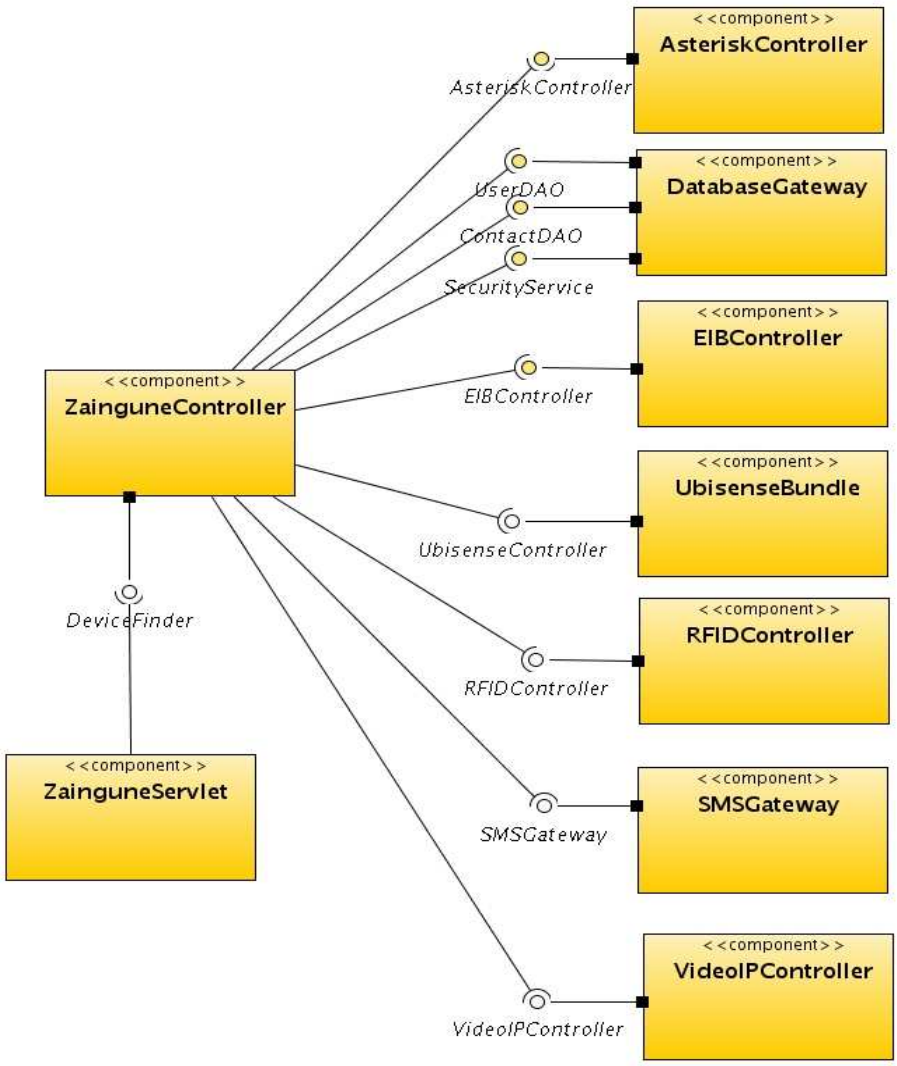
1 Introducción

En este documento se va a explicar el diseño e implementación utilizados en la plataforma Zaingune. Para ello se van a explicar la arquitectura del sistema, el controlador del sistema y las reglas de negocio que este utiliza, la parte servidora de la interfaz web realizada, la interfaz web, el desarrollo de nuevos componentes y la monitorización del sistema.

2 Arquitectura del Sistema Zaingune

El sistema Zaingune es una plataforma compuesta por diferentes componentes. Estos componentes tienen dependencias entre ellos que están representadas mediante el siguiente diagrama:



En el diagrama, en el lado derecho, se encuentran todos los componentes que ofrecen dispositivos físicos y lógicos. A la derecha se encuentran el controlador de Zaingune (ZainguneController), que se encarga de la lógica de control del sistema haciendo uso de los dispositivos Zaingune ofrecidos por el resto de componentes, y el puente entre la plataforma Zaingune y la interfaz Web (ZainguneServlet).

Las descripciones de los componentes que ofrecen dispositivos físicos y lógicos son las siguientes:

- AsteriskController: ofrece un servicio con la interfaz Java llamada "AsteriskController". Este servicio permite realizar llamadas a través de una centralita Asterisk a los usuarios registrados, gestionar usuarios, reproducir textos sintetizados entre otras cosas.

- DatabaseGateway: ofrece tres servicios diferentes. El primero es UserDao que gestiona los usuarios que tienen acceso al sistema. El segundo es ContactDao que gestiona los contactos de los que dispone cada uno de los usuarios del sistema. Por último está SecurityService que se encarga de gestionar las sesiones de usuario.

- EIBController: ofrece la interfaz Java EIBController y diferentes dispositivos físicos. La interfaz EIBController permite gestionar la conexión con el servidor EIB, cargar los dispositivos y acceder a ellos. Los dispositivos físicos ofrecen diferentes funcionalidades según su naturaleza (luz, termómetro, etc.). Además envía eventos de cambios de estado de los dispositivos EIB.

- UbisenseBundle: ofrece la interfaz Java UbisenseController. Este servicio permite obtener la localización de personas. Además envía eventos de aparición, desaparición y movimiento de personas.

- RFIDController: ofrece la interfaz Java RFIDController. Este servicio permite iniciar o parar búsquedas de tarjetas RFID. Cada vez que es detectada una tarjeta RFID envía un evento para notificar quien intenta registrarse.

- SMSGateway: ofrece la interfaz Java SMSGateway. Este servicio permite enviar mensajes SMS a los usuarios que están registrados en la plataforma de mensajería.

- VideoIPController: ofrece la interfaz Java VideoIPController. Este servicio permite gestionar las diferentes cámaras existentes en el sistema. Además envía eventos de detección de movimiento.

Todos los eventos mandados por estos componentes utilizan el servicio EventAdmin de OSGi.

En cuanto al componente ZainguneController, es el encargado de coordinar todos los servicios ofrecidos por los componentes anteriormente comentados. Para ello utiliza el motor de reglas JBoss Rules. Además, recoge todos los eventos generados en el sistema y toma decisiones basándose en estos.

Por otro lado, este componente ofrece la interfaz Java DeviceFinder. Este servicio permite buscar dispositivos lógicos y físicos del sistema según diferentes criterios. Estos dispositivos pueden ser buscados por identificador de servicio, nombre, tipo, etc.

Por último esta el ZainguneServlet que se encarga de unir la plataforma Zaingune con la interfaz Web. Este componente hace uso del servicio ofrecido por ZainguneController para buscar los dispositivos sobre los que el usuario desea interactuar.

3 Controlador del sistema

En este punto se va a explicar como ha sido desarrollado el sistema Zaingune. Para ello, se van a explicar las partes más importantes de ZainguneController.

ZainguneController es el núcleo de la plataforma Zaingune. Básicamente esta dividido en dos partes, la primera es el propio controlador, que hace uso de las reglas de negocio, y la segunda es la implementación de la interfaz DeviceFinder de Zaingune. Esta interfaz será utilizada más adelante por ZainguneServlet, aunque puede ser utilizada por cualquier otro componente del sistema.

3.1 Controlador

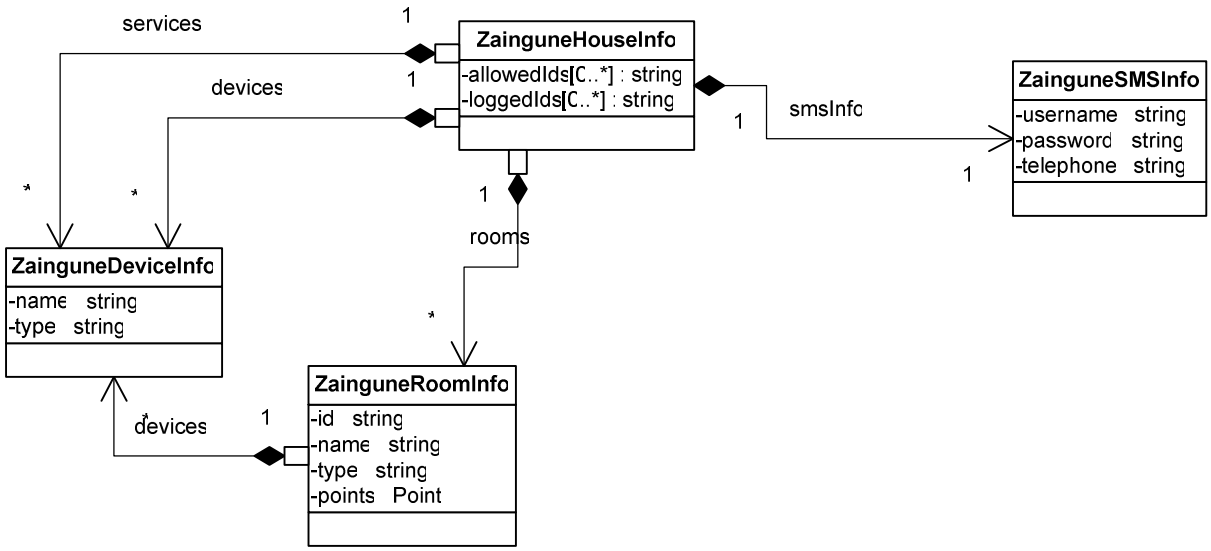
El controlador de Zaingune se encarga de la lógica del sistema. Para ello utiliza el motor de reglas JBoss Rules.

Cuando se arranca Zaingune, el controlador añade a la memoria de trabajo del motor de reglas los servicios existentes, para que este pueda realizar las acciones requeridas. Estas acciones son activadas mediante los eventos que son generados en el sistema. Estos eventos son escuchados por el controlador mediante la implementación de la interfaz HandlerEvent de OSGi y añadidos al motor de reglas.

Para utilizar el motor de reglas en controlador utiliza la interfaz RuleManager que contiene los siguientes métodos:

- public void loadRules(InputStream rules) throws Exception: carga las reglas de negocio del sistema.
- public void insertObject(Object object): añade un objeto a la memoria de trabajo del motor de reglas.
- public void modifyObject(Object object): modifica un objeto de la memoria de trabajo del motor de reglas.
- public void deleteObject(Object object): elimina un objeto de la memoria de trabajo del motor de reglas.
- public void fireRules(): lanza las reglas del motor de reglas. Cuando se añade/modifica/elimina no se lanzan las reglas automáticamente, ésta deben ser lanzadas llamando a este método.

Por otro lado también dispone de una configuración del sistema que es cargada mediante un parser sax. El fichero de configuración esta explicado en el manual de usuario de ZainguneController. El parser sax carga la configuración en las siguientes clases de modelo:



En cuanto a las reglas de negocio, como anteriormente se ha comentado, se ha utilizado el motor de reglas JBoss Rules (anteriormente Drools) en la versión 4.0. Este motor permite desarrollar reglas mediante dos lenguajes, un lenguaje XML y otro llamado DRL. Para este

sistema se ha utilizado DRL ya que su desarrollo es más intuitivo y mantenible, más legible por los humanos e igual de potente que el lenguaje en XML.

Las reglas utilizadas en Zaingune son las siguientes:

```
rule "RFID Event"
        no-loop true
        when
                event: RFIDEvent()
                houseInfo: ZainguneHouseInfo()
                deviceFinder: ZainguneDeviceFinder()
                securityService: SecurityService()
                userDao: UserDAO()
        then
                User user = userDao.getUserByRFIDTag(
                        event.getUserId());
                if (user != null)
                {
                        if (securityService.isLogged(
                                user.getUsername()))
                        {
                                securityService.logout(
                                        securityService.getSessionId(
                                        user.getUsername()) );
                        }
                        else
                        {
                                securityService.login(
                                        event.getUserId());
                        }

                        Collection<DeviceService> cameras= deviceFinder
                                .getDevicesByType("Camera");

                        for(DeviceService camera: cameras)
                        {
                                ((ICameraIP)camera).
                                        enableMotionDetection(
                                        !securityService.isLogged(
                                        user.getUsername()));
                        }
                }

                retract(event);
end

rule "Motion Detection Event"
        no-loop true
        when
                event: MotionDetectionEvent()
                houseInfo: ZainguneHouseInfo()
                zainguneController: ZainguneController()
                deviceFinder: ZainguneDeviceFinder()
                securityService: SecurityService()
```

```
        then
                if (!securityService.isSomeoneLogged())
                {
                        zainguneController.startTimer(20000);
                }

                Collection<DeviceService> cameras =
                        deviceFinder.getDevicesByType("Camera");

                for (DeviceService camera: cameras)
                {
                        ((ICameraIP)camera).
                                enableMotionDetection(false);
                }

                retract(event);
end

rule "Timer End Event --> Asterisk"
        no-loop true
        activation-group "timer"
        salience 1
        when
                event: TimerEvent()
                asteriskController: AsteriskController()
                houseInfo: ZainguneHouseInfo()
                securityService: SecurityService()
        then
                if (!securityService.isSomeoneLogged())
                {
                        asteriskController.callAndTalk(houseInfo.
                                getEmergencyInfo("intrusion").
                                getTelephone(), houseInfo.
                                getEmergencyInfo("intrusion").
                                getText());
                }

                retract(event);
end

rule "Timer End Event --> SMS"
        no-loop true
        activation-group "timer"
        salience 0
        when
                event: TimerEvent()
                smsGateway: SMSGateway()
                houseInfo: ZainguneHouseInfo()
                securityService: SecurityService()
        then
                if (!securityService.isSomeoneLogged())
                {
                        smsGateway.sendSms(
                                houseInfo.getSmsInfo().getUsername(),
                                houseInfo.getSmsInfo().getPassword(),
                                houseInfo.getSmsInfo().getTelephone(),
```

```
                                "Intrusion detectada");
                }

                retract(event);
end

rule "Help --> Asterisk"
        no-loop true
        activation-group "help"
        salience 0
        when
                event: HelpEventOSGiImpl()
                asteriskController: AsteriskController()
                houseInfo: ZainguneHouseInfo()
        then
                asteriskController.callAndTalk(
                        houseInfo.getEmergencyInfo("help").
                        getTelephone(), houseInfo.
                        getEmergencyInfo("help").getText());

                retract(event);
end

rule "Brightness Event"
        no-loop true
        when
                event: BrightnessEvent()
                userLocations: ZainguneUserLocations()
                deviceFinder: ZainguneDeviceFinder()
        then
                if (!userLocations.
                        isLocationEmpty(event.getLocation()))
                {
                        if(event.getBrightness() < 150)
                        {
                                incrementLocationsLightBrightness(
                                        event.getLocation(), 40,
                                        deviceFinder);
                        }
                        else if (event.getBrightness() > 200)
                        {
                                incrementLocationsLightBrightness
                                        (event.getLocation(), -40,
                                        deviceFinder);
                        }
                }

                retract(event);
end

rule "EIB Value Changed Event"
        no-loop true
        when
                event: EIBValueChangedEvent()
                smsGateway: SMSGateway()
                houseInfo: ZainguneHouseInfo()
```

```
        then
                if (event.getDevice().getName().
                        equals("Aviso Inundación") && event.getDevice()
                        .getLastValueReaded().equals("On"))
                {
                        smsGateway.sendSms(
                                houseInfo.getSmsInfo().getUsername(),
                                houseInfo.getSmsInfo().getPassword(),
                                houseInfo.getSmsInfo().getTelephone(),
                                "Inundacion");
                }
                retract(event);
end

rule "EIB Server Disconnected Event"
        no-loop true
        when
                event: EIBServerDisconnectedEvent()
        then
                retract(event);
end
```

3.2 DeviceFinder

La interfaz Java DeviceFinder permite buscar dispositivos y servicios a los diferentes componentes existentes en el sistema. Actualmente únicamente es utilizado por el servlet del sistema. Ésta interfaz Java fue creada para poder desacoplar, entre otras cosas, la interfaz gráfica del sistema ya que más adelante esta interfaz será utilizada en otras plataformas.

En cuanto a la implementación realizada en Zaingune (ZainguneDeviceFinder), utiliza un service tracker para conocer cuales son todos los dispositivos existentes en el sistema. De este modo cada vez que aparece o se modifica un servicio este es clasificado en diferentes hashtables para poder realizar las búsquedas de un modo más eficiente. Cuando un servicio es eliminado, este también es eliminado de todas las hashtables correspondientes.

4 Servidor Web del sistema

En este punto se recoge se detalla la implementación del servidor realizada requerida por la interfaz web. Para ello, se van a explicar las partes más importantes de ZainguneSevlet.

ZainguneServlet es un puente entre el ZainguneController y la interfaz gráfica HTML. Para realizar este puente se han definido unos comandos que serán explicados más adelante.

4.1 Comandos del Servlet

El Servlet es el encargado de la comunicación entre la interfaz Web y la plataforma Zaingune. Como antes se ha comentado se han definido una serie de comandos para la comunicación. Estos son enviados mediante el método post de HTML. El nombre del comando se envía mediante el parámetro "command". Los comandos son los siguientes:

- login: permite iniciar sesión en la plataforma Zaingune. Además requiere dos parámetros más: "username" y "password". También devuelve un "session_id".
- logout: permite terminar la sesión en la plataforma Zaingune. Además requiere de un parámetro llamado "session_id" con el valor devuelto por el comando "login".
- execute_method: permite ejecutar métodos Java en servicios y dispositivos de la plataforma Zaingune. Para que un método pueda ser llamado debe contener la anotación RemoteMethod. Un ejemplo sería el siguiente:

```
@RemoteMethod

public void method() { }
```

Opcionalmente se le puede poner un atributo "type" a la anotación con el valor "RemoteMethodType.EXECUTABLE".

Este método únicamente puede contener parámetros de tipos primitivos aunque puede devolver, además de tipos primitivos, los siguientes tipos complejos:

1. Iterable: toda estructura que implemente la interfaz iterable.
2. Enum: enumeraciones Java.
3. Clases propias: estas clases deberán anotar los métodos (getters) que deseen devolver con "RemoteMethodType.VISUALIZABLE". Estos métodos no pueden tener parámetros y pueden devolver cualquiera de los tipos comentados. Ejemplo:

```
public class Clase{

@RemoteMethodType(type=RemoteMethodType.VISUALIZABLE)

public String getName() { return name;}
```

```
}
```

Para llamar a este comando además del parámetro del comando se debe pasar un comando llamado "xml" que debe cumplir el siguiente XML Schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:complexType name="MethodType">
    <xs:sequence>
      <xs:element name="parameter" type="ParameterType"
                    maxOccurs="unbounded" minOccurs="0" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required" />
    <xs:attribute name="service_id" type="xs:long" use="required"/>
  </xs:complexType>

  <xs:element name="method" type="MethodType"/>

  <xs:complexType name="ParameterType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="type" use="required">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="Boolean"/>
              <xs:enumeration value="Byte"/>
              <xs:enumeration value="Double"/>
              <xs:enumeration value="Float"/>
              <xs:enumeration value="Integer"/>
              <xs:enumeration value="Long"/>
              <xs:enumeration value="Short"/>
              <xs:enumeration value="String"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```

La respuesta a este comando es otro XML que cumple el siguiente XML Schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:complexType name="ReturnValueType">
    <xs:choice minOccurs="1" maxOccurs="1">
      <xs:element name="null" type="EmptyType" />
      <xs:element name="void" type="EmptyType" />
      <xs:element name="method_not_executed" type="EmptyType" />
      <xs:element name="value" type="ValueType" />
      <xs:element name="object" type="ObjectType" />
      <xs:element name="enum" type="EnumType" />
      <xs:element name="list" type="ListType" />
    </xs:choice>
  </xs:complexType>
```

```
  <xs:simpleType name="BasicType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Boolean"/>
      <xs:enumeration value="Byte"/>
      <xs:enumeration value="Double"/>
      <xs:enumeration value="Float"/>
      <xs:enumeration value="Integer"/>
      <xs:enumeration value="Long"/>
      <xs:enumeration value="Short"/>
      <xs:enumeration value="String"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="EmptyType" />
  <xs:element name="return_value" type="ReturnValueType" />
  <xs:complexType name="ObjectType">
    <xs:choice maxOccurs="unbounded" minOccurs="1">
      <xs:element name="value" type="ValueType" />
      <xs:element name="object" type="ObjectType" />
      <xs:element name="enum" type="EnumType" />
      <xs:element name="list" type="ListType" />
    </xs:choice>
    <xs:attribute name="name" type="xs:string" use="required" />
    <xs:attribute name="type" type="xs:string" use="required" />
  </xs:complexType>
  <xs:complexType name="ValueType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="type" type="BasicType" use="required" />
        <xs:attribute name="name" type="xs:string" use="required" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="ListType">
    <xs:choice>
      <xs:element name="object" type="ObjectType" minOccurs="0"
                    maxOccurs="unbounded" />
      <xs:element name="enum" type="EnumType" minOccurs="0"
                    maxOccurs="unbounded" />
      <xs:element name="value" type="ValueType" minOccurs="0"
                    maxOccurs="unbounded" />
      <xs:element name="list" type="ListType" minOccurs="0"
                    maxOccurs="unbounded" />
    </xs:choice>
    <xs:attribute name="name" type="xs:string" use="required" />
  </xs:complexType>
  <xs:complexType name="EnumType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="type" type="xs:string" use="required" />
        <xs:attribute name="name" type="xs:string" use="required" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```

- get_cameras_info: devuelve el listado de cámaras existentes en la plataforma Zaingune. Este comando devuelve un XML que cumple el siguiente XML Schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">

  <xs:complexType name="CamerasType">
    <xs:sequence>
      <xs:element name="camera" type="CameraType" minOccurs="0"
                  maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="CameraType">
    <xs:attribute name="id" type="xs:long" />
    <xs:attribute name="name" type="xs:string" />
  </xs:complexType>

  <xs:element name="cameras" type="CamerasType" />
</xs:schema>
```

- get_gui: devuelve la interfaz HTML de un dispositivo de la plataforma Zaingune.

- get_house_info: devuelve toda la información relativa a la casa. Indica las habitaciones existentes en la casa, la situación de éstas y los dispositivos que contiene cada una de ellas, así como los servicios existentes en la casa. La información es devuelta en un XML que cumple el siguiente XML Schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:complexType name="HouseType">
    <xs:sequence>
      <xs:element name="room" type="RoomType" minOccurs="1"
                  maxOccurs="unbounded" />
      <xs:element name="service" type="DeviceType"
                  maxOccurs="unbounded" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="PointType">
    <xs:attribute name="x" type="xs:float" use="required" />
    <xs:attribute name="y" type="xs:float" use="required" />
  </xs:complexType>

  <xs:complexType name="RoomType">
    <xs:sequence>
      <xs:element name="point" type="PointType" minOccurs="4"
                  maxOccurs="unbounded" />
```

```
      <xs:element name="device" type="DeviceType" minOccurs="0"
                  maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required" />
    <xs:attribute name="type" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="living_room"/>
          <xs:enumeration value="dining_room"/>
          <xs:enumeration value="corridor"/>
          <xs:enumeration value="hall"/>
          <xs:enumeration value="kitchen"/>
          <xs:enumeration value="bedroom"/>
          <xs:enumeration value="toilet"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>

  <xs:complexType name="DeviceType">
    <xs:attribute name="id" type="xs:long" use="required" />
    <xs:attribute name="name" type="xs:string" use="required" />
  </xs:complexType>

  <xs:element name="house" type="HouseType" />
</xs:schema>
```

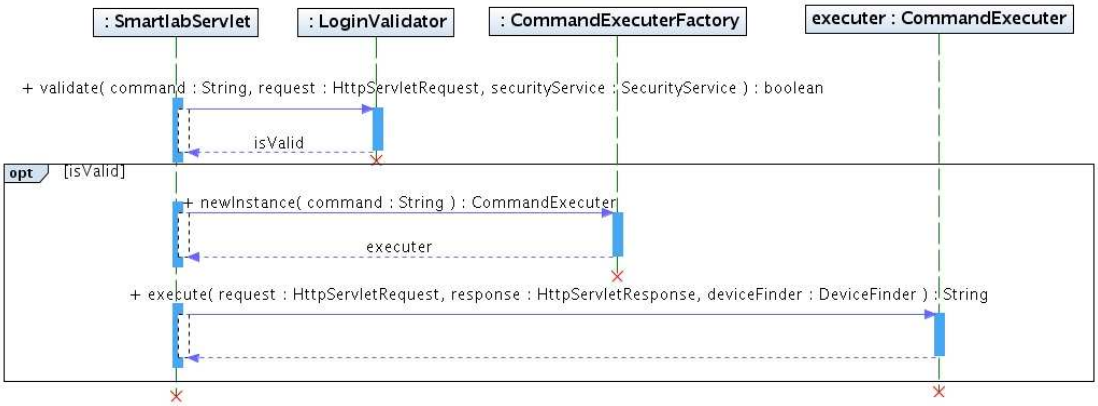
- get_service_id: devuelve el identificador de un servicio mediante el tipo de servicio. Para ello necesita un parámetro llamado "service" que contenga el valor del servicio. Un ejemplo de respuesta sería el siguiente:

```
<service id="17" />
```

4.1.1 Añadir un nuevo comando

Como la plataforma puede extenderse, es posible añadir nuevos comandos al sistema Zaingune. Para ello hay que cumplir ciertos requisitos.

Lo principal es el conocer el funcionamiento del Servlet, explicado por el siguiente diagrama de secuencia:



Como se puede comprobar, primero comprueba si el usuario tiene permiso para ejecutar ese comando. Actualmente el único comando que puede ejecutarse sin haber iniciado sesión es el de "login". Si el usuario tiene permiso, mediante la factoria "CommandExecuterFactory" crea la instancia de "CommandExecuter" concreta y ejecuta el método "execute" de éste.

Por lo tanto, para crear un nuevo comando, se deberá crear una clase que implemente la interfaz "CommandExecuter" y modificar el método "newInstance" de "CommandExecuterFactory" para que cree las nuevas implementaciones de "CommandExecuter".

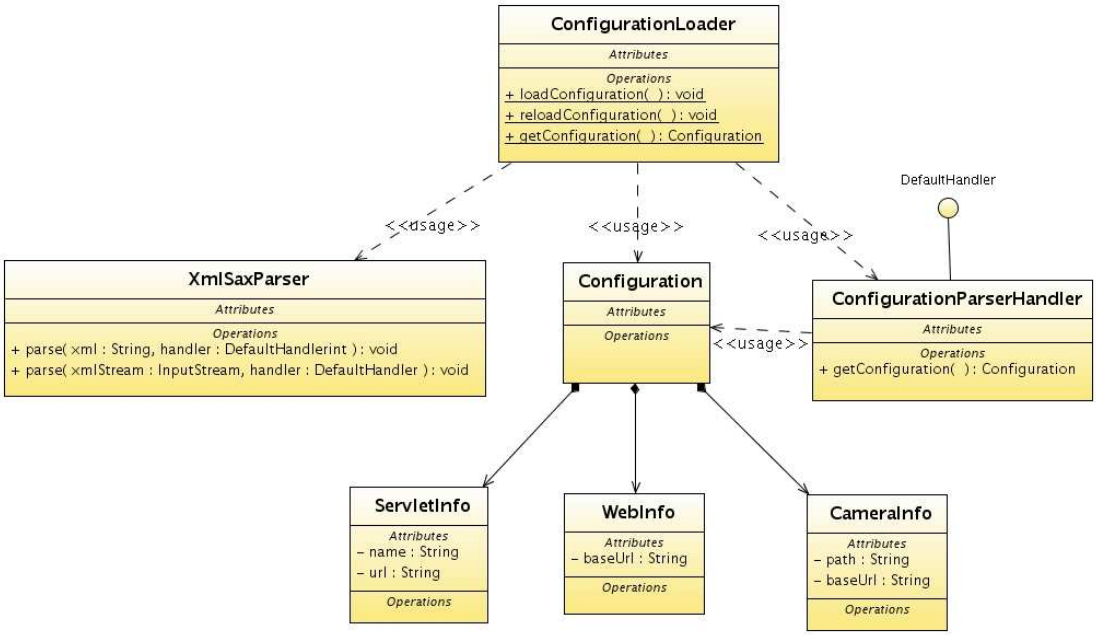
4.2 Configuración

El bundle del Servlet necesita ciertos parámetros de configuración. Para ello utiliza un fichero de configuración XML. Un ejemplo será el siguiente:

```
<configuration>
        <servlet>
                <name>Zaingune Servlet</name>
                <url>/ZainguneServlet</url>
        </servlet>
        <web>
                <base-url>/resources</base-url>
        </web>
        <web-resource>
                <path>file:cameras</path>
                <base-url>/cameras</base-url>
        </web-resource>
        <web-resource>
                <path>file:images</path>
                <base-url>/images</base-url>
        </web-resource>
</configuration>
```

Este XML contiene tres secciones: “servlet”, “web” y “web-resource”. En la primera de éstas, “servlet”, se indica cual es el nombre del servlet, mediante la etiqueta “name”, y en que url va a ser registrado, mediante la etiqueta “url”. En la segunda, “web”, únicamente se define cual va a ser la url de base de los recursos Web (ficheros JavaScript, hojas de estilo, etc.) de los que se dispone. Por último, la etiqueta “web-resource”, contiene dos subelementos, “path”, que indica en dónde se van a almacenar diferentes ficheros (fotos, etc.), y “base-url”, que indica el directorio base en el que se van a publicar dichos ficheros.

Este fichero de configuración es cargado mediante SAX. Para la carga del fichero se puede utilizar la clase ConfigurationLoader. El modelo utilizado para la carga del fichero XML es el siguiente:



4.3 Servidor HTTP

El servlet es registrado en el servidor Web de OSGi. En este se registran todos los recursos Web que van a ser utilizados en la interfaz de usuario HTML.

Ya que por defecto OSGi solo permite registrar rutas internas del bundle sea ha tenido que hacer otra implementación de la interfaz HttpContext (ZainguneHttpContext) para permitir el registro de rutas del sistema de ficheros.

Además también se le ha añadido la posibilidad de aceptar ficheros mediante un formulario, ya que la especificación de OSGi no lo contempla.

5 Interfaz Web

Para la creación de la interfaz de usuario se ha desarrollado una arquitectura de gadgets, donde la ventana principal cuenta con una zona contenedora de pequeñas ventanas o gadgets, que presentan la interfaz de usuario particular de cada dispositivo o elemento conectado al sistema. Este sistema basado en gadgets es totalmente modular, facilitando la creación e incorporación de nuevas interfaces de usuario para el control y/o monitorización de nuevos dispositivos conectados al sistema.

La interfaz de usuario del proyecto Zaingune, está desarrollada con tecnologías web, siguiendo los estándares del World Wide Web Consortium XHTML-1.0-Strict [http://www.w3.org/TR/xhtml1/#a_dtd_XHTML-1.0-Strict], Javascript como lenguaje de scripting y CSS2 [http://www.w3.org/TR/CSS21/]. Por lo tanto, cualquier navegador que respete los estándares del W3C será el único requisito para poder utilizar el sistema.

La creación de un nuevo gadget es sencilla, y únicamente requiere conocimientos de las mencionadas tecnologías de diseño web como xhtml, css y javascript. A continuación se presenta una pequeña guía para los creadores de nuevos gadgets.

5.1 Contenido de un gadget

Cada gadget se carga dentro de un Iframe dentro de la página principal por lo que un documento html totalmente independiente contiene el contendido de cada uno de los gadgtes. Para visualizar su contenido, el sistema carga el documento html del gadget en el iframe correspondiente de la ventana principal.

La interfaz de usuario de cada gadget además del contenido, necesita otros elementos como los estilos correspondientes para dar formato a los contenidos, o el mecanismo de gestión de eventos que permite interactuar al usuario con el sistema, etc. Por tanto, cada gadget estará formado por un conjunto de ficheros:

5.1.1 Documento html

Recoge el contenido del gadget. Este documento se tiene que llamar index.htm. El código de ejemplo que se muestra a continación muestra el contenido del documento html de uno de los gadgets más sencillos, el gadget Thermometer que simplemente visualiza la temperatura de un termómetro:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>SmartLab - Tecológico Fundación Deusto</title>
<link rel='stylesheet' type='text/css'
href='/resources/styles/smartlab/smartlab_gadgets.css'/>
<link rel='stylesheet' type='text/css' href='<%baseUrl%>/thermometer.css'/>
<script type='text/javascript'
src='/resources/scripts/crossbrowser/x_core.js'></script>
<script type='text/javascript'
src='/resources/scripts/crossbrowser/x_event.js'></script>
<script type='text/javascript'
src='/resources/scripts/crossbrowser/xenabledrag.js'></script>
<script type='text/javascript'
src='/resources/scripts/crossbrowser/xgetelementbyid.js'></script>
<script type='text/javascript'
src='/resources/scripts/crossbrowser/xgetelementsbytagname.js'></script>
<script type='text/javascript'
src='/resources/scripts/crossbrowser/xhttprequest.js'></script>
<script type="text/javascript"
src="/resources/scripts/smartlab/gadgets.js"></script>
</head>
<body>
<div class='thermometer-content'>
       <div id="gadgetTitle" class='thermometer-title'> </div>
       <div id="temp-text"> </div>
       <div id='lblError'>
       </div>
</div>
<script type="text/javascript">

var URL="<%servletUrl%>";
var serviceId = "<%serviceId%>";
var title="<%title%>";
var baseUrl="<%baseUrl%>";

</script>

<script type="text/javascript" src="<%baseUrl%>/thermometer.js"></script>
</body>
</html>
```

El html obligatoriamente debe contener los siguientes elementos:

- Referencia a una hoja de estilos general del sistema de gadgets:

```
<link rel='stylesheet' type='text/css'
href='/resources/styles/smartlab/smartlab_gadgets.css'/>
```

- Referencia al fichero de estilos css del gadget:

```
<link rel='stylesheet' type='text/css' href='<%baseUrl%>/thermometer.css'/>
```

- Referencia a librerías javascript utilizadas por la arquitectura de gadgets de Zaingune:

```
<script type='text/javascript'
src='/resources/scripts/crossbrowser/x_core.js'></script>
<script type='text/javascript'
src='/resources/scripts/crossbrowser/x_event.js'></script>
<script type='text/javascript'
src='/resources/scripts/crossbrowser/xenabledrag.js'></script>
<script type='text/javascript'
src='/resources/scripts/crossbrowser/xgetelementbyid.js'></script>
```

```
<script type='text/javascript'
src='/resources/scripts/crossbrowser/xgetelementsbytagname.js'></script>
<script type='text/javascript'
src='/resources/scripts/crossbrowser/xhttprequest.js'></script>
<script type="text/javascript"
src="/smartlab/gadgets/light/debug/scriptaculo/prototype.js"></script>
<script type="text/javascript"
src="/smartlab/gadgets/light/debug/scriptaculo/scriptaculous.js"></script>
<script type="text/javascript"
src="/resources/scripts/smartlab/gadgets.js"></script>
```

• Contenido del gadget. El contenido se recomienda ponerlo dentro de un div en el cuerpo del documento:

```
<div class='thermometer-content'>
...
</div>
```

• Script con los parámetros del gadget. Este pequeño script carga unas variables javascript, que contienen la información particular del dispositivo o elemento particular controlado por el gadget. El ZainguneServlet , antes de enivar el html al navegador sustituye los siguientes códigos: <%servletUrl%>, <%serviceId%>, <%serviceId%>, <%title%>, <%baseUrl%>, por los valores correspondientes al dispositivo controlado:

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

var URL="<%servletUrl%>";
var serviceId = "<%serviceId%>";
var title="<%title%>";
var baseUrl="<%baseUrl%>";

</script>
```

• Referencia al script del gadget, que se explica más adelante:

```
<script type="text/javascript" src="<%baseUrl%>/thermometer.js"></script>
```

5.1.2 Hoja de estilos CSS

La hoja de estilos contiene el formato de todos los elementos contenidos en el gadget. No existe ningún tipo de requisito en cuanto a los estilos, por lo que el diseñador del gadget puede crear todos los estilos que considere necesarios. Se recomienda crear los estilos en un fichero externo con el nombre del gadget. A modo de ejemplo, se muestra la hoja de estilos del gadget Thermometer (thermometer.css):

```
html{
background-color:#000000;
}
.thermometer-content{
	background-color:#000000;
}
#temp-text{
	color:#fb9b29;
	font-size:3em;
```

```
	padding:10px;
	text-align:center;
}
.thermometer-title{
	color:#fb9b29;
	font-size:1.5em;
	padding:10px;
	text-align:center;
}
.thermometer-error
	{
	text-align:center;
	font-size:1em;
	color:#aaa;
	}
```

5.1.3 Script del gadget

Todos los gadgets requieren un script formado por varias funciones que se encarga de recoger y gestionar los eventos realizados por el usuario. Este script necesita conectarse con el servidor del sistema para enviar las peticiones del usuario y para leer periódicamente el estado del dispositivo. Se recomienda crear este script en un fichero externo con el nombre del gadget. A continuación se muestra el script completo del gadget Thermometer (thermoter.js):

```
getTempCall=null;
loadTitle();
getTempValue();

function loadTitle()
{
	document.getElementById("gadgetTitle").innerHTML=title;
}

function setTemp(temp)
{

		document.getElementById("temp-
text").innerHTML=Math.round(parseFloat(temp)*10)/10 + " ºC";
}

function showError(message)
{
	lblError.innerHTML="<p class='thermometer-error'>" + message + "</p>";
}

function getTempValue()
{
	var method;
	var params;
	method='getTemp';
	params = [];
	getTempCall=new Invocacion(URL, serviceId, method, params,
getTempValueResponse,null)
	getTempCall.enviar();
	setTimeout("getTempValue()", 5000);
}

function getTempValueResponse(req, status, thermometer_Object)
```

```
{
       var temperature_Temp;
       if (status == getTempCall.XHR.OK)
       {
              if(req.responseXML.getElementsByTagName("value").length==1)
                     {

       thermometer_Temp=req.responseXML.getElementsByTagName("value")[0].childN
odes[0].data;
                     setTemp(thermometer_Temp);
              }
  }
  else
       {
    if (status & getTempCall.XHR.TIMEOUT)
     showError("Timeout error");

    if (status & getTempCall.XHR.NOXMLCT)
     showError("Noxmlct error");

    if (status & getTempCall.XHR.RSPERR)
      showError();
    }
}
```

Lo primero que tienen que hacer todos los gadgets es visualizar el título a partir de la variable title, cargada en el script de parámeteros ubicado en documento html del gadget. Se recomienda llamar a este método loadTitle. Ejemplo del loadTitle del gadget Thermometer:

```
function loadTitle()
{
       document.getElementById("gadgetTitle").innerHTML=title;
}
```

Otra operación que tienen que hacer todos los gadgets es leer el estado del dispositivo periódicamente. El estado de los gadgets se pueden obtener invocando métodos a la plataforma Zaingune. Los métodos se invocan a través del Servlet, que es el encargado de la comunicación entre la interfaz Web y la plataforma Zaingune. La invocación de métodos se realiza mediante peticiones http al mencionando Servlet. Toda la lógica de las peticiones AJAX se encuentra encapsulada en un objeto Javascript llamado Invocacion, que se encuentra en un .js ya referenciado en el html del gadget (/resources/scripts/smartlab/gadgets.js). Este objeto invocación tiene un contructor con la siguiente sintaxis:

```
new Invocacion(URL, serviceId, method, params, fnCallback, UData)
```

Parámetros:

- URL. String. URL a la que se va a hacer la petición.
- serviceId. Identificativo del dispositivo controlado por el gadget.
- method. String. Método de la plataforma Zaingune que se quiere a invocar.

- params. Array. Parámetros que se van a pasar al método invocado.
- fnCallback. Función javascript que se va a llamar cuando se reciba la respuesta o cuando expire el timeout. Esta función recibe tres parámetros fnCallback(req, status, data);
    - req – El objeto XMLHttpRequest.
    - status – El estado de la petición http.
    - data – el objeto de usuario pasado.
- UData. El objeto de usuario que se pasará a la función de retorno (fnCallback).

El objeto Invocacion también tiene un método sin parámetros llamado enviar que realiza la petición http:

```
enviar()
```

A continuación se muestra como el gadget Thermometer recoge la temperatura periódicamente haciendo uso del mencionado objeto Invocacion:

```
function getTempValue()
{
	var method;
	var params;
	method='getTemp';
	params = [];
	getTempCall=new Invocacion(URL, serviceId, method, params,
getTempValueResponse,null)
	getTempCall.enviar();
	setTimeout("getTempValue()", 5000);
}
```

La función getTempValue lee la temperatura del dispositivo. Como se tiene que leer la temperatura periódicamente, la última instrucción de la función es una llamada a si misma a través de la función setTimeOut de javascript.

En la llamada al constructor del objeto Invocacion, la URL y el serviceId están almacenados en las variables cargadas en el script de parámetros del documento html. El método que hay que invocar en la plataforma Smartalb para obtener la temperatura se llama getTemp. En este caso el método no recibe ningún parámetro por lo que el Array se pasa vacío, pero si el método que se quiere invocar necesita parámetros el argumento params debe ser un Array bidimensional con el tipo y valor de cada uno de los parámetros. Ejemplo:

```
Params=[ ['Integer', 1], ['String', 'on' ]];
```

La función de retorno se llama getTempValueResponse, que será llamada cuando se reciba la respuesta o cuando expire el timeout. Si en la función de retorno fuera necesario referenciar a un elemento del documento html se podría pasar la referencia al elemento a través del parámetro UData. En el método getTempValue no es necesario por lo que se pasa a null.

A continuación se muestra el código de la función de reterno getTempValueResponse que se encarga de recoger la petición:

```
function getTempValueResponse(req, status, thermometer_Object)
{
       var temperature_Temp;
       if (status == getTempCall.XHR.OK)
       {
              if(req.responseXML.getElementsByTagName("value").length==1)
                     {

       thermometer_Temp=req.responseXML.getElementsByTagName("value")[0].childN
odes[0].data;
                     setTemp(thermometer_Temp);
              }
  }
  else
       {
    if (status & getTempCall.XHR.TIMEOUT)
     showError("Timeout error");

    if (status & getTempCall.XHR.NOXMLCT)
     showError("Noxmlct error");

    if (status & getTempCall.XHR.RSPERR)
      showError("Response error");
    }
}
```

Esta función, si el estado de la petición es OK, recoge la temperatura parseando el XML recibido a utilizando XMLDOM. A continuación se llama al método javascript setTemp que lo único que hace es modificar el innerHTML del elemento que visualiza la temperatura.

Si el estado de la petición no es OK, se llama a una función showError que muestra el error correspondiente.

El gadget de ejemplo es muy sencillo, y la información sólo fluye en una dirección: de la plataforma al usuario. Si hubiese sido necesario que el usuario controlase el dispositivo, por ejemplo subiendo o bajando la temperatura, simplemente habría que realizar la Invocación al método correspondiente a través del objeto Invocacion de la misma forma que se hace en el ejemplo mostrado, pero pasando el nombre del método y los parámetros necesarios para que la plataforma Zaingune realice la operación correspondiente sobre el dispositivo.

5.1.4 Otros ficheros.

Si la interfaz de usuario requiere otros recursos como imágenes, u otras librerías javascript que resuelvan una problemática concreta del gadget, simplemente habría que añadir los ficheros correspondientes.

5.2 Internacionalización y Localización

La capa de interacción con el usuario de la plataforma Zaingune está preparada para presentarse en diferentes lenguajes, en función de las necesidades del usuario. A continuación se detallan los pasos que hay seguir para dotar a un gadget la capacidad de multilingüismo:

5.2.1 Crear los ficheros de recursos

Cada gadget debe disponer de uno o más ficheros de recursos que contienen los textos que son dependientes del lenguaje para cada uno de los lenguajes en los que se desea que esté disponible la interfaz de usuario. Estos ficheros se deben llamar de la siguiente manera: lang_[lenguaje].xml, siendo [lenguaje] el código del lenguaje maracado por el estándar ISO 639-1. Por ejemplo, para inglés, el nombre del fichero sería "lang_en.xml", y para español sería "lang_es.xml". Los ficheros de recursos deben cumplir el siguiente XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.org/NewXMLSchema"
xmlns:tns="http://www.example.org/NewXMLSchema"
elementFormDefault="qualified">

    <complexType name="MessagesType">
      <sequence>
            <element name="msg" type="tns:MessageType"></element>
      </sequence>
      <attribute name="lang" type="string" use="required"></attribute>
    </complexType>

    <element name="messages" type="tns:MessagesType"></element>

    <complexType name="MessageType">
      <simpleContent>
            <extension base="string">
                  <attribute name="id" type="string" use="required" />
            </extension>
      </simpleContent>
    </complexType>
</schema>
```

A continuación se muestra a modo de ejemplo, el fichero de recursos para el lenguaje inglés del gadget Surveillance:

```
<?xml version="1.0" encoding="UTF-8"?>
<messages lang="en">
      <msg id="moveUpButton">Move up</msg>
      <msg id="moveLeftButton">Move left</msg>
```

```
        <msg id="homeButton">Home</msg>
        <msg id="moveRightButton">Move right</msg>
        <msg id="moveDownButton">Move down</msg>
        <msg id="backButton">Back</msg>
        <msg id="timeout">Timeout error</msg>
        <msg id="noxmlct">Response content-type is not XML</msg>
        <msg id="rsperr">Response status is not 200</msg>
</messages>
```

5.2.2 Enlazar el fichero de scripts Localization.js

En la cabecera del documento html del gadget hay que hacer referencia a la librería javascript que posibilita la localizacion:

```
<script type="text/javascript"
src="/resources/scripts/smartlab/localization.js"></script>
```

5.2.3 Instanciar el objeto Messages

En el script del gadget, y fuera de toda función para que se ejecute al cargar el gadget se debe instanciar el objeto Messages de la siguiente manera:

```
var m = new Messages();
```

El constructor de este objeto Messages se encarga de acceder al fichero de recursos del lenguaje correspondiente, y reemplazar todos los textos dependiente de lenguaje que estén marcados en el gadget.

5.2.4 Marcar los textos dependientes del lenguaje

El último paso consiste en marcar todos y cada uno de los textos dependientes del lenguaje. Existen dos tipos de textos, los que aparecen de forma estática en el documento html, y los que se generan dinámicamente mediante javascript.

- Textos estáticos contenidos en el documento html. Para marcar estos textos, hay que incluir un código en el lugar del documento html donde se desea que aparezca el correspondiente texto dependiente del leguaje. El código está formado por dos guiones bajos seguidos del el identificativo del mensaje y seguidos de otros dos guiones bajos: __[identificativo mensaje]__. El identificativo del mensaje es el valor del atributo "id" del elemento <msg> correspondiente en los ficheros de recursos. El siguiente ejemplo, asigna al atributo alt de una imagen un texto dependiente del lenguaje.

```
<img id="keyboardButton" src="keyboard.png" alt='__smsSubTitle__'/>
```

- Textos dinámicos generados desde el script. Si se necesita asignar un texto dependiente del lenguaje desde un script, simplemente hay que llamar al método getMessage(messageId), del objeto Message creado en un punto anterior. El

parámetro que recibe esté método es el valor del atributo "id" del elemento <msg> correspondiente en los ficheros de recursos. Ejemplo de llamada al método getMessage:

```
m.getMessage("smsSubTitle")
```

6 Desarrollo de un dispositivo

En este documento se van a explicar que requisitos se deben cumplir para desarrollar dispositivos para la plataforma Zaingune.

El primer requisito que debe cumplir un dispositivo es ser registrado como un servicio OSGi con una propiedad llamada "type". Esta propiedad puede contener dos valores, "device" y "service" que aunque actualmente tienen la misma función, se mantienen ambas por compatibilidad con versiones anteriores (actualmente únicamente se debería utilizar "device"). Añadiendo esta propiedad los dispositivos serán añadidos automáticamente al sistema.

Por otro lado, existen dos interfaces Java. La primera es necesaria para todos los dispositivos (DeviceService), la otra (Displayable) es para los dispositivos que desean ser visualizados en la interfaz gráfica.

6.1 Interfaz DeviceService

La interfaz DeviceService debe ser implementada por todos los dispositivos del sistema (es.deusto.tecnologico.smartlab.devicefinder.model.DeviceService). Esta interfaz contiene los siguientes métodos:

- public String getName(): devuelve el nombre del dispositivo.
- public void setId(Long id): este método es llamado cuando se registra el servicio OSGi. Se le pasa como argumento el identificador del dispositivo.
- public Long getId(): devuelve el identificador que es asignado mediante el método setId.

Esta interfaz es ofrecida por el bundle ZainguneCommons.

6.2 Interfaz Displayable

La interfaz Displayable debe ser implementada por todos los dispositivos que se quieran visualizar en la interfaz gráfica (es.deusto.tecnologico.smartlab.devicefinder.Displayable). Esta interfaz contiene los siguientes métodos:

- public String getDisplayableGui(): devuelve la interfaz HTML del dispositivo. Puede que en un futuro, se le pase un String para poder pedir diferentes tipos de interfaz gráfica.
- public String getDisplayableName(): devuelve un nombre para la interfaz gráfica.
- public Size getDisplayableSize(): devuelve el tamaño de la interfaz HTML. Al igual que en el método getDisplayableGui, en un futuro, se le pasará un String para poder pedir diferentes tipos de interfaz gráfica.

6.2.1 Plantillas HTML

En el método getDisplayableGui para la generación del código HTML se utilizan plantillas. Estas plantillas son código HTML con etiquetas adicionales añadidas que serán reemplazadas en el método mencionado. Estas etiquetas tienen el siguiente formato: "<%nombre_etiqueta%>".

Para facilitar este reemplazo se ha creado una clase de utilidad que lo hace automáticamente (es.deusto.tecnologico.smartlab.util.HtmlPreprocessor). Esta clase contiene los siguientes métodos:

- public static String preprocess(String text, Map<String, String> values)
- public static String preprocess(InputStream is, Map<String, String>) throws IOException

Estos dos métodos devuelven en un String el HTML preprocesado, es decir, cambiando las etiquetas adicionales por un valor. Para realizar esta función se les deben pasar dos parámetros a estas funciones. El primero es el texto a preprocesar, que en el primer método se pasa mediante un String y en el segundo mediante un InputStream. El segundo parámetro es un Map. Las claves de este Map son los nombres de las etiquetas (sin "<%" y "%>") a reemplazar y el valor asociado es el valor por el que se debe reemplazar la etiqueta.

6.3 Ejemplo de creación de dispositivos

En este ejemplo se va a explicar como se desarrollaría un dispositivo para la plataforma Zaingune. En el ejemplo se implementará un dispositivo luz.

Para empezar, y si no ha sido previamente definida, se deberá definir la interfaz que va a implementar el servicio.

```
public interface IBinaryLight
{
        public void setStatus(boolean status);
        public boolean getStatus();
}
```

Una vez definida, esta deberá ser implementada:

```
public class BinaryLight implements IBinaryLight
{
        /* IBinaryLight methods */
        public void setStatus(boolean status)
        {
                if (status)
                        turnOn();
                else turnoff();
        }

        public boolean getStatus()
        {
                return checkStatus();
        }

        /* Private methods */
        ……
}
```

Cuando se ha terminado de implementar el dispositivo, se comenzará con la implementación de la interfaz DeviceService:

```
public class BinaryLight implements IBinaryLight, DeviceService
{
        private String name;
        private Long id;

        /* IBinaryLight methods */
        public void setStatus(boolean status){…}
        public boolean getStatus(){…}

        /* DeviceService methods */
        public String getName()
        { return name; }

        public Long getId()
        { return id; }

        public void setId(Long id)
```

```
        { this.id = id; }

        /* Private methods */
        ……….
}
```

Cuando esta interfaz ha sido definida, ya es posible registrar este dispositivo en la plataforma Zaingune mediante el registro de servicios de OSGi. Se debe añadir la propiedad "type" en el momento del registro. En este caso, como se trata de un dispositivo físico, tiene que tener el valor "device". En caso de haber sido un dispositivo lógico debería tener el valor "service".

Una vez se haya implementada la interfaz DeviceService, es posible implementar la interfaz Displayable. Esta interfaz permite que los dispositivos sean mostrados en la interfaz gráfica. Como la interfaz Displayable extiende la DeviceService, esta última puede ser eliminada de la cabecera del la clase que la implementa. Un ejemplo sería el siguiente:

```
public class BinaryLight implements IBinaryLight, Displayable
{
        private String name;
        private Long id;

        /* IBinaryLight methods */
        public void setStatus(boolean status){…}
        public boolean getStatus(){…}

        /* DeviceService methods */
        public String getName() {…}
        public Long getId() {…}
        public void setId(Long id) {…}

        /* Displayable methods */
        public String getDisplayableGui()
        {
                InputStream is = this.getClass().
                getResourceAsStream("/gadgets/binary/index.htm");

                Hashtable<String, String> values = new
                        Hashtable<String, String>();
                values.put("serviceId", id.toString());
                values.put("title", getName());
                values.put("servlet_url", "/ZainguneServlet");
                values.put("baseUrl", baseUrl);

                try
                {
                        return HtmlPreprocessor.preprocess(is, values);
                }
```

```
                catch (IOException e) {}

                return “”;

        }

        public String getDisplayableName()
        {
                return name;
        }



        public Size getDisplayableSize()
        {
                return new Size(200,150);
        }


        /* Private methods */
        ……
}
```

El método “getDisplayableGui” devuelve la interfaz HTML que será enviada al cliente Web. Para ello utiliza una plantilla (/gadgets/binary/index.htm) sobre la que cambia ciertos valores (llamada método preprocess). La plantilla tiene el siguiente aspecto:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
  <head>
    <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1" />
    <title>SmartLab - Tecnológico Fundación Deusto</title>
    <link rel="stylesheet" type="text/css"
        href="/resources/styles/smartlab/smartlab_gadgets.css"/>
    <link rel="stylesheet" type="text/css"
        href="<%baseUrl%>/light.css"/>
    <script type='text/javascript'
        src='/resources/scripts/crossbrowser/x_core.js'></script>
    <script type='text/javascript'
        src='/resources/scripts/crossbrowser/x_event.js'></script>
    <script type='text/javascript'
        src='/resources/scripts/crossbrowser/xenabledrag.js'></script>
    <script type='text/javascript'
        src='/resources/scripts/crossbrowser/xgetelementbyid.js'>
        </script>
    <script type='text/javascript'
        src='/resources/scripts/crossbrowser/xgetelementsbytagname.js'
        ></script>
```

```
    <script type='text/javascript'
        src='/resources/scripts/crossbrowser/xhttprequest.js'>
        </script>
    <script type="text/javascript"
        src="<%baseUrl%>/scriptaculo/prototype.js"></script>
    <script type="text/javascript"
        src="<%baseUrl%>/scriptaculo/scriptaculous.js"></script>
    <script type="text/javascript"
        src="/resources/scripts/smartlab/gadgets.js"></script>
  </head>
  <body>
    <div class='light-content'>
      <div id='gadgetTitle' class='light-title'>Titulo</div>
      <div id='gd' class='fenster-content' style='text-align:center;'>
        <div id='dc1' class='dc'>
          <div>
            <img id='imgBulb'  src='<%baseUrl%>/bulbon.jpg'
               alt='Bulbon'/>
          </div>
        </div>
        <div id='track1' class='light-track'>
          <div id='handle1' class='light-handle'>
            <img id='botonSlide'
               src='<%baseUrl%>/bola_plata_slider.png'
               alt='Slider button'/>
          </div>
        </div>
        <div id='lblPercent' class='light-percent'>0 %</div>
      </div>
      <div style='text-align:center;'>
        <img id='imgPower' src='<%baseUrl%>/power_on_green.png'
               alt='encender/apagar' onclick='powerOnOff()' />
      </div>
      <div id='lblError'></div>
    </div>

    <script type="text/javascript">
      var URL="<%servletUrl%>";
      var serviceId = "<%serviceId%>";
      var title="<%title%>";
      var baseUrl="<%baseUrl%>";
    </script>

    <script type="text/javascript"
        src="<%baseUrl%>/light.js"></script>
  </body>
</html>
```

Los valores que son cambiados son los que aparecen entre "<%" y "%>" y además son una clave en el Map que es pasado al método "preprocess". Estas etiquetas ("<%" + clave_map + "%>") son reemplazadas con el valor correspondiente a la clave del Map.

Por otro lado, los recursos que van a ser utilizados vía Web deben ser registrados en el servidor Http de OSGi. Un ejemplo sería el siguiente:

```
private void registerHttpResources()
{
  boolean registered = false;
  int tryNumber = 0;

  do
  {
    try
    {
      httpService.registerResources(baseUrl + tryNumber, "/gadgets",
                httpService.createDefaultHttpContext());
      log.log(LogService.LOG_INFO, "Registered resources");
      registered = true;
    }
    catch (NamespaceException e)
    {
      tryNumber++;
    }
  }
  while (!registered);

  baseUrl += tryNumber;
}
```

El método “getDisplayableName” devuelve el nombre que será utilizado en el titulo del gadget y el método “getDisplayableSize” devuelve el tamaño del gadget.

7 Monitorización del sistema

En este punto se va a explicar el sistema de monitorización realizado para el proyecto. Este sistema consta de dos partes: la primera es un bundle de OSGi que permite comprobar si los servicios básicos están funcionando y la segunda, ajena al sistema OSGi, que utiliza este servicio, el servicio de telnet de OSGi y el servidor web de OSGi para comprobar que servicios están funcionando y reiniciarlos en caso de ser necesario.

7.1 Servicio de monitorización interna

Este componente del sistema ofrece un servicio llamada Service Checker que permite conocer cuales de los servicios del sistema no funcionan correctamente y la posibilidad de reiniciarlos.

Para ello utiliza la interfaz ServiceChecker que contiene los siguientes métodos:

- public void checkServices(): Realiza una comprobación de todos los servicios que no estan funcionando.

- public Collection<Service> getStoppedServices(): Devuelve un listado con todos los servicios que se encontraron parados la última vez que fue ejecutado el método *checkServices*.

- public void restartStoppedServices(): intenta reiniciar todos los servicios que están detenidos, para ello reinicia el componente al que pertenece cada uno de los servicios parados. Este método no puede asegurar que todos los servicios sean correctamente reiniciados.

Para que este componente sea capaz de reiniciar los servicios del sistema es necesario realizar una serie de configuraciones. Para ello se utiliza el fichero de configuración "conf.xml" situado en la carpeta "conf". Un ejemplo de fichero es el siguiente:

```
<configuration>
        <services>
                <service interface="es.deusto.tecnologico.zaingune.
                        asterisk.AsteriskController"
                        bundle-symbolic-name="AsteriskController"/>
                <service interface="es.deusto.tecnologico.zaingune.
                        db.ContactDAO"
                        bundle-symbolic-name="DatabaseGateway"/>
                <service interface="es.deusto.tecnologico.zaingune.
                        db.SecurityService"
                        bundle-symbolic-name="DatabaseGateway"/>
                <service interface="es.deusto.tecnologico.zaingune.
                        db.UserDAO"
                        bundle-symbolic-name="DatabaseGateway"/>
                <service interface="es.deusto.tecnologico.zaingune.
                        eib.EIBController"
                        bundle-symbolic-name="zaingune.EIBController"/>
                <service interface="es.deusto.tecnologico.zaingune.
                        email.EmailController"
                        bundle-symbolic-name="EmailController"/>
                <service interface="es.deusto.tecnologico.zaingune.
                        help.HelpService"
                        bundle-symbolic-name="HelpService"/>
                <service interface="es.deusto.tecnologico.zaingune.
                        rfid.RFIDController"
                        bundle-symbolic-name="RFIDController"/>
                <service interface="es.deusto.tecnologico.zaingune.
                        sms.SMSGateway"
                        bundle-symbolic-name="SMSGateway"/>
                <service interface="es.deusto.tecnologico.zaingune.
                        videoip.VideoIPController" bundle-symbolic-
                        name="zaingune.VideoIPController"/>
                <service interface="es.deusto.tecnologico.smartlab.
                        devicefinder.DeviceFinder"
```

```
                        bundle-symbolic-name="ZainguneController"/>
        </services>
        <checkers>
                <checker class="es.deusto.tecnologico.zaingune.
                        eib.checker.EibChecker"/>
                <checker class="es.deusto.tecnologico.zaingune.
                        videoip.checker.CameraChecker"/>

        </checkers>
</configuration>
```

En este fichero de configuración se pueden observar dos apartados: "services" y "checkers".

El apartado "services" contiene un listado de "service". Cada uno de estos servicios contiene dos parámetros, la interfaz que cumple el servicio y con la que es registrado en el sistema, y el nombre de bundle al que pertenece, para poder reiniciar el bundle correspondiente en caso de que el servicio no este en funcionamiento. Todos los "service" deben ser únicos en el sistema.

El apartado "checkers" contiene un listado de "checker". Los "checker" son utilizados para comprobar servicios que son previamente desconocidos o con un número de servicios desconocidos. Cada uno de estos "checker" tienen un atributo "class" que indica la clase que debe ser utilizada para comprobar dichos servicios. Esta clase debe implementar la interfaz *ServiceChecker* previamente explicada.

7.2 Servicio de monitorización externa

El servicio de monitorización externa esta compuesto por cuatro scripts desarrollados en Python.

El primero de ellos permite parar y arrancar Zaingune. Para parar el sistema intenta conectarse por Telnet al servidor y apagarlo limpiamente. Si no consigue conectarse finalizará el proceso. Para arrancarlo ejecuta el comando necesario para ejecutarlo. El código es el siguiente:

```
import getpass
import os
import sys
import telnetlib
import socket


def shutdownZaingune(host, port, user, password):
      try:
            tn = telnetlib.Telnet(host,port)
```

```
            tn.read_until("login: ")
            tn.write(user + "\r\n")

            if password:
                  tn.read_until("password: ")
                  tn.write(password + "\r\n")

            loginIncorrect = False;
            line = ""
            while (not line.startswith("'quit'")) &
                        (not line.startswith("Login incorrect")):
                  line = tn.read_until("\n")

            if line.startswith("'quit'"):
                  tn.write("shutdown\r\n")
            else:
                  os.system('kill-zaingune')
                  return True

            tn.read_all()
      except socket.error, msg:
            os.system('kill-zaingune')
            return True

      return True

def startZaingune():
      os.system("run-zaingune")
```

El segundo de estos scripts comprueba si el servidor web de OSGi esta en funcionamiento. El código es el siguiente:

```
import httplib
import sys
import socket

# return == 0 -> server running
# return == 1 -> server not running or incorrect url

def checkWeb(host, port, url):
      try:
            httpclient = httplib.HTTP(host,port)
            httpclient.putrequest("GET", url)
            httpclient.putheader("Host", host)
            httpclient.putheader("User-agent", "python-httplib")
            httpclient.endheaders()

            returncode, returnmsg, headers = httpclient.getreply()
            if returncode != 200:
                  return 1
            else:
```

```
                  return 0
      except socket.error, msg:
            return 1
```

El tercer script se encarga de comprobar, utilizado la monitorización interna mediante el servidor web, el estado de los servicios del sistema. Para ello necesita hacer login en el sistema, obtener el id del servicio *ServiceChecker* y llamar a sus métodos para obtener la información necesaria. El código del script es el siguiente:

```
# exit == 0 -> Zaingune correctly running, all services up
# exit == 1 -> Cannot connect to the web server
# exit == 2 -> Incorrect login
# exit == 3 -> ServiceChecker not running
# exit == 4 -> Cannot restart all the Zaingune services


import httplib
import urllib
import sys
import re
import socket

def weblogin(httpclient, url, headers, username, password):
      params = {'command':'login', 'username':username,
            'password':password}
      httpclient.request('POST', url,
            urllib.urlencode(params), headers)

      response = httpclient.getresponse()

      cookie = response.getheader('set-cookie')
      headers['Cookie'] = cookie.split(";")[0]

      return response.read().find("unsuccessful_login") < 0

def getServiceCheckerId(httpclient, headers, url):
      params = {'command':'get_service_id','service':'ServiceChecker'}
      httpclient.request('POST', url,
            urllib.urlencode(params), headers)
      xmlresponse = httpclient.getresponse().read()

      pattern = re.compile('id="[0-9]*"')
      tokens = pattern.findall(xmlresponse)
      if len(tokens) > 0:
            return int(tokens[0].split('"')[1])
      else:
            return -1

def getStoppedServices(httpclient, headers, url, checkerid):
      params = {'command':'execute_method',
```

```
            'xml':'<method name="getStoppedServices" service_id="' +
            str(checkerid) +'"/>'}
      httpclient.request('POST', url,
            urllib.urlencode(params), headers)
      xmlresponse = httpclient.getresponse().read()

      pattern = re.compile('<value.*>.*</value>')
      values = pattern.findall(xmlresponse)
      stoppedServices = []

      for value in values:
            stoppedServices.append(value[value.index('>') +
                  1:value.index('</')])

      return stoppedServices

def restartStoppedServices(httpclient, headers, url, checkerid):
      params = {'command':'execute_method',
            'xml':'<method name="restartStoppedServices" service_id="'
            + str(checkerid) +'"/>'}
      httpclient.request('POST', url,
            urllib.urlencode(params), headers)
      httpclient.getresponse()


def checkServices(host, port, username, password):
      url = "/ZainguneServlet"
      try:
            httpclient = httplib.HTTPConnection(host,port)

            headers = {'Content-type':
                  'application/x-www-form-urlencoded'}

            logged = weblogin(httpclient, url, headers,
                  username, password)

            if logged:
                  checkerid = getServiceCheckerId(httpclient,
                        headers, url)
                  if checkerid > 0:
                        stoppedServices = getStoppedServices(
                              httpclient, headers, url, checkerid)

                        if len(stoppedServices) > 0:
                              restartStoppedServices(httpclient,
                                    headers, url, checkerid)
                              stoppedServices = getStoppedServices(
                                    httpclient, headers, url,
                                    checkerid)
                              if len(stoppedServices) > 0:
                                    return 4
                  else:
                        return 3

            else:
                  return 2
```

```
      except socket.error, msg:
            return 1

      return 0
```

Por último queda el script que une los primeros tres scripts. El código es el siguiente:

```
#!/usr/bin/python

import asterisk
import commands
import serviceCheck
import osgiUtils
import web

host = "20.0.0.37"

telnetPort = 2323
telnetUsername = "zaingune"
telnetPassword = "smartlab"

webPort = 8080
webLoginUrl = "/resources/login.htm"
webUsername = "xabier"
webPassword = "laiseca"

webReturn = web.checkWeb(host, webPort, webLoginUrl)

checkServicesReturn = serviceCheck.checkServices(host, webPort,
      webUsername, webPassword)

if (checkServicesReturn != 0) | (webReturn !=0):
      succeded = osgiUtils.shutdownZaingune(host, telnetPort,
telnetUsername, telnetPassword)
      if not succeded:
            print "Cannot stop Zaingune"
      else:
            osgiUtils.startZaingune()
```

Este último script debe ser configurado mediante un sistema tipo cron para ser ejecutado automáticamente.