Universidad politécnica de madrid

Faculdad de Informática

Trabajo fin de grado

Desarrollo sistemático de elementos composicionales para Mashup de aplicaciones

Autor: Santiago Blanco Ventas

Tutor: Francisco Javier Soriano Camino

Resumen

Este Trabajo fin de grado ha tomado como punto de partida el escenario y los casos de uso que han sido seleccionados para el demostrador ("live demo") del proyecto europeo de I+D+I FI-WARE para, a través del diseño y de la implementación del conjunto de widgets y operadores que forman parte de dicho demostrador, educir los principios subyacentes al Desarrollo sistemático de los elementos composicionales que conforman un mashup de aplicaciones Web 2.0.

En concreto, con el escenario escogido se quiere demostrar, en un contexto de Smart Cities, cómo puede monitorizarse el estado y la geolocalización de un conjunto de maquinas de vending y de los técnicos y reponedores que les dan soporte. Este trabajo fin de grado se ocupa, en concreto, de crear todos los componentes de interfaz necesarios para poder ofrecer un mashup de aplicación que ofrezca un cuadro de mando personalizable para llevar a cabo la monitorización y el control de máquinas de vending, técnicos y reponedores.

Teniendo en cuenta el documento de casos de uso para ese escenario, se hizo una revisión de los diferentes widgets y operadores que había que diseñar y desarrollar. El conjunto inicial de widgets se mantuvo invariable, ya que las vistas que hay que mostrar deben ser las que se especifican en el documento anteriormente citado. No obstante, los operadores necesarios para el mashup han ido cambiando a lo largo del desarrollo como consecuencia de las necesidades de los tipos de datos tratados y de las características propias de la plataforma utilizada.

El diseño del mashup se ha dividido en dos tipos de diseños distintos:

- Diseño de vistas: Es un diseño de alto nivel donde se pueden observar las distintas vistas que verá el usuario en cada uno de los widgets (los operadores no tienen vista).

- Diseño de interacciones: Es un diseño de alto nivel donde se representa esquematicamente el flujo de datos entre los elementos composicionales del mashup (tanto widgets como operadores).

Al mismo tiempo que se planteaba el diseño del mashup, y como consecuencia de la experiencia ganada, se han podido ir realizando trabajos como la creación de la "Guía de desarrollo de widgets y operadores" y la "Guía de estilo para el desarrollo de widgets y operadores".Estos documentos, por su importancia, dan pie

al título del presente trabajo ya que suponen una ayuda metodológica al desarrollo sistemático de elementos composicionales para mashup de aplicaciones.
Durante ese periodo, simultaneamente, se pudieron encontrar y consultar diferentes fuentes de información que completan el "Estado del arte" y el documento de "Principios de diseño de aplicaciones composicionales orientadas a mashups" y se realizaron varias herramientas que facilitan la tarea del programador de crear un nuevo widget u operador desde cero. Entre estas últimas, destacan: una herramienta para crear y mantener la estructura de directorios y archivos de un widget y de un operador; y una herramienta que empaqueta todos los ficheros del widget u operador en un solo fichero WGT, dejándolo preparado para ser desplegado en la plataforma.

La mayor parte del trabajo ha consistido en la implementación de los widgets y operadores.

Los widgets implementados son los siguientes:

- Map Viewer.
- Issues List.
- Technicians List.
- Technicians Info.

Los operadores implementados son los siguientes:

- Poi2Issue
- Issue2Poi
- Poi2vCard
- Vcard2Poi
- TechnicianSource
- IssuesSource

También se han realizado tres tipos de tests: unitarios, de integración y de sistema. Además se ha creado documentación de cada elemento composicional para el usuario. En dicha documentación se explica cuales son las funcionalidades de cada uno de los elementos composicionales y cuales son las capacidades de conexión con otros elementos.

El resultado de mi trabajo puede observarse en: http://www.youtube.com/watch?v=r8_Vv_ehJSw

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Esta introducción ofrece una explicación del contexto en el que se engloba el proyecto, las definiciones de los términos que es recomendable conocer antes de leer el resto de la memoria y una descripción detallada de la estructura y el contenido de ésta.

1.1. Contexto del proyecto

1.1.1. ¿Que es FI-WARE?

El proyecto financiado con fondos europeos FI-WARE (Plataforma central del Internet del futuro, del inglés Future Internet Core Platform) se dedica a la creación de una nueva infraestructura de servicios basada en componentes básicos genéricos y reutilizables denominados Capacitadores Generales (Generic Enablers, GE).

Dentro del proyecto existen una serie de capítulos de entre los que destaca en el que esta englobado WireCloud, que es Application/Services ecosystem and Delivery Framework, que trata sobre la creación, publicación, gestión y consumo de servicios del internet del futuro.

1.1.2. Términos importantes

¿Qué es un WireCloud y cuál es su importancia?

Wirecloud es una plataforma de Mashups de aplicación.
WireCloud, desde el punto de vista del usuario, se divide en cuatro vistas:

- Workspace: El espacio de trabajo donde se agregan los widgets y donde se construyen los mashups.
- Wiring: El editor donde se realizan las conexiones entre elementos composicionales (widgets y operadores).
- Catalogue: El catálogo donde se encuentran todos los widgets y operadores que tiene disponible el usuario.

- Marketplace: El lugar donde se pueden comprar widgets y operadores de diferentes tiendas para incluirlos en el catálogo local.

¿Qué es un mashup?

Un mashup [14] es un conjunto de aplicaciones (normalmente denominados widgets o gadgets) o servicios que interaccionan entre si con el objetivo de crear una aplicación más rica, compleja (pero no complicada) y util. No solo significa crear una aplicación, en definitiva, más rica para el usuario, sino que además la integración entre los diferentes elementos debe ser fácil, rápida y sencilla (no simple), usando a menudo APIs abiertos y fuentes de datos para producir resultados enriquecidos que no fueron la razón original para la que fueron producidos los datos originales.

La arquitectura de los mashups se compone de tres capas:

- Presentación: Es la interfaz de usuario del mashup. Las tecnologías usadas, principalmente, son HTML/XHTML, CSS y Javascript.
- Servicios Web: Mediante las APIs de los servicios web podemos acceder a las funcionalidades de los productos. Las tecnologias usadas son, habitualmente, XMLHTTPRequest, XML-RPC, JSON-RPC, SOAP, REST.
- Datos: La lógica de negocio que controla el flujo y la gestión de la información. Las tecnologias usadas son, fundamentalmente, XML y JSON.

Además podemos diferenciar entre dos tipos de mashups dependiendo de donde se haga el tratamiento de la información. De esta forma tenemos Mashups de lado cliente y Mashups de lado servidor.[16]

En lo que respecta a este proyecto los mashups son aplicaciones compuestas por varios elementos composicionales, tanto widgets como operadores, que se relacionan entre si para formar una aplicación más útil y rica para el usuario.

¿Qué es un widget?

En el entorno web un widget es un software desarrollado para la web. Concretamente es una pequeña aplicación que puede ser instalada y ejecutada en una pagina web por un usuario final. Los widgets están inspirados en la idea de la reusabilidad del código, de forma que se puedan incluir en cualquier pagina web que esté preparada para ello tantas veces como se desee. Los widgets también se conocen con el nombre de portlet, gadget, badge, module, webjit, capsule, snippet, mini and flake. Estos widgets son creados normalmente con DHTML, JavaScript o Adobe Flash.[15]

Un widget puede entenderse verse desde el punto de vista del patrón Modelo-Vista-Controlador que consta de tres partes: Presentación: Es la interfaz de usuario que se desarrolla en HTML y CSS. Lógica de negocio: Implementa la funcionalidad que debe tener el widget y la respuesta a los eventos producidos por el usuario. Se

realiza en JavaScript. Representación de la información: El formato de la información y su gestión es la tarea de esta capa. Se realiza en JavaScript.[16]

Por tanto una definición de widget en el ámbito de este proyecto, sería la siguiente: un **widget** es un elemento composicional con una funcionalidad determinada y concreta que forma parte de un mashup dentro de la plataforma de mashups Wirecloud que puede interaccionar con otros elementos composicionales.

¿Qué es un operador?

Un operador es otro tipo de elemento composicional, especialmente diseñado para WireCloud que realiza tareas de background, es decir, no tiene funcionalidades de interacción con el usuario, y por tanto, solo tienen sentido para relacionar unos elementos composicionales con otros. Pueden ser configurables desde el punto de vista del usuario, pero no se puede interaccionar con ellos de otra forma, por tanto en el flujo del mashup jugarán un papel previamente configurado, dadas sus características intrínsecas.
En el contexto de la plataforma WireCloud los operadores solo se pueden agregar al espacio de trabajo y usar en la vista de Wiring.

1.2. Motivación del proyecto

Este proyecto surge de la necesidad de facilitar a los nuevos desarrolladores de elementos composicionales la creación de dichos elementos y proporcionar a los ya versados en este tema, un método sistemático que haga más eficiente su trabajo. Además, dentro de un campo en el que todavía se está investigando y que está en pleno desarrollo, es interesante poder participar en lo que pueden denominarse los cimientos de lo que pretende ser el desarrollo de un tipo específico de aplicaciones del futuro, en distintos dispositivos y en uno de los entornos multiplataforma más potentes y totalmente asentado como es la web.

1.3. Objetivos del proyecto

En este proyecto se persiguen cuatro objetivos que concretamos a continuación:

- Desarrollo de un conjunto de elementos composicionales adecuados para los casos de uso y escenarios considerados.
- Desarrollo de un conjunto de pautas para el diseño de una solución composicional.
- Desarrollo de un conjunto de herramientas y de una guía de estilo para widgets y operadores.
- Creación de una guía de desarrollo que ayude a la creación de widgets de una forma lo más sistemática posible.

1.4. Descripción del resto del documento

Este documento se estructura en 3 capítulos además de este de introducción y objetivos que se explican a continuación.

En el segundo capítulo, se tratará de dar una visión del estado del arte. Por una parte se verá cual es el estado de las principales plataformas existentes para la creación web de mashups web y de los más importantes tipos de widgets considerados por dichas plataformas, que dará una idea general de cómo se encuentra el panorama actual en estas tecnologías. Por otra parte veremos cuáles son las tecnologías que se han usado para llevar a cabo el desarrollo del proyecto.

En el tercer capítulo, se describirá el proceso de desarrollo del proyecto. Con una explicación detallada del trabajo realizado y todo el software generado en el proceso, entre los que destacan los widgets y operadores, por encima de las herramientas de ayuda para su creación. El diseño del mashup y la problemática de los servicios de backend.

En el cuarto y último capítulo, se verán las conclusiones del proyecto, los resultados obtenidos y los pasos futuros que pueden surgir a partir de este trabajo.

Capítulo 2

Estado del arte

2.1. Principales plataformas de Mashup

2.1.1. iGoogle

iGoogle [1] soporta el uso de gadgets especialmente desarrollados para mostrar el contenido a un usuario de la página. Los gadgets interactúan con el usuario y utilizan las API de Google Gadgets. Éstos pueden colocarse en distintas pestañas dentro de iGoogle. Algunos gadgets para Google Desktop también pueden ser utilizados dentro de iGoogle. La API de Google Gadgets es pública y permite a cualquiera desarrollar un gadget para cualquier necesidad.

iGoogle se retirará el 1 de noviembre de 2013. La versión para dispositivos móviles se retiró el 31 de julio de 2012.

2.1.2. Yahoo Pipes

Yahoo! Pipes es una aplicación web de Yahoo! que ofrece una interfaz grafica de usuario para construir mashups de datos que pueden agregar diferentes servicios, crear aplicaciones basadas en la Web de varias fuentes, y publicar dichas aplicaciones. Esta aplicación permite a los usuarios unir y mezclar información de diferentes fuentes y crear reglas para modificar la información (por ejemplo filtrandola)[12].

2.1.3. Windows Sidebar

Windows Sidebar es un motor de widgets de escritorio de Microsoft. Sus widgets son denominados gadgets, y pueden realizar tareas diversas, como mostrar la fecha y hora y mostrar el uso de CPU. Windows viene con una serie de gadgets predeterminados, y cualquier persona puede desarrollar gadgets. Fue introducido con Windows Vista, donde los gadgets se agrupaban en una barra lateral.
En Windows 7 el Sidebar ha sido eliminado y, ahora, los gadgets se pueden colocar

en todo el espacio del escritorio.

2.1.4. Mac OS Dashboard

El Dashboard de Mac OS es el motor de widgets de escritorio de Apple. Cuando Dashboard se activa, en versiones antiguas, el escritorio del usuario es oscurecido y los widgets aparecen en primer plano mediante un fundido. Como las ventanas de aplicaciones, pueden ser arrastrados a diferentes posiciones dentro del escritorio. En la ultima versión de Mac OS X, Lion, es otro escritorio más y, basicamente, con el mismo funcionamiento.

2.2. Principales estándares de widgets

2.2.1. Estándar de widgets del W3C

La especificación del estándar de Widgets del W3C se compone de una serie de documentos de los cuales destaca el denominado "Widget Packaging and XML Configuration"donde se detallan varios aspectos sobre el empaquetamiento de los ficheros que conforman el widget, la estructura de ficheros recomendada y el formato del fichero de configuración que ha de parsear la plataforma.

2.2.2. OpenSocial

OpenSocial es un conjunto de API comunes destinadas a la creación de aplicaciones sociales en múltiples sitios web. OpenSocial está compuesto por API de JavaScript y API de datos de Google. La existencia de este modelo de programación único resulta de gran utilidad tanto para los desarrolladores como para los sitios web. En primer lugar, los desarrolladores sólo tienen que aprender las API una vez para crear aplicaciones que funcionen con cualquier sitio web compatible con OpenSocial. En segundo lugar, como cualquier sitio web puede implementar OpenSocial, los desarrolladores disponen de una amplia red de distribución para llegar a los usuarios. Los sitios web también se benefician mediante la participación de un conjunto mucho más numeroso de desarrolladores externos que el que podrían conseguir sin un conjunto estándar de API. La Compañía Google y sus asociados, ofrecen algunas tecnologías para que Internet en su conjunto llegue a ser un medio más social, respondiendo así al claro interés de los usuarios. Productos, como orkut, son sólo uno de los distintos sitios web que implementan OpenSocial. Actualmente, el código de ejemplo se ofrece con la licencia de Apache 2.0. Además, las licencias de toda la documentación de OpenSocial proceden de Creative Commons, por lo que se puede reutilizar y combinar los servicios como estimes oportuno. En el futuro, se plantea ofrecer el software libre de los componentes necesarios para ejecutar OpenSocial en tu propio sitio web.[13]

2.3. Tecnologías utilizadas

En esta sección se describen una serie de tecnologías utilizadas para la implementación del mashup incluyendo lenguajes de programación, frameworks, etc.

2.3.1. Navegadores

Firefox

Firefox[3] es el **navegador web** de Mozilla. Es libre y de código abierto, además está disponible para MS Windows, Mac OS X y GNU/Linux. Usa el motor de renderizado Gecko[4] (libre también) que es el que se encarga de interpretar una serie de estándares como son:

- HTML4
- HTML5 (parcialmente)
- CSS 2.1
- CSS 3 (parcialmente)
- Javascript 1.8
- DOM 1 y 2
- XML 1.0
- XHTML 1.0

Este navegador web pasa las pruebas de Acid3[5] que comprueba si se respetan ciertos estandares.

Google Chrome

Google Chrome[6] es el **navegador web** de Google. Chrome está construido en el marco del proyecto de software libre Chromium. Este navegador utiliza el motor de renderizado Webkit. Este motor de renderizado tiene 3 componentes:

- Webcore: Se encarga de interpretar el DOM, el código HTML y el SVG.
- JavaScriptCore: Se encarga de interpretar el código JavaScript.
- Drosera: Es el debugger para JavaScript.

Este navegador web pasa las pruebas de Acid3[5] que comprueba si se respetan ciertos estandares.

2.3.2. Plataformas de Mashup

Wirecloud

Es una plataforma de aplicaciones web híbridas. Wirecloud se puede dividir en cuatro conceptos:

- Workspace: El espacio de trabajo donde se agregan los widgets y donde se construyen los mashups.
- Wiring: El editor donde se realizan las conexiones entre widgets y operadores.
- Catalogue: El catálogo donde se encuentran todos los widgets y operadores que tiene disponible el usuario.
- Marketplace: El lugar donde se pueden comprar widgets y operadores de diferentes tiendas para incluirlos en el catálogo local.

Esta plataforma permite a cualquier desarrollador agregar elementos composicionales a una tienda. Dicha tienda mostrará los productos de que dispone en el Marketplace de la plataforma. En dicho Marketplace, el usuario, podrá elegir que widgets/operadores agregará a su catálogo, con los que posteriormente creará sus propios Mashup y comunicará unos elementos composicionales con otros del mismo mashup para dotar de una funcionalidad más rica y compleja a la composición creada.

2.3.3. Lenguajes

HTML

HTML[8] son las siglas de HyperText Markup Languaje (lenguaje de marcado de hipertexto). Es un lenguaje de marcado ampliamente usado para realizar paginas web ya que los navegadores web (clientes HTTP) son capaces de interpretarlo. Actualmente esta disponible la versión 5 de este lenguaje como estándar del W3C[9]. El estandar más extendido es el XHTML 1.0 que es básicamente HTML expresado como XML, con las restricciones que ello supone, con el objetivo de lograr una web semántica. Para conseguir esto se han detallado una serie de normas de obligado cumplimiento que pueden ser comprobadas con verificadores de XHTML:

- Los elementos vacíos deben cerrarse siempre.
- Los elementos no vacíos también deben cerrarse siempre.
- Los elementos anidados deben tener un correcto orden de apertura/cierre (el que se abre último, debe cerrarse primero).
- Los valores de los atributos deben siempre ir encerrados entre comillas (simples o dobles).
- Los nombres de elementos y atributos deben ir en minúsculas.

- No está permitida la minimización de atributos (se usa el nombre del atributo como valor).
- Los atributos desaprobados en HTML 4.01 no forman parte de XHTML.

Ejemplo de código HTML:

```
1 <!DOCTYPE HTML>
2 <html>
3     <head>
4         <title>Ejemplo1</title>
5     </head>
6     <body>
7         <p>ejemplo1</p>
8     </body>
9 </html>
```

CSS

CSS[10] son las siglas de Cascading Style Sheets (hojas de estilo en cascada). Con CSS podemos definir, las fuentes de los textos, su tamaño, su color, el color de fondo. También el espacio que se deja entre diferentes elementos gracias a su modelo de caja, que incluye margenes, bordes, espacio entre borde y el propio elemento, tamaño del elemento en si. Éstas y muchas otras características se pueden cambiar para presentar una página web de diferentes modos tan solo con cambiar el documento CSS enlazado en el fichero HTML.

Es un lenguaje usado para definir la presentación de un documento estructurado escrito en XML (por extensión incluye HTML). El estilo de la página web se puede definir mediante una serie de reglas en un fichero css aparte, lo cual nos permite separar la presentación de la estructura. Separar la presentación del contenido permite, por ejemplo, la posibilidad de poder cambiar la presentación de una web según el dispositivo en el que se encuentre, creando así lo que se está denominando actualmente como "responsive design".

Como podemos ver en el código de ejemplo, la estructura de un fichero CSS es muy simple:

```
1 body {
2     padding-left: 11em;
3     font-family: Georgia, "Times New Roman", serif;
4     color: red;
5     background-color: #d8da3d;
6 }
7
8 h1 {
```

```
9       font-family: Helvetica, Geneva, Arial, sans-serif;
10  }
```

Javascript

Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Es un lenguaje Multi-paradigma, soporta la programación funcional, basado en prototipos, imperativo, débilmente tipado y dinámico, tambien soporta el paradigma orientado a objetos, pero no está tan orientado como en otros lenguajes de programación como Java o C++.
JavaScript fue desarrollado originalmente por Brendan Eich de Netscape con el nombre de Mocha, el cual fue renombrado posteriormente a LiveScript, para finalmente quedar como JavaScript.
Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. Gracias a los motores de renderizado de los navegadores, este lenguaje nos permite modificar la estructura DOM de nuestros documentos HTML, el estilo CSS de las páginas y dotarles de una funcionalidad que sin Javascript sería imposible.

En el siguiente ejemplo podemos comprobar que su sintaxis es muy parecida a C:

```
1  function crearMensaje() {
2      var mensaje = ''Mensaje de prueba'';
3  };
4  creaMensaje();
5  alert(mensaje);
```

Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis muy limpia y que favorezca un código legible. Es multiparadigma: orientado a objetos, imperativo y funcional. usa tipado dinámico y de contero de referencias para la administración de memoria. Python, en entornos de desarrollo web, se usa principalmente para implementar las características del lado servidor que sean necesarias. Destaca principalmente por lo estricto en temas de identación y su diferencia en la sintaxis con respecto a C, sin llaves ni puntos y coma.
A continuación podemos ver un ejemplo de código:

```
1  int factorial(int x) {
2      if (x == 0)
3          return 1;
4      else
5          return x * factorial(x - 1);
```

```
6 }
```

2.3.4. Frameworks

Bootstrap

Bootstrap es un front-end framework creado por Twitter para agilizar el desarrollo de páginas web[11].

Bootstrap ofrece al desarrollador un estilo CSS predefinido listo para usar simplemente adaptando el código HTML a las clases CSS definidas en los ficheros de Bootstrap. La documentación del framework es muy completa y tiene ejemplos que muestran las posibilidades que ofrece al mismo tiempo que facilita su aprendizaje. Además, ofrece la posibilidad de personalizar el estilo definido por defecto para adaptarlo a las necesidades del desarrollador.

Como podemos ver en el siguiente ejemplo, tan solo habría que modificar el código HTML agregando clases a sus etiquetas:

```
1 <div class="btn-group">
2     <button class="btn">1</button>
3     <button class="btn">2</button>
4     <button class="btn">3</button>
5 </div>
```

Jasmine

Es un behavior-driven development (BDD) framework de codigo abierto, que actualmente está en su versión 1.3.1, y sirve para hacer test unitarios en JavaScript. El framework facilita un fichero html en el que se deben incluir los ficheros tanto del código del que se desean realizar los test como los del código de los propios test. Una vez preparado el documento html se pueden ver los resultados de las pruebas en un navegador.

Como se muestra en el ejemplo se pueden realizar pruebas de una forma sencilla y potente:

```
1 describe(''A suite is just a function'', function() {
2     var a;
3     it("and so is a spec", function() {
4         a = true;
5         expect(a).toBe(true);
6     });
7 });
```

Django

Django es un framework de desarrollo web de código abierto, escrito en Python, que cumple en cierta medida el paradigma del Modelo Vista Controlador. Fue desarrollado en origen para gestionar varias páginas orientadas a noticias de la World Company de Lawrence, Kansas, y fue liberada al público bajo una licencia BSD en julio de 2005; el framework fue nombrado en alusión al guitarrista de jazz gitano Django Reinhardt.
La meta fundamental de Django es facilitar la creación de sitios web complejos. Django pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio No te repitas (DRY, del inglés Don't Repeat Yourself). Python es usado en todas las partes del framework, incluso en configuraciones, archivos, y en los modelos de datos.

Selenium

Selenium es un framework de testing para aplicaciones web de código abierto. Fué desarrollado originalmente por Jason Huggins en el 2004.

Selenium WebDriver junto con Django permite realizar pruebas unitarias, de integración y de sistema para entornos web en diferentes navegadores. Automatizando así las pruebas que debería hacer un usuario haciendo click en los diferentes enlaces de una página web o escribiendo el contenido de los campos de un formulario.

A continuanción podemos ver un ejemplo de código:

```
1  class AccountAdminPageTests(unittest.TestCase):
2      def setUp(self):
3          self.selenium = selenium("localhost",
4                                   8000,
5                                   "*chrome",
6                                   "http://localhost:8000/")
7          self.selenium.start()
8          self.selenium.open("/")
9
10     def test_ok(self):
11         self.assertTrue(self.selenium.is_text_present('OK'))
12
13     def tearDown(self):
14         self.selenium.stop()
```

2.3.5. Control de versiones

GIT

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.
Entre las características más relevantes se encuentran:

- Fuerte apoyo al desarrollo no-lineal, por ende rapidez en la gestión de ramas y mezclado de diferentes versiones. Git incluye herramientas específicas para navegar y visualizar un historial de desarrollo no-lineal. Una presunción fundamental en Git es que un cambio será fusionado mucho más frecuentemente de lo que se escribe originalmente, conforme se pasa entre varios programadores que lo revisan.

- Gestión distribuida. Al igual que Darcs, BitKeeper, Mercurial, SVK, Bazaar y Monotone, Git le da a cada programador una copia local del historial del desarrollo entero, y los cambios se propagan entre los repositorios locales. Los cambios se importan como ramas adicionales y pueden ser fusionados en la misma manera que se hace con la rama local.

- Los almacenes de información pueden publicarse por HTTP, FTP, rsync o mediante un protocolo nativo, ya sea a través de una conexión TCP/IP simple o a través de cifrado SSH. Git también puede emular servidores CVS, lo que habilita el uso de clientes CVS pre-existentes y modulos IDE para CVS pre-existentes en el acceso de repositorios Git.

- Los repositorios Subversion y svk se pueden usar directamente con git-svn.

- Gestión eficiente de proyectos grandes, dada la rapidez de gestión de diferencias entre archivos, entre otras mejoras de optimización de velocidad de ejecución.

- Todas las versiones previas a un cambio determinado, implican la notificación de un cambio posterior en cualquiera de ellas a ese cambio (denominado autenticación criptográfica de historial). Esto existía en Monotone.

- Resulta algo más caro trabajar con ficheros concretos frente a proyectos, eso diferencia el trabajo frente a CVS, que trabaja con base en cambios de fichero, pero mejora el trabajo con afectaciones de código que concurren en operaciones similares en varios archivos.

- Los renombrados se trabajan basándose en similitudes entre ficheros, aparte de nombres de ficheros, pero no se hacen marcas explícitas de cambios de nombre con base en supuestos nombres únicos de nodos de sistema de ficheros, lo que evita posibles, y posiblemente desastrosas, coincidencias de ficheros diferentes en un único nombre.

- Realmacenamiento periódico en paquetes (ficheros). Esto es relativamente eficiente para escritura de cambios y relativamente ineficiente para lectura si el reempaquetado (con base en diferencias) no ocurre cada cierto tiempo.

Capítulo 3

Desarrollo

3.1. Análisis y elección de los casos de uso y escenarios

El caso de uso propuesto, en términos generales, trata sobre la monitorización de la posición y el estado de una serie de recursos en un sistema de coordenadas geográficas. Concretamente, se realiza la monitorización del estado de máquinas de vending y los técnicos de mantenimiento de las máquinas.
Dichas maquinas de vending tienen una serie de niveles que indican: la cantidad de sustancias que contienen almacenadas (leche, café, fruta, etc); y la cantidad de cambio de cada moneda. Además tiene una serie de condiciones propias del estado de la maquina, como por ejemplo: apagada, en buen estado y averiada.
Por la parte de los técnicos de mantenimiento se tiene la información de la furgoneta en la que se desplaza, su posición actual, los palés de sustancias que transporta y sus tareas asignadas.

3.2. Identificación del conjunto de widgets y operadores

El documento que ha guiado el desarrollo de la solución ha sido una presentación que describe el escenario y el caso de uso (DemoScript). Mediante este documento se ha conseguido identificar una serie de widgets y operadores que conforman un mashup y dan solución al problema.
Para desarrollar la solución se diseñó un esquema compuesto, por una parte, por 4 widgets que tienen asignada una tarea concreta y, por otra, por 6 operadores que ayudan en la interaccion de unos widgets con otros y reciben la información externa.
La lista de widgets implementados son los siguientes:

- Issues List: La necesidad de relizar el seguimiento de las incidencias de las máquinas de vending es resuelta por este widget.

- Technician List: La función de este widget es mostrar y mantener una serie de técnicos de mantenimiento listados.
- Technician Info: En el que se muestra la información de un técnico de mantenimiento.
- Map Viewer: Es donde se muestra la posición de cada recurso monitorizado dentro de un mapa. En el se pueden observar la posición de cada uno de los técnicos y de las máquinas de vending.

La lista de operadores implementados son los siguientes:

- Poi2Issue: transforma un PoI en una Issue.
- Issue2Poi: transforma una Issue en un PoI.
- Poi2vCard: transforma un PoI en una vCard.
- Vcard2Poi: transforma una vCard en un PoI.
- TechnicianSource: Recibe los datos de los técnicos generados desde el servicio externo para entregarselos al mashup.
- IssuesSource: Recibe las incidencias generadas desde el servicio externo para entregarselos al mashup.

En las siguientes secciones se desarrollarán más en profundidad cada uno de los elementos composicionales.

3.3. Diseño de vistas, lógica e interacciones de widgets y operadores

El diseño del mashup se dividió en dos tipos de diseños distintos:

- Diseño de vistas: El diseño de vistas para cada uno de los widgets es una aproximación de como tendría que ser el widget para tener una idea clara de como se queria interactuar con el usuario.
- Diseño de interacciones: El diseño de interacciones muestra las relaciones y el tipo de datos que se intercambia cada uno de los elementos composicionales con el resto.

A partir de las especificaciones recogidas en un documento, y los diseños iniciales se realizo el proceso de implementación con las tecnologías anteriormente descritas: JavaScript, HTML y CSS.

Para el intercambio de información de unos elementos con otros se optó por elegir tipos de mensajes lo más parecidos a los estándares existenes conocidos, por ejemplo para intercambiar datos de técnicos, se usó el estandar vCard version 3.0, ya que

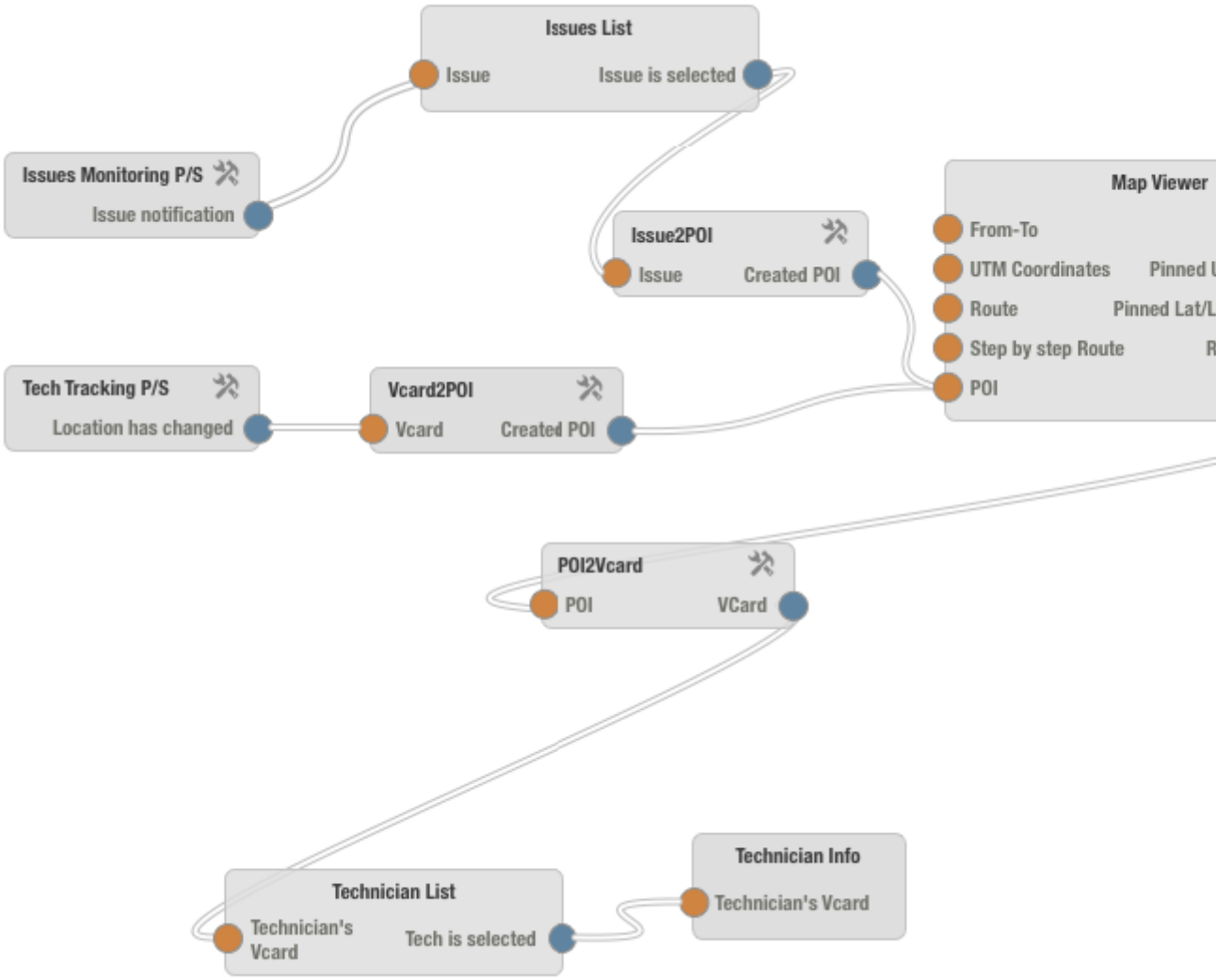

Figura 3.1: Diseño de interacciones

aunque existen otras versiones actuales, la librería desarrollada con el fin de realizar esta taréa debía ser compatible con la versión que usa Gmail.

Los PoIs estan en proceso de estandarización por parte de la W3C y se uso como referencia para crear un tipo de dato propio.

Hay que decir que la plataforma solo permite la transmision de mensajes entre widgets y operadores en formato string, pero no hay ningun problema porque javascript de forma nativa ofrece metodos para transformar JSON a string y viceversa, así que aunque se trate la información en formato JSON dentro de los widgets, simplemente hay que serializarla y desserializarla a traves de estos métodos.

En la figura 3.1 se puede ver el esquema de relaciones de los widgets y operadores.

3.4. Implementación de un conjunto de widgets y operadores

Para implementar todos los elementos composicionales se han usado las siguientes tecnologías:

- JavaScript: El lenguaje de programacion con el que se da la funcionalidad a los widgets.
- HTML: El lenguaje de marcado que sirve para definir el arbol DOM que constituye la estructura del widget.
- CSS: El lenguaje de definición de estilos en cascada que sirve para dar estilo a la vista del widget.

Para llevar un control de versiones del desarrollo se ha usado una herramienta especifica llamada GIT.

3.4.1. Widgets

Recordemos que los widgets son los elementos composicionales con los que ha de interactuar el usuario. Cada uno de los widget tiene una funcionalidad definida independiente de los demás, no obstante cuando se relacionan unos con otros su utilidad se incrementa de forma exponencial.

Issues List

Dado que la monitorización de las incidencias es una parte fundamental para gestionar el estado de las máquinas de vending, se llevó a cabo el desarrollo de este widget independiente que gestina toda la información relacionada con las incidencias.

En la vista del widget se puede encontrar información relacionada con una serie de parámetros de la incidencia tales como: la urgencia de la incidencia, su tipo, una descripción, un identificador de la máquina de vending y cual de los técnico está asignado en el momento actual.

Cada vez que se genera una incidencia, ésta se agrega a la lista de incidencias, de forma que si el usuario quiere asignarle un nuevo técnico lo puede hacer simplemente eligiendolo de la lista desplegable. De la misma forma cuando un nuevo técnico se está monitorizando se agrega a la lista de técnicos.
En la figura 3.2 se puede ver una captura de pantalla del widget.

Technician List

El motivo de la existencia de este widget es que el usuario tenga un conocimiendo de quienes y cuantos técnicos se están monitorizando, por ello es necesario que

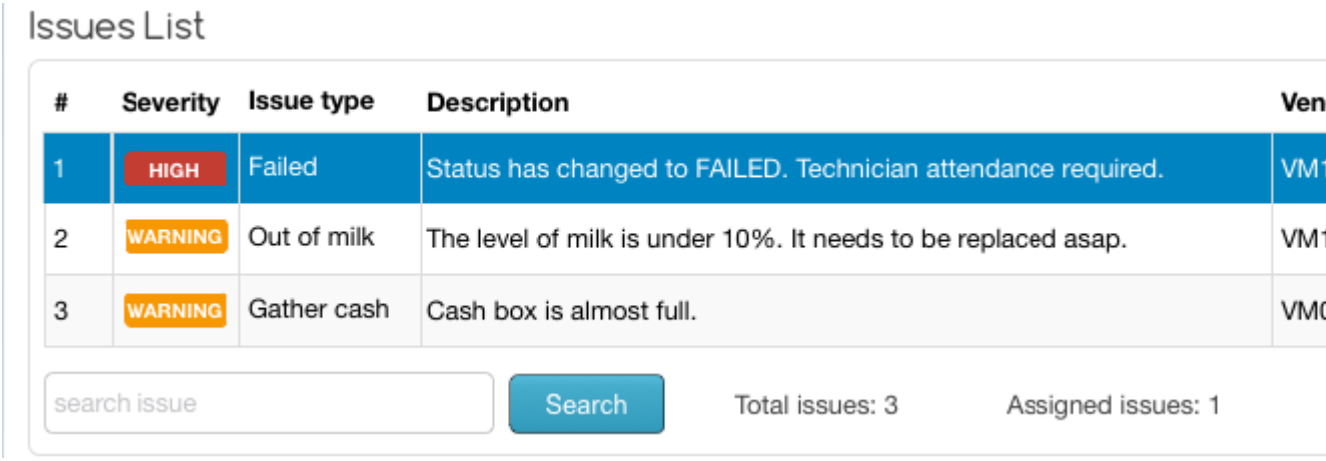

Figura 3.2: Issues List

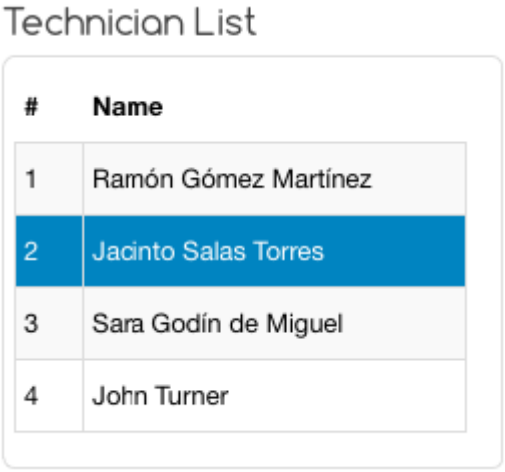

Figura 3.3: Technician List

esté disponible el compendio de todos ellos en algún tipo de lista o tabla.

En este caso se ha optado por crear una lista de técnicos que se pueden seleccionar, para por ejemplo, ver su perfil en el widget Technician Info. Cada vez que llega información de un nuevo técnico al mashup este widget lo muestra en la lista.

En la figura 3.3 se puede ver una captura de pantalla del widget.

Technician Info

De la misma forma que el usuario que está monitorizando el estado de las máquinas de vending a traves del widget "Issues List" puede ver la información de las incidencias, es necesario que pueda obtener también la información de los técnicos.

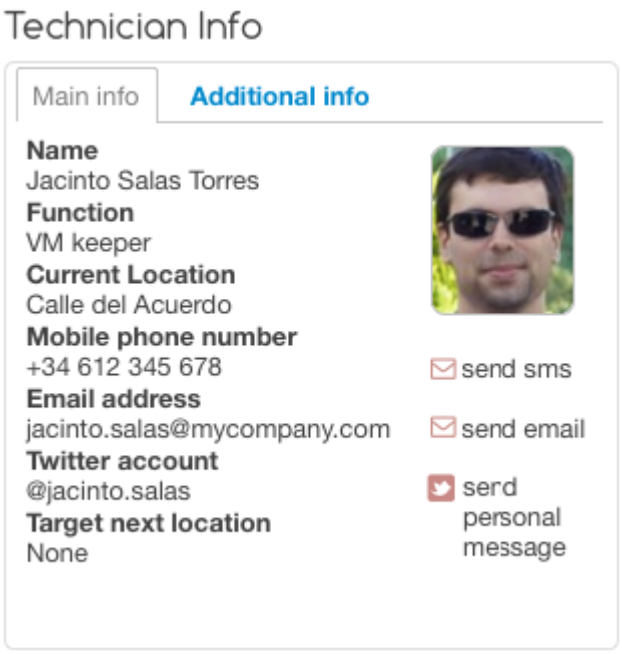

Figura 3.4: Technician Info

El funcionamiento de este widget es el de un widget del tipo visor de datos. Dada cierta información de entrada con un formato determinado se muestra el perfil del técnico con los siguientes campos:

- Nombre: El nombre del técnico de mantenimiento.
- Función: Función que desempeña dentro de la empresa.
- Lugar actual: Lugar donde se encuentra actualmente.
- Teléfono movil: Telefono movil de contacto.
- Email: Dirección de correo electrónico de contacto.
- Twitter: Nombre de su cuenta de Twitter.

En la figura 3.4 se puede ver una captura de pantalla del widget.

Map Viewer

Dado que se quiere monitorizar la posición de cada uno de los técnicos y máquinas de vending es necesario tener un mapa donde se encuentren representados cada uno de los recursos de los que se disponen.

Con la funcionalidad principal de mostrar una representación de cada elemento y donde se encuentra en cada instante se ha implementado este widget, gracias a la API que ofrece Google de Google Maps (v3). Usando esta API se puede obtener

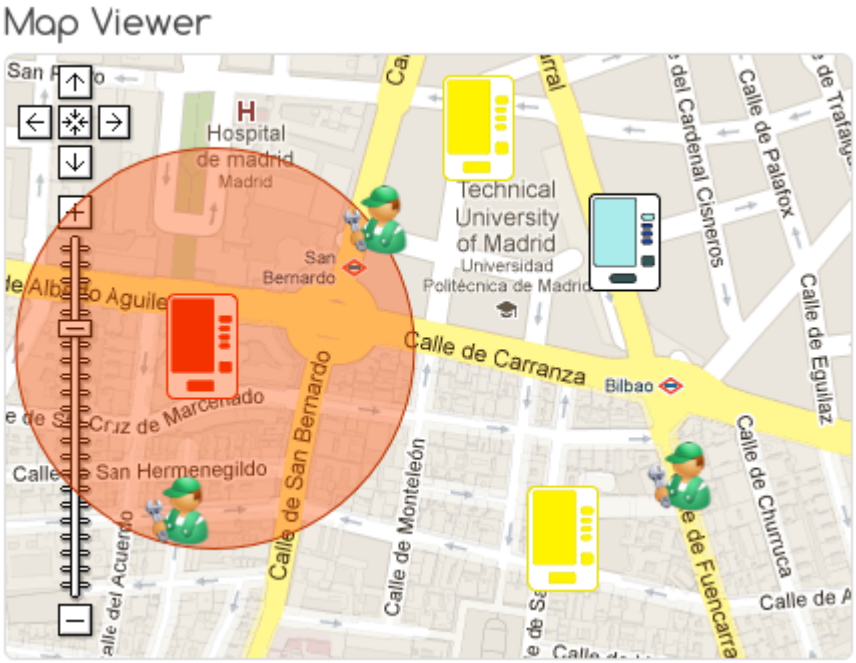

Figura 3.5: Map Viewer

una representación grafica en el lugar indicado mediante Coordenadas o con una localización en lenguaje natural (por ejemplo, “Madrid, C/Bravo Murillo, nº55”).

Cuando se inicia el widget se recibe la información de todas las maquinas de vending y técnicos que existen en nuestro sistema y se muestran en el mapa los iconos que les representan. Las máquinas de vending se representan además con el color del estado en el que se encuentran. Si se pincha en una de las maquinas de vending se muestra información de su estado actual en una ventana, y de la misma forma se muestra la información más relevante del técnico si se pincha sobre su icono.

No solo se pueden ver los iconos de los recursos monitorizados sino que además, al pinchar sobre cada uno de los recursos se muestra un “radio de acción” para mostrar claramente si hay, por ejemplo, cerca de una máquina de vending un técnico.

Otra funcionalidad que puede resultar interesante al usurio es la capacidad del mapa de mostrar rutas, de forma que si se quiere mostrar una ruta desde la posición actual de un técnico a una máquina de vending, se mostrará, y redimensionará el mapa al tamaño adecuado.
En la figura 3.5 se puede ver una captura de pantalla del widget.

3.4.2. Operadores

Antes de hacer una descripción de los operadores es necesario conocer cual es el formato de cada uno de los tipos de datos que se han utilizado, a saber:

- PoI: Point of Interest. Es un JSON creado en base a la especidicación un estandar del W3C.[17]
- Issue: Es un JSON diseñado en base a las necesidades de la aplicacion.
- vCard: Es un formato estándar para el intercambio de información personal, la versión usada es la 3.0, ya que es la que usa actualmente google en Gmail.[18]

Issue to Poi

Este operador tienen como misión transformar los datos que estan en formato de Issue a un formato de PoI (Point of Interest). Basicamente este operador esta diseñado para conectar una fuente de issues (Issues List por ejemplo) a un consumidor de PoIs (Map Viewer por ejemplo).

Poi to Issue

Este operador tienen como misión transformar los datos que estan en formato de PoI a un formato Issue. Basicamente este operador esta diseñado para conectar una fuente de PoIs (Map Viewer por ejemplo) a un consumidor de dwIssues (Issues List por ejemplo).

vCard to Poi

Este operador es análogo al operador Issue to Poi, con la salvedad de tratar con vCards en lugar de con Issues.

Poi to vCard

Este operador es análogo al operador Poi to Issue, con la salvedad de tratar con vCards en lugar de con Issues.

Technician Source

Este operador es el que se encarga de recibir toda la información que se genera en el backend. El sistema utilizado es el de Publish/Subscribe, de manera que el operador se subscribe a un tipo de dato mediante filtros y cuando un "editor" manda un mensaje del tipo especificado se recibe en este operador de forma asíncrona.

Issue Source

Este operador funciona de forma análoga al anterior, con la salvedad del filtro que hay que configurar y el tipo de dato que genera.

3.5. Desarrollo de herramientas de ayuda a la creación de widgets y operadores

Para hacer el desarrollo de widgets y operadores un proceso más rápido y exento de fallos, era necesario crear antes una serie de herramientas. Concretamente fueron cuatro las herramientas desarrolladas.

- createWidget: crea la plantilla de un widget.
- createOperator: crea la plantilla de un operador.
- packageWgt: empaqueta el elemento composicional y lo transforma al formato requerido por la plataforma.
- xml2rdf: transforma el fichero de configuracion xml en otro rdf.

3.5.1. createWidget

Esta herramienta crea un directorio con una plantilla génerica que ha de rellenarse para desarrollar el widget. Las ventajas inherentes de esta herramienta es el ahorro de tiempo que supone para el desarrollador cuando tiene que crear varios widgets. El desarrollador se asegura de que no hay errores al crear el widgets y si los hay, tan solo tiene que corregirlos una vez, en la plantilla.
createWidget esta desarrollado en Bash Script, dado que el desarrollo se realiza en Linux.

3.5.2. createOperator

Esta herramienta es análoga a la anterior pero para operadores con las ventajas anteriormente comentadas. La diferencia fundamental con respecto a la anterior es la plantilla utilizada. Tenía sentido crear una herramienta unica para crear un tipo de elemento composicional u otro ya que la funcionalidad es la misma, pero al ser dos las plantillas que se esperaban tener y sin vistas en un futuro cercano de que vaya a cambiar, junto con la comodidad de poner un mandato u otro con el autocompletado de la consola, se decidió no usar una herramienta única con un parametro extra.
createOperator esta desarrollado en Bash Script, dado que el desarrollo se realiza en Linux.

3.5.3. packageWgt

Esta herramienta comprime todos los ficheros del directorio src del widget u operador en formato zip y lo guarda con extension wgt. De esta forma el elemento composicional objetivo ya está preparado para ser subido a la plataforma Wirecloud.
packageWgt esta desarrollado en Bash Script, dado que el desarrollo se realiza en Linux.

3.5.4. xml2rdf

Wirecloud acepta dos tipos de formato de fichero a la hora de agregar widgets y operadores. El formato xml es mcho más facil de rellenar por una persona que el rdf, por lo cual, se en el proceso de creación del widget se rellena la información necesaria en el fichero de configuracion xml y despues se genera el fichero rdf.
Hay que decir que esta herramienta, a diferencia de las demás, está desarrollada en Python ya que usa librerías desarrolladas para la plataforma y se pretende que acabe integrada en ella.

3.6. Principios de diseño de aplicaciones orientadas a mashups

Este documento es una referencia que se debiera leer antes de comenzar a desarrollar widgets ya que fija las bases de diseño de un elementos composicional de interacción con el usuario (widget). Este documento tiene dos fuentes de inspiración, por un lado la filosofía de desarrollo de widgets web que se explica en la introducción y por otro lado en los principios de diseño de automatas robóticos colaborativos. Ya que, en la plataforma WireCloud los widgets no son solo elementos independientes con una funcionalidad determinada, sino que tambien son capaces de colaborar los unos con los otros intercambiandose información para realizar tareas mucho más complejas.
Este documento es el resultado de la experiencia adquirida en el desarrollo de elementos composicionales para la plataforma WireCloud. Se puede consultar al final de esta memoria ya que forma parte de ella como anexo.

3.7. Guía de desarrollo de widgets y operadores

Esta guía surge de la necesidad de hacer que cualquier desarrollador con los conocimientos adecuados en tecnologías web sea capaz de crear elementos composicionales con una breve toma de contacto en la nueva tecnología que va a utilizar. De la misma forma, al mismo tiempo que sirve para el desarrollador neófito, será una ayuda para el desarrollador que pretende ser eficiente en el desarrollo de nuevos elementos composicionales para la plataforma de mashups Wirecloud. Este sistema de creación de elementos composicionales supone, en terminos de tiempo y esfuerzo, un ahorro significativo a cualquier desarrollador, no solo con la ayuda de las herramientas desarrolladas sino también con una metodología de desarrollo, plasmada en este documento, que se apoya en las herramientas desarrolladas y en los otros dos documentos que conforman esta serie.
El proceso de creación de un widget u operador es siempre el mismo:

1. Se crea la estructura base de directorios y ficheros.

2. Se rellena el fichero de configuración necesario para que la plataforma reconozca el elemento composicional.
3. Se define la estructura HTML base en el fichero index.html.
4. Se crea el estilo CSS para los elementos HTML que existan o que van a existir.
5. Se crea la funcionalidad con JavaScript en uno o varios ficheros del mismo tipo.

Esta guía es el resultado de la experiencia adquirida en el desarrollo de elementos composicionales para la plataforma WireCloud. Se puede consultar al final de esta memoria ya que forma parte de ella como anexo.

3.8. Guía de estilo para el desarrollo de widgets y operadores

Este documento es una referencia para aquellos que pretendan desarrollar un elemento composicional en la plataforma de mashups Wirecloud. En el se propone una estructura jerarquica de directorios y formatos de ficheros para la implementación de widgets y operadores.
Esta guía, como los documentos anteriores, es el resultado de la experiencia adquirida en el desarrollo de elementos composicionales para la plataforma WireCloud. Este documento se puede consultar al final de esta memoria ya que forma parte de ella como anexo.

Capítulo 4

Conclusiones

4.1. Resultados

El objetivo principal de este proyecto era el de crear un conjunto de documentos, que hicieran más eficiente y sistemático el proceso de creación de elementos composicionales para la plataforma de aplicaciones de mashup WireCloud.

Mediante el desarrollo de un conjunto de elementos composicionales previamente descritos en apartados anteriores, y gracias a la experiencia ganada al desarrollar dichos elementos y sus herramientas de ayuda, se ha conseguido analizar cuales son las fases que un desarrollador debe realizar para llevar a cabo la tarea de creación de un elemento composicional y en cada una de ellas identificar cuales son las partes que se pueden automatizar.

Este trabajo de identificación y análisis ha dado como principal resultado la creación de una serie de herramientas y documentos, que en el futuro cercano pueden servir de ayuda a cualquiera que quiera introducirse en el mundo de la programación de aplicaciones para mashup Web 2.0 con en la plataforma WireCloud de una forma mucha más sencilla y eficiente que la que he podido vivir yo.

La serie de tres documentos, que se ha desarrollado en este trabajo, puede servir de introducción para el nuevo desarrollador en este entorno concreto. No obstante, para afrontar el desarrollo de elementos composicionales en otras plataformas de características similares, le sería de gran utilidad haber tenido una toma de contacto con esta documentación por la filosofía subyaciente que se desprende de ellos.

Como resultado adicionales se pueden considerar la creación de un mashup funcional que proporciona al usuario un entorno con el que monitorizar el estado de una serie de maquinas de vending y un conjunto de técnicos de mantenimiento ya creado. Pero no solo eso, sino que los widgets creados por separado se pueden asociar a otros widgets existentes o que se creen en un futuro, dada la independencia de su funcionamiento y su facilidad para la asociación con otros elementos dentro de la

misma plataforma.

http://www.youtube.com/watch?v=r8_Vv_ehJSw

4.2. Conclusiones personales

La idea principal que quiero trasmitir en este espacio es el enriquecimiento personal que he conseguido al realizar este trabajo.

Evidentemente, he aprendido mucho en el contexto de tecnologías, ya que el entorno web, exige el conocimiento de multiples tecnologías y muy variadas, con distintos paradigmas y formas de enterder los problemas. Este trabajo me ha dado la posibilidad de entender paradigmas ya estudiados y descubrir algunos nuevos. A fijado conocimientos tratados durante el grado y han salido a flote conocimientos que parecía que no tenían mucho valor cuando se estudiaban pero que han resultado ser fundamentales para completar este trabajo.

El sinfín de tecnologías que se aprenden en el desarrollo web no desmerece en nada los conocimientos adquiridos en gestión de proyectos, aunque sea un proyecto corto de una duración de aproximadamente seis meses. El aprendizaje de algunas de las herramientas que más se usan en entornos profesionales, como por ejemplo GIT, que se usa para el control de versiones del núcleo de Linux es, cuanto menos, emocionante.

No solo en el aspecto puramente técnico, me ha servido este proyecto, sino también para trabajar habilidades de trabajo en equipo y colaboración. Para tratar con diferentes tipos de personas y trabajadores de distintas características, normalmente con mayores conocimientos que yo, de los cuales se puede aprender mucho. Sino que además me a servido para darme cuentas de cosas que hasta ahora no había vivido.

4.3. Pasos Futuros

Todo el trabajo realizado en este proyecto se puede aprovechar dado que tiene una aplicación eminentemente práctica y didáctica. Algunos pasos futuros lógicos serían:

- Herramienta para la creacion del fichero de configuracion: Sinceramente, rellenar un fichero de configuración a mano es algo tedioso y que se puede mejorar.
- IDE: Durante el desarrollo del proyecto lo que más se echó en falta es un Entorno de desarrollo especifico para este tipo de software.
- Integración del entorno con la plataforma: Si el entorno estuviera integrado con la plataforma sería un sitio ideal y no haría falta depender de ningún programa offline. Dentro de la misma plataforma estaría el entorno de ejecución, de pruebas y el de programación.

Apéndices

Apéndices A

Principios de Diseño

A.1. Introducción

Esta guía forma parte de una serie de tres documentos, a saber:

- "Guía de desarrollo para widgets y operadores"
- "Guía de estilo para widgets y operadores"
- "Principios de diseño para widgets".

Este documento se ha diseñado en base a la experiencia adquirida durante el proceso de diseño e implementación de un conjunto de widgets y operadores que conforman un mashup cuyo objetivo es la monitorización del estado de una serie de recursos distribuidos geográficamente en la plataforma de Mashups Wirecloud. Por ello, siempre hay que tener presente, al leer este documento, que WireCloud es la plataforma de referencia.

A.2. Objetivo

El objetivo principal de este documento es orientar al desarrollador en el diseño tanto de los nuevos elementos composicionales como de los que haya que modificar, sobre cual es la filosofía que debe seguir un elemento composicional.

A.3. Principios de Diseño de Widgets

- Reusabilidad: El widget debe tener la capacidad de ser instanciado en multiples ocasiones y en diferentes entornos.
- Centrado en el usuario: El widget debe tener como prioridad satisfacer una necesidad del usuario.

- Independencia funcional: El widget no debe depender de otros widgets para revelar su funcionalidad.
- Capacidad de colaboración: La funcionalidad del widget debe poder ser aprovechada de alguna forma por otros elementos composicionales.
- Complejidad funcional: El widget debe agrupar un pequeño conjunto de funcionalidades que estén intrínsecamente relacionadas entre si.
- Interactividad: El widget debe llevar a cabo funcionalidades interactivas tanto con el usuario como con otros elementos composicionales.
- Estandarización de las comunicaciones: Los mensajes enviados por el elemento composicional deberá seguir los estándares que existan, en tal caso. Si no existen estandares conocidos se deberá seguir un código de buenas prácticas para mensajes entre elementos composicionales.
- Autogestión: El widget debe ser capaz de detectar y gestionar sus propios errores sin propagarlos al sistema, evitando que el funcionamiento del conjunto no se vea afectado de forma crítica.
- Detección ambiental: El elemento composicional debe detectar si los tipos de datos que se pueden transmitir desde sus fuentes de datos son adecuados.
- Asociación ambiental colaborativa: Las relaciones entre widgets y operadores que estén fuertemente ligadas podrían verse dotados por una perspectiva de caja negra abstrayendo de los detalles de la comunicación entre los elementos composicionales que la conforman, dado que, implícitamente, se podrían considerar como una entidad independiente.

Apéndices B

Guía de desarrollo de widgets y operadores

B.1. Introducción

Esta guía forma parte de una serie de tres documentos, a saber:

- "Guía de desarrollo para widgets y operadores"
- "Guía de estilo para widgets y operadores"
- "Principios de diseño para widgets".

Este documento se ha diseñado en base a la experiencia adquirida durante el proceso de diseño e implementación de un conjunto de widgets y operadores que conforman un mashup cuyo objetivo es la monitorización del estado de una serie de recursos distribuidos geográficamente en la plataforma de Mashups Wirecloud. Por ello, siempre hay que tener presente, al leer este documento, que WireCloud es la plataforma de referencia.

B.2. Objetivo

Siempre que se afronte el proceso de creación de software se debe tener en cuenta que es una tarea creativa y a la vez mecánica. Al mismo tiempo en el que se plantean problemas que exigen del ingenio y la creatividad del ingeniero para resolverlos, se puede observar que la mayor parte del trabajo es repetitiva y se puede automatizar. También es necesario ser consciente de que los desarrolladores cometen errores y suelen repetirlos más o menos frecuentemente. Para tratar de evitar que ciertos errores frecuentes ocurran se debe intentar hacer que el trabajo sea sistemático y, en la medida de lo posible, automático, mejorando de esta manera la eficiencia en el desarrollo. Lo propuesta que a continuación se realiza es una serie de etapas o pasos que deberían completarse, ya sea en el orden propuesto o en el que elija el

desarrollador.

Por tanto, esta guía tiene 2 objetivos fundamentales. Por un lado, ser una referencia para aquellos desarrolladores que se inician en la creación de elementos composicionales sin tener una experiencia previa en este campo. Y por otro lado, esta guía debe servir a los desarrolladores que ya han trabajado creando elementos composicionales en otras plataformas o para Wirecloud a desarrollar de una forma más eficiente widgets y/u operadores.

B.3. Desarrollo de un elemento composicional

1. Definir los requisitos de los widgets/operadores.
   - *a*) Recopilar los requisitos de las diversas fuentes.
   - *b*) Clasificar los distintos tipos requisitos.
     1) Definir requisitos generales.
     2) Definir requisitos específicos de vista e interacción con el usuario.
     3) Definir requisitos específicos de funcionalidad.
     4) Definir requisitos específicos de comunicación/interacción con otros widgets y con la plataforma.
2. Realizar el diseño del widget/operador.
   - *a*) Vistas (no aplica con operadores).
     1) Realizar un croquis/boceto de cada una de las vistas que debiera contener el widget.
     2) Definir con mayor precisión los elementos que debería tener cada una de las vistas.
   - *b*) Funcionalidad.
     1) Decidir en que vistas se van a implementar qué funcionalidades.
     2) Definir cuales de los elementos de cada vista va a dar qué funcionalidades definidas en el documento de requisitos.
   - *c*) Interacciones.
     1) Definir los tipos de datos que pueden ser tratados por el widget/operador dada su funcionalidad.
     2) Definir los tipos de datos que va a ser capaz de servir el widget/operador y que pueden ser de utilidad para otros elementos composicionales.
3. Crear directorio y ficheros necesarios para implementar el widget/operador.

a) Crear la jerarquía de ficheros y directorios de la plantilla(ANEXO 1).

b) Definir los metadatos del widget en el fichero config.xml según la documentación de la plataforma (ANEXO 2).

c) Sustituir images/catalogue.png por una imagen representativa de la funcionalidad del widget/operador.

d) Crear "el esqueleto" de las vistas en el fichero index.html según las necesidades del widget. Omitir en el caso del operador (no tiene vista).

e) Dar funcionalidad al widget/operador mediante el fichero main.js y las librerías que para ello fuera necesario.

f) Dar estilo al widget con style.css y las hojas de estilo que se necesitaran.

4. Empaquetar el widget/operador en el formato wgt.

   a) Comprimir todos los ficheros del widget en formato zip.

   b) Cambiar la extensión del nuevo fichero zip a wgt.

5. Subir el widget/operador a la plataforma WireCloud.

   a) Una vez autenticado en wirecloud, dentro del catálogo, usar la herramienta de subida de widgets desde un paquete.

6. Test.

   a) Comprobar en la vista del catálogo dentro de WireCloud.

      1) La descripción es correcta.
      2) La imagen es correcta.

   b) Comprobar en la vista de wiring dentro de WireCloud.

      1) Tiene el numero de entradas y salidas que debiera tener.
      2) Las entradas y salidas están correctamente etiquetadas y tienen descripción.

   c) Comprobar en la vista del editor dentro de WireCloud.

      1) La vista del widget se ve correctamente.
      2) Comprobar que todas las vistas definidas en el diseño existen y son como debieran ser.

   d) Comprobar que el intercambio de datos a través del wiring es correcto.

      1) Comprobar que los datos generados son los adecuados.
      2) Comprobar que datos correctos llegados desde otro widget/operador son tratados adecuadamente.
      3) Comprobar que datos incorrectos llegados desde otro widget/operador son tratados adecuadamente.

   e) Comprobar que la funcionalidad del widget es realmente la diseñada.

1) Comprobar las interacciones del usuario con la interfaz.
2) Cada uno de los eventos funcionan como deberían funcionar.

7. Documentar el widget/operador.

   a) Describir la funcionalidad general del widget/operador.

   b) Describir las entradas y salidas del widget/operador.

      1) Indicar el tipo de datos que se tratan.
      2) Dar un ejemplo en cada entrada o salida.

   c) Describir las funcionalidades detalladamente con material multimedia (capturas de pantalla, videos, etc) para ayudar al usuario.

   d) Sugerir una serie de widgets/operadores con los que puede conectarse nuestro widget/operador.

B.4. Herramientas

Para ayudarnos en la tarea de automatizar y sistematizar el proceso de desarrollo de los elementos composicionales que creamos, se propone la creación y el uso de una serie de herramientas:

1. Creator. Crea un proyecto nuevo con su jerarquía de ficheros, un conjunto mínimo de los directorios y ficheros básicos que debe contener el widget o el operador. De esta forma, el desarrollador no tiene que crear todos los ficheros manualmente.

2. Importer. Importa todas las dependencias necesarias del proyecto en el fichero HTML y el XML. Sirve para comprobar si se han incluido todos los ficheros que hay en el proyecto o no. Ayuda al desarrollador a recordar si ha incluido todos los ficheros javascript y css en el fichero HTML y la dirección de la imagen y el html en el XML.

3. XML Editor. Para facilitar al desarrollador del widget la edición del fichero de configuración del widget/operador sería aconsejable usar un editor visual que abstrajera de los detalles del formato el contenido esencial del documento.

4. Packager. Comprime todos los ficheros del proyecto en un fichero y le da la extensión ".wgt" aceptada por la plataforma.

5. Deployer. Despliega el paquete ".wgt" en la plataforma definida.

Apéndices C

Guía de estilo para widgets y operadores

C.1. Introducción

Esta guía forma parte de una serie de tres documentos, a saber:

- "Guía de desarrollo para widgets y operadores"
- "Guía de estilo para widgets y operadores"
- "Principios de diseño para widgets".

Este documento se ha diseñado en base a la experiencia adquirida durante el proceso de diseño e implementación de un conjunto de widgets y operadores que conforman un mashup cuyo objetivo es la monitorización del estado de una serie de recursos distribuidos geográficamente en la plataforma de Mashups Wirecloud. Por ello, siempre hay que tener presente, al leer este documento, que WireCloud es la plataforma de referencia.

Dado que la naturaleza de cada uno de los elementos composicionales es distinta, se ha hecho una clasificación intrinseca de estos elementos en este documento.

C.2. Objetivo

El principal objetivo de esta guía es crear un modelo base del que se pueda partir atendiendo a ciertos criterios de calidad y diversas razones técnicas que harán del desarrollo de los elementos composicionales una tarea mucho más mecánica y automatizable, de forma que el trabajo sea más sistemático y, en consecuencia, más eficiente.

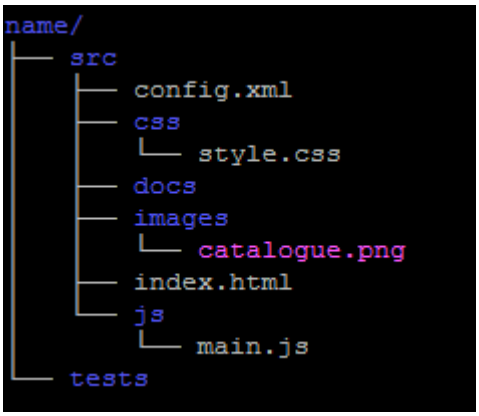

Figura C.1: Jerarquía de directorios para widgets

C.3. Estilo para un elemento composicional

C.3.1. Tipo widget

La jerarquía de directorios y ficheros de un widget será como la que se muestra en la figura C.1.

- name es el directorio donde se encuentran todos los ficheros del widget.
- src es el directorio donde se encuentra el código y la documentación del widget.
- css es el directorio donde se encuentran los ficheros de estilo.
- style.css es el fichero css principal, donde se encuentran las reglas que definen el estilo de la vista del widget.
- docs es el directorio donde se encuentra la documentación auto-contenida del widget, que se podrá consultar a través de la plataforma.
- images es el directorio donde se encuentran las imágenes que usa el widget.
- catalogue.png es la imagen que se muestra en el catalogo de la plataforma.
- js es el directorio donde se encuentran todos los ficheros que contienen el código Javascript.
- main.js es el fichero donde se encuentra el código javascript principal.
- config.xml es el fichero donde se encuentra la descripción que necesita la plataforma para registrar el widget.
- index.html es el fichero donde se encuentra el html del widget.
- tests es el directorio que contiene las pruebas del widget.

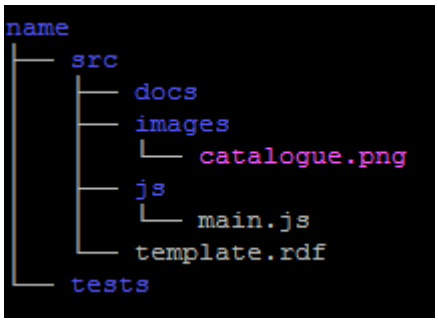

Figura C.2: Jerarquía de directorios para operadores

C.3.2. Tipo operador

La jerarquía de directorios y ficheros de un operador será la que se muestra en la figura C.2.

- name es el directorio donde se encuentran todos los ficheros del operador.
- src es el directorio donde se encuentra el código y la documentación del operador.
- docs es el directorio donde se encuentra la documentación autocontenida del operador, que se podrá consultar a través de la plataforma.
- images es el directorio donde se encuentran las imágenes que usa el operador.
- catalogue.png es la imagen que se muestra en el catalogo de la plataforma.
- js es el directorio donde se encuentran todos los ficheros que contienen el código Javascript.
- main.js es el fichero donde se encuentra el código javascript principal.
- template.rdf es el fichero donde se encuentra la descripción que necesita la plataforma para registrar el operador.
- tests es el directorio que contiene las pruebas del operador. “

Bibliografía

[1] https://igoogle.com/

[2] http://en.wikipedia.org/wiki/JavaScript

[3] http://es.wikipedia.org/wiki/Mozilla_Firefox

[4] http://es.wikipedia.org/wiki/Gecko_(motor_de_renderizado)

[5] http://es.wikipedia.org/wiki/Acid3

[6] http://es.wikipedia.org/wiki/Google_Chrome

[7] http://es.wikipedia.org/wiki/WebKit

[8] http://es.wikipedia.org/wiki/HTML

[9] http://www.w3c.es/

[10] http://www.w3.org/TR/CSS/

[11] http://twitter.github.com/bootstrap/

[12] http://en.wikipedia.org/wiki/Yahoo!_Pipes

[13] http://es.wikipedia.org/wiki/OpenSocial

[14] http://es.wikipedia.org/wiki/Mashup_(aplicación_web_híbrida)

[15] http://es.wikipedia.org/wiki/Mashup_(aplicación_web_híbrida)

[16] http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)

[17] http://www.w3.org/2010/POI/

[18] http://en.wikipedia.org/wiki/VCard

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Tue Feb 11 23:08:34 CET 2014
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)