EUSKO JAURLARITZA
GOBIERNO VASCO

Programa Saiotek 2006

SMARTLAB

Entorno de Trabajo Inteligente Colaborativo y Programable

Arquitectura Bundle Web Interface Gateway

Tecnológico Fundación Deusto
Teknologikoa Deustu Fundazioa

HISTORIAL DE CAMBIOS

Versión	Descripción	Autor	Fecha	Comentarios
V0.1	Versión inicial	Xabier Laiseka	17/12/2007	

TABLA DE CONTENIDOS

1 INTRODUCCIÓN

En este documento se recoge el manual de programador del bundle SmartLabServlet. En éste documento se van a explicar las partes más importantes de SmartLabSevlet.

SmartLabServlet es un puente entre el conjunto de servicios avanzados presentes en la capa 3 y la interfaz gráfica HTML. Para realizar este puente se han definido unos comandos que serán explicados más adelante.

1.1 Comandos del Servlet

El Servlet es el encargado de la comunicación entre la interfaz Web y la plataforma SmartLab. 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 SmartLab. Además requiere dos parámetros más: “username” y “password”. También devuelve un “session_id”.
- logout: permite terminar la sesión en la plataforma SmartLab. 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 SmartLab. 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 SmartLab. 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 SmartLab.

- 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:

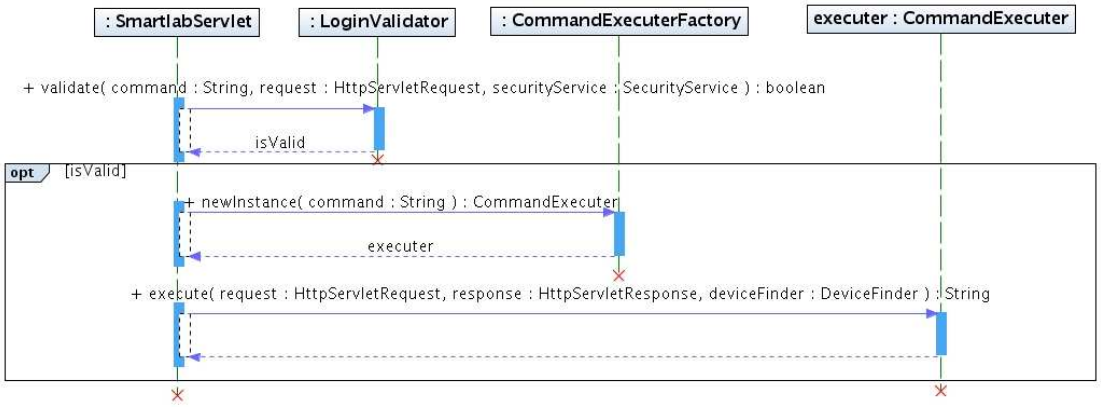
```
<service id=”17” />
```

1.2 Añadir un nuevo comando

Como la plataforma puede extenderse, es posible añadir nuevos comandos al sistema

SmartLab. 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”.

1.3 Configuración

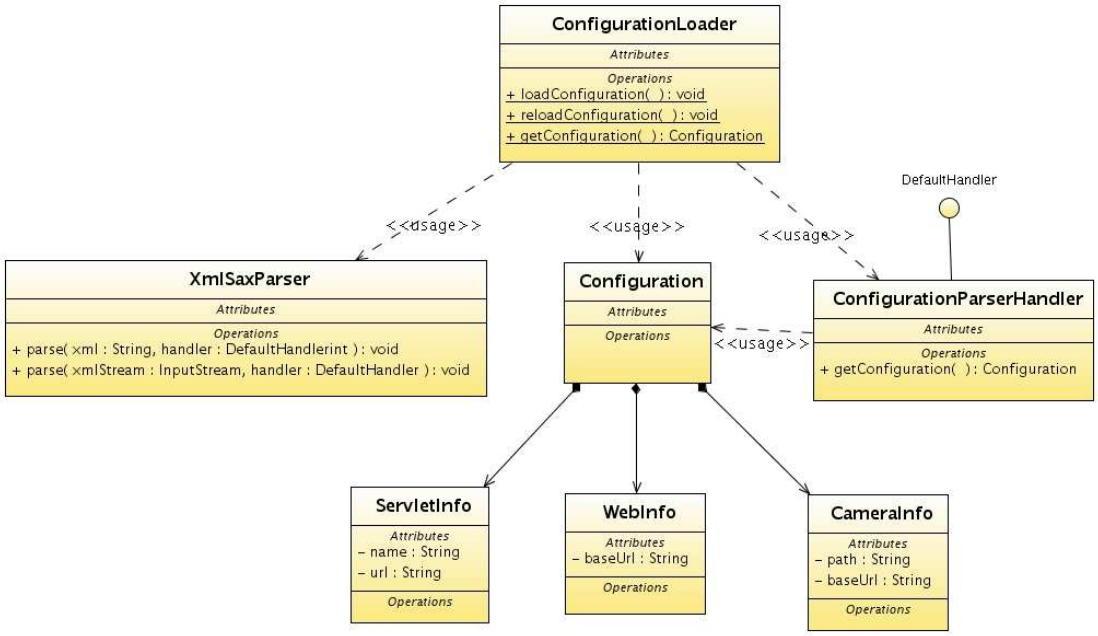
El bundle del Servlet necesita ciertos parámetros de configuración. Para ello utiliza un fichero de configuración XML. Un ejemplo será el siguiente:

```
<configuration>
	<servlet>
		<name>SmartLab Servlet</name>
		<url>/SmartLabServlet</url>
	</servlet>
	<web>
		<base-url>/resources</base-url>
	</web>
	<camera>
		<path>file:cameras</path>
		<base-url>/camera</base-url>
	</camera>
</configuration>
```

Este XML contiene tres secciones: “servlet”, “web” y “camera”. 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 “camera”, contiene dos subelementos, “path”, que indica en dónde se van a almacenar las imágenes capturadas por las cámaras, y “base-url”, que indica el directorio base en el que se van a publicar dichas capturas.

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:



2 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.