



Promoción del desarrollo de SW libre en un entorno de
calidad y confianza adaptando las metodologías, procesos,
modelos de negocio y últimas tecnologías

TSI-020301-2008-22

Entregable D3.5

D3.5 Adecuación de las herramientas a la arquitectura

Instituto Tecnológico de Informática
ATOS Origin

Fecha límite del entregable: 31/12/2008

Fecha de entrega: 31/12/2008

Este trabajo se licencia bajo Creative Commons Attribution-Share Alike 3.0.
Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-sa/3.0/> o
envíe una carta a Creative Commons, 171 Second Street, Suite 300, San Francisco, California,
94105, USA.

Este trabajo está parcialmente financiado por el Ministerio de Industria, Turismo y Comercio
español.

Historial de Cambios

Versión	Fecha	Estado	Autor (Partner)	Descripción
1.0	22/02/2008	Borrador	Pascual L. González Francisco Molerón	
1.1	29/02/2008	Borrador	Pascual L. González Francisco Molerón	Correcciones sobre listado de proyectos
1,2	29/03/2009	Borrador	Manuel Bollaín (UPM) David Musat(UPM)	Nuevas herramientas añadidas
1.3	30/03/2009	Versión final	Maria Alejandra Trujillo (ATOS)	Nuevas herramientas añadidas

RESUMEN EJECUTIVO

En el presente documento se va a describir como se pretende adaptar las diferentes aplicaciones dentro de la forja del proyecto. Como la forja está basada en en la tecnología EzWeb, se definirá una capa REST que encapsulará las funcionalidades de las aplicaciones y se describirán los gadgets necesarios para que la forja de Vulcano tenga la funcionalidad deseada.

Para cada gadget se dará una descripción, cual será el API REST asociada, introduciendo el concepto de recurso, cómo será su interfaz de usuario, cómo se comunicará con el resto de gadgets y por último cual será su implementación.

Información del Documento

Proyecto TSI Número	TSI-020301-2008-22	Acrónimo	Vulcano
Título completo	Promoción del desarrollo de SW libre en un entorno de calidad y confianza adaptando las metodologías, procesos, modelos de negocio y últimas tecnologías		
URL	http://www.ines.org.es/vulcano		
URL del documento			

Entregable	Número	D3.5	Título
Paquete de Trabajo	Número		Título
Tarea	Número		Título

Fecha de Entrega	Contractual	31/12/2008	Entregado	31/12/2008
Estado	Versión 1, fecha 29/02/2008		final <input type="checkbox"/>	
Tipo	Informe <input type="checkbox"/> Demo <input type="checkbox"/> Otro <input checked="" type="checkbox"/>			
Nivel de Diseminación	Público <input checked="" type="checkbox"/> Consorcio <input type="checkbox"/>			
Resumen (para diseminación)				
Palabras Clave				

Autores (Partner)	José Nieto Rubio(ATOS), Pascual L. González (ITI), Francisco Molerón (ITI), David Musat(UPM), Manuel Bollaín(UPM), Maria Alejandra Trujillo(ATOS)			
Responsable de Autoría	José Nieto Rubio		Email	jose.nietor@atosresearch.eu
	Partner	Atos Origin	Tfno	

TABLA DE CONTENIDOS

RESUMEN EJECUTIVO.....	3
TABLA DE CONTENIDOS.....	5
1 INTRODUCCIÓN.....	7
1.1 REST.....	8
1.2 El Gadget.....	9
2 ADECUACIÓN DE DOTPROJECT.....	11
2.1 Descripción.....	11
2.2 API REST.....	12
2.3 Gadget Lista De Tareas.....	12
2.3.1 Descripción.....	12
2.3.2 Interfaz de usuario.....	14
2.3.3 Comunicación.....	14
2.3.4 Implementación.....	15
3 ADECUACIÓN DE PROJECT OPEN.....	16
3.1 Descripción.....	16
3.2 API REST.....	16
3.3 Gadget.....	16
3.3.1 Descripción.....	16
3.3.2 Interfaz de usuario	16
3.3.3 Comunicación	16
3.3.4 Implementación.....	16
4 ADECUACIÓN DE BUGZILLA.....	17
4.1 Descripción.....	17
4.2 API REST.....	17
4.3 Gadgets ListaDeIncidencias.....	19
4.3.1 Descripción.....	19
4.3.2 Interfaz de usuario.....	19
4.3.3 Comunicación.....	19
4.3.4 Implementación.....	20
5 ADECUACIÓN DE LA MIB DE TOPEN	21
5.1 Descripción.....	21
5.2 Gadget de la MIB.....	21
5.2.1 Descripción e interfaz.....	21
5.2.2 Comunicación	21

5.2.3 Implementación.....	22
6 ADECUACIÓN DE WAPITI.....	23
6.1 Descripción.....	23
6.2 API REST.....	23
6.3 Gadget.....	23
6.3.1 Descripción.....	23
6.3.2 Interfaz de usuario	23
6.3.3 Comunicación	23
6.3.4 Implementación.....	23
7 ESQUEMA DE COMUNICACIÓN ENTRE GADGETS.....	24

1 INTRODUCCIÓN

En este documento se describe cómo se van a adaptar las diferentes herramientas a la forja de Vulcano.

dotProject es una aplicación accesible vía web para la gestión de proyectos. Aunque dotProject ofrece muchas funcionalidades nos vamos a centrar en las relacionadas con los usuarios, proyectos y tareas.

Project Open es una aplicación para la gestión de proyectos, totalmente Web.

Bugzilla es una aplicación web para la gestión de errores. En este caso vamos a centrar la información sobre usuarios, proyectos e incidencias.

Topen es un entorno para la validación, monitorización y operación de sistemas intensivos en software de forma remota

Wapiti es un escáner de vulnerabilidades de código abierto que realiza pruebas de caja negra de forma automática.

En este documento se detalla qué parte de la funcionalidad total de las aplicaciones implicadas se va a adaptar. Además, dado que la forja de Vulcano está basada en EzWeb, se define la capa REST que va a encapsular dicha funcionalidad para cada una de las aplicaciones, y por último, se describe cada uno de los gadgets necesarios para dotar a la forja de Vulcano de la funcionalidad deseada.

Se debe tener en cuenta que para llevar a cabo esta tarea vamos a suponer la existencia de dos gadgets dentro de la forja, que podrían llamarse **ListaDeUsuarios** y **ListaDeTareas**.

El gadget **ListaDeUsuarios** contiene, de forma centralizada, la información de los usuarios de la forja. La idea es que la información de los usuarios de la forja esté en un único lugar y sea compartida por el resto de gadgets. Este gadget, por tanto, tiene la capacidad de comunicarse y compartir su información con el resto. En sucesivos apartados se describe un poco más esta comunicación.

El gadget **ListaDeProyectos** contiene, de forma centralizada, la información de los proyectos de la forja. La idea es que la información de los proyectos de la forja, igual que ocurre con la información de usuarios, esté en un único lugar y sea compartida por el resto de gadgets. Este gadget, por tanto, tiene la capacidad de comunicarse y compartir su información con el resto. En sucesivos apartados se describe un poco más esta comunicación.

Y por último, se debe tener en cuenta es que se asume la existencia de un sistema de autenticación de usuarios y control de permisos que permite identificar a los usuarios y establecer qué cosas puede visualizar cada usuario. Este asunto no entra dentro del objetivo de éste entregable.

1.1 REST

REST es una técnica de arquitectura de software para sistemas hipermedia distribuidos como la *World Wide Web*. REST se basa en los mismos principios que han propiciado el éxito de la web:

- Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan *cookies* y otros mecanismos para mantener el estado de la sesión (algunas de estas prácticas, como la reescritura de URLs, no son permitidas por REST)
- Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE. Con frecuencia estas operaciones se equiparan a las operaciones CRUD que se requieren para la persistencia de datos, aunque POST no encaja exactamente en este esquema.
- Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI
- El uso de hipermedios, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente HTML o XML. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

El elemento principal dentro de la arquitectura REST es el recurso. Un recurso es un elemento de información que puede ser accedido utilizando un identificador global (un Identificador Uniforme de Recurso) Para manipular estos recursos, los componentes de la red (clientes y servidores) se comunican a través del protocolo HTTP (GET, POST, PUT y DELETE) e intercambian representaciones de estos recursos.

Añadir una capa REST a una aplicación implica detectar qué funciones o funcionalidades tiene la aplicación y traducirlas a recursos y métodos HTTP sobre éstos recursos.

Esta capa REST proporciona una interfaz sencilla y uniforme al resto de aplicaciones de forma que se consigue un servicio que facilita su consumo por parte de cualquier gadget.

1.2 El Gadget

Teniendo en cuenta el concepto de recurso dentro de la arquitectura REST se define el concepto de gadget como un tipo de recurso que trata con funciones de presentación dentro del entorno en el cual se ejecuta. Dentro de nuestro entorno (el de EzWeb), los gadgets se dividen en dos bloques: un *template* o plantilla y el propio código o implementación del gadget.

La plantilla o *template* es un documento XML con una estructura establecida que define cómo se comporta el gadget con el entorno (usuario, plataforma, otros gadgets, etc).

El código o implementación es un fichero HTML que define lo que hace el gadget haciendo uso de los elementos definidos en su *template*. En él se puede utilizar cualquier tecnología soportada por los navegadores Web (*javascript*, Flash, SVG).

Los gadgets se conectan mediante la capa REST a cada uno de los recursos que forman parte de ella para formar la interfaz gráfica de usuario, a través de la cual, el usuario final puede acceder desde la forja a la funcionalidades que posee cada una de las herramientas

A continuación se muestra un ejemplo de la plantilla de un gadget "localizador" extraída de la documentación de EzWeb:

```
<Catalog.ResourceDescription>
  <Vendor>Morfeo</Vendor>
  <Name>Localizador</Name>
  <Version>1.0</Version>
  <Author>Imayllon</Author>
  <Mail>Imayllon@tid.es</Mail>
  <Description>
    Este gadget presenta la ubicacion en el plano de una determinada dirección
  </Description>
  <ImageURI>
    http://ezweb.hi.inet/gadgets/img/brujula.gif
  </ImageURI>
  <WikiURI>
    http://ezweb.hi.inet/wiki/Localizador
  </WikiURI>
</Catalog.ResourceDescription>
<Platform.Preferences>
  <Preference name="bg" type="text" description="descripcion" label="background color"/>
  <Preference name="textcolor" type="text" description="descripcion" label="text color"/>
</Platform.Preferences>
```

```
</Platform.Preferences>
<Platform.StateProperties>
  <Property name="nota" type="text" label="nota"/>
</Platform.StateProperties>
<Platform.Wiring>
  <Slot name="serviceHired" type="text" label="ServiceHired" friendcode="service"/>
  <Event name="deviceId" type="text" label="deviceId" friendcode="deviceId"/>
  <Event name="deviceStatus" type="text" label="deviceStatus" friendcode="deviceStatus"/>
</Platform.Wiring>
<Platform.Context>
  <Context name="user" type="text" concept="user_name"/>
  <Context name="language" type="text" concept="language"/>
  <GadgetContext name="height" type="text" concept="height"/>
  <GadgetContext name="width" type="text" concept="width"/>
</Platform.Context>
<Platform.Link>
  <XHTML href="http://ezweb.morfeo-project.org/gadgets/myGadgetCode.html"/>
</Platform.Link>
<Platform.Rendering width="1" height="11"/>
```

Destacan las siguientes etiquetas:

- **Catalog.ResourceDescription** : en ella se describen las características del gadget.
- **Platform.Preferences**: aquí se definen las preferencias de usuario del gadget.
- **Platform.StateProperties**: en este bloque se definen variables de estado del gadget.
- **Platform.Wiring**: define las variables que utiliza el gadget para comunicarse con otros gadgets.
- **Platform.Context**: aquí se definen las variables de contexto del gadget.

2 ADECUACIÓN DE DOTPROJECT

2.1 Descripción

DotProject es una aplicación web para la gestión de proyectos. De todas las funcionalidades que ofrece dotProject las que nos interesa adaptar a la forja de Vulcano son las relacionadas con la información de usuarios, proyectos y de tareas. En cuanto a los proyectos nos interesa la información de los proyectos existentes en el sistema y como se relacionan con los usuarios. Esta relación, en dotProject se define de la siguiente forma: un usuario está relacionado con un proyecto si el usuario está asignado a alguna tarea del proyecto. En cuanto a las tareas, nos interesa la información de las tareas almacenadas en el sistema y como se relacionan con los usuarios. Esta relación, dotProject la establece de la siguiente forma: se asignan usuarios a cada tarea.

Se debe tener en cuenta que dotProject contiene su propia información de usuarios del sistema. Esto entra en conflicto con la definición centralizada de usuarios dentro de la forja de Vulcano que se desprende de la existencia del gadget **ListaDeUsuarios**. Es decir, por un lado tenemos la información de usuarios de dotProject y por otro la información de usuarios dentro de la forja. Esto implica que debe haber una sincronización de dicha información. Es decir, todos los usuarios definidos dentro de la forja de Vulcano deben tener su usuario correspondiente dentro de dotProject. En este documento no se va a tratar este problema y se asume que ambas fuentes de usuarios están sincronizadas, de forma que, el identificador de un usuario en una de ellas identifica al mismo usuario en la otra.

Además dotProject contiene su propia información de proyectos del sistema. Esto entra en conflicto con la definición centralizada de proyectos dentro de la forja de Vulcano que se desprende de la existencia del gadget **ListaDeProyectos**. Es decir, por un lado tenemos la información de proyectos de dotProject y por otro la información de proyectos dentro de la forja. Esto implica que debe haber una sincronización de dicha información. Es decir, todos los proyectos definidos dentro de la forja de Vulcano deben tener su proyecto correspondiente dentro de dotProject. En este documento no se va a tratar este problema y se asume que ambas fuentes de usuarios están sincronizadas, de forma que, el identificador de un proyecto en una de ellas identifica al mismo proyecto en la otra.

2.2 API REST

A partir de las funcionalidades descritas anteriormente se definen los siguientes métodos de negocio o acciones que debe satisfacer la interfaz REST:

Función	Descripción
ObtenerTareasUsuarioPorProyecto	Obtiene el listado de tareas asociadas a un usuario y proyecto concreto.

Tabla 1: Funciones de dotProject

A continuación se especifica cada una de las direcciones que forman parte de la API REST de dotProject:

Método	URI	Función
GET	/usuario_id/proyectos/proyecto_id/tareas	obtenerTareasUsuarioPorProyecto

Tabla 2: API REST

Para usar esta nueva API lo único que se debe hacer es realizar las llamadas HTTP oportunas definidas en la lista anterior.

Para el caso de querer obtener el listado de tareas de un proyecto a las que está asignado cierto usuario. Es decir, se debe hacer una llamada GET a la URL indicada en el API REST. El servicio devuelve el listado de tareas en formato XML, para facilitar el manejo de dicha información. Para cada tarea el servicio proporciona los siguientes datos: código de tarea y nombre de tarea.

2.3 Gadget Lista De Tareas

2.3.1 Descripción

Este gadget muestra, mediante el establecimiento de las comunicaciones adecuadas, un listado de tareas correspondientes a un proyecto dado y asignadas a un usuario concreto.

Para hacer posible este comportamiento, se asume la existencia de los gadget **ListaDeUsuarios** y **ListaDeProyectos**. El gadget **ListaDeUsuarios** muestra un listado de usuarios en el que al pulsar sobre uno de ellos “envía” al gadget **ListaDeTareas** la información del usuario seleccionado. El gadget

ListaDeProyectos muestra un listado de proyectos en el que al pulsar sobre uno de ellos “envía” al gadget **ListaDeTareas** la información del proyecto seleccionado.

Una vez obtiene ambos datos, mediante la capa REST definida para dotProject, es capaz de acceder a las tareas del proyecto seleccionado a las que el usuario seleccionado está asociado. Una vez el gadget obtiene esta información, en formato XML, la transforma para componer la interfaz de usuario que muestra estos mismos datos de forma amena para el usuario final.

2.3.2 Interfaz de usuario

La interfaz de usuario del gadget está formada por una tabla donde se lista la información de cada tarea. Para cada tarea, el gadget muestra el código y el nombre de la tarea. Además, si el usuario pulsa sobre el código o nombre de la tarea, la información que identifica la tarea que ha sido seleccionada “es enviada” a la arquitectura para que otros gadgets configurados debidamente hagan uso de ella.

2.3.3 Comunicación

Este gadget, en principio, se comunica con otros tres: los supuestos gadgets **ListaDeUsuarios** y **ListaDeProyectos**, y el gadget **ListaDeIncidencias** que se describe en apartados posteriores.

La comunicación con el gadget **ListadoDeUsuarios** se establece cuando se selecciona un usuario concreto del listado. En este momento la información de qué usuario ha sido seleccionado se “envía al gadget”. El gadget utiliza este dato además del dato del proyecto seleccionado para completar su función.

En términos de plantillas de gadgets esto implica que, este gadget, en el apartado de Wiring de su plantilla, define una variable de solo lectura o SLOT que podría llamarse *usuario_id*, que establece el identificador del usuario seleccionado en el supuesto gadget de **ListaDeUsuarios**. Se puede decir que el gadget espera la llegada de la información del usuario en su entrada.

La comunicación con el gadget **ListadoDeProyectos** se establece cuando el usuario selecciona un proyecto concreto del listado de proyectos de este gadget. En ese momento la información de qué proyecto ha sido seleccionado se “envía al gadget”. El gadget utiliza este dato junto con el dato que indica qué usuario ha sido seleccionado para completar su función.

En términos de plantillas de gadgets esto implica que, este gadget, en el apartado de Wiring de su plantilla, define una variable de solo lectura o SLOT que podrían llamarse *proyecto_id*, que establece el identificador del proyecto seleccionado en el gadget de **ListaDeProyectos**. Se puede decir que el gadget espera la llegada de la información del proyecto en su entrada.

La comunicación con el gadget de **ListadoDeIncidencias** se establece cuando el usuario selecciona una tarea concreta del listado de tareas del propio gadget. En ese momento, la información de qué tarea ha sido seleccionada se actualiza dentro de la arquitectura, de manera que los gadgets que utilizan esta información se dan cuenta de la nueva selección.

En términos de plantillas de gadgets esto implica que, este gadget, en el apartado de Wiring de su plantilla, define una variable de lectura y escritura o EVENT que podría llamarse *tarea_id*, que establece el identificador de la tarea

seleccionada. Se puede decir que el gadget exporta u ofrece a su salida la información de qué tarea ha sido seleccionada.

La Ilustración 1: Esquema de comunicación entre gadgets muestra un esquema de comunicación entre los gadget de la forja donde se puede ver como se comunica este gadget con el resto.

2.3.4 Implementación

En lo que se refiere al código, el gadget, en algún momento hace una llamada al servicio de dotProject. Es decir, accede a una de las URI definidas en el API de la capa REST que encapsula las funcionalidades de dotProject que necesitamos. En concreto, cuando el gadget encuentra en su entrada los datos *usuario_id* y *proyecto_id*, que indican que un usuario y un proyecto han sido seleccionados, compone la URI de la llamada **GET /{usuario_id}/proyectos/{proyecto_id}/tareas** definida en el API para tal propósito. Para realizar esta composición substituye la cadena *usuario_id* y *proyecto_id* que se encuentran entre llaves por los valores respectivos de las variables *usuario_id* y *proyecto_id* que tiene en su entrada. Una vez compuesta la URI realiza la llamada a la misma. Fruto de esta llamada obtiene un conjunto de tareas en formato XML. El gadget, a partir de estos datos genera el código HTML que muestra las tareas obtenidas al usuario final, a través del cual éste puede interactuar con él.

3 ADECUACIÓN DE PROJECT OPEN

3.1 Descripción

3.2 API REST

3.3 Gadget

3.3.1 Descripción

3.3.2 Interfaz de usuario

3.3.3 Comunicación

3.3.4 Implementación

4 ADECUACIÓN DE BUGZILLA

4.1 Descripción

Bugzilla es una herramienta basada en Web de seguimiento de incidencias (bugs). De todas las funcionalidades que ofrece esta aplicación la que nos interesa incluir dentro de la arquitectura de Vulcano es la relacionada con la información de las incidencias y su relación con los usuarios y proyectos.

En concreto es interesante obtener las incidencias asociadas a una proyecto y a un usuario concreto.

Se debe tener en cuenta que Bugzilla, igual como ocurre con dotProject, contiene su propia información de usuarios del sistema. Esto entra en conflicto con la definición centralizada de usuarios dentro de la arquitectura de Vulcano que se desprende de la existencia del gadget **ListaDeUsuarios**. Esto quiere decir que por un lado tenemos la información de usuarios de Bugzilla y por otro la información de usuarios dentro de la forja. Esto implica que debe haber una sincronización de dicha información. Es decir, que cada usuario definido dentro de la forja de Vulcano debe tener su usuario correspondiente dentro de Bugzilla. En este documento no se trata este problema y se asume que ambas fuentes de usuarios están sincronizadas, de forma que, el identificador de un usuario en una de ellas identifica al mismo usuario en la otra.

Se debe tener en cuenta, además, que Bugzilla define, igual que ocurre con dotProject, su propia información de proyectos (los proyectos en la terminología de Bugzilla se llaman productos). Igual que ocurre con dotProject, esta fuente de proyectos entra en conflicto con la fuente de proyectos que existe dentro de la forja de Vulcano que se desprende de la existencia del gadget **ListaDeProyectos**. Esto implica que debe haber una sincronización entre ambas fuentes. Es decir, cualquier proyecto definido dentro de la arquitectura de Vulcano debe tener su correspondiente dentro de Bugzilla. En este documento no se trata este problema y se asume que ambas fuentes de proyectos están sincronizadas, de forma que, el identificador de un proyecto en una de ellas identifica al mismo proyecto en la otra.

4.2 API REST

A partir de las funcionalidades descritas anteriormente se definen los siguientes métodos de negocio o acciones que debe satisfacer la interfaz REST:

Función	Descripción
ObtenerIncidenciasPorProyecto	Obtener listado de incidencias que están asociadas a algún proyecto en concreto.
obtenerIncidenciasPorUsuario	Obtener listado de incidencias que están asociadas a algún usuario en concreto.

Tabla 3: Funciones Bugzilla

A continuación se muestra la API REST de Bugzilla:

Método	URI	Función
GET	bugs/proyectos/{proyecto_id}/	ObtenerIncidenciasPorProyecto
GET	bugs/usuario/{usuario_id}/	obtenerIncidenciasPorUsuario

Tabla 4: API REST

Para acceder a esta nueva API lo único que se debe hacer es realizar las llamadas HTTP convenientes definidas en la lista anterior.

En concreto cuando se quiere obtener el listado de incidencias asociadas a un proyecto o a un usuario se debe hacer una llamada GET a la correspondiente URI especificada en el API REST. Haciendo esto, el servicio, devuelve el listado de incidencias en formato XML, de forma que pueda ser manejada la información de forma sencilla. Para cada incidencia el servicio proporciona los siguientes datos: el código de la incidencia y el nombre de la incidencia.

4.3 Gadgets ListaDeIncidencias

4.3.1 Descripción

Estos gadgets muestra, mediante el establecimiento de las comunicaciones adecuadas, un listado de incidencias correspondientes a un proyecto o a un usuario en concreto. Al igual que en el gadget anterior, para hacer posible este comportamiento se asume la existencia de los gadgets **ListaDeUsuarios** y **ListaDeProyectos**. El comportamiento de estos gadget es idéntico al definido para el caso anterior. El primero, muestra un listado de usuarios en el que al pulsar sobre uno de ellos "envía" al gadget **ListaDeIncidenciasPorUsuario** la información del usuario seleccionado; una vez obtiene el dato, mediante la capa REST definida para Bugzilla, es capaz de acceder a las incidencias asignadas al usuario seleccionado. El segundo muestra un listado de proyectos en el que al pulsar sobre uno de ellos "envía" al gadget **ListaDeIncidenciasPor Proyecto** la información del proyecto seleccionado; una vez obtiene el dato, mediante la capa REST definida para Bugzilla, es capaz de acceder a las incidencias asignadas al proyecto seleccionado.

Una vez el gadget obtiene la información, la transforma para componer la interfaz de usuario, que muestra estos mismos datos de forma amena para el usuario final.

4.3.2 Interfaz de usuario

Estos gadgets muestra una tabla donde se lista la información de cada incidencia. Para cada incidencia, el gadget muestra el código y el nombre de la incidencia.

4.3.3 Comunicación

Este gadget se comunica con los gadgets **ListaDeUsuarios** y **ListadeProyectos**.

La comunicación con **ListadoDeUsuarios** se establece cuando se selecciona un usuario concreto del listado. En ese momento la información de qué usuario ha sido seleccionado se "envía al gadget" correspondiente. El gadget utiliza este dato para completar su función.

En términos de plantillas de gadgets esto implica que, este gadget, en el apartado de Wiring de su plantilla, define una variable de solo lectura o SLOT que podría llamarse *usuario_id*, que establece el identificador del usuario

seleccionado en el supuesto gadget de **ListaDeUsuarios**. Se puede decir que el gadget espera la llegada de la información del usuario en su entrada.

La comunicación con el **ListadoDeProyectos** se establece cuando el usuario selecciona un proyecto concreto del listado. En ese momento la información de qué proyecto ha sido seleccionado se "envía al gadget" correspondiente. El gadget utiliza este dato para completar su función.

En términos de plantillas de gadgets esto implica que, este gadget, en el apartado de Wiring de su plantilla, define una variable de solo lectura o SLOT que podría llamarse *proyecto_id*, que establece el identificador del proyecto seleccionado en el gadget **ListaDeProyectos**. Se puede decir que el gadget espera la llegada de la información del proyecto en su entrada.

La *Ilustración 1: Esquema de comunicación entre gadgets* muestra un esquema de comunicación entre los gadget de la forja donde se puede ver como se comunica este gadget con el resto.

4.3.4 Implementación

En lo que se refiere al código, el gadget, en algún momento hace una llamada al servicio de Bugzilla. Es decir, accede a alguna de las funciones definidas en el API de la capa REST que encapsula las funcionalidades de Bugzilla que necesitamos. En concreto, para este caso, cuando los gadgets encuentran en su entrada los datos *usuario_id* o *proyecto_id*, que indican que un usuario o un proyecto han sido seleccionados, compone la correspondiente URI de llamada **GET bugs/usuario/{usuario_id}/** o **GET bugs/proyectos/{proyecto_id}/** definidas en el API. Para realizar alguna de estas composiciones sustituye en las cadenas *usuario_id* o *proyecto_id* que se encuentran entre llaves por los valores respectivos de las variables *usuario_id* o *proyecto_id* que tiene en su entrada. Fruto de estas llamadas se obtiene un conjunto de incidencias en formato XML. Los gadgets, a partir de estos datos generan el código HTML que muestra las incidencias obtenidas al usuario final, a través del cual éste puede interactuar con él.

5 ADECUACIÓN DE LA MIB DE TOPEN

5.1 Descripción

La MIB de TOPEN es una base de datos que permite almacenar los resultados de los procesos de monitorización, testeo y pruebas a sistemas de operaciones. Para ello, se ejecutan una serie de aplicaciones que envían datos al motor de la base de datos.

La base de datos está implementada en MySQL, y para hacer la conexión con la forja, se van a manejar servlets. Eso supone que toda la conexión a la base de datos, extracción y recogida de los mismo se hará mediante Java, y utilizando Eclipse como plataforma de desarrollo. Por esta razón, no se utilizará el recubrimiento REST, sino que será una aplicación web de consulta, sin variables de almacenamiento.

La url introducida en el gadget localizador indicará la url de la aplicación web.

5.2 Gadget de la MIB

5.2.1 Descripción e interfaz

Se mostrará una lista con cada uno de los procedimientos de test que existen. Por cada procedimiento que el usuario elija se mostrará el contenido y una lista con las ejecuciones realizadas. De estas ejecuciones realizadas, el usuario podrá elegir una de ellas, mostrándose así los comandos y los resultados de esas ejecuciones.

La arquitectura interna del gadget consta de un servlet que realiza la petición a una base de datos implementada en MySQL mostrando la información solicitada en formato html. La implementación del servlet se realiza con las clases de la librería javax.servlet y javax.servlet.http.

Para realizar la conexión con la base de datos y realizar la extracción de la información se utilizan todas las clases contenidas en la librería java.sql, además de instalar el driver de java correspondiente al jodbc y de instalar el conector para realizar un origen de datos desde el sistema operativo hasta el servicio que accede a la base de datos MySQL.

5.2.2 Comunicación

Este gadget en principio no tiene comunicación con ningún otro.

5.2.3 Implementación

En lo que se refiere al código, el gadget, en algún momento hace una llamada a la aplicación web de la url del mismo. A partir de ese momento el usuario selecciona una opción y la ejecuta sucesivamente. Se consultará la MIB a través de un servlet, almacenándose los datos. El gadget, a partir de estos datos genera el código HTML que muestra la información seleccionada por el usuario, permitiéndole realizar nuevas consultas.

6 ADECUACIÓN DE WAPITI

6.1 Descripción

6.2 API REST

6.3 Gadget

6.3.1 Descripción

6.3.2 Interfaz de usuario

6.3.3 Comunicación

6.3.4 Implementación

7 ESQUEMA DE COMUNICACIÓN ENTRE GADGETS

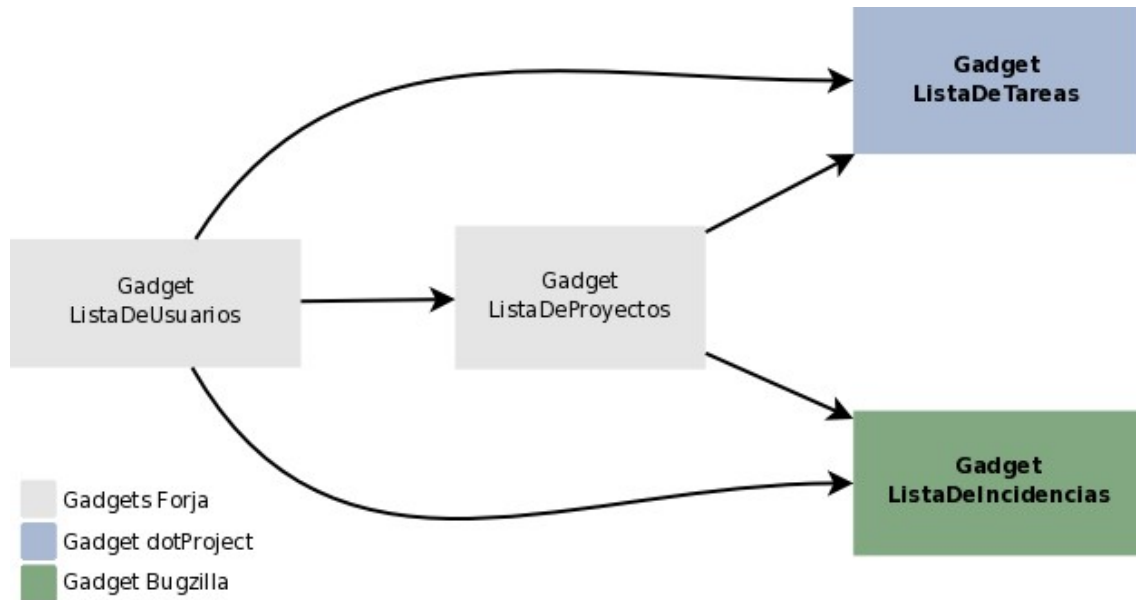


Ilustración 1: Esquema de comunicación entre gadgets