

Ilustración 30 Evento de Tipo de Dispositivo

7.2.4.10 Pantalla de Evento Regla

Ídem que el anterior. La elección de la regla se hace igual que en la pantalla principal.

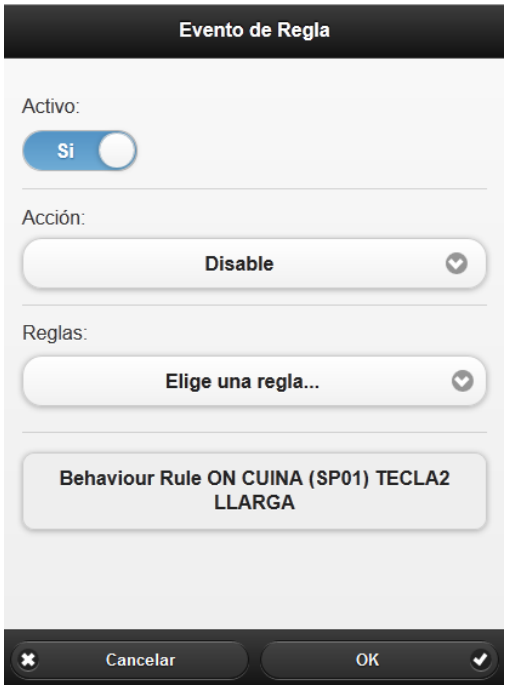

Ilustración 31 Evento de Regla

7.2.4.11 Pantalla de Evento de Escena

Ídem a los demás eventos.

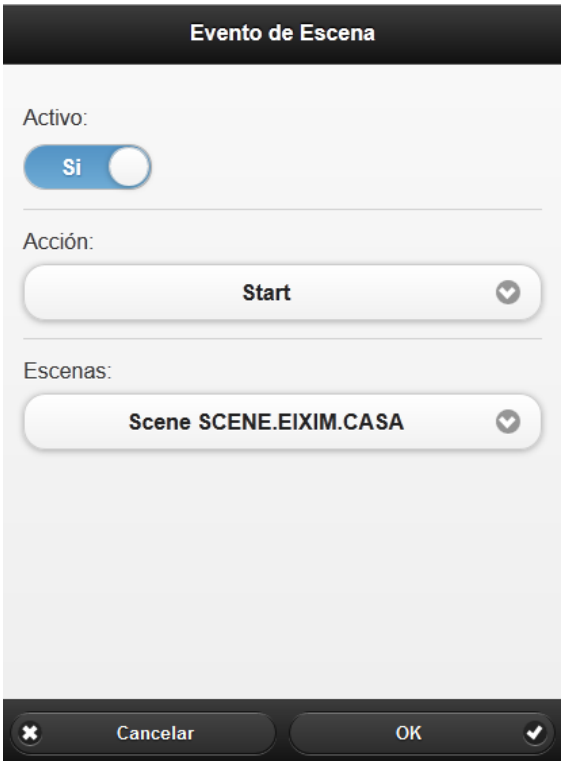

Ilustración 32 Evento de Escena

7.2.4.12 Pantalla de Gestión de Listas de Condiciones - Acciones

En esta pantalla se gestionarán las listas de condiciones – acciones de una regla. Aparecerán las listas de Condiciones - Acciones de la regla si esta tuviera alguna.

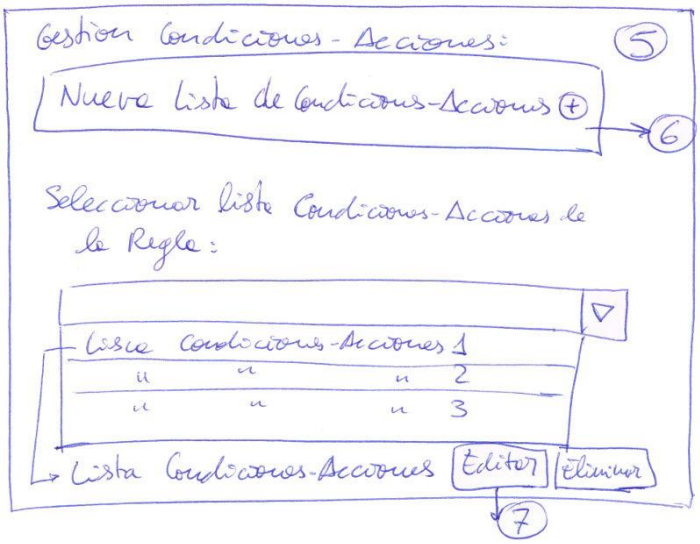

Ilustración 33 Diseño Gestión Condiciones - Acciones

La implementación es similar a la inserción de reglas. Al cargar la página esta pide todas las listas de condiciones - acciones de la regla actual al *servlet* y las muestra en un elemento *select*. Si se selecciona alguna lista del desplegable aparecerán los botones de "Editar" y "Eliminar" como se muestra en las siguientes ilustraciones.

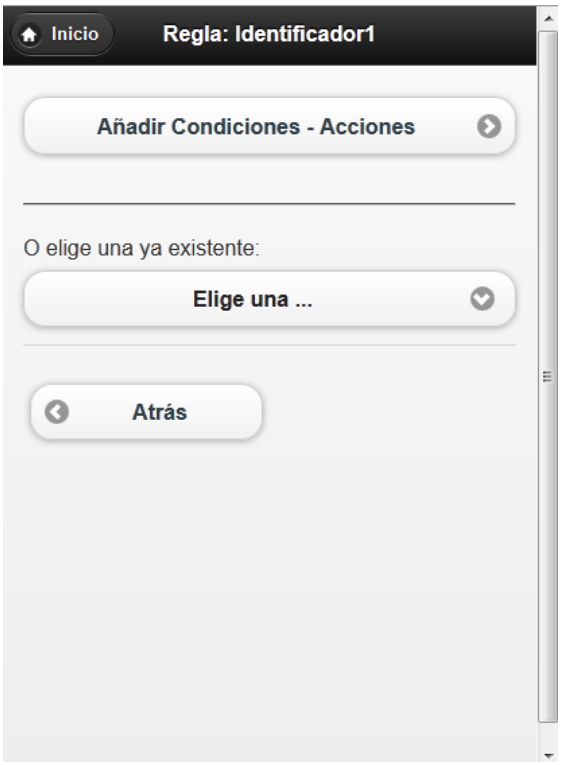

Ilustración 34 Añadir Condición - Acción

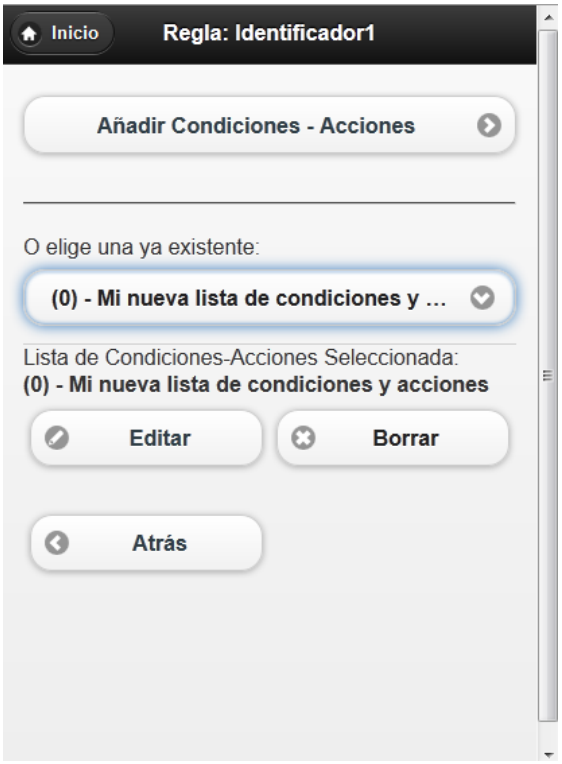

Ilustración 35 Editar Condición - Acción

```
function refreshCA(idselect){
    var op = $(idselect+' option').detach();//Quito los elemento del
select
    $(idselect+' option').remove();//Borro todas las opciones
    $(idselect).append(op[0]);//Añado la primera opción
    httpGetCAs(idselect); // Pido todas las listas de CAs
    $(idselect).selectmenu('refresh');
}
```

Esta función es la que se llama al mostrarse la página. La pongo porque es un poco diferente a como se hace en otros componentes *select*.

Le pasamos como parámetro el identificador del *select* en el que queremos añadir la listas de condiciones-acciones. En la primera línea de la función sacamos todos los elementos *option* del componente a una variable, luego se borran todas las *option* para dejarlo vacio. Añado la primera *option* que es la que pone "Elige una..." y a continuación pido todas las listas de condiciones - acciones al *servlet* y las añado en la función "*httpGetCAs()*" y para finalizar actualizo el componente visual llamando a la función `selectmenu('refresh')`.

7.2.4.13 Pantalla de Añadir Lista de Condiciones - Acciones

Al hacer clic en el botón "Añadir Condiciones – Acciones" de la pantalla anterior nos llevará a esta pantalla donde podremos añadir una descripción a la nueva lista de condiciones – acciones y además se podrá decir si está activa o no.

Al dar a OK se llamará a una función que hará una petición al servidor para añadir la nueva lista a la regla actual.

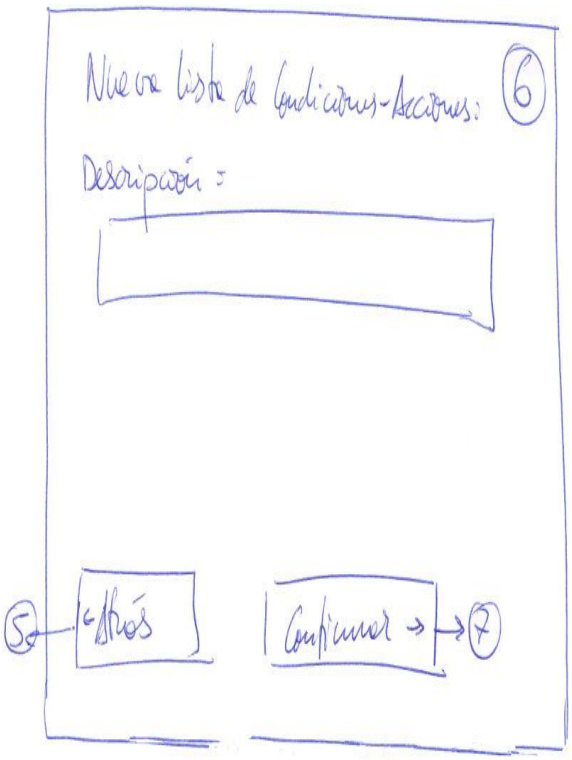

Ilustración 36 Diseño Nueva lista Condición-Acción

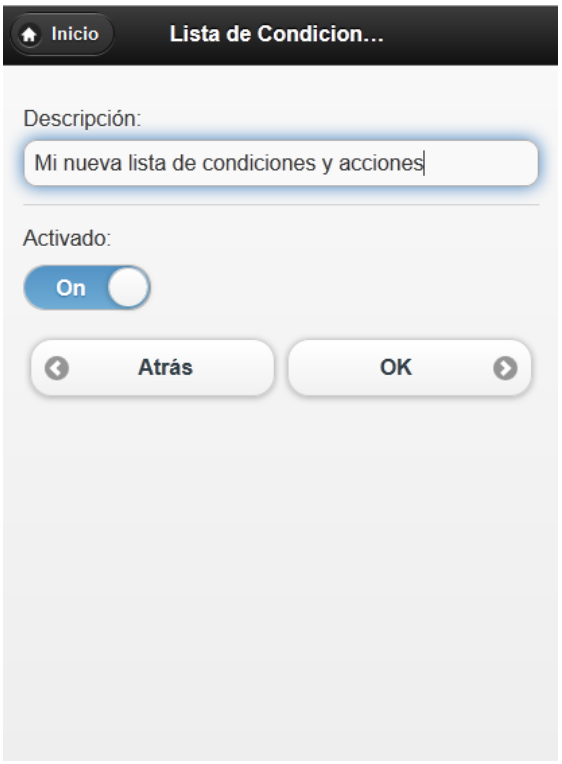

Ilustración 37 Nueva lista Condición-Acción

7.2.4.14 Pantalla de Gestión de Condiciones

Al mostrarse esta pantalla se pedirá al servidor que obtenga todas las condiciones de la lista de condiciones – acciones actual. La implementación de esta pantalla es similar a la de Gestión de Eventos.

Hay una lista desplegable con todos los botones de añadir cada tipo de condición y una lista desplegable con todas las condiciones que se vayan añadiendo o que ya tuviera la lista.

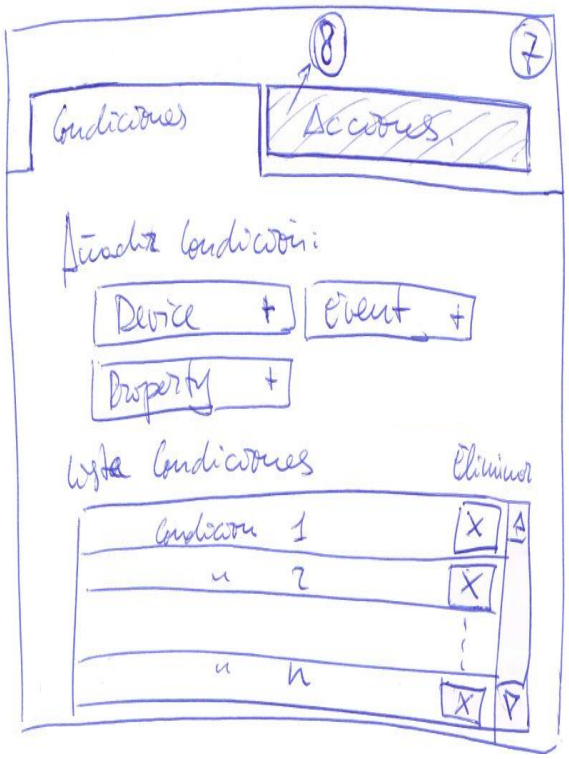

Ilustración 38 Diseño Condiciones

Ilustración 39 Condiciones

7.2.4.15 *Pantalla de Condición de Dispositivo*

En esta pantalla se configurarán las opciones correspondientes a una condición de dispositivo. Como en todas las pantalla los elementos de lista se rellenan con la información que llega del servidor.

Podemos configurar si está activo o no, seleccionar un dispositivo, el operador para dispositivos y si va a ser lo contrario del operador o no.

Como en condiciones se pueden añadir varios valores el botón de "Añadir Valor" mostrará un diálogo con la pantalla de añadir valor, con las opciones correspondientes. Como esta pantalla se va a reutilizar por el resto de condiciones mostraré el código de cómo se ha implementado.

```
<div data-role="fieldcontain">
    <label>Lista de Valores:</label>
    <a href="conditionDialogAddValue.php?lista=condD_listaDVs"
id="btnAdd_CondD_DV" data-role="button" data-icon="plus" data-
transition="pop">
        Añadir Valor
    </a>
    <ul id="condD_listaDVs" data-role="listview" data-inset="true"></ul>
</div>
```

En el botón que llama a este diálogo en el enlace se le pasará como parámetro *lista* la lista en donde se quiera que se muestren los valore que se añadan, `href="conditionDialogAddValue.php?lista=condD_listaDVs"`.

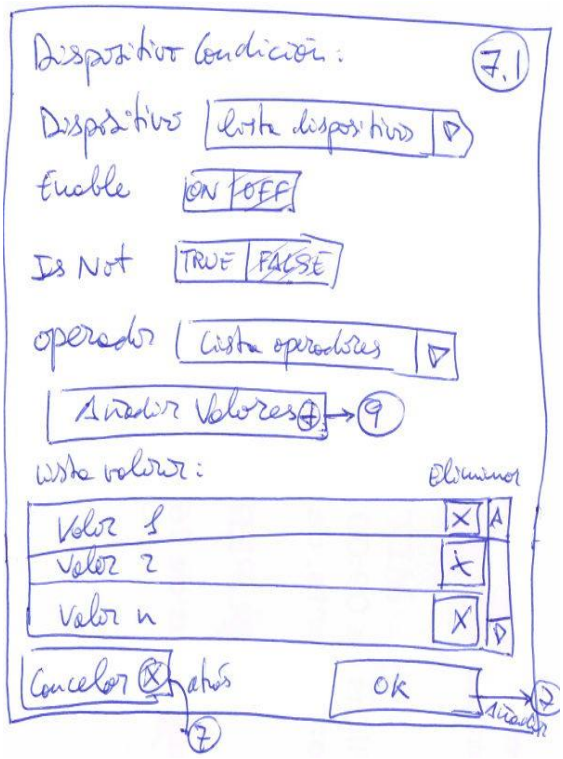

Ilustración 40 Diseño Condición de Dispositivo

Ilustración 41 Condición de Dispositivo

7.2.4.16 Pantalla de Condición de Evento

Esta pantalla se implementará como la anterior, pero para los eventos.

Ilustración 42 Condición de Evento

7.2.4.17 Pantalla de Condición de Propiedad

Esta pantalla se implementará como la anterior, pero para las propiedades.

Ilustración 43 Condición de Propiedad

7.2.4.18 Pantalla de Añadir Valor

Esta pantalla es la que será llamada para añadir un valor a la lista de valores de quien la llame. A continuación como se ha resuelto.

Cuando es llamada se le pasa un parámetro *lista* en el que se indica en que lista se quiere añadir el valor que se va a seleccionar. Este valor se lo asignaremos a un *input* de tipo *hidden*, o sea que estará oculto, mediante un script en php.

```
<input id="idlista" value="<?php echo $_REQUEST['lista'] ?>"
type="hidden">
```

Luego en el código javascript del botón OK se comprobará que el tipo de dato elegido es correcto conforme a lo introducido y finalmente se añadirá ese valor a la lista. Cada elemento de la lista contendrá un *input* oculto con el valor introducido en formato json que servirá para cuando se añada el elemento que ha llamado a esta pantalla.

```
$("#btnAdd_caDV").on('click', function ( e ){
    if ($('#cond_dvIntAux').is(':visible') &&
!IsInteger($('#cond_dvIntAuxVal').val())){
        $('#cond_epErrorMsg').text('El número tiene que ser un entero.');
        $('#cond_popupError').popup('open');
        return false;
    }
    else if ($('#cond_dvFloatAux').is(':visible') &&
!IsFloat($('#cond_dvFloatAuxVal').val())){
        $('#cond_epErrorMsg').text('El número tiene que ser en coma
flotante.');
        $('#cond_popupError').popup('open');
        return false;
    }
    else if($('#cond_dvStatusAuxVal').is(':visible') &&
$('#cond_dvStatusAuxVal').val() == ""){
        $('#cond_epErrorMsg').text('Elige un estado.');
        $('#cond_popupError').popup('open');
        return false;
    }
    else if($('#cond_dvStringAux').is(':visible') &&
$('#cond_dvStringAuxVal').val() == ""){
        $('#cond_epErrorMsg').text('No se puede guardar una cadena
vacia.');
        $('#cond_popupError').popup('open');
        return false;
    }
    else if(pageCond_dv == ""){
        $('#cond_epErrorMsg').text('No hay nada seleccionado.');
        $('#cond_popupError').popup('open');
        return false;
    }
    // Añadir el valor a la lista
    else{
        var type = $('#cond_Datavalue fieldset :checked').val();
        var value = $(pageCond_dv).val();
        var js = '{&quot;'+type+'&quot;:&quot;'+value+'&quot;}';
        var item = '<li><h4>'+type+': '+value+'</h4>'+
            '<input name="value[]" type="hidden" value="'+js+'">'+
            '<div data-role="controlgroup" data-type="horizontal"
class="btnRight">'+
            '<a href="#" name="delete" data-role="button" data-
icon="delete" data-iconpos="notext">Borrar</a>'+
            '</div></li>';
        $(listaid).append(item);
        $(listaid).listview('refresh').trigger('create');
    }
});
```

En el php proxy, comentado en un apartado anterior, que envía finalmente los datos al servidor se tendrán que tratar estas cadenas json porque al utilizar las comillas el formulario las convierte en comillas literales, y el json queda así {\"tipo\":\"valor\"} y esto aunque se puede enviar no se puede procesar como cadena de texto json.

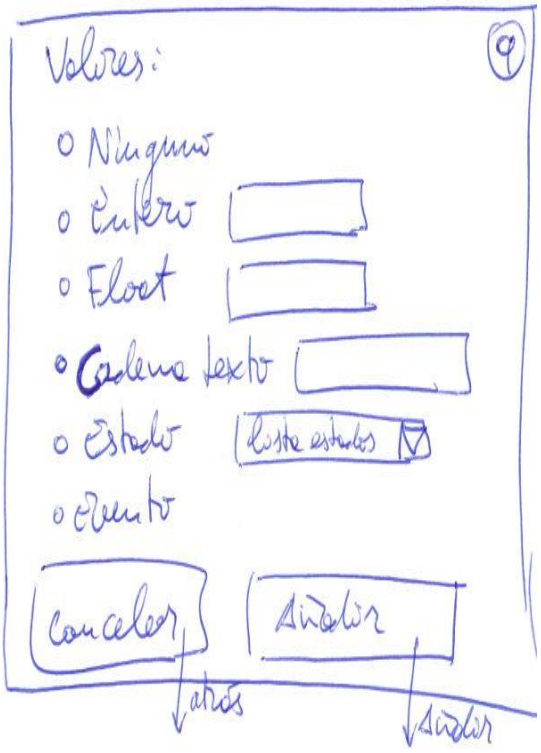

Ilustración 44 Diseño Insertar Valor

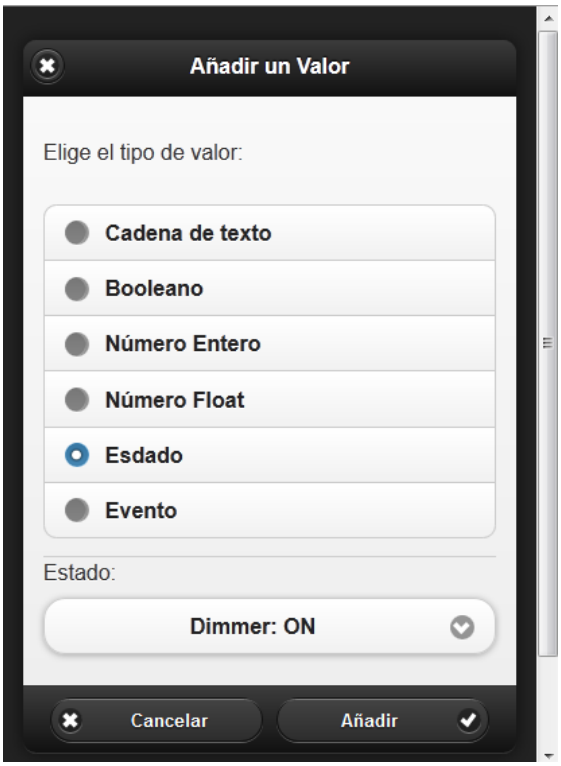

Ilustración 45 Insertar Valor

Para que jQuery Mobile abra esta página como un diálogo el contenedor que engloba a toda la página tiene que tener el rol de "*dialog*":

```
<div data-role="dialog" id="conditionDialogAddValue"> </div>
```

Otra forma que hay para que una página tenga el rol de dialogo es que el enlace que lo llama tenga la propiedad `data-rel="dialog"`.

El código aquí empleado es el mismo que está en Evento de Propiedad. El código que hace que se muestre cada elemento al seleccionar su respectivo botón es el siguiente:

```
$('#cond_Datavalue fieldset').on('change', 'input', function() {
    $('input').each(function(){
        $('#'+this.id+'Aux').toggle(this.checked);
        if($(this).is(':checked')){
            pageCond_dv = '#'+this.id+'AuxVal';
        }
    });
});
```

Aquí se añade el código al contenedor en vez de a cada botón y en el evento *change*. Se comprueba cada botón y se muestra con la función *toggle* el componente que esté activado y se asigna a una variable el valor del identificador activo.

Para realizarlo de esta forma se ha seguido una estrategia de nombrado de identificadores en el que cada componente que se muestra se llama igual que el botón que lo activa más el texto “Aux” y cada valor correspondiente se llama igual más el texto “AuxVal”.

7.2.4.19 Pantalla de gestión de Acciones

En esta pantalla se gestionaran las acciones. La implementación se realizaría igual que las condiciones.

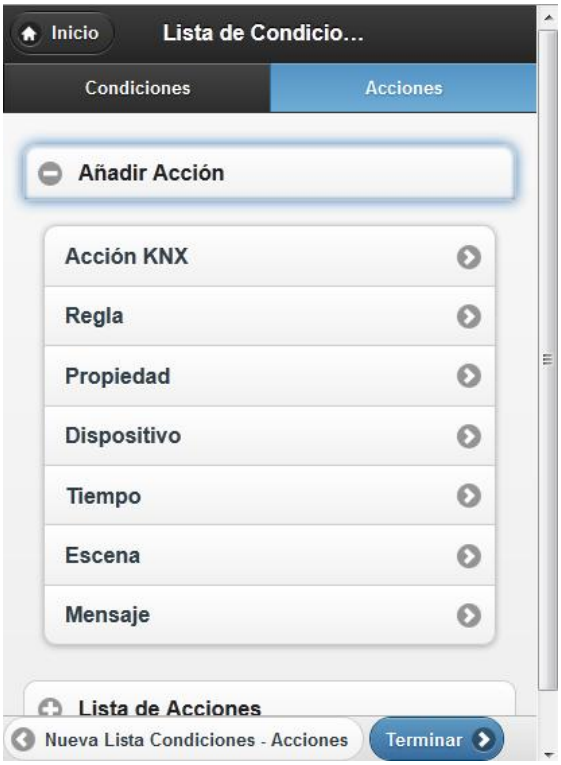

Ilustración 47 Gestión de Acciones

Ilustración 46 Diseño de Gestión de Acciones

7.2.4.20 *Pantalla de información*

En esta pantalla solo se muestra la información referente a la aplicación.

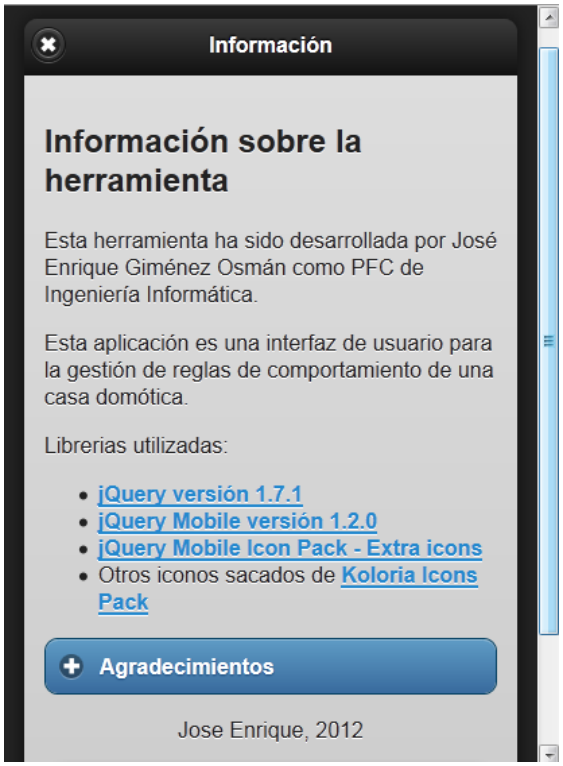

Ilustración 48 Pantalla Acerca de la aplicación

7.3 Aplicación *Servlet*

En este aparatado se va a profundizar en el código implementado para realizar la otra parte del proyecto, la aplicación *servlet*.

El proyecto se ha nombrado con el mismo tipo de nomenclatura que el resto de proyectos que forman del SmartHome: "SmartHomes.HTTPRuleAPIServlet". También se ha estructurado de forma parecida a los proyectos ya existentes para que el impacto de la integración fuera el mínimo posible. Se ha cogido como ejemplos los proyectos "SmartHomes.HTTPAPIServlet" y "SmartHomes.HTTPStatusAPIServlet".

7.3.1 Configuración del proyecto

Lo primero que se hará será incluir la librería json a utilizar en el proyecto. La librería elegida ha sido Json-Smart. Para incluirla en el proyecto lo primero que hay que hacer es bajarse el archivo "json-smart-1.1.1.jar" de su web http://code.google.com/p/json-smart/. Luego se creará un directorio llamado "lib" dentro del directorio del proyecto. Una vez se tenga se editará el archivo "MANIFEST.MF" y en la sección correspondiente a "Runtime" en el apartado Classpath se añadirá la librería.

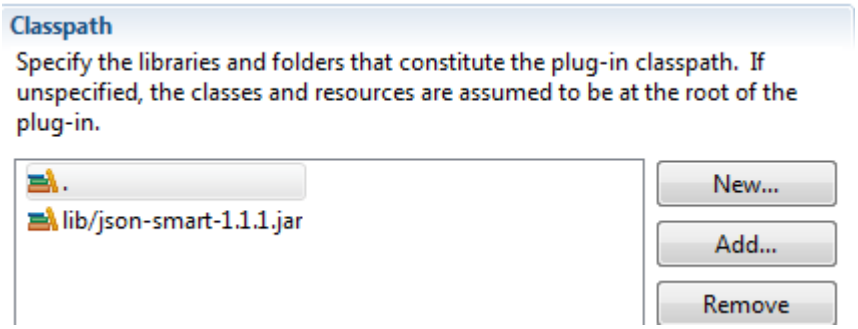

Ilustración 49 Inclusión de librería en el classpath del *Servlet*

Con esto se consigue que al ejecutarse el *servlet* también incluya la librería de json, si no daría error al ejecutar el *servlet*.

También hay que añadir la librería para que sea utilizada en tiempo de compilación. Esto se hace en la sección "Build" del archivo que estamos modificando seleccionando el archivo de la librería.

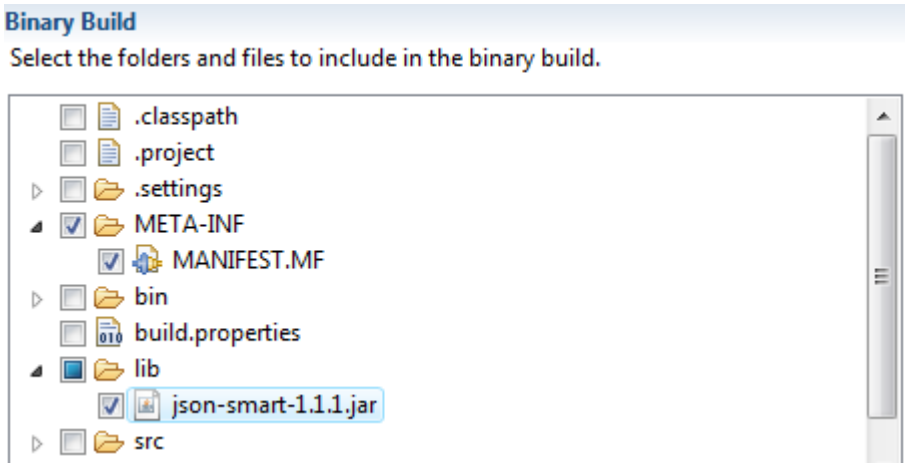

Ilustración 50 Sección Build del archivo MANIFEST.MF

A lo largo de la implementación se han hecho uso de clases pertenecientes varios proyectos y para poder usarlos y que el *servlet* no de error se han tenido que añadir a la sección de "Dependencies" del archivo MANIFEST.MF, estos proyectos son los enmarcados en amarillo en la ilustración siguiente, el resto de dependencias son del framework.

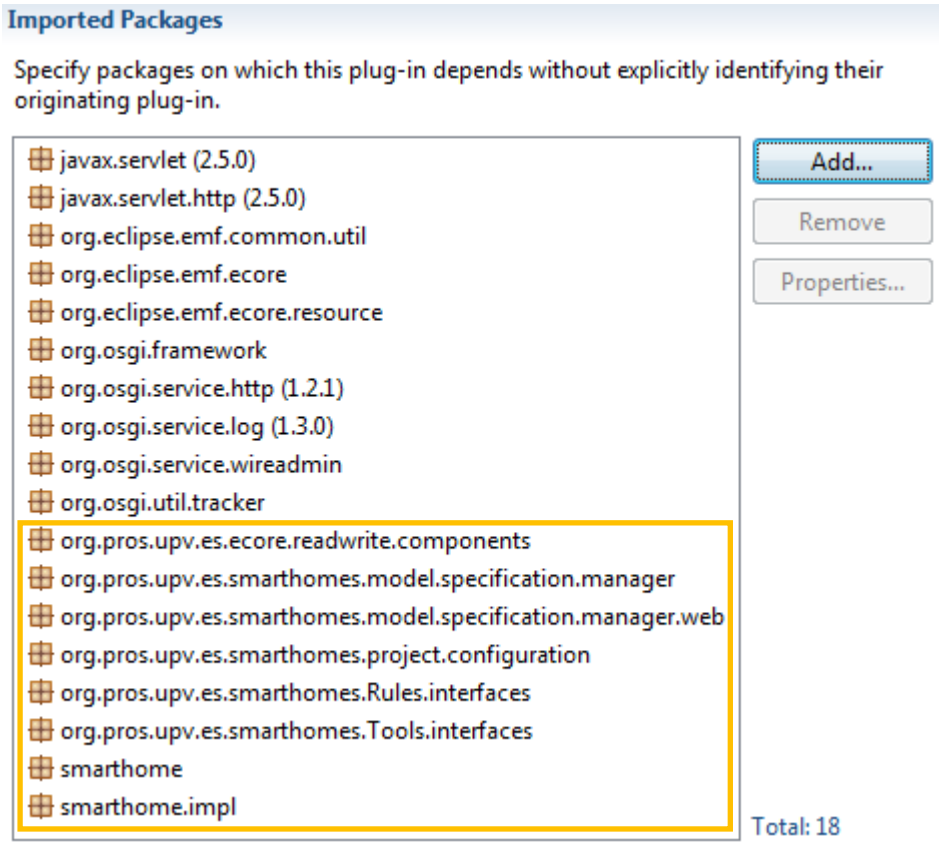

Ilustración 51 Dependencias del proyecto

7.3.2 Estructura de clases del proyecto

La aplicación *servlet* está dividida en dos tipos de clases. Una es la parte de la aplicación que hereda de *servlet*, que es la encargada de obtener los parámetros enviados por la aplicación web, mediante el API creado para este proyecto, y responder a estas peticiones y la otra se encarga de gestionar el archivo de configuración de la casa domótica, la SmartHome, ya que en ella es donde se realizarán todos los cambios que el usuario quiera hacer.

En las ilustraciones siguientes se han separado las clases en dos partes, como se describe en el párrafo anterior, aunque todas las clases están conectadas.

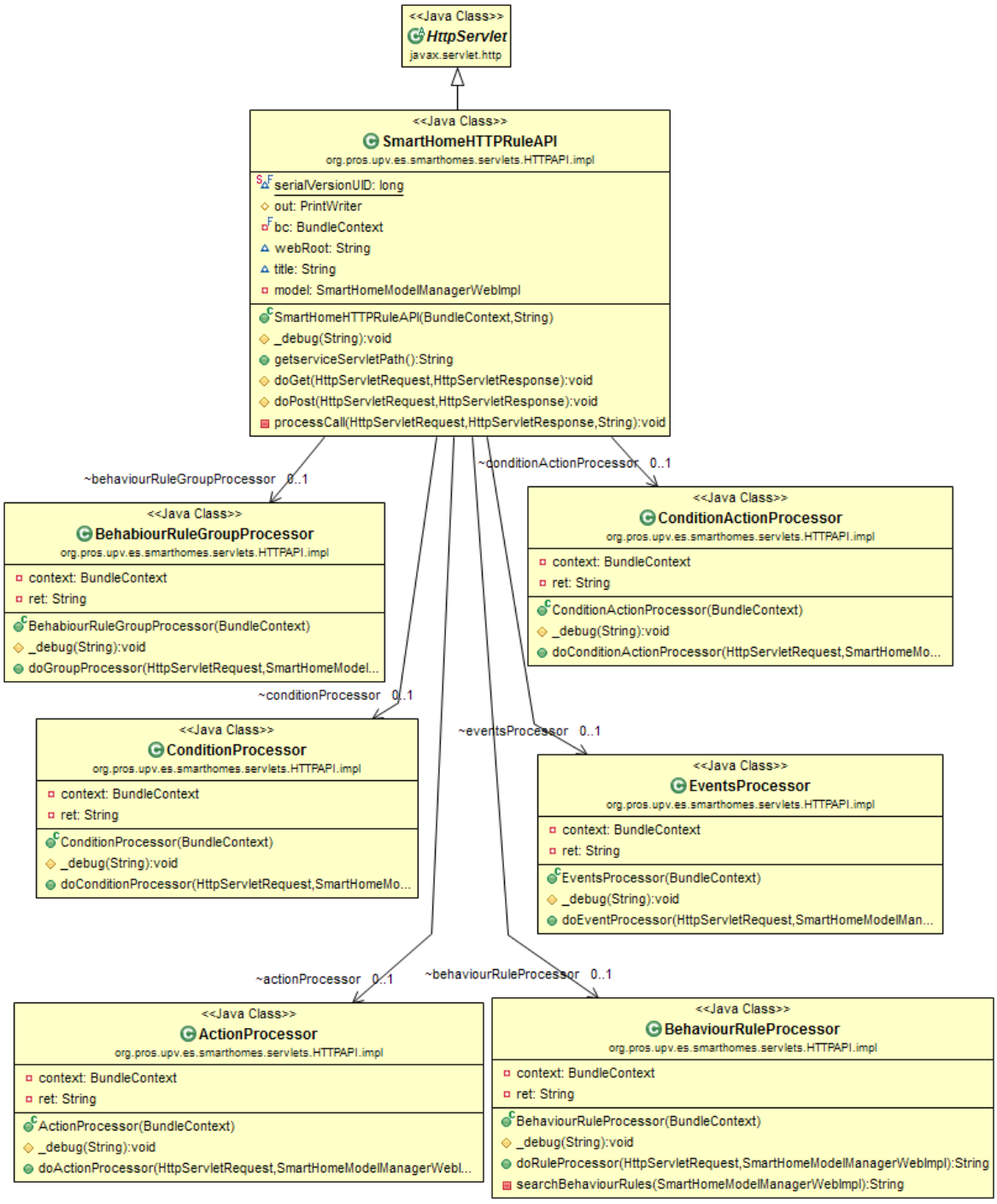

Ilustración 52 Clases de la Aplicación *Servlet* que recogen la información (Parte 1)

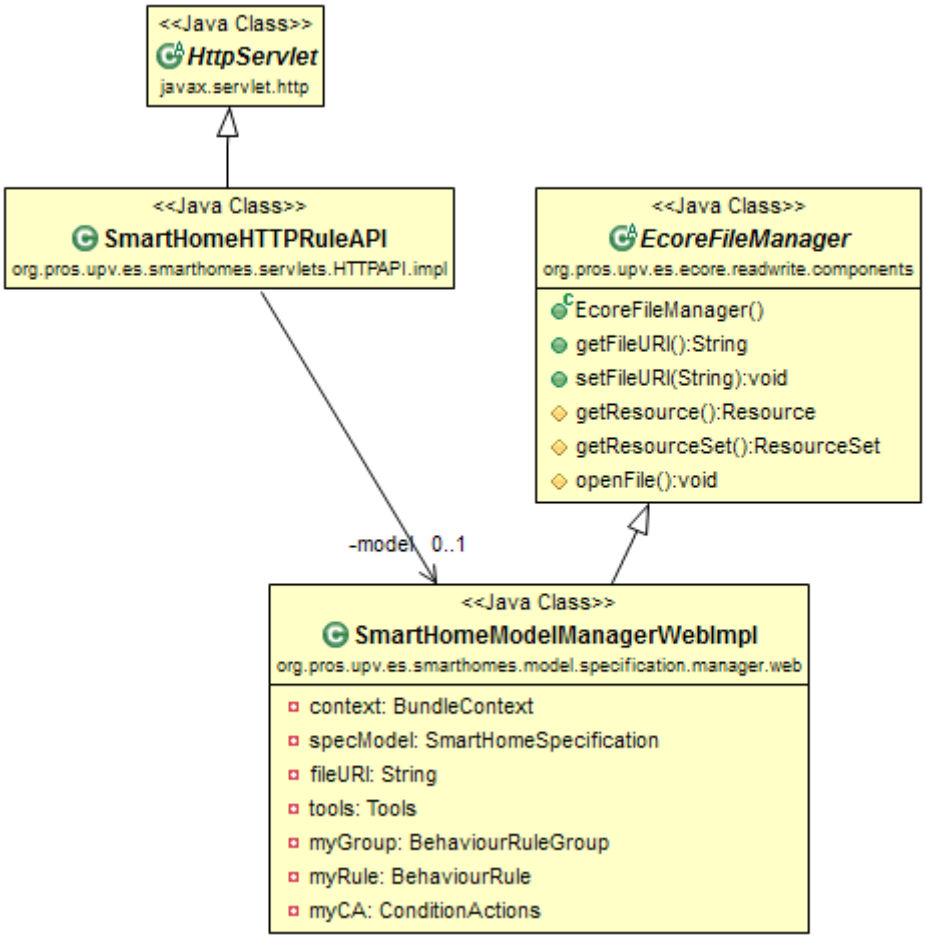

Ilustración 53 Clases de la Aplicación *Servlet* que gestionan la SmartHome (Parte 2)

A continuación una ilustración con la relación de archivos y organización de los mismos en la aplicación *servlet*.

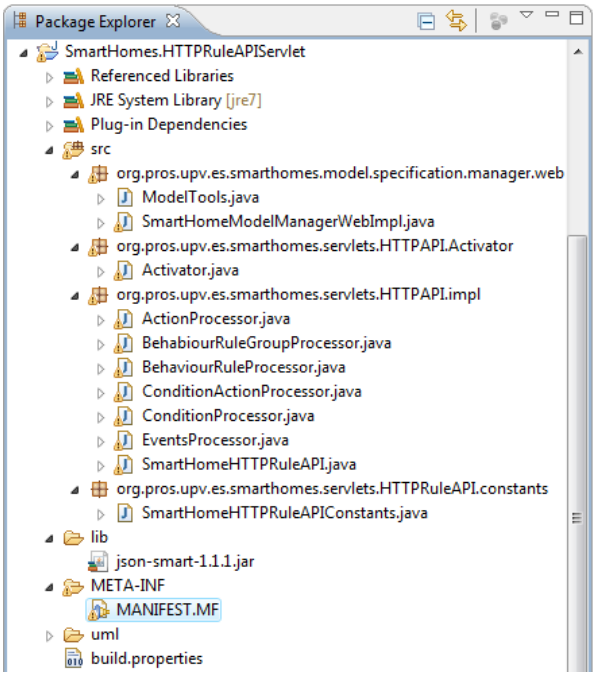

Ilustración 54 Archivos que componen la aplicación *servlet*

7.3.2.1 Clase SmartHomeHTTPRuleAPI

Esta clase extiende de *httpServlet*. En el constructor de esta clase se crearán el resto de objetos que necesita la aplicación, además se llamará a la función *getModel(null)* para asignar el fichero correspondiente del modelo de la SmartHome y se asignará el grupo de reglas por defecto con *getMyGroup()*.

```
public SmartHomeHTTPRuleAPI(BundleContext context, String webRoot) {
    this.bc = context;
    this.webRoot = webRoot;

    //model
    model = new SmartHomeModelManagerWebImpl(context);
    model.getModel(null);
    model.getMyGroup();

    // Processors
    this.behaviourRuleProcessor = new BehaviourRuleProcessor(context);
    this.behaviourRuleGroupProcessor = new
BehabiourRuleGroupProcessor(context);
    this.eventsProcessor = new EventsProcessor(context);
    this.conditionActionProcessor = new
ConditionActionProcessor(context);
    this.conditionProcessor = new ConditionProcessor(context);
    this.actionProcessor = new ActionProcessor(context);
}
```

En esta clase hay que sobrescribir dos funciones que son las encargadas de escuchar lo que le llega al *servlet* desde la aplicación web, estas son *doGet* y *doPost*.

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    processCall(request, response, "GET");
}
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    processCall(request, response, "POST");
}
```

En ellas se llamará a la misma función *processCall* para procesar los datos que recibe el *servlet*. Esta función es la que se va a encargar de repartir los datos que llegan a los objetos correspondientes según el tipo que los datos que estén llegando.

Como podemos observar en la variable *resposta* se guardará lo que devuelva el *servlet* como respuesta a la petición, esta respuesta se escribirá en la variable *out* que es del tipo *PrintWriter*. Esto muestra en el navegador el contenido de la variable *resposta,* que como ya se ha comentado en la sección de

la aplicación web es lo que recogerá la llamada de la función php *get_file_content()*. El contenido de esta variable siempre será texto o texto en formato json.

```
private void processCall(HttpServletRequest request, HttpServletResponse
response, String postOrGet) throws ServletException, IOException {
    String resposta = null;

    String type =
request.getParameter(SmartHomeHTTPRuleAPIConstants.TYPE);

    if (type.equalsIgnoreCase(SmartHomeHTTPRuleAPIConstants.RULE_TYPE)){
        resposta = this.behaviourRuleProcessor.doRuleProcessor(request,
model);
    }
    else if (
type.equalsIgnoreCase(SmartHomeHTTPRuleAPIConstants.GROUP_TYPE)){
        resposta =
this.behaviourRuleGroupProcessor.doGroupProcessor(request, model);
    }
    else if (
type.equalsIgnoreCase(SmartHomeHTTPRuleAPIConstants.EVENT_TYPE) ){
        resposta = this.eventsProcessor.doEventProcessor(request, model);
    }
    else if (
type.equalsIgnoreCase(SmartHomeHTTPRuleAPIConstants.CONDITIONACTIONS_TYPE
) ){
        resposta =
this.conditionActionProcessor.doConditionActionProcessor(request, model);
    }
    else if (
type.equalsIgnoreCase(SmartHomeHTTPRuleAPIConstants.CONDITION_TYPE) ){
        resposta = this.conditionProcessor.doConditionProcessor(request,
model);
    }
    else if (
type.equalsIgnoreCase(SmartHomeHTTPRuleAPIConstants.ACTION_TYPE) ){
        resposta = this.actionProcessor.doActionProcessor(request,
model);
    }
    out = response.getWriter();
    out.write(resposta);
    out.close();
}
```

7.3.2.2 SmartHomeModelManagerWebImpl

Esta clase es la que centralizará todas las operaciones que se hagan en el modelo de la casa. Esta clase extiende de una clase del SmartHome, *EcoreFileManager* y cuya finalidad es la de obtener todos los recursos que se encuentran en el archivo xml de configuración del SmartHome.

Al extender de esta clase tenemos acceso a la función *setFileURI()* que se encarga de cargar los recursos desde el fichero.

```
public void setFileURI(String fileURI) {
    this.fileURI = fileURI;
    this.openFile();
}

protected void openFile() {
    _xmiFac = new XMIResourceFactoryImpl();
    rSet = new ResourceSetImpl ();

rSet.getResourceFactoryRegistry().getExtensionToFactoryMap().put("*",_xmi
Fac);
    uri = URI.createURI(this.fileURI);
    this.resource = this.rSet.getResource(uri, true);

}
```

Al ser una clase muy extensa solo se van a comentar las funciones más importantes que gestionan el modelo del SmartHome.

7.3.2.2.1 getModel()

Esta función obtiene todos los recursos y los asigna a la variable *specModel*.

```
public void getModel(String fileURI) {
    if ( fileURI == null ) {
        this.setFileURI(this.fileURI);
    } else {
        this.setFileURI(fileURI);
    }
    specModel = (SmartHomeSpecification)
this.getResource().getContents().get(0);
    System.out.println("Pocessing SmartHome Specification for web: " +
specModel.getName());
}
```

7.3.2.2.2 newBehaviorRule(id, name)

Esta función se encarga de crear una nueva regla en la casa. Lo primero que hace antes de crearla es buscar si existe como servicio o está en el modelo, si no existe creamos una nueva regla con el *SmartHomeFactory* y le asignamos el identificador y el nombre que le pasan por parámetros. Se asigna a la variable regla del la clase y se asigna al grupo que esté seleccionado en la clase en ese momento. Creamos una lista de eventos vacía y se la asignamos. Finalmente guardamos los recursos creados y procesamos el grupo para que la nueva regla este cargada como servicio.

```
public boolean newBehaviorRule(String id, String name){
    //Buscamos si existe ese id (lo busco en los servicios en vez de en
el modelo)
    org.pros.upv.es.smarthomes.Rules.interfaces.IRule existingRule =
this.tools.searchBehaviourRule(this.context, id);
    BehaviourRule rule = seachBehaviourRule(id); // Buscamos en el modelo
    if ( existingRule != null || rule != null) {
        this._debug("Found an existing Rule with the same ID: " + id);
        return false;
    }
    // Creamos nueva behabiourRule
    myRule = SmartHomeFactory.eINSTANCE.createBehaviourRule();
    myRule.setId(id);
    myRule.setName(name);

    myRule.setEventList(SmartHomeFactory.eINSTANCE.createEventList());
    myGroup.getRules().add(myRule);

    System.out.println("BehaviourRule: "+myRule.getName()+" added in
group "+myGroup.getName()+".");
    // Guardamos los recursos
    this.save();
    // Para activar la nueva rule creada
    // Procesamos todo el grupo en el que se encuentra
    RulesProcessor rp = new RulesProcessor(context);
    rp.processBehaviourRuleGroup(myGroup);
    return true;
}
```

7.3.2.2.3 deleteBehaviourRule(id)

Esta función como su propio nombre indica borra una regla del modelo y la elimina como servicio.

```
public boolean deleteBehaviourRule(String id){
   //Buscamos en los servicios
   IRule existingRule = this.tools.searchBehaviourRule(this.context, id);
   if (existingRule != null) {
       existingRule.remove();
       this._debug("Remove existing Rule with the ID: " + id);
       // Buscamos en el modelo
       BehaviourRule br = seachBehaviourRule(id);
       if (br != null){
           BehaviourRuleGroup brg = getGroupOfBehaviourRule(id);
           Boolean r = brg.getRules().remove(br);
           if(r) {
               this.save();// Guardamos los recursos
               this._debug("Rule "+br.getName()+ " deleted from Group
"+brg.getName());
               return true;
           }
           else this._debug("Rule "+br.getName()+ " can´t deleted from
Group "+brg.getName());
           return false;
       }
   }
   return false;
}
```

7.3.2.2.4 setBehaviourRule(id)

Esta función asigna una regla al manager, pero para hacerlo primero tiene que comprobar que esa regla existe en el modelo por lo que primero se busca la regla. Si se encuentra también asignamos el grupo al que pertenece esta regla al manager. Con la función *getGroupOfBehaviourRule(id)* se busca a qué grupo de reglas pertenece una regla. Devuelve cierto si se ha asignado y falso en caso contrario.

```
public boolean setBehaviourRule(String id){
    BehaviourRule br = seachBehaviourRule(id);
    if (br != null){
        BehaviourRuleGroup brg = getGroupOfBehaviourRule(id);
        this.setMyGroup(brg);
        this.myRule = br;
        this._debug("Rule "+br.getName()+ " and Group "+brg.getName()+"
selected.");
        return true;
    }
    return false;
}
```

7.3.2.2.5 getMyGroup()

Esta función asigna el grupo por defecto en el manager, su identificador está configurado en la clase de constantes, si este grupo no existe en el modelo lo crea.

```
public BehaviourRuleGroup getMyGroup(){
    if(myGroup == null){
        myGroup =
getBehaviourRuleGroup(SmartHomeHTTPRuleAPIConstants.MY_GROUP);
        // Si no está en el modelo lo creamos
        if(myGroup != null){
            System.out.println("Finded group \"" + myGroup.getName() +
"\"");
        }
        else {
            myGroup =
SmartHomeFactory.eINSTANCE.createBehaviourRuleGroup();
            myGroup.setName(SmartHomeHTTPRuleAPIConstants.MY_GROUP);

specModel.getBehaviourRulesList().getBehaviourRuleGroups().add(myGroup);
            this.save();
            System.out.println("Created Group " + myGroup.getName());
        }
    }
    return myGroup;
}
```

7.3.2.2.6 getBehaviourRuleGroup(id)

Devuelve el grupo con el identificador que se le pasa por parámetro o *null* en caso de no encontrarlo. Esta función realiza una búsqueda en todos los grupos y subgrupos de reglas del modelo.

```
public BehaviourRuleGroup getBehaviourRuleGroup(String idgroup){
    BehaviourRuleGroup brGroup = null;

    Iterator<BehaviourRuleGroup> it_brGroups =
specModel.getBehaviourRulesList().getBehaviourRuleGroups().iterator();
    while ( it_brGroups.hasNext() ){
        BehaviourRuleGroup brg = it_brGroups.next();
        brGroup = _getBehaviourRuleGroup(brg, idgroup);
        if(brGroup != null) return brGroup;
    }
    return brGroup;
}
```

Esta función hace uso de la función *_getBehaviourRuleGroup()* que realiza una búsqueda en el grupo e identificador que se le pasa por parámetros. Esta búsqueda es recursiva dentro del grupo, ya que un grupo de reglas puede contener más grupos.

```
private BehaviourRuleGroup _getBehaviourRuleGroup(BehaviourRuleGroup
brGroup, String group){
    BehaviourRuleGroup brSubGroup = null;

    if ( brGroup == null )
        return null;

    if(brGroup.getName().equalsIgnoreCase(group)){
        return brGroup;
    }

    Iterator<BehaviourRuleGroup> it_brGroups =
brGroup.getSubGroups().iterator();
    while ( it_brGroups.hasNext() ){
        brSubGroup = _getBehaviourRuleGroup(it_brGroups.next(), group);
        if(brSubGroup != null) return brSubGroup;
    }
    return brSubGroup;
}
```

7.3.2.2.7 seachBehaviourRule(id)

Esta función se encarga de buscar en el modelo la regla cuyo identificador se le pasa por parámetro. Esta función busca dentro de todos los grupos de reglas la regla en cuestión y si no la encuentra devuelve *null*.

```
public BehaviourRule seachBehaviourRule(String ruleId){
    BehaviourRule br = null;
    Iterator<BehaviourRuleGroup> it_brGroups =
specModel.getBehaviourRulesList().getBehaviourRuleGroups().iterator();
    while ( it_brGroups.hasNext() ){
        BehaviourRuleGroup brGroup = it_brGroups.next();
        br = _searchBehaviourRule(brGroup, ruleId);
        if (br != null) return br;
    }
    return br;
}
```

Esta función hace uso de una función recursiva para encontrar la regla.

```
private BehaviourRule _searchBehaviourRule(BehaviourRuleGroup brGroup,
String ruleId){
    BehaviourRule br = null;
    if ( brGroup == null )
        return null;
    EList<BehaviourRule> brs = brGroup.getRules();
    if ( brs == null )
        return null;

    Iterator<BehaviourRule> it_br = brs.iterator();
    while ( it_br.hasNext() ){
        br = it_br.next();
        if (br.getId().equalsIgnoreCase(ruleId)){
            return br;
        }
    }

    EList<BehaviourRuleGroup> brGroups = brGroup.getSubGroups();
    Iterator<BehaviourRuleGroup> it_brGroups = brGroups.iterator();
    while ( it_brGroups.hasNext() ){
        br = _searchBehaviourRule(it_brGroups.next(), ruleId);
        if (br != null) return br;
    }
    return null;
}
```

Como se ha podido comprobar hasta este momento las funciones de búsqueda en el modelo tiene una parte recursiva que será implementada de la misma manera en todas las funciones.

7.3.2.2.8 getGroupOfBehaviourRule(id)

Devuelve el grupo al que pertenece el identificador de la regla que se pasa por parámetro.

7.3.2.2.9 searchBehaviourRuleGroups()

Busca todos los grupos de reglas y los almacena en una cadena de texto en formato json que es lo que devolverá la función.

```
public String searchBehaviourRuleGroups(){
    JSONArray ja = new JSONArray();

    Iterator<BehaviourRuleGroup> it_brGroups =
specModel.getBehaviourRulesList().getBehaviourRuleGroups().iterator();
    while ( it_brGroups.hasNext() ){
        BehaviourRuleGroup brg = it_brGroups.next();
        this._searchBehaviourRuleGroups(brg, ja);
    }
    return ja.toJSONString();
}

public void _searchBehaviourRuleGroups(BehaviourRuleGroup brGroup,
JSONArray ja) {
    if ( brGroup == null )
        return;
    ja.add(brGroup.getName());
    Iterator<BehaviourRuleGroup> it_brGroups =
brGroup.getSubGroups().iterator();
    while ( it_brGroups.hasNext() ){
        this._searchBehaviourRuleGroups(it_brGroups.next(), ja);
    }
}
```

Mientras recorre todos los grupos se van añadiendo los identificadores de los grupos en la variable JSONArray, y al finalizar con la función de la librería de json *toJSONString()* se convierten todos los datos a una cadena de texto en formato json. Esta función también hace uso de otra que trabaja de manera recursiva para recorrer todos los grupos.

7.3.2.2.10 deleteEvent(BehaviourRule br, int index)

Borra de la regla, que se pasa por parámetro, el evento en la posición que marca el parámetro index. Al estar los eventos dentro de una lista y no tener identificador se ha optado por acceder a ellos mediante su posición en la lista.

```
public Boolean deleteEvent(BehaviourRule br, int index){
    try {
        br.getEventList().getEvents().remove(index);
    } catch (IndexOutOfBoundsException e) {
        return false;
    }
    this.save();
    return true;
}
```

Lanza una excepción si el índice pasado no está dentro del rango del que están los eventos. Como por ejemplo si solo hay tres eventos y se pide borrar el

evento en la posición seis, aunque esto es muy difícil que pase es aconsejable tenerlo en cuenta para evitar posibles fallos.

7.3.2.2.11 getEventList(BehaviourRule)

Esta función devuelve todos los eventos de una regla en una cadena de texto en formato json. En esta función se ha tenido que identificar cada tipo de evento porque por la limitación de la librería de json no todos los eventos se podían pasar directamente a json y se ha tenido que procesar algunos a mano sacando sus propiedades y metiéndolas en un objeto json.

Como se puede comprobar en el código solo los eventos de dispositivo, regla y escena no se han podido pasar directamente, el resto de eventos sí.

```
public String getEventList(BehaviourRule br){
    JSONArray ja = new JSONArray();
    int count = 0;
    Iterator<Event> el = br.getEventList().getEvents().iterator();
    while (el.hasNext()){
      Event event = el.next();
      String eClass = event.getClass().getSimpleName();
      JSONObject jo = new JSONObject();
      jo.put("index", count);
      jo.put("type", eClass);
      jo.put("enabled", event.isEnabled());
      if(eClass.equalsIgnoreCase(DeviceEventImpl.class.getSimpleName())){
          DeviceEvent e = (DeviceEvent) event;
          JSONObject joe = new JSONObject();
          joe.put("action", e.getAction());
          joe.put("direction", e.getDirection());
          joe.put("device", e.getDevice().getId());
          jo.put(SmartHomeHTTPRuleAPIConstants.EVENT_TYPE, joe);
      }
      else
if(eClass.equalsIgnoreCase(BehaviourEventImpl.class.getSimpleName())){
          BehaviourEvent e = (BehaviourEvent) event;
          JSONObject joe = new JSONObject();
          joe.put("action", e.getAction());
          joe.put("rule", e.getRule().getId());
          jo.put(SmartHomeHTTPRuleAPIConstants.EVENT_TYPE, joe);
      }
      else
if(eClass.equalsIgnoreCase(SceneEventImpl.class.getSimpleName())){
          SceneEvent e = (SceneEvent) event;
          JSONObject joe = new JSONObject();
          joe.put("action", e.getAction());
          joe.put("scene", e.getScene().getId());
          jo.put(SmartHomeHTTPRuleAPIConstants.EVENT_TYPE, joe);
      }
      else{
          jo.put(SmartHomeHTTPRuleAPIConstants.EVENT_TYPE, event);
      }
      ja.add(jo);
      count++;
    }
    return ja.toJSONString();
}
```

7.3.2.2.12 searchKNXGroupAddress()

Devuelve todos los grupos de direcciones KNX en una cadena de texto json.

7.3.2.2.13 searchDevices()

Devuelve todos los dispositivos de la casa en una cadena de texto json.

7.3.2.2.14 searchRules()

Esta función devuelve todos los objetos cuyo padre es de tipo regla del modelo, o sea no solo las reglas, sino también las escenas y las reglas de reconfiguración, en una cadena de texto json. Esta función simplemente une todas las búsquedas de cada elemento en el modelo utilizando la función *merge*.

```
public String searchRules(){
    String brules = this.searchBehaviourRules();
    JSONArray or = (JSONArray) JSONValue.parse(brules);

    String scenes = this.searchScenes();
    JSONArray os = (JSONArray) JSONValue.parse(scenes);

    String reconf = this.searchReconfigurationRules();
    JSONArray rr = (JSONArray) JSONValue.parse(reconf);

    or.merge(os);
    or.merge(rr);
    return or.toJSONString();
}
```

7.3.2.2.15 searchBehaviourRules()

Devuelve todas las reglas del modelo en una cadena de texto json. Cada elemento contendrá el identificador y el nombre de la regla.

```
private String searchBehaviourRules(){
    JSONArray ja = new JSONArray();
    Iterator<BehaviourRuleGroup> it =
specModel.getBehaviourRulesList().getBehaviourRuleGroups().iterator();
    while ( it.hasNext() ){
        BehaviourRuleGroup bg = it.next();
        this._searchBehaviourRules(bg, ja);
    }
    return ja.toJSONString();
}
```

Para ello a parte del objeto *JSONArray*, se crea para cada regla un objeto *JSONObject* en el que se guardarán el nombre e identificador de la regla.

Este será el objeto se añada al *JSONArray*.

```
private void _searchBehaviourRules(BehaviourRuleGroup bg, JSONArray ja) {
    if (bg == null)
        return;
    EList<BehaviourRule> brs = bg.getRules();
    if ( brs == null )
        return;

    Iterator<BehaviourRule> it_br = brs.iterator();
    while ( it_br.hasNext() ){
        JSONObject jo = new JSONObject();
        BehaviourRule br = it_br.next();
        jo.put(SmartHomeHTTPRuleAPIConstants.ID, br.getId());
        jo.put(SmartHomeHTTPRuleAPIConstants.NAME, "Behaviour Rule
"+br.getName());
        ja.add(jo);
    }

    EList<BehaviourRuleGroup> brGroups = bg.getSubGroups();
    Iterator<BehaviourRuleGroup> it_brGroups = brGroups.iterator();
    while ( it_brGroups.hasNext() )
        this._searchBehaviourRules(it_brGroups.next(), ja);
}
```

7.3.2.2.16 searchScenes()

Devuelve todas las escenas del modelo en una cadena de texto json. Como en la anterior, de cada escena se guardará el identificador y el nombre. La implementación es igual a la función anterior.

7.3.2.2.17 searchReconfigurationRules()

Devuelve todas las reglas de reconfiguración del modelo en una cadena de texto en formato json. La implementación es igual a la función anterior.

7.3.2.2.18 getReconfigurationRule(id)

Devuelve la regla de reconfiguración con el identificador que se le pasa por parámetro, si no la encuentra se devuelve *null*.

7.3.2.2.19 searchProperties()

Devuelve todas las propiedades del modelo en una cadena de texto en formato json.

7.3.2.2.20 getProperty(String id)

Devuelve la propiedad cuyo identificador se le pasa por parámetro. Si no la encuentra en el modelo se devuelve *null*.

7.3.2.2.21 getEventDirections()

Devuelve todas las opciones que hay en el modelo para *EventDirections* en una cadena de texto en formato json. Estas opciones son las que se pondrán en listas para ser elegidas en la aplicación web.

```
public String getEventDirections(){
    JSONArray ja = new JSONArray();
    Iterator<EObject> it_obj =
SmartHomeFactory.eINSTANCE.getSmartHomePackage().getEventDirections().eCo
ntents().iterator();
    while ( it_obj.hasNext() ){
        ja.add(it_obj.next());
    }
    return ja.toJSONString();
}
```

7.3.2.2.22 getDeviceActions()

Lo mismo que la función anterior pero para *DeviceActions*.

7.3.2.2.23 getBehaviourRuleActions()

Ídem que la anterior pero para *BehaviourRuleActions*.

7.3.2.2.24 getSceneActions()

Ídem que la anterior pero para *SceneActions*.

7.3.2.2.25 getPropertyEventTypes()

Ídem que la anterior pero para *PropertyEventTypes*.

7.3.2.2.26 getDeviceStatus()

Ídem que la anterior pero para *DeviceStatus*.

7.3.2.2.27 getDeviceTypes()

Ídem que la anterior pero para *DeviceTypes*.

7.3.2.2.28 newEvent(BehaviourRule br, String eventype, String params)

Esta función añade un nuevo evento a una regla. El parámetro *eventype* es el tipo de evento que se va a añadir y *params* es una cadena de texto en formato json con todas las propiedades del evento a añadir. Sabremos el contenido de este parámetro porque depende del tipo de evento que se esté añadiendo.

```
public Boolean newEvent(BehaviourRule br, String eventype, String
params){
    JSONObject jo = (JSONObject) JSONValue.parse(params);
    if
(eventype.equalsIgnoreCase(SmartHomeHTTPRuleAPIConstants.KNX_TYPE)){
        KNXEvent e = SmartHomeFactory.eINSTANCE.createKNXEvent();

        Boolean bol = Boolean.valueOf((String)jo.get("enabled"));
        e.setEnabled(bol);

        String dir = (String)jo.get("direction");
        e.setDirection(EventDirections.get(dir.toUpperCase()));

        String ad = (String)jo.get("address");
        KNXGroupAddress knxga = getKNXGroupAddress(ad);
        e.setGroupAddress(knxga);

        Boolean done = br.getEventList().getEvents().add(e);
        if (done) this.save();
        return done;
    }
```

En el código se observa que para un tipo de evento se procesan distintas propiedades de este. Por ejemplo en el evento KNX primero creamos el tipo de evento KNX y luego en la variable *JSONObject jo* se convierte la cadena de texto json en un objeto json del que es más fácil extraer los elementos. Como por ejemplo se sabe que *enabled* es un booleano y al extraerlo se guarda en booleano y se asigna la propiedad al evento KNX. Y así con todas las propiedades de cada tipo de evento.

El único evento un poco distinto es el de *PropertyEvent* porque en este se puede añadir un atributo que en el que el tipo de valor que puede variar, pero como el tipo también lo obtenemos por json es muy fácil asignarlo.

A continuación un fragmento del código que realiza lo descrito anterior mente.

```
    // Añado el datavalue si está presente
    String dv = (String)jo.get("datavalue");
    String v = (String)jo.get("value");
    //boolean
    if (dv.equalsIgnoreCase("boolean")){
        BooleanValue d = SmartHomeFactory.eINSTANCE.createBooleanValue();
        d.setValue(Boolean.valueOf(v));
        e.setValue(d);
    //string
    } else if (dv.equalsIgnoreCase("string")){
        StringValue d = SmartHomeFactory.eINSTANCE.createStringValue();
        d.setValue(v);
        e.setValue(d);
    //integer
    } else if (dv.equalsIgnoreCase("integer")){
        IntegerValue d = SmartHomeFactory.eINSTANCE.createIntegerValue();
        d.setValue(Integer.valueOf(v));
        e.setValue(d);
```

Se comprueba de que tipo es el dato en el *datavalue* y luego extraemos del valor del dato del campo *value*. Así se puede asignar el tipo de dato correcto.

7.3.2.2.29 getScene(String id)

Devuelve una escena cuyo identificador es el pasado por parámetro. Si no lo encuentra devuelve *null*.

7.3.2.2.30 getRule(String id)

Devuelve la regla cuyo identificado se pasa por parámetro. Esta función se diferencia de una parecida nombrada anteriormente en que esta función está compuesta por tres búsquedas. Busca en reglas, escenas y reglas de reconfiguración.

```
private Rule getRule(String id) {
    Rule r = seachBehaviourRule(id);
    if (r != null) return r;

    ReconfigurationRule rr = getReconfigurationRule(id);
    if (rr != null) return rr;

    Rule s = getScene(id);
    if (s != null) return s;

    return null;
}
```

Primero busca en reglas y si lo encuentra la devuelve, si no busca en reconfiguración de reglas y si no continua con escenas.

7.3.2.2.31 getKNXGroupAddress(String id)

Devuelve el KNXGroupAddress cuyo identificador es pasado por parámetro o *null* si no lo encuentra.

7.3.2.2.32 getDevice(String id)

Devuelve el dispositivo con el identificador que se le pasa a la función o *null* si no lo encuentra.

7.3.2.2.33 newConditionAction(BehaviourRule br, String description, Boolean enable)

Crea una nueva lista de *ConditionActions* con los valores que se le pasa por parámetro. Además también creamos las listas vacías de *Conditions* y *Actions*.

```
public boolean newConditionAction(BehaviourRule br, String description,
Boolean enable){
    ConditionActions ca =
SmartHomeFactory.eINSTANCE.createConditionActions();
    ca.setDescription(description);
    ca.setEnabled(enable);

ca.setConditionList(SmartHomeFactory.eINSTANCE.createConditionList());
    ca.setActionList(SmartHomeFactory.eINSTANCE.createActionList());
    br.getCondition_actions().add(ca);
    this.myCA = ca;

    System.out.println("BehaviourRule: "+br.getName()+" added
ConditionAction: "+description+".");
    // Guardamos los recursos
    this.save();
    return true;
}
```

7.3.2.2.34 getConditionActions(BehaviourRule br)

Devuelve todas las listas de *ConditionActions* que hay en una regla. Devuelve su posición en la lista y su descripción en formato json.

```
public String getConditionActions(BehaviourRule br) {
    JSONArray ja = new JSONArray();
    int i = 0;
    Iterator<ConditionActions> itca =
br.getCondition_actions().iterator();
    while(itca.hasNext()){
        ConditionActions ca = itca.next();
        JSONObject jo = new JSONObject();
        jo.put(SmartHomeHTTPRuleAPIConstants.ID, i);
        jo.put(SmartHomeHTTPRuleAPIConstants.DESC, ca.getDescription());
        ja.add(jo);
        i++;
    }
    this._debug(ja.toJSONString());
    return ja.toJSONString();
}
```

7.3.2.2.35 setConditionAction(Integer index)

Asigna en el manager la lista de *ConditionActions* en la posición que se pasa por parámetro, de la regla seleccionada en ese momento en el manager.

```
public boolean setConditionAction(Integer index) {
    EList<ConditionActions> lca = myRule.getCondition_actions();
    try {
        myCA = lca.get(index);
    } catch (IndexOutOfBoundsException e) {
        return false;
    }
    return true;
}
```

7.3.2.2.36 deleteConditionActions(BehaviourRule br, int index)

Elimina la lista de *ConditionActions* en la posición *index* de la regla que se le pasa por parámetro.

```
public boolean deleteConditionActions(BehaviourRule br, int index){
    try {
        br.getCondition_actions().remove(index);
    } catch (IndexOutOfBoundsException e) {
        return false;
    }
    this.save();
    return true;
}
```

7.3.2.2.37 getDeviceConditionOperators()

Devuelve una cadena de texto json con todos los operadores de condición de dispositivo del modelo. El resto de este tipo de funciones la implementación es igual.

```
public String getDeviceConditionOperators(){
    JSONArray ja = new JSONArray();

ja.add(SmartHomeFactory.eINSTANCE.getSmartHomePackage().getDeviceConditio
n_Operator());
    return ja.toJSONString();
}
```

7.3.2.2.38 getEventConditionOperators()

Ídem que el anterior pero para *EventConditionOperators*.

7.3.2.2.39 getPropertyConditionOperators()

Ídem que el anterior pero para *PropertyConditionOperators*.

7.3.2.2.40 getConditionList(ConditionActions ca)

Devuelve una cadena de texto json con todas las condiciones y sus propiedades que hay en la lista de *ConditionActions* que se pasan por parámetro. En esta función se procesan a mano todas las propiedades de las condiciones en un objeto json.

Primero se comprueba que hay una lista de condiciones y si no la hay se crea una vacía y se devuelve un array json vacío.

Solo se muestra una parte del código para la implementación de un tipo de condición, *DeviceCondition*, el resto se hace igual.

```
public String getConditionList(ConditionActions ca){
    JSONArray ja = new JSONArray();
    int count = 0;
    // Las reglas ya creadas puede que no tenga lista de condiciones
    Iterator<Condition> it = null;
    try {
        it = ca.getConditionList().getConditions().iterator();
    } catch (Exception e) {
        // Si error no tiene lista de Conditions

ca.setConditionList(SmartHomeFactory.eINSTANCE.createConditionList());
        return ja.toJSONString();
    }
    while (it.hasNext()){
        Condition cond = it.next();
        String eClass = cond.getClass().getSimpleName();
        JSONObject jo = new JSONObject();
        jo.put("index", count);
        jo.put("type", eClass);
        jo.put("enabled", cond.isEnabled());

if(eClass.equalsIgnoreCase(DeviceConditionImpl.class.getSimpleName())){
            DeviceCondition c = (DeviceCondition) cond;
            JSONObject joc = new JSONObject();
            joc.put("values", c.getValues());//es una lista
            joc.put("isnot", c.isIs_not());
            joc.put("operator", c.getOperator());
            joc.put("device", c.getDevice().getId());
            jo.put(SmartHomeHTTPRuleAPIConstants.CONDITION_TYPE, joc);
        }
```

7.3.2.2.41 deleteCondition(ConditionActions ca, int index)

Elimina una condición de índice *index* de una lista de *ConditionActions*.

7.3.2.2.42 String2Datavalue(JSONObject js)

Convierte la cadena de texto json en un objeto *DataValue*. En el código

```
private DataValue String2Datavalue(JSONObject js){
    DataValue dv = null;
    //boolean
    if (js.containsKey("boolean")){
        BooleanValue d = SmartHomeFactory.eINSTANCE.createBooleanValue();
        d.setValue(Boolean.valueOf((String)js.get("boolean")));
        return d;
    //string
    } else if (js.containsKey("string")){
        StringValue d = SmartHomeFactory.eINSTANCE.createStringValue();
        d.setValue((String)js.get("string"));
        return d;
    //integer
    } else if (js.containsKey("integer")){
        IntegerValue d = SmartHomeFactory.eINSTANCE.createIntegerValue();
        d.setValue(Integer.valueOf((String)js.get("integer")));
        return d;
```

mostrado a continuación solo aparecen la conversión de los tipos booleano, cadena de texto y entero, para el resto de tipos se hace igual.

7.3.2.2.43 newCondition(ConditionActions ca, String condtype, String params)

Crea una nueva condición en la lista de condiciones de la *ConditionActions* que se le pasa por parámetro, del tipo *condtype* y con los atributos que se le pasan por *params*, este último es en json. En esta parte del código de la función podemos observar que una condición puede contener varios valores *DataValue*. Estos se irán añadiendo uno a uno con un bucle que recorrerá el array de valores.

```
public Boolean newCondition(ConditionActions ca, String condtype, String
params){
    JSONObject jo = (JSONObject) JSONValue.parse(params);

    if
(condtype.equalsIgnoreCase(SmartHomeHTTPRuleAPIConstants.DEVICE_TYPE)){
        DeviceCondition c =
SmartHomeFactory.eINSTANCE.createDeviceCondition();

        Boolean bol = Boolean.valueOf((String)jo.get("enabled"));
        c.setEnabled(bol);

        Boolean not = Boolean.valueOf((String)jo.get("isnot"));
        c.setIs_not(not);

        String id = (String)jo.get("device");
        Device de = getDevice(id);
        c.setDevice(de);

        Integer o = Integer.parseInt((String)jo.get("operator"));
        c.setOperator(BRDeviceCondition_Operators.get(o));

        JSONArray value = (JSONArray)jo.get("value");
        if (value != null){
            for (int i = 0; i < value.size(); i++){
                JSONObject js = (JSONObject)value.get(i);
                DataValue dv = String2Datavalue(js);
                if (dv != null)
                    c.getValues().add(dv);
            }
        }
        Boolean done = ca.getConditionList().getConditions().add(c);
        if (done) this.save();
        return done;
    }
```

7.3.2.3 BehaviourRuleProcessor

La principal tarea de esta clase es procesar las peticiones de tipo *rule* y la función principal que se encargará de esto es *doRuleProcessor*.

```
public String doRuleProcessor(HttpServletRequest request,
SmartHomeModelManagerWebImpl model) {
    ret = "";
    String actionType =
request.getParameter(SmartHomeHTTPRuleAPIConstants.ACTION);
    this._debug("Action: "+actionType);

    // Return Rules
    if ( actionType == null || actionType == "" ||
actionType.equalsIgnoreCase(SmartHomeHTTPRuleAPIConstants.GET_RULES_ACTIO
N)){
        ret = this.searchBehaviourRules(model);
        this._debug("Rules: "+ret);

    // New Rule
    } else
if(actionType.equalsIgnoreCase(SmartHomeHTTPRuleAPIConstants.NEW_ACTION))
{
        String id =
request.getParameter(SmartHomeHTTPRuleAPIConstants.ID);
        String name =
request.getParameter(SmartHomeHTTPRuleAPIConstants.NAME);
        if(name == null || name == "") name = id;

        boolean r = model.newBehaviorRule(id, name);
        JSONObject js = new JSONObject();
        js.put(SmartHomeHTTPRuleAPIConstants.ID, id);
        js.put(SmartHomeHTTPRuleAPIConstants.DONE_ACTION, r);
        this._debug(js.toJSONString());
        return js.toJSONString();
    }....
```

Una vez se ha obtenido la acción se procede a realizar las llamadas correspondientes al manager para realizar la acción. Por ejemplo si es obtener todas las reglas se llamará a *searchBehaviourRules(model)* y obtendrá todas las reglas del modelo en formato de cadena de texto json y ya solo queda devolverla.

Otra acción como por ejemplo *newBehaviorRule(id, name)*, se extraerán de la llamada los parámetros de identificador y del nombre, si esta regla no tiene nombre se le asignará el identificador como nombre. Como esta función devuelve *true* o *false*, lo que haremos será empaquetar esta respuesta en formato json. El formato de respuesta será un objeto json en el que se pasará el identificador y otro parámetro llamado *done* donde irá la respuesta. Por ejemplo:

```
{"id":"idrule","done":true}
```

Para las acciones de borra y seleccionar el código será similar.

7.3.2.4 BehabiourRuleGroupProcessor

Esta clase es como la anterior pero procesará los grupos de reglas.

La función principal *doGroupProcessor* solo tendrá que procesar las peticiones de búsqueda de grupos y de selección de un grupo en el manager.

```
public String doGroupProcessor(HttpServletRequest request,
SmartHomeModelManagerWebImpl model) {
    ret = "";
    String actionType =
request.getParameter(SmartHomeHTTPRuleAPIConstants.ACTION);
    this._debug("Action: "+actionType);

    // Return Groups
    if ( actionType == null || actionType == "" ||
actionType.equalsIgnoreCase(SmartHomeHTTPRuleAPIConstants.GET_GROUPS_ACTI
ON)){
        ret = model.searchBehaviourRuleGroups();
        this._debug("Groups: "+ret);

    // Select Group
    } else
if(actionType.equalsIgnoreCase(SmartHomeHTTPRuleAPIConstants.SELECT_ACTIO
N)){
        String name =
request.getParameter(SmartHomeHTTPRuleAPIConstants.NAME);

        BehaviourRuleGroup brGroup = model.getBehaviourRuleGroup(name);
        model.setMyGroup(brGroup);
    }
    return ret;
}
```

7.3.2.5 EventsProcessor

Ídem que la anterior pero para los eventos. Esta clase además de gestionar la creación y eliminación de un nuevo evento en una regla, devuelve todos los tipos de datos que un evento puede contener.

7.3.2.6 ConditionActionProcessor

Ídem que la anterior pero para las listas de *ConditionActions*. La función *doConditionActionProcessor* gestionará la creación y eliminación de listas de de *ConditionActions* en una regla así como la selección de la lista en el mánager.

7.3.2.7 ConditionProcessor

La gestión que hace esta clase es parecida a la que se hace de eventos pero para las condiciones de una lista de *ConditionAction*.

7.3.2.8 ActionProcessor

Ídem que la anterior pero para acciones.

7.3.2.9 SmartHomeHTTPRuleAPIConstants

En esta clase se definen todas las constantes utilizadas en el API entre la aplicación web y el *servlet*. Extracto del código de esta clase.

```
public interface SmartHomeHTTPRuleAPIConstants {
	// PARAMETRES
	public final String TYPE = "type";
	public final String ACTION = "action";

	public final String ID = "id";
	public final String NAME = "name";
	public final String DESC = "description";
	public final String ENABLED = "enabled";

	public final String INDEX = "index";
```

7.4 Pruebas

Durante toda la implementación se han ido realizando pruebas en distintos navegadores para verificar el correcto funcionamiento del código que se ha ido desarrollando.

Se ha ido probando tanto en el navegador Firefox como en el navegador Iron Portable basado en el navegador de google Chrome, ambos la versión de escritorio que en el momento de realizar la aplicación eran la 17.0.1 y la 21.0.1200.0 respectivamente, como con el navegador por defecto del dispositivo móvil HTC Wildfire con la versión 2.2 de Android, obteniéndose resultados satisfactorios del funcionamiento de la aplicación en todos los casos.

8 Dificultades y posibles mejoras

8.1 Dificultades surgidas

Una de las principales dificultades que surgieron fue el entendimiento y comprensión de la estructura del modelo de la SmartHome ya que es un modelo muy amplio y la gran cantidad de objetos que lo componen dificulta su comprensión. Véase parte del modelo en los anexos.

Otra de las dificultades a supera fue la forma de realizar el intercambio de información del servidor de la aplicación web y el servidor donde se ejecuta el *servlet* que es el que obtiene la información y realiza todas las operaciones directamente sobre el modelo del SmartHome.

Otra de las dificultades tiene que ver con el diseño de la aplicación, ya que tenía que poderse utilizar en pantallas táctiles de dispositivos móviles, por lo tanto pantallas pequeñas, fue la elección del componente en el que se iba a mostrar la información o que se iba a actuar sobre ella tenía que ser fácil de seleccionar e intuitivo para el usuario. Se solucionó utilizando en la medida de lo posible botones grandes y pocos componentes a la vez en la pantalla, y gestionando la introducción o modificación de reglas como un asistente paso por paso.

En la implementación también surgieron problemas referentes a la utilización de la librería de conversión de json por que no se podían pasar los objetos directamente al *parser* pero se solucionó creando la conversión a mando, codificando cada propiedad del objeto en cuestión en un objeto json.

8.2 Posibles mejoras

Una posible mejora sería la simplificación de las constantes que se usan en el API entre la aplicación web y el *servlet*. Por ejemplo en vez de usar varias constantes como *id*, *index*, *name* unirlas todas en una sola llamándola por ejemplo *id*. Con ello mejoraría el entendimiento del uso del API.

Otra mejora seria que la aplicación fuera capaz de gestionar varios usuarios a la vez. Aunque esta mejora queda fuera del objetivo de este proyecto.

9 Conclusiones

Desarrollar este proyecto final de carrera a supuesto todo un reto para mí por la distintas tecnologías que se han usado en el desarrollo de cada parte del proyecto, ya que eran completamente distintas.

He tenido que familiarizarme con dos entornos de desarrollo distintos, Netbeans y Eclipse.

He intentado y conseguido entender el código desarrollado por otras personas, sobre todo cuando el proyecto es tan grande como en el que he tenido que integrar la aplicación *servlet*.

Se ha desarrollado en varios lenguajes de programación distintos a la vez, html, javascript, php, java; con distintos framework, *jQuery*, *jQuery Mobile*, Json-Smart.

Se ha conseguido completar los objetivos propuestos para este proyecto y para ser la primera versión funciona correctamente.

Todo el esfuerzo ha valido la pena al ver funcionando la aplicación en un dispositivo móvil.

10 Bibliografía

Toda la información utilizada y consultada para la realización de este proyecto se ha conseguido a través de páginas web de internet y de la ayuda de mi director de proyecto.

A continuación una pequeña muestra de páginas web consultadas:

- Web de iWebKit: http://snippetspace.com/portfolio/iwebkit/
- Web JQuery Mobile: http://jquerymobile.com/gbs/
- Web del IDE Netbeans: http://netbeans.org/
- Web del IDE Eclipse: http://www.eclipse.org/modeling/emf/
- Web del WampServer: http://www.wampserver.com/en/
- Web del Json-Smart: http://code.google.com/p/json-smart/
- Web de comprobación de json: http://json.parser.online.fr/
- Web editor prototipos: http://codiqa.com/embed/editor/
- Web del manual de php: http://php.net/manual/es/
- Web para consultas sobre html: http://www.w3schools.com/
- Artículos:
    - http://www.ticbeat.com/sim/2016-habra-mas-dispositvos-moviles-personas/
    - http://www.android.es/crecimiento-de-ios-y-android-en-una-infografia.html
    - http://www.hongkiat.com/blog/mobile-web-design/
    - http://www.cesardelacruz.es/ajax-cross-domain-jsonp-php/
    - http://blog.unijimpe.net/utilizar-crossdomainxml/
    - http://www.webresourcesdepot.com/cross-domain-javascript-with-simple-php-proxy/
    - https://github.com/softius/php-cross-domain-proxy
    - http://php.net/manual/es/function.file-get-contents.php
    - http://www.emenia.es/metodo-on-de-jquery-tutorial/
    - http://graphpaperpress.com/tips/using-jquerys-on-event-to-prevent-default-actions/
    - http://www.emenia.es/metodo-on-de-jquery-tutorial/

11 Anexos

11.1 Poner en marcha

11.1.1 Aplicación web

En este apartado se va a describir como se pone todo el sistema en funcionamiento.

Para la aplicación web hay que utilizar un servidor web que pueda ejecutar código php. Se puede usar el que se quiera pero por sencillez se va a utilizar el *WampServer* ya que viene configurado con todo lo que se necesita.

Una vez instalado el programa hay que acceder a su directorio de instalación, por ejemplo "C:\wampserver\", entrar en el directorio llamado "www". Dentro de este directorio copia la carpeta contenedora de la aplicación web. Quedaría tal que así: "C:\wampserver\www\mobile_website".

Ahora solo queda ejecutar el archivo wampserver.exe y abrir el navegador y escribir en la dirección: "localhost\mobile_website".

Con eso accedemos a la pantalla de inicio de la aplicación web.

11.1.2 Aplicación servlet

Para la aplicación servlet hay que copiar el directorio del proyecto eclipse "SmartHomes.HTTPRuleAPIServlet" al directorio del workspace de eclipse donde se encuentre el resto de proyectos que componen el SmartHomes.

Abrir la aplicación eclipse modeling tools y ejecutar la configuración "Domotica (web)". El eclipse lleva su propio servidor de servlets por lo que se cargará en memoria todos los servlets presentes en el workspace.

Una vez arrancado el servlet ir a la pestaña "Console" de eclipse y ejecutar el comando de carga de un modelo de casa SmartHomes.

Una vez en marcha y el modelo cargado ya se puede ir al navegador web y empezar a trabajar con la aplicación web de creación de reglas.

11.2 Esquemas del modelo del SmartHome

En este anexo se van a mostrar partes del modelo del SmartHome que se han utilizado en la aplicación.

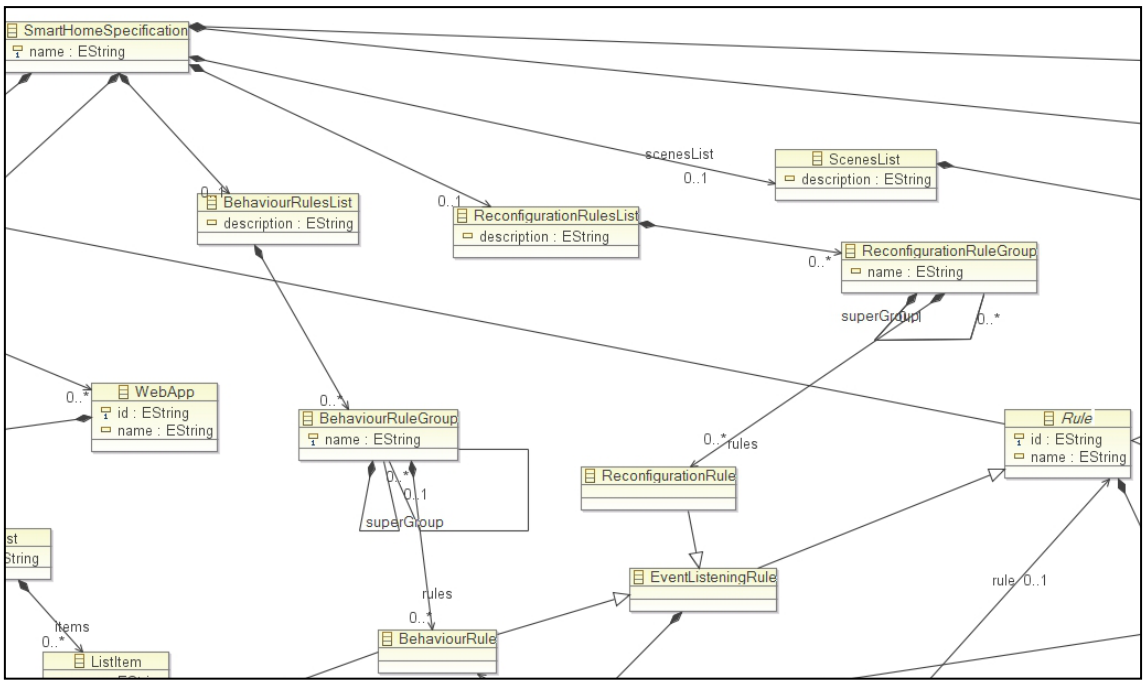

Ilustración 55 Clase principal del SmartHome

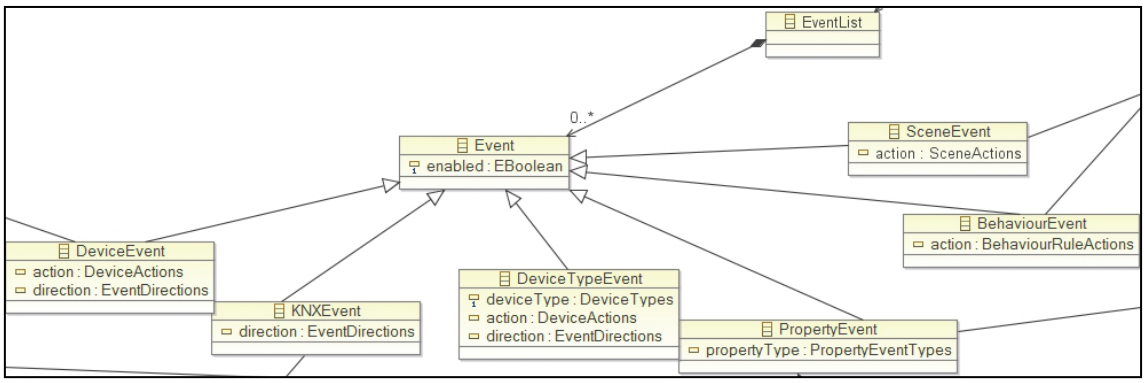

Ilustración 56 Clase de Eventos e hijos

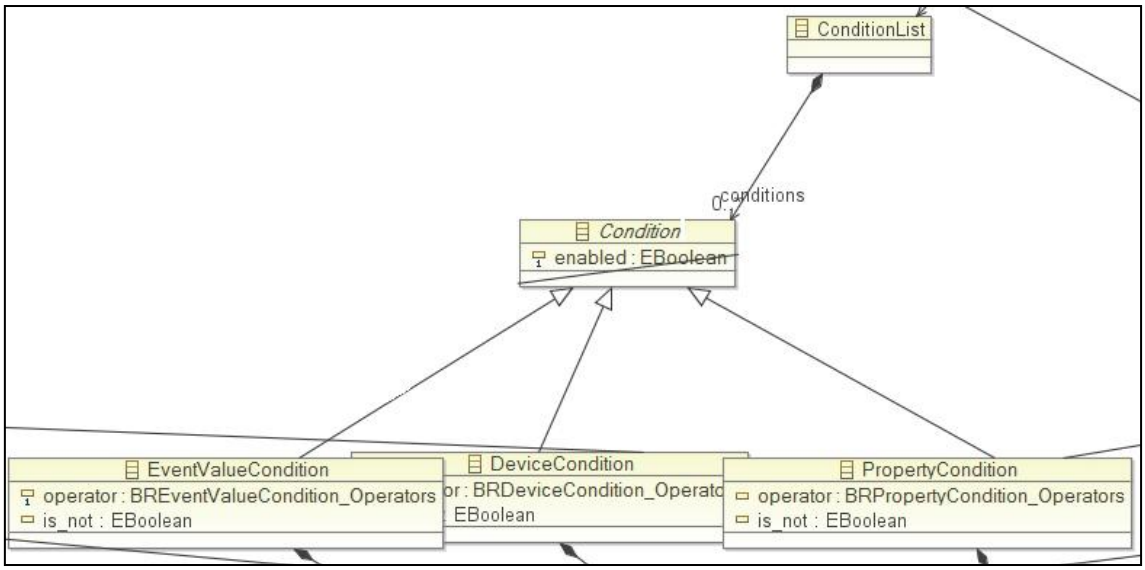

Ilustración 57 Clase de Condiciones e hijos

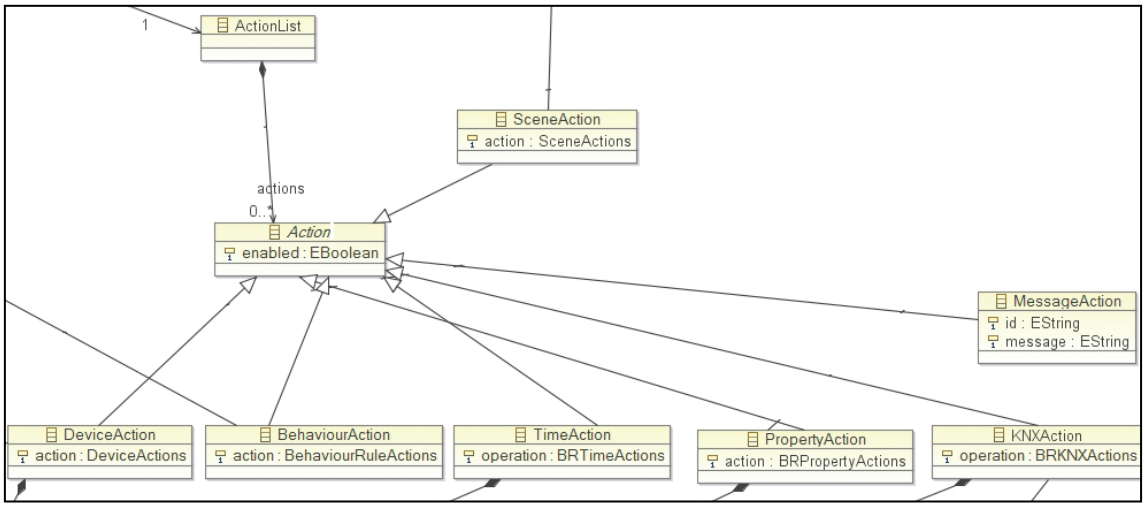

Ilustración 58 Clase de Acciones e hijos

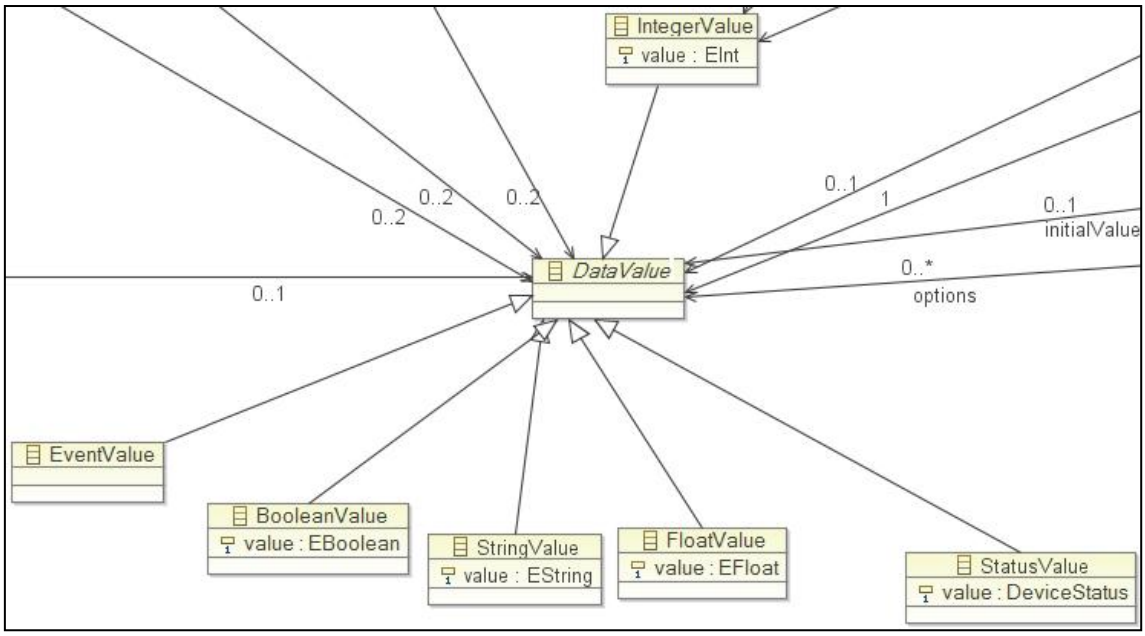

Ilustración 59 Clase de DataValue e hijos