
EnrutaBus: Aplicación urbana de carácter OpenData para la búsqueda de rutas



Universidad de Huelva

Proyecto Fin de Carrera Ingeniero en Informática

Autor: Sebastián González Aranda
Tutor: Gonzalo Antonio Aranda Corral

Escuela Politécnica Superior de la Rábida
Universidad de Huelva

Septiembre 2011

A mis padres

Cuéntame y lo olvidaré.

Enséñame y lo recordaré.

Involúcrame y lo comprenderé.

Resumen

EnrutaBus es un proyecto en el que partiendo de la filosofía que envuelve a los *Open Data*, desarrollamos una aplicación web que nos proporciona el camino a seguir entre dos paradas de autobús dadas.

Esta filosofía de reutilización y apertura de datos es sustentada gracias a los datos ofrecidos por un servicio web de la red de autobuses de Gijón. De esta forma, nuestro proyecto queda enmarcado en el campo de la informática urbana, el cual, trata sobre el uso (y generación) de información física y digital sobre la ciudad como fuente de nuevas aplicaciones que puedan ser usadas por el ciudadano a través de dispositivos fijos o móviles.

La metodología de desarrollo de nuestro proyecto queda recogida en tres aspectos fundamentales:

1. Recuperación de datos del servicio web de los que se nutrirá nuestra aplicación.
2. Presentación de los datos en formato web/móvil.
3. Creación de una aplicación que aporte una funcionalidad añadida a la información presente. Un planificador de rutas.

Luego, apoyándonos en los datos extraídos, obtenemos una plataforma enfocada a los usuarios de la red de autobuses de la comunidad de Gijón accesible desde cualquier dispositivo fijo o móvil al servicio del ciudadano.

Palabras clave

Open data, Gijón, autobuses, transporte, informática urbana.

Índice General

1. Introducción.....	7
1.1. Motivación.....	8
1.2. Objetivo	9
1.3. Trabajos relacionados	9
2. Open data.....	11
2.1. Introducción.....	11
2.2. Fuentes Open Data.....	31
2.3. Agentes Infomediarios.....	49
2.4. Productos y servicios open data.....	51
2.5. Usuarios de Open data	53
2.6. Clasificación del Opendata	56
2.7. Situación actual de los Open Data en España.....	57
2.8. Catálogos Open Data	62
3. Open Data Gijón.....	65
3.1. Introducción.....	65
3.2. ¿Qué es <i>datos.gijon.es</i> ?.....	66
3.3. ¿Qué es y para qué sirve un servicio web?	67
3.4. Estructura del servicio web BusGijón	68
4. Arquitectura del proyecto	77
4.1. Descripción general	77
4.2. Tecnologías y herramientas	82
5. Análisis y persistencia de datos	85
5.1. Análisis de los datos	85
5.2. Base de datos	90
6. Recuperación de datos	97
6.1. Descripción	97
6.2. Arquitectura lógica	99
7. Aplicación web.....	103
7.1. Descripción	104
7.2. Arquitectura lógica	107

7.3. Funcionalidad móvil	116
8. Planificador de rutas.....	119
8.1. Creación del grafo.....	120
8.2. Búsquedas empleadas	124
8.3. Rendimiento.....	145
8.4. Ruta entre dos paradas	154
8.5. Puente PHP/Java.....	158
9. Conclusiones y trabajos futuros.....	159
9.1. Trabajo futuro	159
9.2. Valoración final	162
Anexos.....	165
Bibliografía.....	167

Índice de cuadros

Figura 1: Aspectos del enfoque Open.....	12
Figura 2: Cadena de valor Open Data	14
Figura 3: Estandar de clasificación de Berners-Lee	24
Figura 4: Linking Open Data cloud diagram.....	28
Figura 5: Ejemplo de búsqueda en el buscador facetado de <i>CTIC</i>	30
Figura 6: Mapa de las fuentes analizadas por el origen de los datos y su ámbito	32
Figura 7: Mapa de iniciativas Open Data a nivel mundial	33
Figura 8: Mapa de iniciativas Open Data en España.....	38
Figura 9: Distribución de fuentes de datos públicas AGE por organismo	39
Figura 10: Europa y Africa.....	62
Figura 11: Asia	62
Figura 12: América.....	63
Figura 13: Esquema de comunicación en un servicio web.....	67
Figura 14: Página principal del servicio web busGijon.....	68
Figura 15: Consulta realizada al servicio web - Líneas.....	69
Figura 16: Consulta realizada al servicio web - Paradas	69
Figura 17: Consulta realizada al servicio web - Trayectos.....	70
Figura 18: Consulta realizada al servicio web - ParadasTrayectos	70
Figura 19: Consulta realizada al servicio web - LlegadasParadas.....	71
Figura 20: Consulta realizada al servicio web - Posiciones	72
Figura 21: Consulta realizada al servicio web - estadoBus	72
Figura 22: Consulta realizada al servicio web - estadoLinea	73
Figura 23: Consulta realizada al servicio web - estadoParada	73
Figura 24: Mapa mundial de las zonas UTM	86
Figura 25: Uso de la librería Jcoord	87
Figura 26: Representación del concepto: desfase.....	88
Figura 27: Depuración de datos - Dirección.....	89
Figura 28: Resultado de depuración de datos - Dirección.....	89
Figura 29: Diagrama entidad-relacion.....	91
Figura 30: Diagrama de integridad referencial.....	94
Figura 31: Consumidor de datos - Relaciones entre sus principales clases.....	99
Figura 32: Pantalla de inicio de la aplicación web	104
Figura 33: Menú proyecto - aplicación web.....	105
Figura 34: Menú WebServices - aplicación web.....	105
Figura 35: Menú visualizar en Gmaps - aplicación web	106
Figura 36: Menú planificador - aplicación web	106
Figura 37: Consulta de excepciones en la aplicación web	106
Figura 38: Diagrama aplicación web 1.....	107
Figura 39: Diagrama aplicación web 2.....	107
Figura 40: Vista de la tabla 'Lineas'	108

Figura 41: Vista de la tabla 'Paradas'	109
Figura 42: Vista de la tabla 'Trayectos'	109
Figura 43: Vista de la tabla 'Tramos'	110
Figura 44: Vista de la tabla 'Autobus'	110
Figura 45: Vista de la tabla 'Localizacion'	111
Figura 46: Vista de la tabla 'Carrera'	111
Figura 47: Imagen descriptiva para la visualización de líneas	112
Figura 48: Imagen descriptiva del histórico de localización de un bus.....	113
Figura 49: Imagen descriptiva del planificador de rutas	114
Figura 50: Esquema de relaciones entre ficheros PHP de la web	115
Figura 51: Estructura 'esMovil.php' en el servidor.....	116
Figura 52: Vista de Opera mobile.....	117
Figura 53: Página de inicio - Vista móvil.....	118
Figura 54: Página de consulta - Vista móvil	118
Figura 55: Página de consulta 2 - Vista móvil	118
Figura 56: Listado de paradas entre dos paradas dadas.....	119
Figura 57: Consulta de paradas a la base de datos	120
Figura 58: Consulta de líneas a la base de datos	120
Figura 59: Consulta de trayectos a la base de datos	120
Figura 60: Consulta de tramos a la base de datos.....	121
Figura 61: Diagrama de clases del paquete Grafo	121
Figura 62: Atributos y métodos del algoritmo A*.....	126
Figura 63: Pseudocódigo de método 'algoritmoPathfinding'	128
Figura 64: Pseudocódigo del método 'verticeConMenorF'	129
Figura 65: Pseudocódigo del método 'setEcuacion'.....	130
Figura 66: Pseudocódigo del método 'recursivo'	131
Figura 67: Pseudocódigo del método 'obtenerRecorrido'	131
Figura 68: Ilustración de la búsqueda en profundidad	133
Figura 69: Atributos y métodos de la búsqueda en profundidad.....	134
Figura 70: Pseudocódigo del método 'getRecorridoProfundidad'	136
Figura 71: Atributos y métodos del algoritmo de Floyd	139
Figura 72: Pseudocódigo del método 'metodoBase'	141
Figura 73: Pseudocódigo del método 'inicializarCostes'	142
Figura 74: Pseudocódigo del método 'getValorEtiqueta'	143
Figura 75: Pseudocódigo del método 'getCaminoFloyd'	144
Figura 76: Consulta de linea - version 1 línea.....	146
Figura 77: Consulta de trayecto - versión 1 línea.....	146
Figura 78: Consulta de parada - versión 1 línea.....	147
Figura 79: Medición de memoria - A* - ver 1 línea.....	147
Figura 80: Medición de memoria - Profundidad - ver 1 línea.....	148
Figura 81: Medición de memoria - Floyd - ver 1 línea	148
Figura 82: Paradas vs Memoria - versión 1 línea	149
Figura 83: Medición de memoria – A* - ver todas	151
Figura 84: Medición de memoria - Profundidad - ver todas	151

Figura 85: Medición de memoria - Floyd - ver todas.....	152
Figura 86: Parada vs Memoria - versión todas las líneas	152
Figura 87: Múltiples opciones - transiciones.....	155
Figura 88: Opción única - transiciones.....	155
Figura 89: Esquema PHP/JavaBridge	158

1. Introducción

Los organismos públicos generan una enorme cantidad de informaciones sobre cada aspecto de nuestras vidas; desde cómo se destinan los impuestos que pagamos hasta las estadísticas sobre accidentes de bicicleta en las vías de los centros urbanos. Gran parte de estos datos nunca sale a la luz y permanece en la oficina de un burócrata cogiendo polvo. Pero eso está cambiando gracias a la apertura de datos u *'open data'*.

El objetivo es proporcionar toda la información pública a la ciudadanía, en un formato estructurado, fácil de manipular, convertir en servicios y crear valor añadido. Ésta apertura de datos permite la construcción de aplicaciones de valor añadido a partir de la información liberada como fuente. Sobre estas aplicaciones se pueden construir una serie de herramientas para la consulta/visualización/acceso a los datos y reutilizarlas para otras aplicaciones ya existentes, lo que se llama en inglés *'mashup'*, que se podría traducir por *'remezcla o aplicación web híbrida'* y que da una gran potencia y flexibilidad a estas aplicaciones.

Las aplicaciones pueden ser hechas por la administración u otros (particulares, organizaciones, empresas, colectivos ...) en el momento que se libera su acceso y pueden tener objetivos muy diferentes: desde el uso personal por parte de los ciudadanos, por ejemplo la recepción en el móvil de retenciones de transporte u ofertas próximas, pasando por asociaciones de enfermos respiratorios que controlan los índices de contaminación o empresariales que controlan los transportistas con navegadores GPS a los vehículos y predecir los tiempos de entrega.

Otros ejemplos de la riqueza y la variedad de las aplicaciones que se pueden hacer a partir de los datos públicos abiertas son los mapas y gráficos interactivos que se utilizan en el llamado periodismo de bases de datos y en numerosas servicios de búsqueda, como un buscador de equipamientos que permite localizar escuelas, hospitales o centros deportivos, entre otros. También forman parte de las aplicaciones generadas con datos abiertas los titulares incorporados vía RSS blogs, webs o aplicaciones móviles que muestran de esta manera contenidos facilitados automáticamente por la administración.

Una de las aplicaciones más inmediatas y de uso frecuente por los ciudadanos de una ciudad; es la relativa a la red de transporte público de dicha ciudad. Conocer la información en tiempo real sobre las líneas y paradas de autobuses de una ciudad, nos presenta un servicio sobre el que los ciudadanos podrán conocer las paradas más cercanas a su ubicación (a través de una lista, mapa de Google o realidad aumentada) y saber la posición y tiempo de llegada de los autobuses que se dirigen hacia cualquier parada.

Los datos generados por la red de transporte público (autobuses) de Gijón son la base sobre la que implementar nuestra aplicación. Estos datos son tomados de un servicio web que tiene a disposición del público la ciudad de Gijón, *BusGijón*¹. Servicio web sobre el que construir una aplicación accesible desde cualquier tipo de dispositivo

¹ <http://docs.gijon.es/sw/busgijon.asmx>

fijo o móvil que permita consultar a tiempo real las particularidades que de una red de autobuses puede esperarse.

Una de las principales aplicaciones a construir sería un programa consumidor de datos. Este consumidor de datos actuaría sobre el servicio web antes mencionado, almacenando los datos en nuestra base de datos particular diseñada a demanda del futuro funcionamiento de nuestra aplicación. El programa consumidor de datos recopilará datos, de manera ininterrumpida y registrando todo tipo de fallos, sujeto a una determinada duración.

Otra gran tarea fue conseguir tener accesibles y de manera ordenada los datos recopilados. Esto es, mostrando información sobre líneas, paradas, trayectos, tramos, autobuses, localizaciones y carreras. Todo ello ubicado en una web accesible desde cualquier tipo de dispositivo, ya sea fijo o móvil. Pudiendo realizar determinados filtros a los datos a consultar, a la vez que mantenemos una alta cohesión entre los datos mostrados. Todas estas particularidades han sido enfocadas para que el futuro usuario pueda encontrarse cómodo en todo momento consultando nuestros datos.

Por último, se propuso añadirle una funcionalidad característica a la web. Esta funcionalidad encontraría la ruta a seguir entre dos paradas dadas.

1.1. Motivación

Los datos del servicio web *busGijón* (datos de los que se nutrirá el proyecto que aquí venimos exponiendo) ya se encuentran utilizados en forma de aplicación¹ que muestra en tiempo real, información sobre las líneas y paradas de autobuses de la ciudad. Esta aplicación web está caracterizada por representar (haciendo uso de la API de Google Maps) cualquier línea de las que se encuentran en la red de transportes de Gijón. Así, puede consultarse a tiempo real las distintas aproximaciones de los autobuses presentes en la línea consultada. Por último, nos ofrece la posibilidad de consultar cualquier tipo de parada apoyándonos en un servicio de búsqueda con sugerencias de éstas.

Nuestra motivación, tras observar el funcionamiento de esta aplicación web, viene impulsada por la necesidad de cubrir ciertas carencias observadas en ella. Esto es, ofrecer una presentación completa de los datos presentando información acerca de todas y cada unas de las partes que componen la red de autobuses citada. Ofreciendo la información de manera más directa y accesible valiéndonos de la potencia expresiva de las tablas, así como de las representaciones gráficas sobre mapas. Todas estas mejoras son ofertadas y presentadas en formato web accesible tanto desde dispositivos fijos como móviles.

¹ <http://datos.gijon.es/aplicaciones/busgijon/busgijon.jsp>

La información procedente del servicio web, de la que se nutre nuestra aplicación web, debe ser capturada de alguna forma, a la vez que depurada y adaptada para su uso por nuestra aplicación web. Es decir, habremos de construir un programa consumidor de datos que, además de capturarlos y depurarlos, deba de hacerlos persistir.

Finalmente, una de las grandes motivaciones de este proyecto ha sido el cubrir una necesidad real no presente en la aplicación¹ web citada: Responder a la necesidad de cualquier usuario de la red de autobuses de Gijón de conocer qué líneas ha de tomar para ir desde una parada hasta otra, así como conocer las paradas existentes entre esas dos paradas y los transbordos a tomar. Es decir, se hacía imprescindible el insertar en nuestra aplicación web un modulo/herramienta/opción planificadora de rutas. Necesidad real de la vida cotidiana trasladada a nuestro proyecto, y por tanto principal elemento diferenciador para con la aplicación web de la que en un principio venimos haciendo referencia.

1.2. Objetivo

El objetivo pues, consistía en la realización de una aplicación web, accesible desde cualquier tipo de dispositivo, que nos permitiese la consulta y planificación de viajes en la red de transportes de Gijón. Este planificador o enrutador de viajes entre dos paradas dadas supone la característica diferenciadora de nuestra aplicación a nivel nacional, dentro del ámbito de la red de transportes públicos (autobuses).

Además, debíamos de hacer posible la creación de un programa consumidor de datos que nos permitiese la creación de nuestra propia base de datos; base de datos que permitiría sustentar nuestra aplicación web.

1.3. Trabajos relacionados

A continuación exponemos diversos proyectos dignos de mención, todos ellos relacionados con la movilidad urbana en el ámbito de los transportes públicos:

- **Roadify:**

Galardonada como una de las mejores aplicaciones de la ciudad de New York en el año 2011. *Roadify* es una aplicación dedicada no solo a informar del estado del tráfico rodado y la situación del transporte público en la ciudad, sino también a aconsejar entre las diferentes opciones. Apoyada en Google maps, nos ofrece una interfaz amigable e intuitiva para el usuario.

¹ <http://datos.gijon.es/aplicaciones/busgijon/busgijon.jsp>

- **NextStop:**

Al igual que ocurre con *Roadify*, *NextStop* es una aplicación para iPhone dirigida a los usuarios de la red de metro de New York. A simple vista el usuario podrá conocer los tiempos y que líneas se aproximan a la posición en la que él se encuentre. Con esta información a la mano del usuario, es fácil de averiguar la ruta más corta entre el punto A y el B. Además, la aplicación está disponible de manera gratuita en la App Store de iTunes.

- **infobustussam:**

Aplicación que da cobertura a la red de transporte público de la ciudad Sevilla, más concretamente su red de autobuses. Nos encontramos ante una aplicación destinada a la consulta de tiempos llegadas de un autobús a una determinada parada, además de la visualización y consultas de las líneas de la ciudad. Ésta, representa una robusta aplicación de uso para los ciudadanos de la ciudad, pero que en ocasiones se ve mermada por las múltiples interrupciones de su servicio web.

- **Bus Gijón:**

Aplicación del ayuntamiento de Gijón para iPhone y iPad. Muestra en tiempo real información sobre las líneas y paradas de autobuses de la ciudad. Esta aplicación pone a disposición de los ciudadanos una información completa y actualizada sobre los autobuses de la ciudad de Gijón. La aplicación disponible en Apple Store, forma parte de la iniciativa '*open data*', promovida por el ayuntamiento de Gijón y que pone a disposición pública la información que gestiona como base para la mejora de la participación ciudadana. A destacar el excelente servicio web del que se nutre la aplicación.

- **Emtusahuelva:**

Reciente aplicación web que da soporte a la red de autobuses de la ciudad de Huelva. En ella, podremos consultar los itinerarios de las distintas líneas existentes en la ciudad, todo ello haciendo uso de Google maps (al igual que en el resto de aplicaciones ya citadas). Además tenemos la opción de consultar los tiempos de llegada, por parte de un autobús, a una determinada parada.

2. Open data

2.1. Introducción

2.1.1. ¿Qué es y cómo surgió el open data?

El *open data* es una filosofía y una práctica que requiere que ciertos datos sean de libre acceso para todo el mundo, sin limitaciones técnicas o legales.

Para acometer el ejercicio de sus servicios, las administraciones públicas disponen de muchos conjuntos de datos que, en buena parte, son o pueden estar abiertos a la sociedad sin que implique ningún problema de privacidad.

Estos datos pueden ser reutilizados para otras finalidades, lo cual constituye el principal objetivo de los procesos de apertura de información pública.

Se define la **apertura de datos públicos** (también conocido por las palabras inglesas *open data*) como el proceso que pone al alcance de la sociedad los datos públicos de los que dispone la Administración, en formatos digitales, estandarizados y abiertos, siguiendo una estructura clara que permita su comprensión y reutilización.

“Open data es el proceso que pone al alcance de la sociedad los datos públicos de los que dispone la Administración, en formatos digitales, estandarizados y abiertos”

El origen de este movimiento se remonta al 21 de Enero de 2009. En su primer día en el despacho oval, el presidente de los EE.UU. Barack Obama envió un comunicado¹ a los directores de las agencias y departamentos federales sobre *Transparencia y Gobierno Abierto* en el que se resaltaba que toda la información producida por el gobierno de los EE.UU. debe hacerse pública “por defecto”.

Una de las consecuencias de ese comunicado fue la posterior creación de *data.gov*², el sitio donde se encuentra el catálogo de datos abiertos del gobierno federal. Sin duda, este sitio puso el listón sobre el que se medirían los demás.

Las acciones de la iniciativa¹ de gobierno abierto de la Casa Blanca han dinamizado el sector público a nivel mundial. Cada vez son más los gobiernos que anuncian y

¹ <http://edocket.access.gpo.gov/2009/pdf/E9-1777.pdf>

² <http://www.data.gov/>

comienzan iniciativas de transparencia y gobierno abierto y que publican sus catálogos de datos. La iniciativa² europea sobre la reutilización de la información del sector público (en base a la directiva³ europea al respecto) está jugando un papel muy importante a este lado del Atlántico.

A través del liderazgo de la iniciativa⁴ mundial de eGovernment en W3C y la participación en iniciativas como las arribas mencionadas, **CTIC** (ver *Marco tecnológico* en este documento) ha ido desarrollando métodos y técnicas para la puesta en marcha y explotación de sitios “*data.gov.**” a nivel nacional y mundial.

2.1.1.1. Contexto del Open Data dentro de lo ‘open’

El Open Data es una faceta de los ecosistemas abiertos ‘Open’ que incluyen al **Open Source** (o puesta en común de código de aplicaciones mediante licencias copyleft), el **Open Innovation** o mecanismos de cooperación en el ámbito I+D+i y al **Open Governance**, una de cuyas facetas, es el Open Data que es analizado en esta overview. En la siguiente figura usted podrá obtener una visión más amplia de su situación.

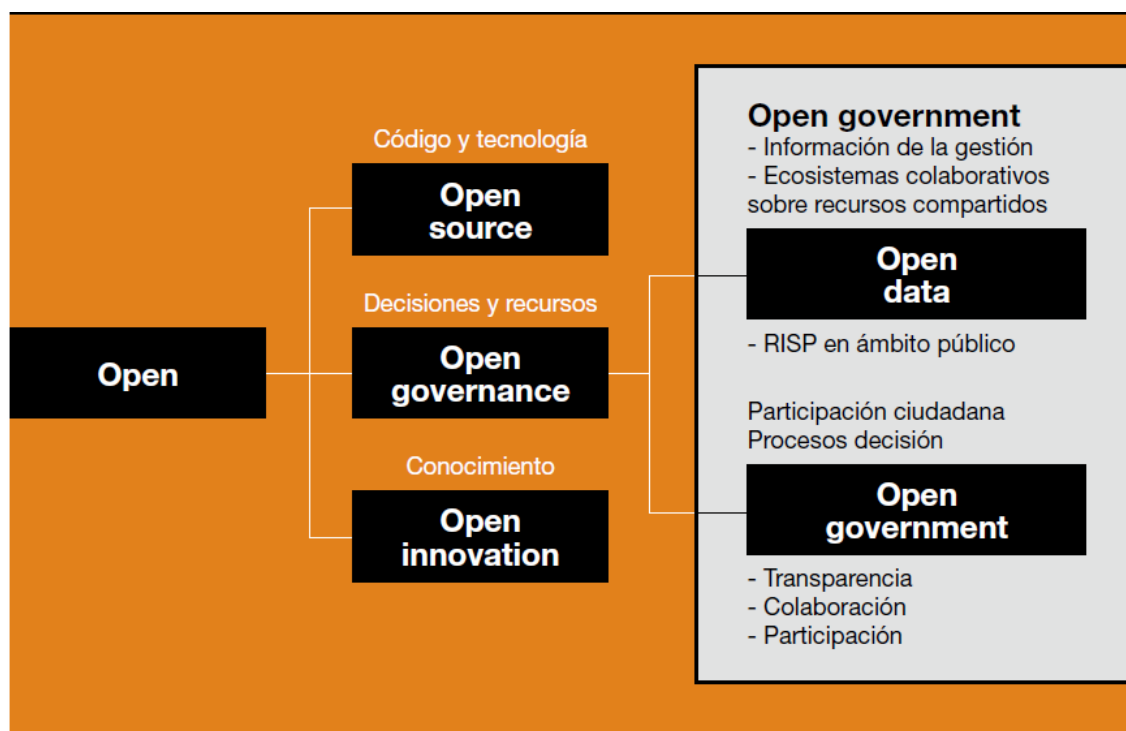


Figura 1: Aspectos del enfoque Open

¹ <http://www.whitehouse.gov/open>

² http://ec.europa.eu/information_society/policy/psi/index_en.htm

³ http://ec.europa.eu/information_society/policy/psi/docs/pdfs/directive/com09_212/com09_212_en.pdf

⁴ <http://www.w3.org/egov/>

2.1.2. ¿Cómo y por qué se liberan los datos?

El por qué de la liberalización de los datos, o lo que es lo mismo, el objetivo que guarda tras de sí éste movimiento recae casi en su totalidad sobre las administraciones públicas. Y es que, Desde ellas se ha visto la **necesidad** que tienen las personas de **acceder a** cierta **información** hasta ahora no disponible. Por eso tienen la voluntad de avanzar hacia un gobierno más abierto, basado en los valores de **transparencia, participación, servicio y eficiencia**.

También se quiere promover la creación de valor con la reutilización de la información pública, facilitar la ordenación interna de los sistemas de información dentro de las Administraciones y fomentar la interoperabilidad entre servicios del sector público.

Al mismo tiempo, la **liberación de datos** supone también un **potencial económico** ya que constituye una base esencial para muchos productos de información digital. Es imprescindible asegurar la reutilización eficiente de la información pública para poder aprovechar todo su potencial y desarrollar nuevos productos, servicios y mercados que logren un mayor desarrollo económico y mayor creación de puestos de trabajo.

Pero, ¿cómo ponemos estos datos a disposición del ciudadano?. Tal y como recomienda el W3C en su última guía¹ de publicación, se tienen que liberar datos útiles y en formatos reutilizables. Cualquier formato es bienvenido, pero cuanto mejor estructurados y enriquecidos estén los datos, más fácil será reutilizarlos y construir aplicaciones que los traten automáticamente. Debido a esto existen formatos y proyectos de datos abiertos que se consideran de menor valor que otros.

Según la clasificación² propuesta por Tim Berners-Lee (ver *Marco Tecnológico en este documento*), los formatos más adecuados son el RDF, basado en XML y susceptible de pasar fácilmente al siguiente nivel llamado “datos enlazados” (*linked data*), que a su vez es la base técnica para alimentar la denominada web semántica, un estándar en el que cada dato contiene información asociada que lo relaciona automáticamente con otros.

Hacer notar que el W3C también recomienda crear un catálogo con la descripción precisa de todos y cada uno de los conjuntos de datos (*data sets*) que libera un gobierno y, a la vez, libera también el propio catálogo en formato RDF.

¹ <http://www.w3.org/TR/2009/WD-gov-data-20090908/>

² <http://inkdroid.org/journal/2010/06/04/the-5-stars-of-open-linked-data/>

Por último, es oportuno recordar que lo que se ha de liberar son datos, no documentos; esta afirmación implica que no tiene sentido abrir archivos en formatos pdf, Word o similares.

Un servicio open data no es un servicio open access, sino un sitio donde se ofrecen datos en bruto o en crudo (raw data) para poder ser reutilizados y crear otros servicios públicos.

“Un documento es una colección de datos reunida de forma temporal para un fin determinado donde se mezclan datos e información de presentación de dichos datos, información que no es interesante cuando el objetivo es la reutilización de los datos”

Por este motivo los documentos no pueden ser ofrecidos como información en un portal Open Data.

2.1.2.1. Cadena de valor del Open Data

Desde que los datos son generados, hasta que son liberados, reutilizados y puestos a disposición del usuario final; habremos pasado por una serie de fases. Estas fases o ‘cadena de montaje’ quedan representadas en la siguiente figura:



Figura 2: Cadena de valor Open Data

El primer componente de la cadena son las fuentes de datos, que pueden tener origen en organizaciones públicas o privadas, dividiéndose estas últimas en organizaciones con ánimo de lucro (empresas) o sin él (funciones y ONG).

El segundo elemento de esta cadena de valor son los esquemas legales y los mecanismos técnicos que permitan la reutilización, que incluye una evaluación legal de las fuentes, la definición de licencias de reutilización, etc (ver capítulo). Así mismo hay que implantar la infraestructura tecnológica para la publicación y puesta a disposición del público de acuerdo con las condiciones legales establecidas.

Los infomediarios, son los creadores de los productos y servicios basados en las fuentes. De nuevo pueden ser entidades de carácter público, privado o del tercer sector y, por tanto, sus objetivos pueden variar desde el mero interés económico, hasta el desarrollo de la propia sociedad, o la promoción de la transparencia o la participación, entre otros.

Basados en estos datos los infomediarios pueden generar, bien nuevos conjuntos de datos mediante el tratamiento y la combinación, aplicaciones para dispositivos móviles, tablets u otros dispositivos, o puramente servicios, bien sean disponibles en línea o como asesoría, que constituyen el cuarto elemento de esta cadena de valor.

El último eslabón de esta cadena son los usuarios, bien de ámbito ciudadano (con modelos de negocio gratuitos o sufragados por publicidad) o desde el ámbito profesional, que utilizan los productos como una parte de su propio negocio.

2.1.3. Reutilización y sector público

Ya sea denominado “Open Government Data” (OGD) o “Reutilización de la Información del Sector Público” (RISP) nos encontramos con que la **Reutilización de la Información del Sector Público** es el objetivo principal de la iniciativa Open Data.

La reutilización de la información del sector público consiste en la puesta a disposición de la ciudadanía de la documentación que obra en poder de las Administraciones Públicas, con fines distintos del propósito inicial que tenían esos documentos para los que se produjeron. De este modo, queda excluido de la reutilización, el intercambio de documentos entre distintos órganos u organismos de la Administración Pública.

Mediante la reutilización, el sector público pone a disposición de la ciudadanía la información que está en su poder, con las reservas específicas en los casos de protección de datos de carácter personal, seguridad nacional, etc. Posibilitando, así, la aparición de

nuevos negocios basados en la **información “en bruto”** que originariamente las Administraciones Públicas pusieron a disposición de la ciudadanía.

Estos **nuevos negocios** se fundamentarán, en buena medida, en los contenidos del sector público, a los que se añadirán un plus, un “valor añadido” por parte de los reutilizadores/infomediarios, haciendo de la información resultante un servicio apetecible para los diferentes segmentos del mercado.

Fundamentalmente, esta puesta a disposición de la información del sector público puede ayudar, de un lado, a generar crecimiento económico, a raíz del uso que de ella pueden hacer las empresas y los ciudadanos; y, de otro, a generar negocios basados en la creación de nuevos productos y servicios derivados de ella.

La reutilización se encuadra, de este modo, como una de las políticas de fomento de la Sociedad de la Información, y se plantea desde el inicio no sólo como una forma de hacer más transparentes y abiertas las Administraciones Públicas, sino, también y fundamentalmente, como una forma de conseguir ayudar a las empresas europeas y españolas a aprovechar el potencial de esta información y contribuir, con ella, al desarrollo económico y a la creación de empleo.

2.1.3.1. Concursos

Dentro de las posibles estrategias existentes para el fomento de la reutilización y consumo de los datos, una de las actuaciones básicas que se pueden encontrar en multitud de proyectos es la promoción de un **concurso de aplicaciones** en el que se desafíe a los participantes a desarrollar aplicaciones que exploten los datos expuestos y resulten de utilidad para los ciudadanos. Al mismo tiempo, este tipo de concursos son también un instrumento útil a la hora de ayudar a las empresas y emprendedores a **vislumbrar el potencial existente en la explotación de los datos abiertos**.

Varios precedentes exitosos, como el caso pionero del concurso¹ promovido desde Washington, o el más reciente e igualmente exitoso concurso² promovido por el Banco Mundial, son grandes alicientes que dan lugar a que sean varias las instituciones que actualmente tiene concursos en marcha.

¹ <http://www.appsfordemocracy.org/>

² <http://appsfordevelopment.challengepost.com/>

▪ **Desafío AbreDatos**



A nivel nacional se hace necesario mencionar el **Desafío AbreDatos**. Concurso de 48 horas para el desarrollo express de servicios tecnológicos al ciudadano basado en el uso de datos públicos que cuenten con al menos una fuente gestionada por un organismo público español, a nivel estatal, regional o local.

Puede encontrarse en la siguiente dirección: <http://www.abredatos.es/>

▪ **APPs for Democracy**



A nivel internacional es de obligada mención la **APPs for Democracy**. Idea lanzada en Washington en 2008 surgida a raíz del planteamiento de qué hacer con el catálogo de datos de las agencias gubernamentales para que dicho catálogo fuese de utilidad para la ciudadanía y las empresas.

A raíz de ellos, organizaron un concurso en el que cualquiera podía crear una aplicación basándose en los datos públicos que ofrecían las agencias.

Con una inversión de 50.000\$ en publicidad y premios se consiguieron **47 aplicaciones para iPhone, Facebook y páginas web**, devolviendo a la ciudad un valor en innovación estimado en más de 2 millones de dólares.

Puede encontrarse en la siguiente dirección: <http://www.appsfordemocracy.org/>

Si desea ampliar información al respecto de los concursos más relevantes a nivel nacional e internacional, debe dirigirse al capítulo '*concursos*' presente en nuestro blog¹.

¹ <https://datosenabierto.wordpress.com/estado-del-arte/>

2.1.4. Marco Legal

2.1.4.1. Legislación usada para la liberación de datos

Dentro del ámbito público, los antecedentes pueden iniciarse en el año 2000, donde basado en los datos del estudio PIRA (Oct-2000) para la Unión Europea se promovió la adopción de una Directiva. Finalmente fue aprobada 3 años más tarde, con la denominación 2003/98/CE, del parlamento y del Consejo, de 17 de Noviembre, de reutilización de la información del sector público.

Directiva que tiene su reflejo en el ámbito nacional en la ley 37/2007 de 16 de noviembre¹, sobre reutilización de la información del sector público y en el Real Decreto 1671/2009, de 6 de noviembre² que la desarrolla parcialmente.

Así mismo, el Esquema Nacional de Interoperabilidad, establecido por el Real Decreto 4/2010³, de 8 de enero, y el Esquema Nacional de Seguridad establecido por el Real Decreto 3/2010⁴, de 8 de enero, proporciona un marco legal para el ámbito más tecnológico de su puesta en marcha.

Actualmente, se encuentra en fase de consultas un Borrador del Real Decreto por el que se desarrolla la Ley 37/2007, de 16 de noviembre⁵, sobre reutilización de la información del sector público.

Podrá encontrarse una descripción más detallada de las distintas directivas legislativas nombradas en el capítulo de Anexos.

¹ <http://www.boe.es/boe/dias/2007/11/17/pdfs/A47160-47165.pdf>

² <http://www.boe.es/boe/dias/2009/11/18/pdfs/BOE-A-2009-18358.pdf>

³ <http://www.boe.es/boe/dias/2010/01/29/pdfs/BOE-A-2010-1331.pdf>

⁴ http://www.boe.es/aeboe/consultas/bases_datos/doc.php?id=BOE-A-2010-1330

⁵ http://www.mityc.es/dgdsi/es-ES/participacion_publica/Paginas/Cerradas/ConsultaBorradorLey37.aspx

2.1.4.2. Términos de uso y licencias / ¿Cómo hacer uso de los datos?

Con el concepto reutilización de la información generada por el sector público nos referimos al uso que pueden hacer personas, empresas y organizaciones para crear nuevos productos y/o servicios para otros públicos o destinatarios. La **Ley 37/2007** (que transpone la Directiva 2003/98/CE), de reutilización de la información del sector público atienden esta relevancia de la información para el desarrollo económico.

Modalidades de reutilización:

En cada conjunto o subconjunto de datos registrado en el catálogo de un portal ha de especificarse el tipo de reutilización que aplica entre los cuatro que recomienda la *Guía de reutilización del Proyecto Aporta* del Ministerio de Ciencia y Tecnología (Plan Avanza):

1. General: sin ningún tipo de restricción más allá de las condiciones básicas establecidas en el artículo 8 de la Ley 37/2007 (citación de la fuente, no alteración ni desnaturalización de la información y especificación de la fecha de última actualización).
2. Los datos considerados obras (con derechos de propiedad intelectual) se ponen a disposición de terceros mediante licencias Creative Commons, las de difusión más abierta, que permitan tanto la finalidad comercial, como la creación de obras derivadas.
3. En determinados casos, como es el caso de fotografías tomadas por terceros, la licencia de reutilización es la licencia Creative Commons de Reconocimiento – SinObraDerivada BY-ND, es decir, que se permite «el uso comercial de la obra pero no la generación de obras derivadas».
4. Finalmente, se establece una modalidad con solicitud previa según las condiciones del ente generador, del cual se facilitará el contacto para acordar los términos de uso específicos.

En cada una de las modalidades de reutilización citadas, existen una serie de condiciones de uso dependientes de la tipología del catálogo de datos. Podremos encontrar más información sobre estas **Condiciones de uso** en el capítulo de Anexos.

2.1.5. Marco Tecnológico

En este nivel daremos cabida a la plataforma tecnológica que sustenta y hace posible la materialización de los Open Data. Desde los **formatos** en los que son liberados los datos por parte de los distintos portales, pasando por los **sistemas de acceso** que facilitan la consulta de estos, hasta dar una visión de una de las fundaciones pilares que ayudan al impulso de este movimiento en España – **Fundación CTIC**.

También será tratado un **estándar de clasificación** que permite la evaluación de un portal de datos en función de la calidad de estos. Por último será mostrada la propuesta de **linked data** que surge dentro del marco general de la web semántica.

2.1.5.1. Formatos disponibles

La información se publica en formatos de datos estructurados para facilitar que pueda ser utilizada de forma automática por los lenguajes de programación. De esta manera, se intenta cumplir el objetivo de reutilizar al máximo la información publicada.

Así pues, disponemos de formatos estructurados (propietarios y libres), y formatos no estructurados.

Formatos estructurados:

Estos son los formatos más utilizados para publicar los datos, distinguimos dos tipos de formatos estructurados:

Formatos Propietario:

Son formatos de archivo que requieren herramientas que no son públicas.

- **SHP** (Shapefile)

Shapefile es un formato propietario estándar de datos espaciales, desarrollado por la compañía ESRI, que almacena tanto la geometría como la información alfanumérica. Este formato no está preparado para almacenar información topológica.

Más información: <http://es.wikipedia.org/wiki/Shapefile>

- **XLS** (Microsoft Office Excel)

Microsoft Office Excel es un formato propietario de Microsoft que muestra la información en celdas organizadas en filas y columnas, y cada celda contiene datos o fórmulas, con referencias relativas o absolutas a otras celdas.

Más información: http://es.wikipedia.org/wiki/Microsoft_Excel

Formatos Libres:

Son formatos de archivo que se pueden crear y manipular para cualquier software, libre de restricciones legales.

- **XML** (eXtensible Markup Language)

Es un metalenguaje extensible de etiquetas desarrollado por el W3C que permite definir lenguajes para diferentes necesidades. Es el estándar para el intercambio de información estructurada entre diferentes plataformas.

Más información: www.w3.org/standards/xml/core

- **CSV** (Comma – separated values)

Valores separados por coma. Los ficheros CSV son un tipo de documento en formato abierto sencillo para representar datos en formato de tabla. Las columnas se separan por comas (o punto y coma) y las filas por saltos de línea.

Más información: tools.ietf.org/html/rfc4180

- **RSS** (Really Symple Syndication)

Es un formato XML para la distribución de contenidos de páginas web. Facilita la publicación de información actualizada a los usuarios suscritos a la fuente RSS sin necesidad de usar un navegador, utilizando un software especializado en este formato.

Más información: <http://es.wikipedia.org/wiki/RSS>

- **JSON** (JavaScript Object Notation)

Es un formato ligero para el intercambio de datos basado en la notación literal de objetos de JavaScript. Su sintaxis es simple, por lo que facilita el tratamiento en los navegadores. Además, su concisión reduce el tamaño de flujo de datos entre cliente y servidor.

Más información: json.org/json-es.html

- **RDF** (Resource Description Framework)

Es una especificación del W3C para el modelado de información y la descripción de recursos, que se hace con la forma de sujeto-predicado-objeto. La combinación de RDF con otras herramientas permite añadir significado a las páginas y es una de las tecnologías esenciales para la web semántica.

Más información: www.w3.org/standards/techs/rdf#w3c_all

- **TMX** (Translation Memory eXchange)

Estándar de XML, que es un DTD que sirve para el intercambio de memorias de traducción. Creado por el comité OSCAR (Open Standards for Container/Content Allowing Re-use).

Más información: www.w3.org/2002/02/01-i18n-workshop/LocFormats#TMX
- www.lisa.org/Translation-Memory-eXchange-TMX.34.0.html

- **KML** (Lenguaje de Etiquetado de Ojo de Cerradura)

Gramática XML y formato de archivo para la creación de modelos y almacenamiento de funciones geográficas como puntos, líneas, imágenes, polígonos y modelos que se mostrarán principalmente en aplicaciones de mapas. Se utiliza para compartir lugares e información entre aplicaciones. Es el estándar del Open Geospatial Consortium y se puede utilizar a través de Google Earth. Los archivos KML se distribuyen comprimidos como archivos KMZ.

Más información:

code.google.com/intl/es/apis/kml/documentation/kmlreference.html

www.opengeospatial.org/standards/kml/

Formatos no estructurados:

En los casos en los que no sea posible publicar los datos en formatos estructurados, estos se publicarán en otro tipo de formatos en tecnologías propietarias o no abiertas, como por ejemplo en Word, PDF, enlaces a otros sitios web, mapas interactivos, etc.

2.1.5.2. Sistemas de acceso

Los lenguajes disponibles para facilitar el acceso a estos datos son lenguajes de consulta de archivos tales como:

- **SPARQL** (Protocolo Simple y Lenguaje de consulta de RDF)

Lenguaje estandarizado para la consulta de grafos RDF, normalizados por el W3C. Es una recomendación oficial del W3C desde enero de 2008 para el desarrollo de la web semántica.

Al igual que sucede con SQL, es necesario distinguir entre el lenguaje de consulta y el motor de almacenamiento y recuperación de los datos. Por este motivo, existen múltiples implementaciones de SPARQL, generalmente ligados a entornos de desarrollo y plataformas tecnológicas.

En un principio SPARQL únicamente incorpora funciones para la recuperación de sentencias RDF. Sin embargo, algunas propuestas también incluyen operaciones para el mantenimiento (creación, modificación y borrado) de datos.

Ejemplo de consulta

```
PREFIX  dc: <http://purl.org/dc/elements/1.1/>
SELECT  ?title
WHERE   { <http://ejemplo.org/libros> dc:title ?title }
```

Estos resultados pueden ser tratados por cualquier aplicación para construir el catálogo según las necesidades específicas.

Más información: www.w3.org/standards/techs/sparql

Guía de uso: <http://www.dajobe.org/2005/04-sparql/SPARQLreference-1.8.pdf>

- **Web services – API** (Interfaz de programación de aplicaciones)

Son interfaces de programación de aplicaciones (API) o la API de la web que se accede a través de HTTP a se ejecuta en un sistema remoto de alojamiento de los servicios solicitados. Los servicios web son sistemas de software diseñados para apoyar la interacción interoperable máquina a máquina sobre una red. Tiene una interfaz descrita en un formato procesable por una máquina y otros sistemas interactúan con el servicio web de una manera prescrita por su descripción utilizando mensajes SOAP, transmitidos a través de HTTP con una serialización XML en conjunto con otras normas relacionadas con la web.

Más información: www.w3.org/TR/ws-gloss/ - www.w3.org/standards/techs/wsdl

- **WMS** (Servicio de Mapas Web)

Formato que produce mapas de datos referenciados espacialmente, de forma dinámica, a partir de información geográfica. Es estándar internacional. Los mapas WMS se generan normalmente en un formato de imagen como PNG, GIF o JPG, y opcionalmente como gráficos vectoriales en formato SVD o WebCGM.

Más información: www.opengeospatial.org/standards/wms

2.1.5.3. Estandar de clasificación de Tim Berners-Lee

Tim Berners-Lee propuso una clasificación de los portales de datos abiertos en 5 niveles en función de cómo ofrecen la información y qué formato utilizan. Cuanto mayor sea el nivel más fácil será reutilizar la información de forma automática:

- ★ make your stuff available on the web (whatever format)
- ★★ make it available as structured data (e.g. excel instead of image scan of a table)
- ★★★ non-proprietary format (e.g. csv instead of excel)
- ★★★★ use URLs to identify things, so that people can point at your stuff
- ★★★★★ link your data to other people's data to provide context

Figura 3: Estandar de clasificación de Berners-Lee

Publicado ★☆☆☆☆

Existen datos publicados en formatos no estructurados.

Para utilizar los datos se requiere un esfuerzo extra para tratar el documento. Ejemplos: formatos de imagen (JPG, PNG, GIF,...), formatos de vídeo (AVI, MPG, MP4, ...), formatos de música (MP3, WMA,...) y muchos otros.

Las ventajas de este nivel son que los datos se pueden visualizar, imprimir o almacenar localmente, además de ser fácilmente publicables.

Datos estructurados ★★☆☆☆

Los datos están publicados en formatos estructurados. El problema es que los datos todavía están dentro de un documento y en este caso con una licencia propietaria y por lo tanto se requieren herramientas que no son públicas para acceder a los datos.

Las ventajas de este nivel es que los datos se pueden procesar directamente si se dispone del software propietario, esto es, se pueden exportar a otros formatos estructurados y todavía son fáciles de publicar.

Formatos abiertos ★★★☆☆

Los datos están publicados en la web en formato estructurado, pero ahora los formatos de publicación son abiertos y no propietarios. Cualquier persona puede utilizar (además de acceder) los datos de forma sencilla. Todavía se trata de datos “de la web” y no en la web y por lo tanto, el contenido depende del contexto.

Los formatos de publicación de datos en formato abiertos son, por ejemplo, CSV o XML.

Las ventajas de este nivel es que los datos se pueden manipular de la forma que se desee sin estar limitado por las particularidades del software. El problema es que por parte del publicador, la publicación puede requerir procesos de conversión para exportar los datos de formatos propietarios, con la ventaja de que todavía es muy sencillo publicar los datos.

Utilización de URI's para identificar los datos ★★★★★☆

En este nivel, los datos ya están “en la web”. Podemos representar información estructurada, los documentos son validables y los datos adquieren significado dependiendo de las etiquetas que se utilicen para la confección del documento.

Las ventajas de este nivel son que los datos se pueden vincular con los de cualquier otro lugar (de la Web o localmente) y que los usuarios pueden reutilizar parte de los datos. El principal problema es para el publicador, ya que la publicación de estos datos requiere de un cierto tiempo estructurando y separando los datos, asignando URI's a los datos o pensando como presentarlos. Con la ventaja de tener un mayor control sobre los datos para poder optimizar su acceso.

Enlazar con otros datos (Linked Data) ★★★★★★

En este nivel, el objetivo es enlazar los datos más relevantes utilizando URI's para asociarlas a un contexto concreto, de forma que permitimos relacionar los datos originales con otros datos nuevos. El formato más utilizado es RDF.

Las ventajas de este nivel es que pueden descubrir más datos relacionados durante la consulta de los datos originales y se puede aprender a partir de los esquemas de los datos. Por parte del publicador de los datos se tienen que invertir recursos al vincular los datos a otros datos existentes en la Web con la ventaja de que ahora los datos son más fácilmente localizables (porque pueden ser vinculados por otros datos), incrementando así el valor de los datos publicados.

2.1.5.4. Linked data

En el proceso de publicación de los datos, tan útil como el propio valor del dato o de la información es su significado. Esta meta-información o metadatos completan la publicación avanzada de información para su reutilización.

Las semánticas (es decir, conjuntos de metadatos) todavía no pueden considerarse de uso generalizado en la publicación de fuentes. Sin embargo, los estándares RDF, OWL y diversas implementaciones sobre XML, entre otros, pueden considerarse los mecanismos más extendidos.

La clave para conseguir una Web autodescriptiva¹, consiste en identificar los vínculos existentes entre los distintos conjuntos de datos y representarlos adecuadamente. Para ello podremos contar nuevamente con las tecnologías de Web Semántica, puesto que sus características de enlazado, como por ejemplo la integración automática de datos provenientes de varios orígenes, o la posibilidad de realizar consultas a través de diversas fuentes de forma transparente, son únicas y no las proporciona ningún otro modelo.

Los vínculos que se establezcan entre conjuntos de datos representados mediante tecnologías de Web Semántica nos permitirán navegar fácilmente entre los distintos datos relacionados, además de proporcionarnos opciones de búsqueda y consulta más avanzadas que las convencionales y, gracias a que los resultados de las consultas serán datos estructurados y no solo simples textos o enlaces, podrán a su vez ser reutilizados por cualquier aplicación.

Los principios que deben cumplir los datos para ser considerados “Linked data” son:

- Los recursos deben ser unívocamente identificables a través de su URI. Las URIs deben estar basadas en el esquema HTTP para hacer su gestión descentralizada y para hacer su acceso ubicuo a través de la web.
- Los recursos deben ser descritos mediante RDF² que es el modelo de datos de la web semántica. De entre las diferentes representaciones RDF, al menos la serialización oficial en XML, RDF/XML, debe estar disponible para cada recurso.
- Para crear una auténtica web de datos es necesario que los datos estén enlazados. Nuestros recursos deben incluir referencias en forma de enlaces RDF a otras fuentes de datos y, en la medida de lo posible, deberían ser referenciados desde recursos externos.

¹ <http://www.w3.org/2001/tag/doc/selfDescribingDocuments>

² <http://www.w3.org/TR/REC-rdf-syntax/>

A continuación mostraremos el **diagrama de la nube de open datas enlazados**:

El siguiente diagrama¹ muestra la colección de datos actual que existe de forma abierta y accesible – Open Data - y como se interconectan entre sí.

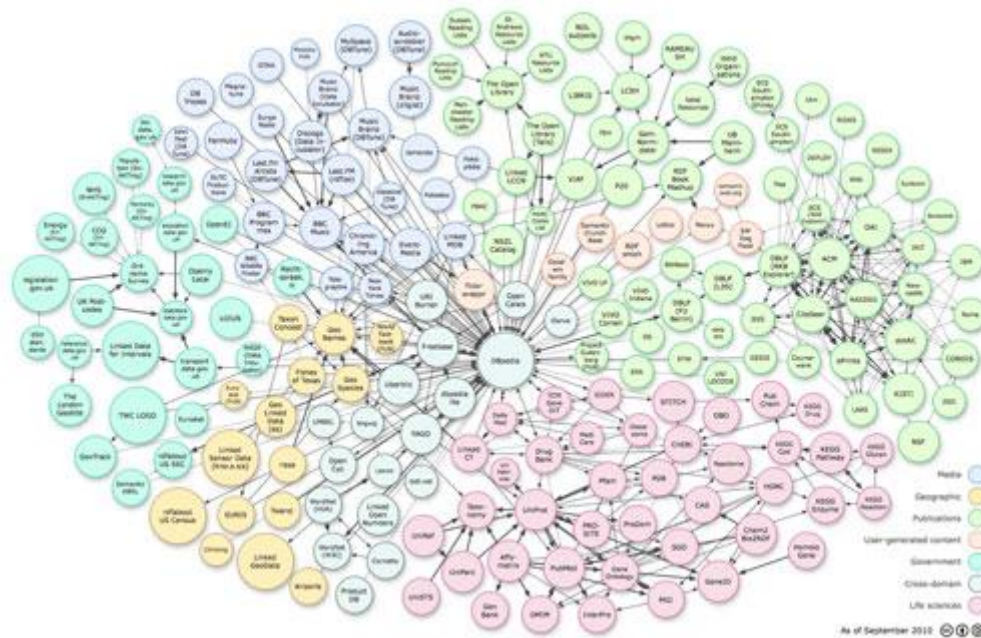


Figura 4: Linking Open Data cloud diagram

¹ <http://iqingenieros.com/empresadesoftware/the-linking-open-data-cloud-diagram/>

2.1.5.5. Fundación CTIC

Dentro del panorama nacional, por su destacado papel en la participación en foros internacionales de normalización, merece destacarse el papel de la **Fundación CTIC Centro Tecnológico**.

CTIC Centro Tecnológico, tiene como objetivo la mejora de la competitividad empresarial a través de la investigación y la innovación tecnológica, así como el asesoramiento a entidades públicas y privadas para conseguir sus objetivos mediante el uso de estándares y tecnología Web.

CTIC es la sede de W3C en España, la cual ha fundado y liderado la iniciativa mundial de W3C sobre eGovernment y participa activamente en la normalización de los estándares para la web, especialmente en aquellos relacionados con el Linked Data.

CTIC es líder en la aplicación de estos estándares a iniciativas Open Data y tiene una metodología propia para desarrollar e implantar estrategias Open Data en las Administraciones Públicas que ya ha dado resultado en varias iniciativas en el territorio nacional (Asturias, Euskadi y Cataluña entre otras) y en proyectos en otros países como Chile o Ghana. **CTIC** también desarrolla la estrategia del catálogo de datos de la Administración General del Estado y comenzará nuevos proyectos Open Data en África y Latinoamérica a lo largo del 2011.

Buscador facetado – CTIC

Para localizar los diferentes catálogos Open Data existentes en el mundo, la fundación **CTIC** ha desarrollado un buscador facetado¹ (*Faceted search*) sencillo, usando Exhibit, donde los catálogos de datasets pueden ser posicionados sobre un mapa y filtrados por sus propiedades: Jurisdicción, país y madurez de la iniciativa.

Esta aplicación usa datos en RDF guardados en la triple-store, a través del SPARQL endpoint². Por otro lado, la información geográfica (nombre de lugares, coordenadas, etc.) proceden de la base de datos de Geonames³.

Además, este buscador, describe el grado de evolución de cualquier catalogo dentro de la horquilla de web semántica definida por Tim Berners-Lee (ver *Estandar de clasificación de Tim Berners-Lee* en este documento).

¹ <http://datos.fundacionctic.org/sandbox/catalog/faceted/>

² <http://data.fundacionctic.org/sparql>

³ <http://www.geonames.org/>

En la siguiente imagen podemos observar como el buscador de **CTIC** nos indica el número de catálogos actuales y su nivel de madurez definidos en función de la horquilla de Tim Berners-Lee.

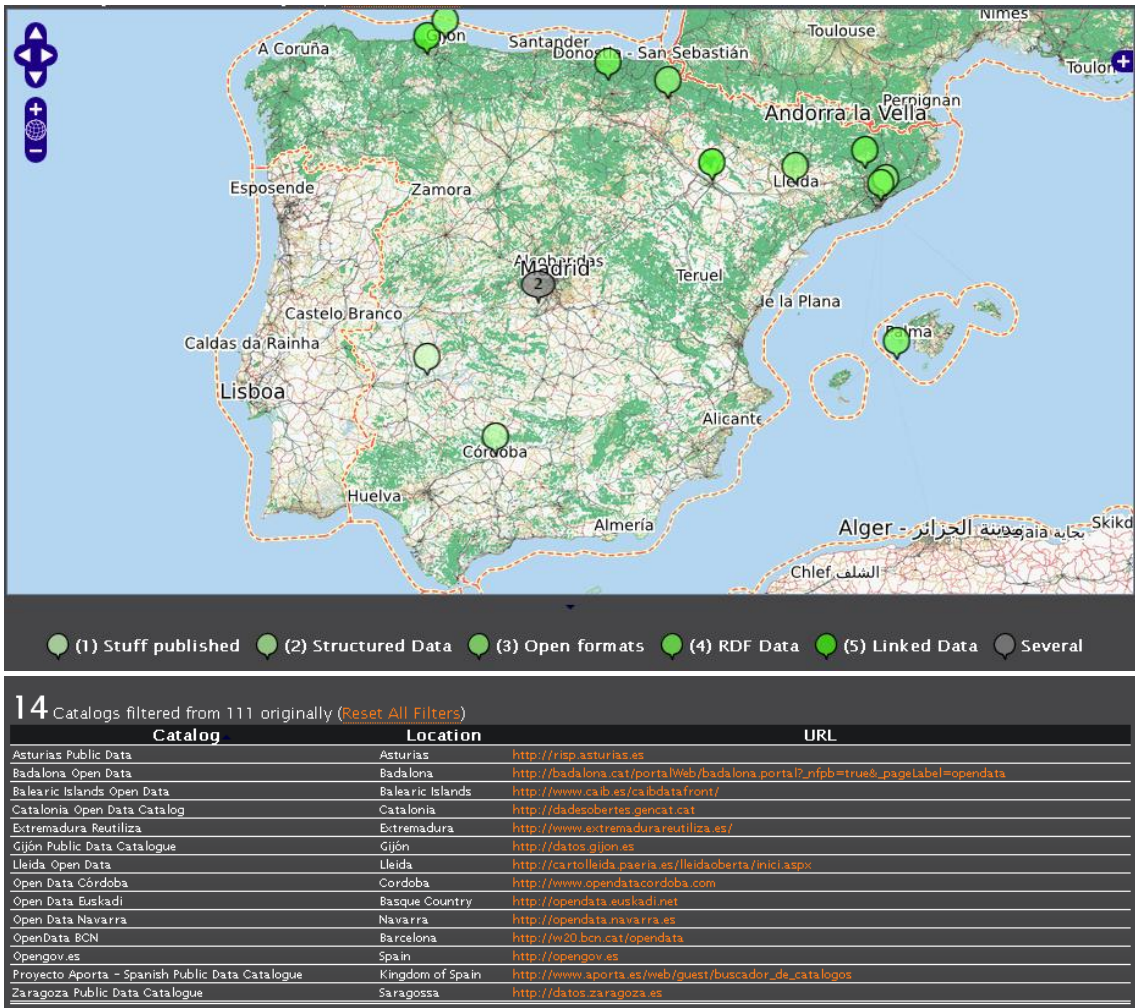


Figura 5: Ejemplo de búsqueda en el buscador faceteado de CTIC

2.2. Fuentes Open Data

La apertura y reutilización de la información del sector público es aún una actividad reciente. De hecho, el número de iniciativas públicas generales, de gobiernos y administraciones, dedicadas en la Red a esta apertura de datos apenas supera el centenar a nivel mundial.

A continuación pasaremos a enumerar una serie de características comunes a todos los portales/fuentes Open Data alrededor del mundo, para dar paso a la vista de los portales Open Data más influyentes tanto a nivel Internacional como a nivel nacional.

2.2.1. Características comunes

En primer lugar todos los portales disponen de una sección dedicada a dar **información sobre el proyecto**. En especial se puede destacar la sección dedicada a la información del proyecto¹ de la página web del **Reino Unido** que incluye, junto con la información básica del proyecto ya las respuestas a una serie de preguntas generales, una **declaración de apoyo institucional** al máximo nivel con un vídeo del Primer Ministro británico.

Otra sección habitual en este tipo de portales es la de **Novedades**. Al respecto, es interesante la opción² que ha hecho el portal de EEUU, en el que se combinan diferentes fuentes de información y en el que se incluyen los nuevos conjuntos de datos que se incorporan al catálogo.

Asimismo, la mayor parte de estos portales dedican una sección a las aplicaciones que se han desarrollado a partir de la reutilización de sus datos. Tanto el portal³ del Reino Unido como el portal⁴ de la ciudad de Rennes son unos **buenos ejemplos** de cómo se puede ofrecer este tipo de información.

Además, el portal de **Rennes** contiene un **manual** de las aplicaciones⁵ contenidas dentro del propio sitio, que puede ser muy útil para **desarrolladores** y **potenciales reutilizadores**. En otros portales esta información a veces no está centralizada en una sección sino que se encuentra dispersa entre los diferentes apartados de la web.

¹ <http://data.gov.uk/about>

² <http://www.data.gov/whatsnew>

³ <http://data.gov.uk/apps>

⁴ <http://www.data.rennes-metropole.fr/vos-applications/>

⁵ <http://www.data.rennes-metropole.fr/le-concours/1-edition-2011/les-applications-en-competition/>

En cuanto a las **vías de participación** que ofrecen los portales analizados, se señala que estas opciones a día de hoy todavía son de **carácter muy básico** con una mínima integración de las **redes sociales**, por lo que este punto seguro que evolucionará en los próximos meses. A modo de ejemplo, el Reino Unido ofrece una plataforma¹ completa para compartir, proponer y valorar las **ideas** de los usuarios.

Por supuesto, todos estos portales contienen al menos un **catálogo de información pública**, junto con sus respectivas **fichas descriptivas** sobre los conjuntos de datos, así como un apartado en el que se recogen las condiciones de reutilización.

A modo de ejemplos significativos del primer punto de la Cadena de valor de Open Data (ver *Cadena de valor del Open Data* en este documento), se incluyen a continuación una relación de fuentes Open Data. Se analiza en cada uno de ellos sus distintas dimensiones y situaciones de madurez en la publicación de datos en la siguiente figura:



Figura 6: Mapa de las fuentes analizadas por el origen de los datos y su ámbito

A continuación, podemos establecer una clasificación a nivel internacional y nacional en la que apreciar los más influyentes portales open data.

¹ <http://data.gov.uk/ideas>

2.2.2. Ámbito Internacional

Entre los más importantes portales/fuentes Open data a nivel internacional podemos destacar los siguientes:

- Data.gov – EEUU
- Data.gov.uk – UK
- Nueva Zelanda
- Banco mundial
- DBpedia
- City – go – round | Rockefeller Foundation
- Freebase

Asimismo, en el siguiente mapa podremos observar las distintas iniciativas a nivel internacional que en sucesivos puntos serán detalladas:



Figura 7: Mapa de iniciativas Open Data a nivel mundial

▪ Data.gov (Estados Unidos de América)



Gran repositorio¹ de información de carácter público y ámbito nacional con gran desarrollo de información georeferenciada. Incluye informaciones de tipo económico, social, meteorológicas, de comercio, medioambientales, vivienda, etc.

Data.gov es un repositorio de información de origen público lanzado en Mayo de 2009. En enero de 2011 incluye más de 3.085 datasets², y más de 302.000 datasets de información georeferenciada³.

El mayor impulso al mismo fue recibido con la directiva de apertura de datos que obligaba a todas las agencias de la administración de los EEUU a publicar 3 datasets de alto valor en 45 días (diciembre de 2009).

En general, la información se proporciona de forma gratuita, con una cláusula de no responsabilidad en cuanto a la fiabilidad de la misma cuando es ‘mezclada’ con otros tipo de fuentes o tratada por las entidades privadas y con una cláusula de recomendación de citación del origen de la información.

Incluye una multitud de entidades públicas y departamentos de la administración pública americana.

Globalmente los datasets de este repositorio son descargados más de 14.000 veces a la semana.

¹ Sitio centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos.

² Colección de datos, normalmente presentada de forma tabular. Cada columna representa una variable concreta y cada fila corresponde a un miembro determinado del conjunto de datos en cuestión.

³ Posicionamiento con el que se define la localización de un objeto espacial en un sistema de coordenadas y datum determinado.

▪ Data.gov.uk (Reino Unido)



Gran repositorio de información de carácter público y ámbito nacional con gran desarrollo en la publicación de información económica. Incluye también una sección para aplicaciones que los utilizan.

Data.gov.uk es un repositorio de datos de origen público con más de 6.900 conjuntos de datos (datasets Mayo 2011).

Incluye la posibilidad de ser consultado vía SPARQL (Ver *Marco Técnico* en este documento) lo que posibilita una explotación automatizada.

En proceso de implantación hay una normativa que publicará todos los gastos de la administración superiores a 25.000 libras además de otras medidas de ahorro.

Así mismo, **data.gov.uk** utiliza el software *CKAN* (creado por la Open Knowledge Foundation) que se está convirtiendo en uno de los referentes en cuanto a la gestión de catálogos de fuentes de datos. Dicho software está siendo utilizados, además de en el Reino Unido, en Noruega, Alemania, Canadá, Francia. En el anexo se incluye un listado de catálogos de datos utilizando *CKAN*.

▪ Nueva Zelanda



Entidad privada que tiene como objeto la difusión de fuentes de datos públicas. Para cada una de las informaciones disponibles incluyen descripción de su disponibilidad, los distintos formatos de descarga, XLS, CSV, KML (en el caso de que sea esa la vía de acceso), así como su precio, si la información tiene componente georeferenciada, y si tiene API para el acceso automatizado. Actualmente reúne más de 70 catálogos de datos.

Puede encontrarse en la siguiente dirección: <http://cat.open.org.nz/>

▪ Banco mundial



Entidad pública supranacional que proporciona información para la mejor disponibilidad de sus clientes (países).

<http://datos.bancomundial.org> es la sección que el banco mundial ha abierto para hacer disponibles sus repositorios de datos sobre su actividad (datos sobre el desarrollo socio económico de los países) proporcionando información sobre más de 2.000 indicadores.

Algunos de estos datos están disponibles mediante su API y otros como descarga directa.

▪ DBpedia



Entidad privada independiente, que incluye tanto profesionales como voluntarios y que proporciona datos de forma libre como objetivo de su existencia. **DBpedia** pertenece a las iniciativas de la fundación **Wikipedia**.

Wikipedia no solo es el único sitio sin ánimo de lucro entre los 10 más visitados de internet, sino que con sus más de 3,5 millones de artículos (solo en inglés) se ha convertido en referencia mundial del saber.

Dbpedia es una iniciativa de la Fundación Wikipedia para generar datasets a partir de la información contenida en **Wikipedia** bajo los mismos principios de colaboración y licenciamiento abierto de sus contenidos.

Actualmente (Mayo 2011), contiene más de 460.000 entidades geoposicionadas y más de 6,5 millones de referencias RDF a **Freebase**.

Como otros casos aquí contemplados es consultable vía SPARQL. Sin necesidad de interactuar vía SPARQL, desde herramientas como Google Docs también se pueden incorporar datos de **Wikipedia** como puede verse en este ejemplo.

▪ City-go-round (Rockefeller foundation)



Iniciativa privada desde el tercer sector (fundación) para la mejora de la sociedad proporcionando informaciones y aplicaciones abiertas.

<http://www.citygoround.org> es un ejemplo de publicación de datos por parte privada, aunque el origen de los mismos en buena parte sea público.

Su temática es la de fuentes de datos para la generación de aplicaciones para el uso del transporte público.

De momento, está centrado en el transporte interno en grandes ciudades de Estados Unidos, lo cual permite y fomenta el desarrollo de aplicaciones que los utilicen. Actualmente es financiada por la Rockefeller foundation¹.

▪ Freebase



Entidad privada independiente que incluye tanto profesionales como voluntarios y que proporciona datos de forma libre como objetivo de su existencia.

Freebase a diferencia de *DBpedia*, con quien mantiene importantes lazos, incluye no sólo repositorios de datos desde *Wikipedia*, sino también desde otras fuentes.

Es coordinada por la empresa Metaweb Technologies, que ha sido adquirida por Google en Julio de 2010.

<http://www.freebase.com/> es un repositorio con más de 22 millones de entidades de datos licenciadas bajo Creative Commons. Una entidad, en terminología de *Freebase*, es cualquier persona, lugar o cosa recogida dentro del repositorio de datos.

¹ Prominente organización filantrópica y fundación privada estadounidense.

2.2.3. Ámbito Nacional

Entre los más importantes portales/fuentes Open data a nivel nacional podemos destacar los siguientes:

- Aporta.es
- Opendata.Euskadi.net
- Gencat.cat
- Risp.Asturias.es
- Open Data Navarra
- ExtremaduraReutiliza.es
- OpendataCordoba.com
- Datos.Gijon.es
- Zaragoza.es/ciudad/risp
- Instituto Geográfico Nacional
- Agencia Española de Meteorología

Asimismo, en el siguiente mapa podrá apreciarse la descentralización del movimiento Open Data de los portales ya citados:



Figura 8: Mapa de iniciativas Open Data en España

▪ Aporta.es



Catálogo de fuentes de datos de origen público de la Administración General del Estado.

Con el objetivo de impulsar el sector de la reutilización de la información del sector público (RISP) en nuestro país, el Gobierno lanzó en 2009 el Proyecto Aporta www.aporta.es, promovido por los Ministerios de Política Territorial y Administración Pública y de industria, Turismo y Comercio.

Actualmente, mantiene el catálogo¹ oficial de fuentes públicas de la Administración General del Estado susceptibles de reutilización. A Mayo de 2011 enumeraba 723 fuentes repartidas de acuerdo al siguiente gráfico.

Este catalogo puede localizarse en la siguiente dirección: <http://aporta.es>.

A continuación incluimos un gráfico con la distribución de las fuentes de datos por la entidad pública de origen.

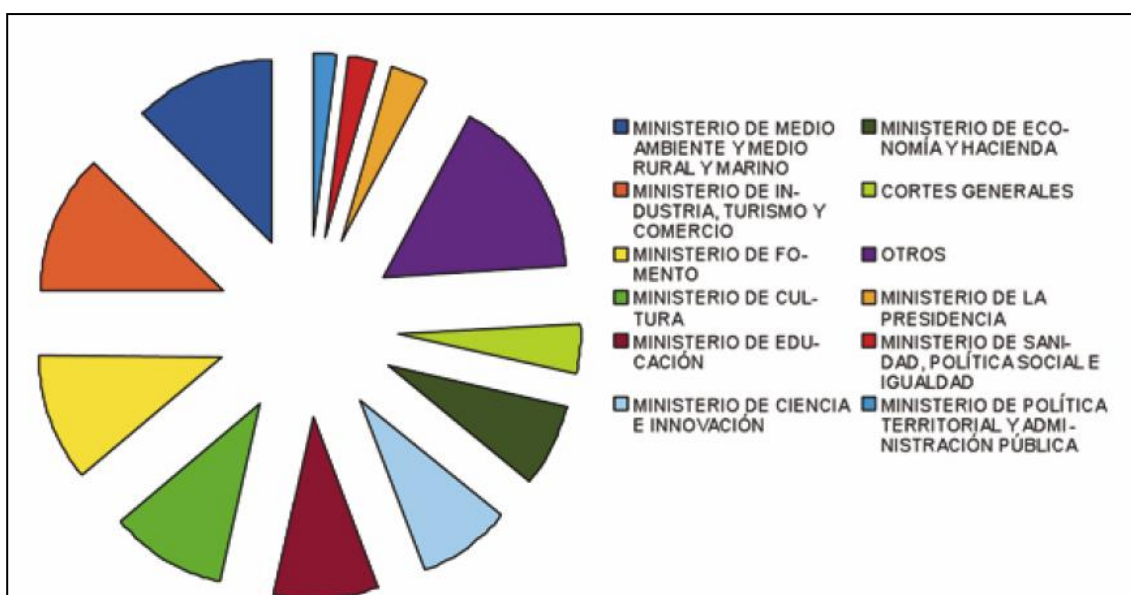


Figura 9: Distribución de fuentes de datos públicas AGE por organismo

¹ http://www.aporta.es/web/guest/buscador_de_catalogos

▪ Opendata.euskadi.net



Repositorio de datos de origen público de la Administración Autonómica del País Vasco.

Repositorio de fuentes de datos que cuenta en la actualidad con un conjunto de 1,391 datos. Los conjuntos de datos expuestos en este portal se ofrecen bajo licencias de propiedad abiertas, que permiten su redistribución, reutilización y aprovechamiento con fines comerciales.

Uno de los proyectos pioneros de España, desarrollado por el equipo técnico del Gobierno del País Vasco, con la ayuda de **CTIC** (ver *Marco Tecnológico* en este documento) para las partes de identificación y del modelado semántico de los metadatos del catálogo.

En este repositorio destaca la ejemplar documentación sobre generación y normativa de codificación y utilización.

Podemos consultar desde cláusulas legales, hasta normativas para la ficha descriptiva de los distintos juegos de datos, pasando por la descripción de la Infraestructura de entrega de contenidos.

Puede encontrarse en la siguiente dirección: <http://opendata.euskadi.net>.

▪ Gencat.cat



Repositorio de datos de origen público de la Generalitat de Cataluña. <http://dadesobertes.gencat.cat/> es gestionado desde la Dirección General de Atención Ciudadana, donde están publicados los datos de carácter público.

Los datos proceden de diferentes organismos públicos de la Generalitat y están agrupados en el catálogo de datos. En dicho repositorio hay publicados varios conjuntos de datos en RDF. Entre estos se observan datos semánticos sobre los más de 26.000 edificios públicos¹ que son representados mediante el estándar vCard en RDF², más de 700 trámites y servicios³, y muchos otros datos en formatos estructurados y fácilmente reutilizables (XML, RSS, APIs, etc.).

Una de las novedades que incluye Dades Obertes de Gencat es la representación del catálogo de los datos usando el vocabulario internacionalmente reconocido *dcat*⁴, Dataset Catalog, que ofrece la capacidad de la descripción de los catálogos de conjuntos de datos de forma semántica en RDF. Todo el catálogo ha sido modelado usando *dcat* y está disponible en formato RDF-XML⁵. CTIC está desarrollando la estandarización mundial de este vocabulario en el seno del grupo de trabajo sobre eGovernment de W3C⁶ (fundado por CTIC en 2007), esto permitirá la interoperabilidad con otros sistemas similares en el resto del mundo.

Aunque el portal aún no dispone de un punto de consulta SPARQL (ver *Marco Tecnológico en este documento*), todos los datos representados en RDF están disponibles en documentos RDF-XML y la iniciativa en sí contempla el compromiso de la utilización en un futuro cercano de Linked Data en toda su potencia.

Este proyecto ha contado con una coordinación muy dinámica entre la dirección técnica y estratégica de CTIC (ver *Marco Tecnológico en este documento*), y el excelente trabajo realizado por el equipo técnico y legal de la Dirección General de Atención Ciudadana de la Generalitat de Catalunya, ampliamente comprometidos, lo que ha permitido producir esta primera fase en un periodo muy corto de tiempo.

Puede encontrarse en la siguiente dirección: <http://dadesobertes.gencat.cat/>

¹ http://dadesobertes.gencat.cat/es/dades-obertes/dataset_000026.html

² <http://www.w3.org/TR/vcard-rdf/>

³ <http://dadesobertes.gencat.cat/es/dades-obertes/tramits.html>

⁴ <http://vocab.deri.ie/dcat-overview>

⁵ <http://dadesobertes.gencat.cat/recursos/datasets/catalog.rdf>

⁶ http://www.w3.org/egov/wiki/Main_Page

▪ risp.asturias.es



Repositorio de datos de origen público del Gobierno del Principado de Asturias.

El Gobierno del Principado de Asturias fue pionero en publicar un portal de datos abiertos con su Datos de Asturias¹. CTIC elaboró íntegramente el proyecto, desde el análisis de los datos que compondrían inicialmente el catálogo de datos, hasta el desarrollo de la herramienta que permite almacenar datos en un almacén semántico. Este proyecto fue el **primero a nivel mundial** compuesto íntegramente mediante tecnologías Linked Data ²(o de la Web Semántica).

Algo novedoso de este portal es el punto de consulta SPARQL³, un punto de acceso único que permite consultar y obtener todos los datos almacenados usando un lenguaje estándar de consulta, esto permite hacer ejemplos de una forma sencilla y potente.

Otra de las novedades de este sistema es la generación del catálogo. Gracias a que toda la información de los conjuntos de datos está modelada mediante el vocabulario voidD⁴, a través de una consulta sencilla se puede descubrir el tipo de datos que se incluyen y generar automáticamente el catálogo de datasets.

Puede encontrarse en la siguiente dirección: <http:// risp.asturias.es>

¹

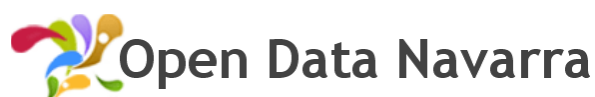
<https://sede.asturias.es/portal/site/Asturias/menuitem.77b6558ac8616446e44f5310bb30a0a0/?vgnextoid=05badd42ece45210VgnVCM10000097030a0aRCRD&vgnnextchannel=05badd42ece45210VgnVCM10000097030a0aRCRD&i18n.http.lang=es>

² <http://linkeddata.org/>

³ <http:// risp.asturias.es/sparql>

⁴ <http://vocab.deri.ie/void/guide>

▪ Open Data Navarra



Repositorio de datos de origen público del Gobierno de Navarra. Open Data Navarra es el resultado del compromiso del Gobierno de Navarra para exponer datos públicos sin restricciones técnicas ni legales que hasta ahora no estaban disponibles.

El Gobierno de Navarra se suma a esta iniciativa en la que ya están inmersas otras administraciones públicas de todo el mundo, dando un paso más por cumplir sus objetivos de implantar y difundir la Administración electrónica.

Entre los conjuntos de datos publicados¹, destaca interesante información geográfica e **información transparente** sobre las retribuciones en 2011 de sus altos cargos o el histórico desde 1987 de Expedientes de Regulación de Empleo autorizados.

Puede encontrarse en la siguiente dirección: http://www.navarra.es/home_es/Open-Data/

▪ Extremadurareutiliza.es



Repositorio de datos de origen público del Gobierno Extremeño. Proyecto piloto de la junta de Extremadura que pone en marcha este portal de reutilización que supone el primer paso en la apuesta del Gobierno Extremeño por el crecimiento económico y la transparencia a través de esta herramienta.

En él, los ciudadanos y las empresas podrán encontrar no sólo información general sobre la reutilización, sino también un catálogo de datos que contenga toda la información que pueda ser susceptible de ser reutilizada, en unos formatos abiertos y accesibles para la ciudadanía.

Puede encontrarse en la siguiente dirección: <http://www.extremadurareutiliza.es/>

¹ http://www.navarra.es/home_es/Open-Data/Datos-abiertos/

▪ **Opendatacordoba.com**



Repositorio de datos de origen público del instituto Municipal de Desarrollo Económico de Córdoba.

El instituto Municipal de Desarrollo Económico de Córdoba es un instituto de carácter público cuyo objetivo es contribuir al desarrollo económico y social del municipio de Córdoba, trabajando en una doble vertiente:

Por una parte, el apoyo a los/las emprendedores/as que estén dispuestos a asumir el reto de la aventura empresarial, así como por otra parte el impulso y gestión de Políticas Activas de Empleo, con la finalidad de mejorar las posibilidades de acceso al mercado laboral de las personas desempleadas, promover la formación de trabajadores/as e integrar los intereses de personas, colectivos y empresas en el desarrollo económico y social de la ciudad

Esta plataforma de datos y toma de decisiones persigue precisamente promover el desarrollo económico y social de la ciudad, mediante la puesta a disposición de todo tipo de documentos, set de datos y aplicaciones para que los agentes económicos y sociales de la ciudad tengan, en un repositorio común, el mayor número de datos, documentos y herramientas de soporte a la toma de decisiones de los diferentes agentes socio-económicos de la ciudad.

Puede encontrarse en la siguiente dirección:

<http://www.opendatacordoba.com/>

▪ Datos.gijon.es



Repositorio de datos de origen público del Ayuntamiento de Gijón. Este repositorio se enmarca dentro de las últimas iniciativas surgidas a nivel local el 16 de Noviembre de 2010, donde la Junta de Gobierno dio luz verde a la resolución que facilitará la reutilización de la información pública del Ayuntamiento.

La participación de CTIC (ver *Marco tecnológico* en este documento) ha consistido en el desarrollo de un plan RISP mediante un proceso de consultoría e identificación de conjuntos de datos.

Entre los aspectos técnicos más destacados del portal está la asociación de los contenidos estructurados del nuevo portal del Ayuntamiento con vocabularios semánticos a través del gestor de contenidos de la plataforma. Esta relación permite la publicación automática de datos en RDF, bien a través de ficheros estáticos generados o a través de RDFa¹ embebido en las páginas web.

Por otro lado, también se ha preparado la publicación del catálogo de datos mediante Linked Data (ver *Marco tecnológico* en este documento) usando el vocabulario RDF para la descripción de conjuntos de datos dcat².

Puede encontrarse en la siguiente dirección: ***<http://datos.gijon.es/>***

¹ Conjunto de extensiones de XHTML propuestas por W3C para introducir semántica en los documentos.

² <http://vocab.deri.ie/dcat-overview>

■ Zaragoza.es/ciudad/risp



Repositorio de datos de origen público del Ayuntamiento de Zaragoza. Estamos ante una plataforma RISP¹ completa que incluye datos estructurados de gran calidad e información modelada y expuesta siguiendo la aproximación Linked Data. Cuenta con un almacén de tripletas RDF instalada sobre sus bases de datos y un punto de consulta SPARQL para lectura de los datos expuestos. Este almacén de datos semánticos y el modelado de los mismos en formatos estándar es utilizado internamente para mejora de la interoperabilidad de la información administrativa.

Dispone de un catálogo expresado en formato RDF utilizando un vocabulario internacionalmente reconocido para el modelado de catálogos de conjuntos de datos – dcat² - . La sección de aplicaciones incluye algún ejemplo de reutilización, como el planificador de rutas turísticas³ y buscadores integrados en redes sociales como Facebook.

Otra de las novedades incluidas en el portal es la utilización de RDFa, una tecnología que permite declarar datos incrustados en las páginas HTML, de forma que se puedan procesar automáticamente sin afectar a la representación visual.

El catálogo incluye cientos de conjuntos de datos en formato abierto y una decena en formato semántico: puntos de interés turístico, organigrama, trámites y servicios, direcciones de Zaragoza y ofertas de empleo, entre otros.

Puede encontrarse en la siguiente dirección: <http://www.zaragoza.es/ciudad/risp/>

¹ Reutilización de la información del sector público

² <http://vocab.deri.ie/dcat-overview>

³ http://www.zaragoza.es/turruta/Turruta/index_Ruta

▪ Instituto Geográfico Nacional



Agencia pública de carácter nacional en el ámbito de catalogación del territorio e información geográfica.

El *Instituto Geográfico Nacional* <http://www.ign.es> ofrece no sólo repositorios de datos para su descarga sino también un buen número de servicios en línea donde es posible acceder a multitud de datasets, que incluyen entre otros:

- IDEE¹ Infraestructura de Datos Espaciales de España.
- Buscador² de nombres geográficos.
- IBERPIX³ Servidor de imágenes.
- Cartociudad⁴ Callejero, información censal y postal de todo el territorio nacional.
- SIOSE⁵ Sistema de Información sobre Ocupación del Suelo de España.
- Sistema de información geográfica (Signa) que incluye información sobre morfología, comunicaciones, zonas protegidas, límites administrativos.

Centro de descargas⁶ (que proporciona informaciones, tanto sin licencia de uso, como con licencia de uso no comercial y comercial).

¹ http://www.idee.es/show.do?to=pideep_pidee.ES

² <http://www.idee.es/IDEE-Gazetteer/index.html?locale=es>

³ <http://www2.ign.es/iberpix/visoriberpix/visorign.html>

⁴ <http://www.cartociudad.es/portal/1024/serviciosOGC.htm>

⁵ <http://www.ign.es/siose/>

⁶ <http://centrodedescargas.cnig.es/CentroDescargas/index.jsp>

■ Agencia Española de Meteorología



Agencia pública de carácter nacional con información meteorológica.

La *Agencia Española de Meteorología* <http://www.aemet.es/> ha comenzado en diciembre de 2010 la liberación masiva de sus datos desde un sitio específico¹ de su web.

Las informaciones incluyen datos de observación, Boletines del Sistema Mundial de Telecomunicaciones de la OMM, datos de Radiación solar, sondeos de ozono, datos de la red de radares, rayos, modelos numéricos, series climatológicas, etc.

¹ <ftp://ftpdatos.aemet.es/>

2.3. Agentes Infomediarios

El término **Infomediario** (“intermediario de información”) fue acuñado por primera vez en el libro “Net Worth”, cuyos autores John Hagel y Marc Singer (Harvard Business School Press, Enero 1999), analizan los profundos cambios en el desarrollo de nuevos modelos de negocio derivados de la aparición del comercio electrónico.

Una **empresa infomediaria** es aquella dedicada a analizar y tratar información procedente del sector público, para crear productos de valor añadido destinados a terceras empresas o a la ciudadanía en general.

En España se han identificado a 230 empresas, catalogadas en subsectores, en función del ámbito de información que reutilizan: Negocio/ Económico, Jurídico/ Legal, Geográfico/ Cartográfico, Meteorológico, Socio demográfico/ Estadístico y de Transportes.

La información que reutilizan procede mayoritariamente de organismos nacionales, aunque la mitad de las empresas reutilizan también datos internacionales. A nivel nacional, estas compañías se concentran principalmente en Madrid, Cataluña y País Vasco.

Con el objeto de proyectar una imagen más clara de lo que es una infomediaria, presentaremos un par de casos de éxito. Éstos han sabido tratar la información procedente del sector público, para de esta forma crear productos de valor añadido:

■ Euroalert



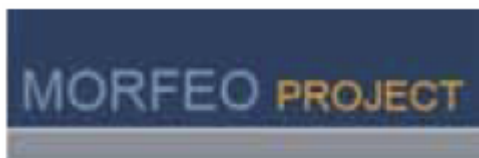
Fundación: 1999.

Miembros: Empresas y entidades públicas.

Web: Euroalert.net

Euroalert.net ofrece acceso diario a información, contenidos y servicios relevantes relacionados con la Unión Europea, en español y en inglés. Euroalert está trabajando en el proyecto “[10ders Information Services](#)” que pretende construir un prototipo de plataforma pan-europea que agregue todos los concursos públicos de los 27 estados miembros de la UE, con el fin de diseñar productos y servicios de información baratos y accesibles que ayudarán a las pymes a ser más competitivas en este mercado, lo que pondrá a disposición de pequeñas y medianas empresas de toda Europa oportunidades por valor del 17% del PIB de la UE.

▪ **Linked / Open data en proyecto Morfeo**



Fundación: 2010

Miembros: Empresas y entidades públicas.

Web: <http://linked-open-data.morfeo-project.org/>

El proyecto Morfeo es una agrupación de empresas, asociaciones empresariales, organizaciones y entidades públicas para el desarrollo de software de fuentes abiertas y promoviendo los estándares abiertos.

Desde Septiembre de 2010, se ha creado un nuevo capítulo para el desarrollo de soluciones tecnológicas en el ámbito de linked y open data. Este capítulo tiene como objetivo en Morfeo definir y estandarizar aplicaciones y plataformas para el uso de datos abiertos (opendata) tanto de origen público como privado. Por ello, promueve el incremento de las tecnologías disponibles que respeten estos estándares y los servicios que las utilicen. Al mismo tiempo, busca incrementar la inversión en I+D y la identificación de proyectos estratégicos, así como la identificación de estudios clave o documentación necesaria para acercar estas tecnologías a sus participantes.

Si desea conocer más casos de éxito, puede dirigirse al blog¹ de nuestro proyecto.

¹ <https://datosenabierto.wordpress.com/2011/06/24/overview-agentes-infomediarios-nivel-nacional/>

2.4. Productos y servicios open data

Éstos son el resultado final del proceso de reutilización. Proceso que en un principio partió de los datos de administraciones públicas o empresas privadas y que ha desembocado en productos con un valor añadido y de utilidad a la sociedad.

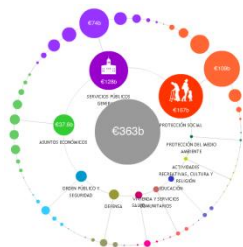
El mayor grueso de las aplicaciones open data están enfocadas a sectores de movilidad, geolocalización, transporte y meteorológico. Además de otro tipo de aplicaciones de carácter divulgativo o de conocimiento general, dando a conocer datos sobre influencia política y control presupuestario.

De entre un nutrido listado de ejemplos, nacional e internacional, hemos seleccionado dos ejemplos representativos de este tipo de aplicaciones finales. Si usted desea conocer el resto de aplicaciones, puede dirigirse a la siguiente dirección:

<https://datosenabierto.wordpress.com/estado-del-arte/>

En ella, usted podrá encontrar el listado de aplicaciones clasificado a nivel nacional e internacional. A continuación paso a mostrar dos ejemplos:

▪ Donde van mis impuestos



Aplicación de ámbito nacional, no georeferenciada y orientada al control presupuestario. La aplicación visualiza los gastos anuales de la Administración Central del Estado (ministerios, agencias dependientes y otros organismos públicos, como el Congreso) y la Seguridad Social, tal y como aparecen recogidos en los presupuestos Generales.

Es una réplica del portal británico <http://wheredoesmyMoneygo.org>, creado por la Open Knowledge foundation. La visualización sigue el estándar COFOG de las naciones unidas para las partidas de gastos identificadas.

Para más información puede visitar la siguiente dirección:
<http://www.dondevanmisimpuestos.es>

■ Roadify



Aplicación dedicada no solo a informar del estado del tráfico rodado y la situación del transporte público en la ciudad, sino también a aconsejar entre las diferentes opciones que han sido elegidas.

Aplicación creada en New York y georeferenciada que combina redes sociales y mapas.

Para más información puede visitar la siguiente dirección:
<http://www.roadify.com/>.

2.5. Usuarios de Open data

Si bien todos los ciudadanos y empresas de una sociedad son potenciales usuarios de los datos abiertos, merece especial atención el planteamiento reactivo que plantean los agentes sociales. Éstos, buscan la **protección social** frente a los efectos de la desigualdad y desequilibrios sociales.

A continuación pasamos a citar los agentes sociales más característicos:

▪ Pro bono público

Pro Bono Publico

Por la apertura de datos públicos

Fundación: 2009.

Miembros: Personas.

URL: <http://www.probp.org/>

De la expresión latina *pro bono* “para el bien público”, nace esta asociación que busca la apertura de datos públicos y el uso de la tecnología para aumentar la transparencia como forma de mejorar nuestra vida en sociedad.

Esta plataforma, está promoviendo acciones como la apertura del Registro Mercantil Español¹. Apoyándose en que una de las funciones principales del Registro es publicitar la información empresarial que posee, para mayor seguridad jurídica y económica, Pro Bono Público entiende que España es uno de los países donde más complicado resulta acceder a la información, ya que el Registro Mercantil es de acceso restringido a los usuarios que se registren y hay que pagar para poder ver la mayoría de la información.

Otra iniciativa de Pro Bono Publico es el **Desafío AbreDatos**, el principal concurso en España de generación de aplicaciones a partir de datos públicos, cuya finalidad es fomentar el uso de estándares abiertos, el acceso a la información de las administraciones públicas e incentivar la creación de nuevos servicios y aplicaciones a partir de la información del sector público.

¹ http://es.wikipedia.org/wiki/Registro_mercantil_de_Espa%C3%B1a

▪ Access info Europe



Fundación: 2006.

Miembros: Personas.

URL: <http://www.access-info.org>

Access Info Europe es una organización de derechos humanos dedicada a la promoción y protección del derecho de acceso a la información en Europa como herramienta para la defensa de nuestras libertades civiles y los derechos humanos, para facilitar la participación pública en la toma de decisiones, y para exigir al gobierno una rendición de cuentas.

Con este objetivo, **Acces Info** lleva a cabo proyectos de distinta naturaleza, incluyendo monitoreos de transparencia, análisis de las leyes de acceso a la información, promoción social del derecho de acceso a la información, creación de guías sobre el uso de este derecho o un servicio de información sobre el mismo.

Access Info trabaja desde hace ya más de dos años en el estudio de la relación y el impacto del movimiento Open Data en la teoría y la práctica del derecho de acceso a la información. En concreto, en 2011 Access Info publicó junto con la Open Knowledge Foundation un informe, “Beyond Access¹”, que trata esta relación y donde se establece la necesidad de que las leyes de acceso a la información garanticen el acceso a bases de datos. Además, en este informe también se establecen los requisitos formales que deben tener los datos que se publiquen para ser considerados accesibles y reutilizables (“open”), estos deben ser publicados en formato electrónico, procesables electrónicamente, utilizando programas de formato abierto y libres de cualquier derecho de propiedad.

¹ http://es.wikipedia.org/wiki/Registro_mercantil_de_Espa%C3%B1a

▪ Coalición pro acceso



Fundación: 2006.

Miembros: Organizaciones¹ (46). Personas pueden adherirse a sus principios.

URL: <http://www.proacceso.org>

La **Coalición Pro Acceso** es una plataforma formada por organizaciones de la sociedad civil e individuos que se constituyó en Octubre de 2006 con el fin de promover la adopción e implementación de una Ley de Acceso a la Información en España.

Su aproximación al Open Data es, por tanto, social y no industrial, y aunque es una plataforma que agrupa a asociaciones y otras entidades jurídicas, está abierta a que ciudadanos² suscriban sus 9 principios.

1. El derecho a la información es un derecho de todas y todos.
2. El derecho se aplica a todas las entidades públicas.
3. Realizar solicitudes debe ser sencillo, rápido y gratuito.
4. Los funcionarios tienen la obligación de ayudar a los solicitantes.
5. Principio de publicidad de la información: el secreto y la denegación de la información son la excepción.
6. Las denegaciones de acceso a la información deben ser limitadas y estar debidamente motivadas.
7. Toda persona tiene el derecho de recurrir las denegaciones de acceso o la no contestación a las solicitudes realizadas.
8. Las entidades públicas, a iniciativa propia, deben poner a disposición del público información básica y esencial sin que sea necesario realizar una solicitud.
9. El derecho de acceso a la información debe ser garantizado por un órgano independiente.

¹ <http://www.proacceso.org/about-2/>

² <http://www.proacceso.org/firmantes/>

2.6. Clasificación del Opendata

Aunque prácticamente todo tipo de informaciones son susceptibles de entrar en un proceso de Open Data, siempre que se respeten las restricciones legales de seguridad y de protección de datos personales, se enumeran aquí algunos ejemplos de informaciones que ya son reutilizadas o lo son potencialmente.

INFORMACIÓN ECONÓMICA

- Información sobre empresas (registro mercantil).⁵⁵
- Información de concursos públicos.⁵⁶
- Información de adquisiciones.
- Sinistralidad laboral.
- Presupuestos.⁵⁷

INFORMACIÓN GEOGRÁFICA

- Información direcciones (carreteras y calles).⁵⁸
- Fotografías del territorio.⁵⁹
- Datos geológicos e hidrográficos.
- Datos topográficos.

INFORMACIÓN TRANSPORTE Y TRÁFICO

- Situación de congestión.
- Obras y desvíos.⁶⁰
- Cámaras en carretera.⁶¹
- Concentración de accidentes.⁶²

INFORMACIÓN LEGAL

- Resoluciones legales.⁶³
- Tratados y convenios.
- Resoluciones de consumo.

INFORMACIÓN METEOROLÓGICA

- Datos climatológicos (temperatura, precipitación viento, humedad, etc).⁶⁴

INFORMACIÓN SOCIEDAD

- Datos demográficos.⁶⁵
- Datos socio sanitarios.⁶⁶
- Datos culturales.
- Datos seguridad ciudadana.
- Datos patrimoniales.

CONTENIDOS DIGITALES

- Patrimonio RTVE.*⁶⁷
- Otro patrimonio audiovisual histórico.

En el ámbito público, en general, todas aquellas que produce la administración y que no contengan información sensible por seguridad o privacidad de personas, como se establece en el apartado 3.3 de la ley 37 / 200768 son potencialmente reutilizables.

2.7. Situación actual de los Open Data en España

En este punto analizamos la situación de los open data en España. Éste análisis hará hincapié en la situación de la administración pública, pasando por los aspectos técnicos que posibilitan la implantación de esta filosofía de reutilización. Además veremos la implantación del sector infomediario en el país, así como los beneficios de la apertura de datos para la ciudadanía Española. Por último terminamos describiendo brevemente el futuro del Open Data en España.

2.7.1. El papel de la Administración

La Administración va superando poco a poco su obsesión por la posesión de los datos¹ y **ha comenzado a asumir que los datos no le pertenecen y que su papel se limita al de una mera gestora de la información**. Pocos dudan ya de que abrir los datos contribuye a mejorarlos, corregir errores, incrementar la interoperabilidad y crear valor añadido gracias a cruzar distintas fuentes de información. El debate dentro de la Administración se centra ahora en otros aspectos, como por ejemplo la preocupación por las posibles consecuencias que deriven del uso de datos que contienen errores y la responsabilidad que eso conlleva.

Debemos también ser cuidadosos con que la Administración no se convierta en un competidor privilegiado de la empresa privada, y para ello **no debería encargarse de la explotación directa de los datos, ni de la tecnología necesaria para realizar dicha explotación, sino tan solo de su adecuada exposición al público a través de los medios más convenientes**. Sin embargo, esto abre también el debate de hasta qué punto algunos servicios públicos básicos podrían o deberían delegarse en la empresa privada.

Ahora toca afianzar las iniciativas² y asegurar su supervivencia para no queden en una mera moda pasajera, incluyendo la apertura de datos como otro nuevo canal permanente de comunicación con el ciudadano y haciendo de la reutilización un valor rentable para todos.

¹ <http://mashable.com/2011/01/12/data-ownership/>

² <http://opendata.blog.euskadi.net/blog-es/noticias/primer-aniversario-de-open-data-euskadi/>

2.7.2. Los aspectos técnicos

Todavía hoy en día se detecta una **gran dificultad para ejercer la labor del infomediario**, debido principalmente a la baja calidad de los datos y la falta de preparación para su reutilización. La diversidad de vocabularios y la proliferación de formatos difíciles de tratar de forma automatizada consumen también una gran parte de los recursos en las iniciativas sin aportar ningún valor añadido.

Existe **unanimidad en cuanto a la importancia de la armonización en las iniciativas**, la utilización de suficiente meta información para poder valorar adecuadamente los datos y la creación de estándares al respecto¹ para conseguir **unificar y documentar vocabularios** en las áreas donde todavía no existen. Con todo ello conseguiríamos que el potencial de reutilización fuese mayor y ampliar el mercado.

Tampoco hay que perder de vista que **el *Open Data* necesita de una aproximación integral**, y no únicamente desde una perspectiva técnica. Los retos técnicos son en general menores que los conceptuales, ya que el camino a seguir está hoy en día más claro en la parte técnica con respecto a otros aspectos organizativos, conceptuales, políticos y de evangelización.

Debemos sin embargo también tener en cuenta que, si bien con respecto a la *transparencia* lo fundamental es el volumen de información abierto, en cuanto a la *reutilización* es también primordial contar con una buena base tecnológica. Es por ello que resulta muy importante también llegar a un **compromiso adecuado entre la inmediatez de las iniciativas *raw data*² y la potencia ofrecida por el *Linked Data*³**, llegando a un equilibrio que permita la sostenibilidad a largo plazo⁴.

Por tanto, no podemos obviar los problemas técnicos y debemos darles solución, cuestión que en general no es trivial porque no partimos de cero, sino de sistemas de información pre-existentes que originalmente no estaban pensados para ser expuestos. Esto conlleva un trabajo adicional además de tener que afrontar otros retos relacionados con la automatización de los procesos, la robustez y escalabilidad de los sistemas, la actualización de la información, etc.

¹ <http://www.w3.org/2011/gld/charter>

² Datos en crudo, también conocidos como datos fuentes o datos atomizados, son datos que no han sido procesados para ser utilizados.

³ <http://www.w3c.es/divulgacion/guiasbreves/LinkedData>

⁴ <http://lab.linkeddata.deri.ie/2010/star-scheme-by-example/>

2.7.3. El sector de los infomediarios

Si bien la Administración todavía no está completamente preparada para la reutilización, da la impresión de que el sector empresarial tampoco lo está.

El sector de los infomediarios en España está actualmente muy focalizado hacia nichos concretos relacionados con sectores como geografía, economía, jurídico, consultoría, etc. Se echan en falta nuevas iniciativas que ayuden a ampliar el rango de sectores y abrir **nuevas líneas de negocio** a partir de los nichos de mercado que quedan todavía por cubrir gracias a la apertura de información y a la transparencia.

Las empresas reutilizadoras juegan un papel fundamental en la cadena de generación de valor y es imprescindible contar con ellas, pero para ello debemos modificar la visión actual del modelo de negocio un tanto anticuada y distinta a la que podemos encontrarnos fuera del país, olvidándonos cuanto antes de vender datos y pasar a vender servicios.

Los **dispositivos móviles y la Web única**¹ jugarán también un papel importante, no solo como herramientas de consumo, sino también como un medio fundamental de generación de datos por parte de la ciudadanía, pasando esta a desempeñar también un papel activo y no solo como consumidora.

También hay que tener en cuenta que el desarrollo de aplicaciones es tan sólo una parte del potencial del *Open Data*, y que **existen otras vías de negocio todavía por explorar** como la publicidad, los servicios de suscripción, la transformación de los datos, etc.

Finalmente, existen también algunos aspectos que lastran la posibilidad de generación de negocio por parte de los infomediarios, como por ejemplo la **falta de definición y claridad en el contexto legal** o la **posibilidad de que se impongan tasas para el acceso a los datos**. Una mayor claridad y unificación en los términos de las licencias y un modelo de retorno de inversión que no se basara en la imposición de tasas ayudarían a la evolución positiva del sector.

¹ <http://www.w3.org/TR/mobile-bp/#OneWeb>

2.7.4. Los beneficios para la ciudadanía

Quizás el reto más importante sea cómo hacer llegar realmente al ciudadano los beneficios de la apertura de datos ya que **si fuese consciente de los potenciales beneficios demandará Open Data de forma natural**. La difusión y formación desde la base es por tanto fundamental y necesaria tanto de forma interna como externa. Otro factor importante es la escucha activa¹ para identificar los intereses de ciudadanos² e infomediarios³.

La gente entiende mejor el open data a través de aplicaciones concretas y es por ello que iniciativas del tipo de AbreDatos⁴, si bien tienen un potencial limitado a la hora de generar negocio entorno al Open Data, sí constituyen un buen escaparate de cara al público y un componente demostrador de cara a la ciudadanía que les ayuda a ser conscientes de los beneficios. No obstante, no podemos estancarnos ahí, y debemos promover **políticas completas de participación, difusión y fomento del consumo** que involucren a todos los agentes.

La información liberada y su potencial utilidad es también un factor primordial de cara a atraer la atención del público. **Existen datos muy interesantes y demandados por la sociedad que siguen sin publicarse**, como por ejemplo información relacionada con la seguridad ciudadana, el turismo, tráfico y transporte público o agendas de eventos.

No obstante, también hay una sensación general de que nadie tiene muy claro cuáles son los datos realmente útiles y dónde están, de ahí la tendencia hacia políticas *raw data* hoy en día, con la intención de dar a conocer lo antes posible cuál es la información disponible. **Una política de transparencia activa ayudaría a facilitar esta labor fundamental de identificación**, y en este aspecto también juegan un papel fundamental los catálogos de dato⁵s a la hora de facilitar la localización de la información.

¹ <http://zaragoza.uservoice.com/forums/64643-datos-que-me-gustar-a-reutilizar>

² <http://navarra.uservoice.com/forums/90007-open-data-navarra>

³ <http://navarra.uservoice.com/forums/96955-open-data-navarra-empresas>

⁴ <http://www.abredatos.es/>

⁵ <http://datos.fundacionctic.org/sandbox/catalog/faceted/>

2.7.5. Futuro próximo del open data en España

En los próximos días, semanas y meses seguiremos viendo nuevos proyectos y muchas y excitantes novedades sobre la materia, como la publicación de un estudio del sector infomediario realizado por el ONTSI¹ que nos ofrecerá importantes datos, nuevos proyectos interesantes en materia de estadística², un campo en el que la reutilización es muy importante, y el nuevo portal de reutilización nacional de España - **datos.gob.es** – que cuenta con la participación del equipo Open Data @ CTIC³.

Podemos pues concluir que **España cuenta con una comunidad Open Data activa y cada vez más madura que justifica su posición destacada entre las iniciativas internacionales** y los retos a los que nos enfrentamos son similares a los de otras iniciativas y comunidades con grados de madurez similares, por lo que debemos seguir evolucionando para superarlos.

¹ <http://www.ontsi.red.es/index.action>

² <http://ine.es/>

³ <http://datos.fundacionctic.org/about/>

2.8. Catálogos Open Data

En esta sección presentaremos los más importantes portales Open Data agrupados según el continente en el que se encuentren – UE, Asia, EEUU - . De esta forma se facilita la localización de los catálogos englobados en su enclave geográfico.

Figura 10: Europa y Africa

2.8.1. Catálogos UE Open Data

Reino unido

<http://data.gov.uk/>
<http://www.lichfielddc.gov.uk/data>
<http://picandmix.org.uk/categories/>
<http://www.opendatani.info/>
<http://data.london.gov.uk/>
<http://opendata.warwickshire.gov.uk/>
http://www.manchester.gov.uk/info/500215/open_data
<http://www.sunderland.gov.uk/index.aspx?articleid=4112>

España

<http://opengov.es/>
<http://www.proyectoaporta.es/web/guest/catalogo-de-informacion-publica>
<http://www.zaragoza.es/ciudad/servicios/conjuntodatos.htm>
<http://opendata.euskadi.net/w79-home/es>
<http://risp.asturias.es/catalogo/index.html>
<http://datos.gijon.es/>

2.8.2. Catálogos Asia

Australia

<http://data.australia.gov.au/>

Nueva Zelanda

<http://www.data.govt.nz/>



Figura 11: Asia

2.8.3. Catálogos América

Canadá

<http://www.data.gc.ca/default.asp?lang=En>
<http://openparliament.ca>
<http://data.edmonton.ca/>
<http://www.mississauga.ca/data>
<http://www.nanaimo.ca/datafeeds>
http://www.ottawa.ca/online_services/opendata/index_en.html
<http://www.toronto.ca/open>
<http://data.vancouver.ca/>

Norte América

<http://data.wa.gov/>
<http://data.oregon.gov/>
<http://data.ca.gov/>
<http://www.utah.gov/data/>
<http://openbooks.az.gov/app/transparency/index.html>
<http://www.colorado.gov/data/>
<http://www.nd.gov/gis/>
<http://www.nebraska.gov/data/>
<http://www.kansas.gov/KanView/>
<http://www.ok.gov/about/data.html>
<http://www.texas.gov/en/Connect/Pages/open-data.aspx>
<http://www.mn.gov/data/>
<http://www.mo.gov/data/>
<http://wwwprd.doa.louisiana.gov/LaTrac/portal.cfm>
<http://www.michigan.gov/data/>
<http://inmap.indiana.edu/viewer.htm>
<http://www.sos.state.oh.us/betterLives.aspx>
<http://opendoor.ky.gov/search/Pages/s>
<http://www.tn.gov/opengov/>
<http://open.alabama.gov/>
<http://www.open.georgia.gov/>
<http://www.floridahasarighttoknow.com/>
<http://www.ncopenbook.gov/>
<http://datapoint.apa.virginia.gov/>
<http://www.nysenate.gov/opendata/>

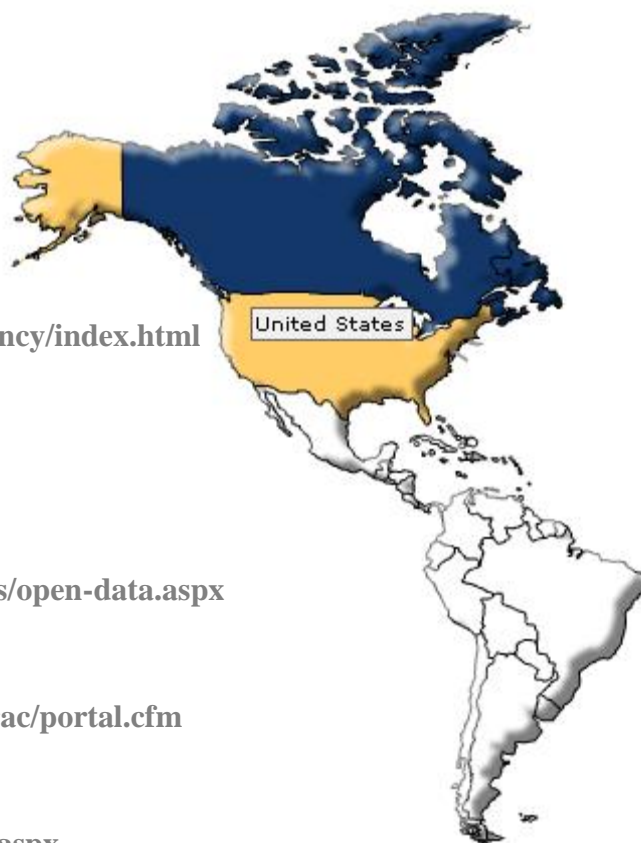


Figura 12: América

<http://www.maine.gov/data/>
<http://www.mass.gov/data/>
<http://www.nyc.gov/data>
<http://data.octo.dc.gov/>

3. Open Data Gijón

3.1. Introducción

La idea básica del proyecto consiste en la creación de una aplicación web urbana, accesible desde cualquier dispositivo móvil o fijo. Ésta aplicación web ha de hacer uso del enfoque proporcionado por la filosofía *OpenData*. Filosofía de reutilización de datos abiertos al público (por parte de administraciones públicas o privadas) para transformarlos en productos de valor añadido y de utilidad para la sociedad.

Tras varias semanas de búsqueda encontramos un portal Open data, *datos.gijon.es*, que se ajustaba a nuestras necesidades. Éste nos ofrecía una serie de catálogos públicos (generados por colectivos públicos y/o privados) listos a reutilizar y convertir en algún tipo de aplicación que ofreciese una funcionalidad añadida y de utilidad.

De entre el conjunto de catálogos públicos ofertados en dicha fuente de datos, encontramos un dataset que se ajustaba al principal requerimiento que nuestra aplicación web tendría. Éste requerimiento responde a la necesidad de una fuente de datos consistente y robusta, basada en el flujo de información a tiempo real de una red de autobuses. Y esto es justamente lo que encontramos en el dataset *BusGijón*¹, un conjunto de datos en tiempo real de paradas y autobuses basado en el servicio que proporciona EMTUSA² a la ciudad de Gijón.

Varios son los formatos en los que el portal *datos.gijon.es* pone a disposición pública los datos procedentes del dataset *BusGijón*, cuyo sistema de información es diferente al sistema seguido en el portal. Estos formatos son XML y JSON, ya tratados en el capítulo 2.

El sistema de información del cual proceden los datos que el portal *datos.gijon.es* pone a disposición pública es el servicio web *BusGijón*. Este sistema de información es la fuente de datos originaria de la que se nutrirá nuestra aplicación y que tiene por dirección <http://docs.gijon.es/sw/busgijon.asmx>.

¹ http://datos.gijon.es/risp_datasets/show/busgijontr

² Empresa de Transportes Urbanos Sociedad Anónima

3.2. ¿Qué es *datos.gijon.es*?

Es un portal temático localizado en el sitio web del Ayuntamiento de Gijón, <http://datos.gijon.es>. Este portal RISP (reutilización de la información del sector público) es el punto informativo y divulgativo central de los servicios de reutilización de información municipal. Contiene un catálogo¹ de datos con información específica para cada uno de los conjuntos de datos que pueden ser reutilizados, como es la licencia y términos de uso, fechas de creación, procedencia, etc. Además, alberga parte de las aplicaciones² piloto que consumen los datasets³ publicados, así como servicios de soporte para la participación ciudadana.

Este portal tiene su origen como una de las últimas iniciativas surgidas a nivel local el 16 de Noviembre de 2010, donde la Junta de Gobierno dio luz verde a la resolución que facilitó la reutilización de la información pública del Ayuntamiento.

El desarrollo de un plan **RISP** (reutilización de la información del sector público), llevado a cabo mediante un proceso de consultoría e identificación de conjunto de datos, fue ejecutado gracias a la participación del Centro Tecnológico de la Información y Comunicación – **CTIC**.

Entre los aspectos técnicos más destacados del portal está la asociación de los contenidos estructurados del nuevo portal del Ayuntamiento con vocabularios semánticos a través del gestor de contenidos de la plataforma. Esta relación permite la publicación automática de datos en RDF⁴, bien a través de ficheros estáticos generados o a través de RDFa embebido en las páginas web.

Por otro lado, también se ha preparado la publicación del catálogo de datos mediante Linked Data usando el vocabulario RDF para la descripción de conjuntos de datos dcat⁵.

Como ya se ha comentado, de entre los catálogos de datos disponibles en esta web, tomamos como fuente de datos para nuestro proyecto el catalogo ‘*BusGijón*’. Como se explicó anteriormente, el proyecto, se nutre de un servicio web del mismo nombre, objeto de nuestro estudio.

¹ <http://datos.gijon.es/site/43/page/1808-catalogo-de-datos>

² <http://datos.gijon.es/site/43/page/1809-aplicaciones>

³ Colección de datos, normalmente representados de forma tabular. Cada columna representa una variable particular y cada fila corresponde a un miembro determinado del conjunto de datos en cuestión.

⁴ Lenguaje de objetivo general para representar la información en la web

⁵ <http://vocab.deri.ie/dcat-overview>

3.3. ¿Qué es y para qué sirve un servicio web?

Un web service es básicamente una función o procedimiento que puede ser accedida via web por cualquier programa o aplicación sin importar en que plataforma reside el servicio o en que lenguaje ha sido desarrollado.

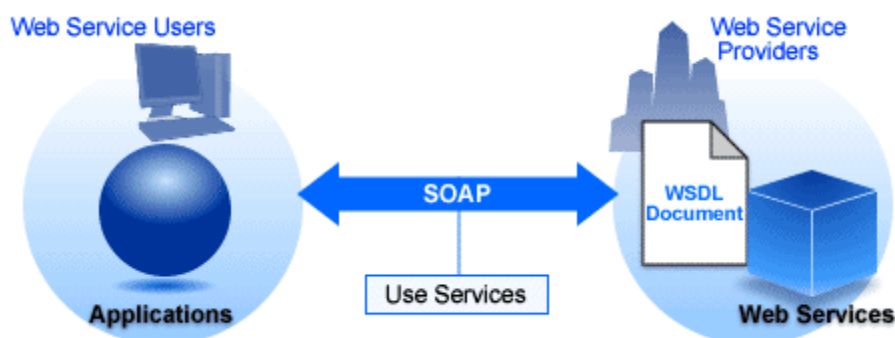


Figura 13: Esquema de comunicación en un servicio web

Para lograr este acceso o comunicación, aparte de basarse en estándares, un web service se comunica mediante mensajes **XML** utilizando el protocolo SOAP. Para establecer un diálogo coherente entre el WSC (Web Service Cliente), que envía la petición y recibe la respuesta y el WSS (Web Service Server), el que ejecuta el proceso y envía la respuesta, se utiliza SOAP (Simple Object Access Protocol), que es una codificación basada en XML.

Cualquier servicio web, en vez de obtener peticiones desde un navegador y devolver páginas web como respuesta, recibe peticiones, mediante un mensaje formateado con SOAP, desde otras aplicaciones realiza la labor que le han pedido y devuelve un mensaje de respuesta también con formato SOAP. En este proyecto usaremos una implementación de SOAP a nivel cliente en el lenguaje Java.

El interfaz de un web service se describe en un documento XML utilizando el estándar WSDL. En este fichero, que recibe el mismo nombre WSDL, se definen los detalles de cómo usar un web service, como las operaciones, el formato de los mensajes, el tipo y el protocolo de transporte del web service. El documento WSDL que usará nuestra aplicación cliente será <http://docs.gijon.es/sw/busgijon.asmx?wsdl>. A la dirección del servicio web hemos añadido una marca que indique que lo queremos es un lenguaje descriptor del servicio web, de ahí las siglas WSDL.

3.4. Estructura del servicio web BusGijón

Para crear un servicio de valor añadido a partir de información pública y abierta a la ciudadanía, antes es preciso analizar la estructura que presenta. Es por ello que hemos de analizar la estructura del servicio web de la red de autobuses de Gijón (*BusGijón*¹), con el objetivo de comprender qué tipo de datos podemos obtener de dicho servicio web.

Como ya se ha comentado, este servicio web, basado en los datos a tiempo real de paradas y autobuses del servicio que proporciona la empresa EMTUSA a la ciudad de Gijón, es una de las piedras angulares de nuestro proyecto. La dirección de este servicio web documentado es la siguiente: <http://docs.gijon.es/sw/busgijon.asmx>.

Al acceder a la página principal de este servicio web, vemos una pantalla como la de la siguiente imagen, la cual corresponde a una descripción del conjunto de operaciones disponibles a ser consultadas por un servicio web cliente. Ese servicio web cliente será nuestro programa consumidor de datos, el cual será detallado en sucesivas secciones.

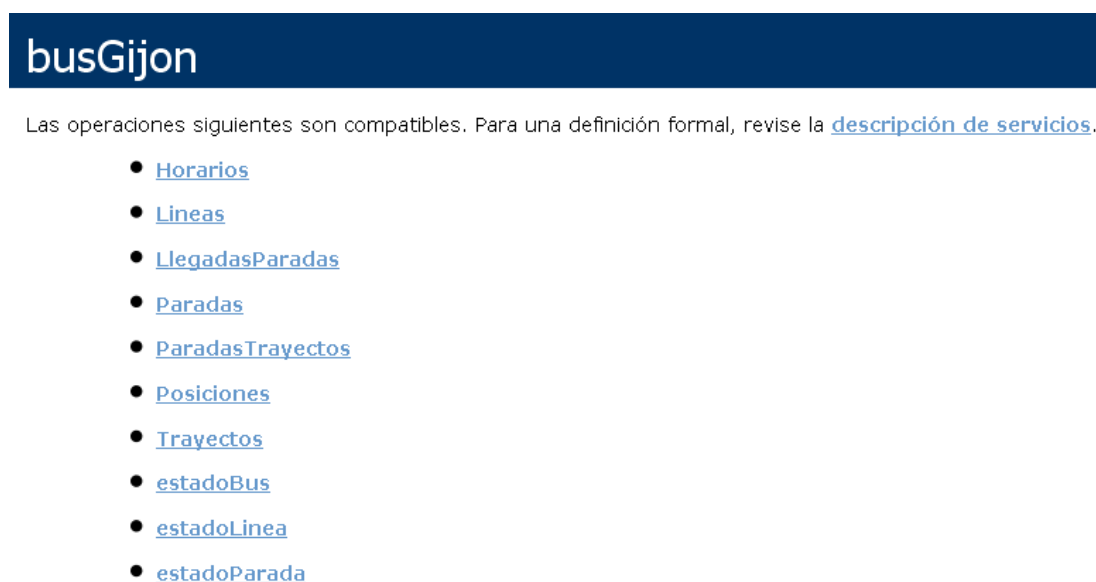


Figura 14: Página principal del servicio web busGijon

¹ <http://docs.gijon.es/sw/busgijon.asmx>

En dicho servicio web tenemos a nuestra disposición 10 operaciones, también pueden denotarse por funciones, listas para devolvernos los datos a tiempo real característicos de cada una de ellas.

A continuación paso a mostrar una descripción de cada una de las operaciones presentes en 'busGijón', además de los tipos que contienen:

- **Horarios:** Operación que ofrece información horaria sobre las distintas líneas y trayectos a lo largo de una semana.

FechaInicio : dateTime
FechaFin : dateTime
Hora : String
idLinea : Integer
idTrayecto : Integer
NumeroExpedicion : Integer

- **Líneas:** Operación que arroja las líneas existentes en la red de autobuses de Gijón. En total son 32 líneas.

idLinea : Integer
Descripcion : String
Tipo : String

```
idLinea : 1 : Tipo : NORMAL : Descripción : CERILLERO - HOSPITAL DE CABUEÑES
```

Figura 15: Consulta realizada al servicio web - Líneas

- **Paradas:** Ofrece información descriptiva y de geolocalización sobre las 605 paradas existentes.

idParada : Integer
Descripcion : String
utmX : Integer
utmY : Integer

```
idParada : 1 : [utmX, utmY] : 281881, 4827378 : Descripción : MUSEL
```

Figura 16: Consulta realizada al servicio web - Paradas

- **Trayectos:** Ofrece información sobre los viajes de ida y vuelta pertenecientes a una línea concreta.

idTrayecto : Integer
idLinea : Integer
Descripcion : String
idCabecera : Integer

Dirección : Integer

Destino : String

```
idTrayecto : 1 : idLinea : 1 : idCabecera : 180 : idDirección : 1 :  
idDestino : HOSP. CABUENES : idDescripción : L-1. CERILLERO - HOSPITAL  
DE CABUENES  
  
idTrayecto : 2 : idLinea : 1 : idCabecera : 34 : idDirección : 2 :  
idDestino : CERILLERO : idDescripción : L-1. HOSPITAL DE CABUENES -  
CERILLERO  
  
idTrayecto : 6 : idLinea : 1 : idCabecera : 34 : idDirección : 2 :  
idDestino : CERILLERO : idDescripción : L-1. HOSPITAL DE CABUENES -  
CERILLERO  
  
idTrayecto : 7 : idLinea : 1 : idCabecera : 34 : idDirección : 2 :  
idDestino : CERILLERO : idDescripción : L-1. HOSPITAL CABUENES-  
CERILLERO (POR SEM. NEGRA)
```

Figura 17: Consulta realizada al servicio web - Trayectos

- **ParadasTrayectos:** Proporciona información sobre las paradas que forman parte de una trayecto. Estas paradas mantienen un orden que permiten definir el trayecto perfectamente.

idParada : Integer

idLinea : Integer

idTrayecto : Integer

Orden : Integer

Descripción : String

utmX : Integer

utmY : Integer

```
idLinea: 1 :idParada: 180 :idTrayecto: 1 :idOrden: 0 :[utmX, utmY]:  
281411, 4824479 :idDescripcion: CAMINO DE RUBÍN  
  
idLinea: 1 :idParada: 181 :idTrayecto: 1 :idOrden: 1 :[utmX, utmY]:  
281572, 4824481 :idDescripcion: INEM  
  
...  
  
idLinea: 1 :idParada: 34 :idTrayecto: 1 :idOrden: 30 :[utmX, utmY]:  
289352, 4822636 :idDescripcion: HOSPITAL DE CABUENES
```

Figura 18: Consulta realizada al servicio web - ParadasTrayectos

- **LlegadasParadas:** Nos ofrece las distintas llegadas a las paradas que el servicio web estima oportuno en el momento de realizar la consulta.

idAutobus : String
 Matricula : String
 Modelo : String
 idLinea : Integer
 idTrayecto : Integer
 idParada : Integer
 Minutos : Integer
 Distancia : Double
 HoraActualizacion : String
 FechaActualizacion : DateTime

```

idAutobus:280      :      FechaActualización:      2011-09-15T00:00:00      :
HoraActualización : 19:27:11 : idTrayecto : 1 : idParada : 1 : Minutos
: 231 : Distancia : 9721.433345 : idLinea : 34 : Matricula Bus : O-
0.516-CJ : Modelo Bus : N094UB4X2
  
```

Figura 19: Consulta realizada al servicio web - LlegadasParadas

- **Posiciones:** Proporciona la localización, fecha, hora, distancia a la siguiente parada...etc, de n autobuses elegidos de forma aleatoria de entre los que se encuentren operativos en ese momento en la red de autobuses.

idAutobus : String
 Matricula : String
 Modelo : String
 idLinea : Integer
 idTrayecto : Integer
 idParada : Integer
 OrdenParada : Integer
 utmX : Integer
 utmY : Integer
 HoraActualizacion : String
 FechaActualizacion : DateTime
 idSiguienteParada : Integer
 Minutos : Integer
 Distancia : Double

```

FechaActualización : 2011-09-15T00:00:00 : HoraActualización :
19:26:47 : idAutoBus : 301 : Matricula : 3760 CGY : Modelo : NO94UB4X2
: Distancia : 0.760647 : idLinea : 1 : idParada : 34 :
idSiguienteParada : 138 : idTrayecto : 1 : Minutos : 0 : OrdenParada :
30 : [utmX, utmY] : 283137, 4824306

```

Figura 20: Consulta realizada al servicio web - Posiciones

- **estadoBus:** Operación que en función del identificador de autobús pasado por parámetros, nos arrojará la línea y trayecto al cual éste se encuentra adscrito y todas las paradas a las que se aproximará en sucesivos minutos.

idAutobus : String
 utmX : Integer
 utmY : Integer
 HoraActualizacion : String
 FechaActualizacion : DateTime

idParada : Integer
 idLinea : Integer
 idTrayecto : Integer
 Minutos : Integer
 Distancia : Double
 HoraActualizacion : String
 FechaActualizacion : DateTime

```

idAutobus276
Paradas de dicho bus: 16
idParada a la que se dirige : 18 : Minutos restantes hasta llegar : 0 : Distancia : 305.631923
idParada a la que se dirige : 19 : Minutos restantes hasta llegar : 2 : Distancia : 589.316501
idParada a la que se dirige : 20 : Minutos restantes hasta llegar : 3 : Distancia : 775.655832
idParada a la que se dirige : 21 : Minutos restantes hasta llegar : 4 : Distancia : 1045.918081
idParada a la que se dirige : 22 : Minutos restantes hasta llegar : 5 : Distancia : 1330.929039
idParada a la que se dirige : 23 : Minutos restantes hasta llegar : 6 : Distancia : 1646.907161
idParada a la que se dirige : 24 : Minutos restantes hasta llegar : 7 : Distancia : 1855.275864
idParada a la que se dirige : 25 : Minutos restantes hasta llegar : 8 : Distancia : 2155.460607
idParada a la que se dirige : 596 : Minutos restantes hasta llegar : 10 : Distancia : 2723.828021
idParada a la que se dirige : 27 : Minutos restantes hasta llegar : 11 : Distancia : 3447.740452
idParada a la que se dirige : 28 : Minutos restantes hasta llegar : 12 : Distancia : 4036.60623
idParada a la que se dirige : 192 : Minutos restantes hasta llegar : 13 : Distancia : 4449.553343
idParada a la que se dirige : 193 : Minutos restantes hasta llegar : 14 : Distancia : 4762.191873
idParada a la que se dirige : 30 : Minutos restantes hasta llegar : 15 : Distancia : 5398.991415
idParada a la que se dirige : 31 : Minutos restantes hasta llegar : 16 : Distancia : 5727.336413
idParada a la que se dirige : 33 : Minutos restantes hasta llegar : 17 : Distancia : 6119.374916

```

Figura 21: Consulta realizada al servicio web - estadoBus

- **estadoLinea:** Requiere del identificador de una línea para ver los trayectos que tiene dicha línea, además de los autobuses que transitan por dicha línea en el momento de la consulta.

idTrayecto : Integer

idAutobus : String

La siguiente captura de datos está referida a la línea número: 2

```
Requiere el id de una Linea, para ver que trayectos tiene y sus buses asociados
Trayecto : 3
idAutobus: 337
idAutobus: 338

Trayecto : 4
idAutobus: 337
idAutobus: 338
idAutobus: 339
```

Figura 22: Consulta realizada al servicio web - estadoLinea

- **estadoParada:** Operación que requiere del identificador de una parada para devolver los autobuses que se aproximan a ella.

idAutobus : String

idLinea : Integer

idTrayecto : Integer

Minutos : Integer

utmX : Integer

utmY : Integer

HoraActualizacion : String

FechaActualizacion : DateTime

La siguiente captura de datos está referida a la parada número: 708

```
idAutobus que se aproxima! : 296 : Minutos : 17 : [utmX, utmY] : 2871134822544 : Distancia :4480.997068
idAutobus que se aproxima! : 271 : Minutos : 18 : [utmX, utmY] : 2829854824530 : Distancia :4303.022561
```

Figura 23: Consulta realizada al servicio web - estadoParada

Estas 10 operaciones pueden agruparse de 3 formas distintas atendiendo a:

1. **Tipo de datos:** Estáticos o dinámicos. Es decir, la información devuelta por una de estas funciones/operaciones puede ser que no cambie en el tiempo, como la información proporcionada por las 'Líneas', o puede ser cambiante con el transcurrir del tiempo como ocurre con la información devuelta por la operación 'Posiciones'.

Estáticos: Líneas, Paradas, Trayectos, ParadasTrayectos, Horarios.

Dinámicos: LlegadasParadas, Posiciones, EstadoBus, EstadoLinea, EstadoParada.

2. **Redundancia en los datos:** Esto es, repetición de la misma información devuelta por distintas operaciones. Es decir, según nuestras consultas realizadas a todas y cada una de las operaciones, existen operaciones que contienen información que ya se encuentra presente en otras operaciones. Esta redundancia permite reducir el número de operaciones a consultar centrándonos en las operaciones importantes.

De todas las operaciones, aquellas que contienen información presente en otras operaciones son: LlegadasParadas, EstadoLinea y EstadoParada. El resto de operaciones forman parte del grupo a considerar para llevar a cabo nuestro proyecto.

3. **Parametrización de las operaciones:** sin parámetros, o con parámetros. Es decir, existen dos tipos de funciones atendiendo a la forma en la que las consultamos, funciones que necesitan de un parámetro específico al ser consultadas y funciones que no necesitan del paso de ningún parámetro. Las únicas funciones que necesitan de parámetros son: EstadoBus, EstadoLinea y EstadoParada.

Ahora bien, de las 10 operaciones podemos prescindir de 'Horarios'. El motivo es que no vamos a hacer uso de los horarios presentes en la red de autobuses a la hora de la realización de nuestra aplicación web, ya que la información que ofrece no la consideramos relevante para su inclusión en el proyecto. Luego, contamos con 9 operaciones.

De esas 9 operaciones restantes, hemos de descartar por redundancia de datos las siguientes operaciones:

- **LlegadaParada:** El motivo de descarte de esta operación es que la operación 'EstadoBus' ofrece la misma información. Es decir, la existencia de la función 'posiciones' nos proporciona la localización de n autobuses (proporcionados de forma aleatoria), por lo que, si por cada uno de estos autobuses dados por localizaciones, consultamos su estado, valiéndonos de 'EstadoBus', obtendremos la misma información que obtendríamos con 'LlegadaParadas' más las localizaciones exactas.

- **EstadoLinea:** En esta operación, el motivo que nos lleva a desecharla es que la información que proporciona al consultarla es la misma que obtenemos gracias a la combinación de las operaciones '*Posiciones*' y '*EstadoBus*'.
- **EstadoParada:** El motivo de descarte para esta operación es exactamente el mismo que para la anterior operación descartada. En ella se nos proporciona información que ya tenemos gracias a la combinación de las operaciones '*Posiciones*' y '*EstadoBus*'.

Después de esta criba, las operaciones que restan a usar en nuestro proyecto son las siguientes:

- **Líneas**
- **Paradas**
- **Trayectos**
- **ParadasTrayectos**
- **Posiciones**
- **EstadoBus**

Así pues, después de obtener las operaciones precisas a consultar, nos encontramos en disposición de poder almacenar estos datos en una base de datos creada a demanda de las necesidades de nuestro proyecto. Creación que será detalla en sucesivas secciones.

4. Arquitectura del proyecto

En este capítulo trataremos, en líneas generales y sin entrar en muchos detalles, la estructura establecida en nuestro proyecto.

4.1. Descripción general

Nuestro propósito en este proyecto pasa por reutilizar los datos, accesibles de manera pública, ofrecidos por el ayuntamiento de Gijón a través del servicio web *busGijón*. De esta forma se ve extrapolada la filosofía de los Opendata en nuestro proyecto, ya que, el fin último de este uso de datos es la generación de una aplicación con un valor añadido.

Nuestra aplicación web, que aporta un valor añadido a la información extraída del servicio web *busGijón*, está envuelta en un sistema formado por tres componentes:

1. Programa recuperador de datos cuya principal misión es: consumir, depurar y hacer persistir la información leída de *busGijón* en una base de datos. Esta aplicación consumidora será ejecutada en un servidor en el que se ejecutará, de forma residente, hasta la fecha marcada por un parámetro pasado en el momento de su ejecución. Dicho programa es realizado en el lenguaje de programación Java.
2. Presentación de estos datos a través de la creación de una página web. Ésta será accesible desde cualquier dispositivo fijo o móvil. Para la construcción de la web¹ nos valdremos, principalmente, del lenguaje de programación PHP.
3. Creación de un módulo planificador de rutas que nos dará las indicaciones necesarias para ir desde una parada hasta otra. Éste proporcionará las indicaciones necesarias a la aplicación web, que será la que finalmente muestre el resultado solicitado por el usuario. Dicho módulo será el que aporte ese valor añadido y de salto cualitativo a la aplicación web. Además, la web, puede verse como la interfaz receptora de la consulta que da soporte al módulo planificador de rutas. Finalmente hacer notar, que para comunicar este módulo (programado en Java) con la aplicación web (programada en PHP) hemos de hacer uso de un protocolo de red que permite conectar PHP con Java y viceversa, hablamos de PHP/JavaBridge.

¹ <http://sanfernando.cs.us.es/~sebastian>

Podemos añadir que la aplicación web es el componente dependiente del programa recuperador de datos, puesto que necesita de una base de datos consistente de la cual nutrirse. Además es el componente que depende del módulo planificador de rutas para así dar cabida a las consultas realizadas por los usuarios en la web, consultas relativas a planificación de viaje entre dos paradas.

Así pues, nuestro proyecto estará integrado básicamente por esas tres componentes. Éstas han sido alojadas, de forma conjunta, en un servidor¹ público proporcionado por el director de proyecto. Es en este servidor donde, además, se mantiene la base de datos que posibilita la persistencia de los datos consumidos por nuestra aplicación consumidora.

Tras observar las pautas por las que se rigen las componentes integrantes de nuestro proyecto, llega el momento de conocer las fases en las que se ha estructurado su realización, la realización del proyecto. Estas fases son:

- **Análisis y persistencia de datos.**

En este estadio del proyecto nos centramos en el análisis de la información leída del servicio web de forma manual. En estas lecturas realizadas a *busGijón* observamos ciertas peculiaridades en la forma en la que se nos presenta los datos. Puesto que estas ‘anomalías’ son tratadas con más detalle en el capítulo del mismo nombre no me detendré en enumerar ninguna de ellas. Dichas ‘anomalías’ son restricciones que se han de imponer en el momento de realizar la consumición de datos, de forma que se estará realizando una criba o depurado en dicho momento.

Tras esta criba o depurado de datos, éstos, se han de grabar en una base de datos para tenerlos disponibles y es por eso, que fruto de las diversas consultas manuales al servicio web, se consigue definir una estructura consistente y libre de redundancias que permita el almacenamiento de estos datos.

Luego, tanto el depurado de datos, como la estructura donde irán almacenados éstos, serán usados por el programa recuperador de datos. Éste será el que finalmente haga uso tanto de las restricciones impuestas para la depuración de la información, como de la estructura de datos donde almacenar esta información depurada.

- **Recuperación de datos.**

En esta fase se ha de alojar el programa recuperador de datos en el servidor de datos proporcionado por nuestro tutor, donde será puesto en marcha y ejecutado de forma ininterrumpida hasta la fecha marcada en el momento de su puesta en marcha. Durante la ejecución ira consumiendo información del servicio web,

¹ <http://sanfernando.cs.us.es/>

depurándola y almacenándola en la estructura de datos ya creada. Además, nuestro recuperador de datos, registrará durante el transcurso de su funcionamiento, cualquier tipo de error o excepción que ocurra, ya sea por error de nuestro servidor¹ o error del servidor que mantiene el servicio web *busGijón*.

Hacer notar que el programa consumidor ha estado prácticamente 2 meses recopilando datos del servicio web, y aún continua.

- **Aplicación web.**

En este estadio del proyecto nos encargamos de presentar la información que el programa consumidor de datos ha recopilado o está recopilando. Esta web está desarrollada en php con los elementos javascript necesarios para poder usar Google Maps, además del uso de jQuery para tablas y autocompletados en formularios. También está accesible vía móvil gracias framework de desarrollo para aplicaciones móviles basado en jQuery, jQTouch. Remarcar la dirección de la aplicación web del proyecto: <http://sanfernando.cs.us.es/~sebastian>

En la web, la información se encuentra representada atendiendo a:

- *Configuración:* Información acerca de los elementos que forman parte de la red de autobuses (líneas, paradas, trayectos, tramos y autobuses). Información representada en formato tabla.
- *Estado de los autobuses:* Información referente a las últimas posiciones de los autobuses pertenecientes a la red e información relativa a las distintas aproximaciones de un autobús a una parada, indicándose distancia de llegada, tiempo...etc.
- *Representación de líneas y sus trayectos:* Representación en formato gráfico sobre el mapa de Google Maps de las distintas líneas que componen el servicio de autobuses, así como de los trayectos de cada una de ellas.
- *Localizaciones de los autobuses:* Representación en formato gráfico sobre el mapa de Google Maps de las últimas localizaciones de un autobús seleccionado, cuya resolución será seleccionada a demanda.
- *Planificador de rutas:* Herramienta de la web donde el usuario realizará la consulta de la ruta existente entre las dos paradas insertadas en sus respectivos campos. Estos campos ofrecerán sugerencias de las descripciones de las paradas junto a su identificador, todo ello valiéndose de la herramienta jQuery. Esta opción de la aplicación web es la

¹ <http://sanfernando.cs.us.es/>

“representación gráfica” de la lógica que esconde el módulo *planificador de rutas*.

- **Planificador de rutas.**

Esta fase es la última y una de las más importantes. En ella se implementa el modulo que permite la búsqueda de rutas entre las dos paradas proporcionadas por el formulario de la aplicación web (opción *planificador de rutas*). Este modulo implementado en java, requiere de la implementación de una búsqueda que proporcione el camino más eficiente/corto entre esas dos paradas. A su vez, la búsqueda ha de explorar sobre un conjunto de datos que represente la red de autobuses.

El sistema que mejor representa nuestra red de autobuses es un grafo, en el que las paradas serán vértices y cada par de ellas representará una arista. Luego, será sobre este grafo sobre el que realizaremos las exploraciones propias de nuestra búsqueda. Pero... ¿Qué búsqueda emplear?

Tenemos un conjunto de tres algoritmos de búsquedas: algoritmo A*, búsqueda en profundidad y algoritmo de Floyd. De este conjunto hemos de seleccionar la búsqueda más eficiente que permita encontrar sobre el grafo el mejor camino entre dos paradas. Esta selección o medida de rendimiento será en función de la memoria consumida durante la ejecución del algoritmo y el número de paradas que arroje su búsqueda. Además mediremos nuestro rendimiento en función de dos tipos de grafos: un primer grafo que únicamente contemple una línea, con todos sus trayectos, y un segundo grafo que contemple todas las líneas de la red de autobuses. Dejando aparte la búsqueda de rendimiento, el grafo que usaremos para la búsqueda de nuestras rutas será el grafo que contemple todas las líneas (junto a todos sus trayectos).

El algoritmo más eficiente y con el que realizaremos las búsquedas de rutas será el algoritmo A*. Dicho algoritmo, al igual que el resto, nos proporciona una lista de paradas, una tras otra, con su descripción y en formato texto. Esta lista de descripciones de paradas no es ningún tipo de indicación entendible por un usuario, es decir, se necesitan indicaciones que nos permitan transcribir la lista de paradas en las líneas a las que pertenece cada una de estas paradas, así como los transbordos a realizar entre líneas. La metodología de estas transcripciones son detalladas en el capítulo 8 (ver 8.4 Ruta entre dos paradas).

Como ya se comento, este modulo (completamente en Java), ha de interactuar con PHP, más concretamente con el fichero '*GoogleMaps_planificador.php*'. Esta interacción implica el paso de las indicaciones de ruta en formato texto al fichero PHP que será el que finalmente lo muestre y quede visualizado en la web. Para ello nos valemos de un protocolo de red que permite conectar en ambas direcciones tanto PHP como Java.

Una indicación de ruta entre dos paradas tales como *Camino de Rubín* y *Pedro Duro* será la siguiente:

Línea 90 | 9 : **CAMINO DE RUBÍN**, CERILLERO, PARQUE J. BESTEIRO, ALGODONERA, ECUADOR, CUATRO CAMINOS, SANTA OLAYA. Km's: 11

[TRANSBORDO]

Línea 6 | 1 : SANTA OLAYA, PLAZA DE LA LUZ, REVILLAGIGEDO, ESTACIÓN DEL NORTE, MUSEO DEL FERROCARRIL, **PEDRO DURO**. Km's: 12

Esto nos está indicando que para ir desde la parada *Camino de Rubín* hasta *Pedro Duro*, hemos de montarnos en la línea número 90, trayecto 9 que se compone de las paradas que se indican tras los dos puntos. Realizamos transbordo en la última parada: *Santa Olaya* para continuar en la línea número 6, trayecto 1 para finalizar nuestro viaje en la parada *Pedro Duro*.

Todas las fases de las que se compone nuestro proyecto se encuentran detalladas los capítulos homónimos.

En este proyecto, desde un primer momento se exigió que el desarrollo fuese similar al de un proyecto más o menos profesional en tanto que debía existir una comunicación entre alumno y coordinador más o menos periódica. Para este fin, se creó un blog¹ alojado en el portal Wordpress tanto para informar de las nuevas ideas surgidas referentes al proceso de desarrollo, como de los logros conseguidos, problemas encontrados, etc. A fecha de entrega de este proyecto el contenido del blog se considera finalizado y puede ser leído libremente por todo aquel que lo considere de interés.

¹ <https://datosenabierto.wordpress.com/>

4.2. Tecnologías y herramientas

Para poder llevar a cabo el desarrollo del proyecto fue necesario establecer un conjunto de aplicaciones/herramientas mínimo:

- **php:** Es un lenguaje de programación interpretado del lado del servidor que está enfocado para el desarrollo de aplicaciones webs dinámicas. Está disponible para la mayoría de sistemas operativos y arquitecturas.
- **Java:** Lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores.
- **Mysql:** Es el gestor de base de datos de libre distribución más extendido. Utiliza el modelo relacional y se integra perfectamente con php y otros muchos lenguajes.
- **Javascript:** Es un lenguaje interpretado del lado del cliente. Su objetivo es complementar a los lenguajes del lado del servidor, generalmente para mejorar la interfaz de usuario y para facilitar la comunicación asíncrona. Todos los navegadores modernos incluyen un intérprete de javascript.
- **JQuery:** JQuery es una biblioteca de Javascript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. Ofrece una serie de funcionalidades basadas en Javascript que de otra manera requerirían de mucho más código.
- **jQTouch:** Es una librería para el desarrollo de aplicaciones web móviles que imita el estilo y el comportamiento de las aplicaciones nativas del iPhone OS. En realidad se basa en características de HTML5 y CSS3 soportadas por el motor renderizado Webkit, presente en Safari Mobile, por lo que se visualizará correctamente en cualquier navegador que funcione con este motor.
- **API de Google Maps:** Conjunto de funciones y procedimientos que nos ofrece Google para hacer uso de prácticamente todos los servicios que ofrece la compañía a día de hoy. En concreto, en este proyecto, hemos usado la API de Google Maps en su versión 3.0 para generar mapas dinámicos con una enorme cantidad de funciones.
- **Apache:** Servidor web de gran importancia y difusión debido a la multitud de extensiones y proyectos relacionados que existen alrededor de él. Presenta

características altamente configurables, bases de datos de autenticación y negociado de contenido.

- **NetBeans:** Entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación java. Existe además un número importante de módulos para extenderlo.
- **Php/JavaBridge:** El puente PHP/Java es un protocolo de red que permite conectar en ambas direcciones scripts en PHP con clases Java. De esta manera, desde cualquier fichero php tendremos acceso a cualquier clase Java, cuyo .class se encuentre incluido en la carpeta '*JavaBridge/Web-inf/classes*'. Esta carpeta JavaBridge posibilita el funcionamiento del protocolo de red.

5. Análisis y persistencia de datos

Conocida la estructura del servicio web a utilizar en nuestra aplicación, hemos de analizar sus datos, con el consiguiente depurado, para finalmente crear una estructura acorde a los datos obtenidos y así hacerlos persistir en nuestra base de datos.

Del depurado y posterior creación de la estructura de la base de datos se encargará nuestra aplicación consumidora de datos, en sucesivas secciones.

5.1. Análisis de los datos

Antes de pasar a estudiar el número de tablas, las relaciones existentes entre ellas y todo lo que se desprende del servicio web que hemos de hacer persistir en forma de base de datos, hemos de observar que tipo de ‘tamices’ o modificaciones hemos de imprimirles a los datos que obtenemos del servicio web para crear posteriormente nuestra base de datos. De forma que habremos depurado los datos ofrecidos por el servicio web *busGijón*¹.

La necesidad de un pre-tratamiento de datos, antes de la creación de la estructura que los albergue, es de especial importancia para un correcto y simplificado acceso posterior de nuestra aplicación a éstos. El origen de esta necesidad de depurado surgió durante el estudio y observación de las distintas operaciones a consulta que posee el servicio web *busGijón*. Origen caracterizado en las siguientes particularidades:

- **Formato UTM para las localizaciones geográficas.**
- **Latitud con valores negativos.**
- **Campo: idSiguienteParada. Valores negativos.**
- **Campo: Direccion. Descripción numérica 1 y 2**
- **Cabecera de la descripción de los trayectos a eliminar.**

Esta serie de particularidades son problemas menores, de sencilla solución, a excepción del más laborioso que fue: obtener las localizaciones geográficas en formato UTM.

La representación de las localizaciones geográficas, paradas, autobuses...etc , aplicables a nuestra aplicación web se representan gracias a la API de Google Maps, la cual, necesita de coordenadas geográficas. Y aquí es donde se radica uno de nuestros problemas, que Google Maps no utiliza el formato UTM, sino que usa coordenadas geográficas.

¹ <http://docs.gijon.es/sw/busgijon.asmx>

Es decir, las distintas localizaciones geográficas proporcionadas por las operaciones del servicio web *BusGijón* están en un formato no entendible por la API de Google Maps. Es decir, vienen en formato **UTM** (*Universal Transverse Mercator*), cuando nosotros lo que necesitamos son **Coordenadas geográficas** (Latitud, Longitud).

Éstas coordenadas **UTM** se localizan en los campos '*utmX : integer*' y '*utmY : integer*', presentes en casi todas las operaciones/funciones que contiene el servicio web *BusGijón* que aquí estamos tratando.

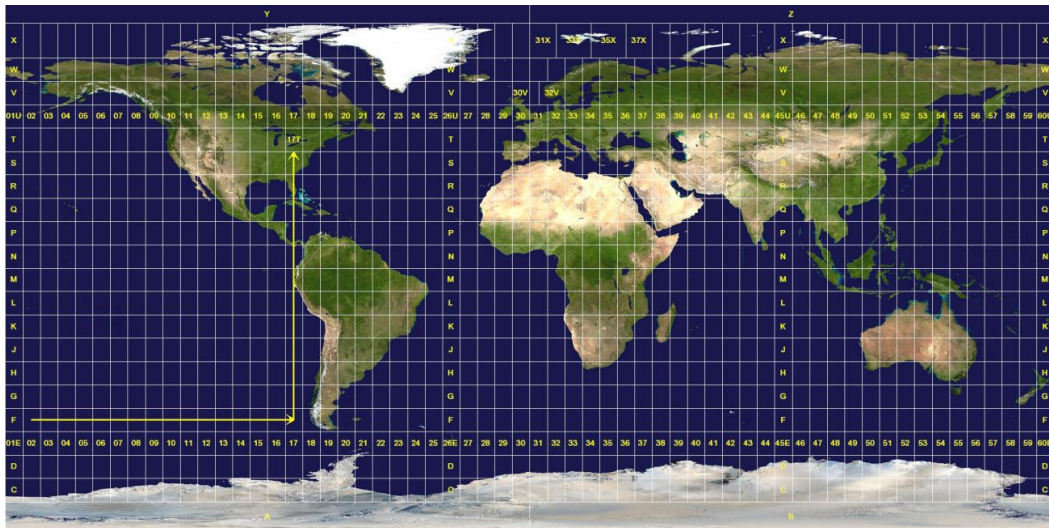


Figura 24: Mapa mundial de las zonas UTM

En este formato, las magnitudes se expresan en metros al nivel del mar. Para especificar un punto del mapa en **UTM** hay que dar, además de la X e Y, el huso y la zona o hemisferio correspondiente según el mapa de la imagen. **A Gijón, de donde proceden nuestros datos, le corresponde el huso 30 y el hemisferio Norte.**

El problema que se plantea con este tipo de magnitud es que es incompatible con la medida cartográfica (Latitud, longitud) que soporta Google Maps. Por tanto, hemos de encontrar la forma de convertir los valores **UTM** a Coordenadas geográficas (Lat, Lng).

5.1.1. Soluciones propuestas

A continuación, daremos solución a todas y cada una de las particularidades observadas en el estudio del servicio web a tratar en este proyecto. Estas soluciones irán encabezadas por la descripción del problema del que se parte:

- **Formato UTM para las localizaciones geográficas:**

Para dar solución al problema de conversión antes propuesto, hemos utilizado en nuestro proyecto una librería Java, Jcoord¹, que entre otras funciones nos permite convertir de *UTM* a *Coordenadas Geográficas*. Aunque la transformación realizada viene con algo de desfase respecto de las coordenadas geográficas que Google Maps nos proporciona.

Antes de continuar con la explicación, nos gustaría mostrar las llamadas a realizar (a modo ejemplificador) para lograr la conversión aquí expuesta. Llamadas realizadas en el lenguaje de programación Java, haciendo uso de la citada librería:

```
//Importamos la librería Jcoord
import uk.me.jstoot.jcoord.UTMRef;

//Definimos nuestra variable a usar
private UTMRef conversor;

/*Realizamos la siguiente inicialización*/
conversor = new UTMRef();

/*Pasamos por parámetros:
- Coordenadas UTM leídas del servicio web
- Hemisferio en el cual se encuentra Gijón. Norte 'N'
- Huso horario que le corresponde a Gijón. 30*/
conversor.setValores(aux.getUtmx(), aux.getUtmy(), 'N', 30);

/*Ahora tenemos lista nuestra variable 'conversor' para
la obtención de la Latitud y Longitud*/
private double Lat = conversor.toLatLng().getLat();
private double Lng = conversor.toLatLng().getLng();
```

Figura 25: Uso de la librería Jcoord

Es decir, cuando conseguíamos las coordenadas geográficas de una parada de autobús concreta (por supuesto, haciendo uso de la librería antes citada), ésta se encontraba con un ligero desplazamiento/desfase respecto de la posición de la parada que Google Maps nos proporcionaba. Nos guiamos por Google Maps ya que será el sistema que utilicemos para representar las distintas localizaciones de autobuses, paradas...etc.

¹ <http://www.jstott.me.uk/jcoord/>

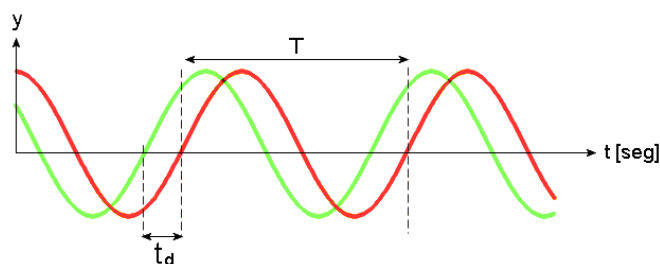


Figura 26: Representación del concepto: desfase

La solución al desfase que mostraban nuestras conversiones, esto es, pasar de *UTM* al sistema de coordenadas geográficas (Latitud, longitud), fue tomar varias localizaciones (en este caso, paradas de autobuses) en formato geográfico ya transformado (Lat, Lng) y compararlas con las coordenadas geográficas que nos proporciona Google Maps.

Cotejando todas esas posiciones, (desplazadas vs óptimas), obtuve que como máximo, la corrección a aplicar a las coordenadas geográficas resultantes de la transformación aplicada por la librería Jcoord es de:

- **0.002** para la Latitud
- **0.00125** para la Longitud

A los valores que la librería nos proporciona, hemos de restarle este desfase. Así pues, con esta conversión básica ya realizada, disponemos de los datos de forma limpia para proceder a su estudio y almacenamiento en la base de datos.

▪ **Latitud con valores negativos:**

La siguiente característica a tratar se trata de un efecto derivado de la conversión a coordenadas geográficas. Es decir, observamos, que en ocasiones obteníamos Latitudes negativas (algo que no puede darse). Así pues, en el programa *consumidor de datos* (que pasará a detallarse en sucesivos apartados) antes de realizar las inserciones de las localizaciones geográficas en la base de datos, hemos de comprobar que éstas sean mayor que 0. De esta forma tenemos cubierta la siguiente peculiaridad que se nos presentaba en la lectura de datos:

▪ **Campo: idSiguienteParada. Valores negativos:**

Otra de las peculiaridades a tratar derivaba de una de las operaciones/funciones presentes en el servicio web *BusGijón*. Esta operación es '*Posiciones*', la cual, de entre todos los campos que se nos ofrece al consultarla, encontramos '*idSiguienteParada : integer*'. Este campo nos informa de la siguiente parada a la cual se dirige un determinado autobús que también viene indicado en su respectivo campo.

Pues bien, la problemática que se nos planteaba con este campo, era que en determinadas ocasiones mostraba valores negativos. Por lo que, en el programa **consumidor de datos** que será el que trate todas y cada una de las inserciones en nuestra base de datos, hemos de comprobar que este campo sea distinto de cero (al igual que hacíamos con Latitud, comentado más arriba).

- **Campo: Direccion. Descripción numérica empleada 1 y 2:**

La descripción numérica mostrada por el campo '*Direccion*', presente en la operación '*Trayectos*', nos indica el sentido en que se realiza dicho el Trayecto consultado. Nos muestra como valores o un 1, o un 2.

```
idTrayecto: 1 idLinea: 1 Direccion: 1 Destino: HOSP. CABUENES
Descripcion: L-1. CERILLERO - HOSPITAL DE CABUEÑES

idTrayecto: 2 idLinea: 1 Direccion: 2 Destino: CERILLERU
Descripcion: L-1. HOSPITAL DE CABUEÑES - CERILLERO
```

Figura 27: Depuración de datos - Dirección

Así pues, la solución planteada a este problema es bien sencilla. En nuestra **aplicación cliente del servicio web** únicamente habremos de cambiar el dígito 1 por la cadena '*ida*' y el dígito 2 por la cadena '*vuelta*'.

```
idTrayecto: 1 idLinea: 1 Direccion: ida Destino: HOSP. CABUENES
Descripcion: L-1. CERILLERO - HOSPITAL DE CABUEÑES

idTrayecto: 2 idLinea: 1 Direccion: vuelta Destino: CERILLERU
Descripcion: L-1. HOSPITAL DE CABUEÑES - CERILLERO
```

Figura 28: Resultado de depuración de datos - Dirección

- **Cabecera de la descripción de los trayectos a eliminar:**

La última de las particularidades a tratar, surge a raíz de la consulta a la operación '*Trayectos*'. Es decir, cuando visionamos el campo '*Descripcion*' de cualquier Trayecto obtenemos algo tal que:

L-1. HOSPITAL DE CABUEÑES - CERILLERO

Es decir la abreviatura identificativa de la línea va solapada al comienzo de la descripción de todos y cada uno de los trayectos. Así pues, una vez identificado nuestro problema, la solución, una vez más, vuelve a estar en el tratamiento dado en la **aplicación cliente del servicio web**. Ya que cuando en la aplicación consumidora, se lea cualquier trayecto para almacenarlo en la base de datos, ejecutamos la eliminación de dicha cabecera, quedándonos todas las descripciones tal que:

HOSPITAL DE CABUEÑES – CERILLERO

5.2. Base de datos

Tras realizar un depurado de los datos procedentes del servicio web *busGijón*¹ nos encontramos en disposición de crear la estructura de datos que albergará estos datos gracias al uso de nuestra aplicación consumidora de datos, aplicación detallada en el capítulo 5. A continuación se detallaran las tablas surgidas del servicio web y los distintos diagramas que de ellas se desprenden.

5.2.1. Tablas

Para representar con total precisión la información extraída del servicio web, hemos de conocer qué tipo de tablas se necesitan. Algunas tablas no guardan la misma estructura que presentan las operaciones al ser leídas de *bugGijón*, es decir, éstas habrán sido ‘divididas’ para mantener la simplicidad. A continuación pasamos a enumerar las tablas finales que quedarán almacenadas:

Tabla #1: `Lineas`. Tabla extraída de la operación con el mismo nombre.

Tabla #2: `tipoLineas`. Tabla que contendrá los distintos tipos de líneas.

Tabla #3: `Paradas`. Tabla extraída de la operación con el mismo nombre.

Tabla #4: `Trayectos`. Tabla extraída de la operación con el mismo nombre.

Tabla #5: `Extremos`. Tabla que contendrá origen y destino de cada trayecto.

Tabla #6: `Tramos`. Tabla extraída de la operación ‘*ParadasTrayectos*’.

Tabla #7: `Autobus`. Tabla extraída de la operación ‘*Posiciones*’.

Tabla #8: `Localizacion`. Tabla extraída de la operación ‘*Posiciones*’.

Tabla #9: `PosicionViva`. Tabla con los mismos campos que ‘*Localizacion*’. Únicamente almacenará las últimas posiciones.

Tabla #10: `Carrera`. Tabla extraída de la combinación de las operaciones ‘*Posiciones*’ y ‘*EstadoBus*’, de forma que van quedando patente los distintos recorridos de cada autobús.

Tabla #11: `Excepciones`. Tabla que pretende almacenar cualquier tipo de excepción que ocurra durante la monitorización o recopilación de información del servicio web.

¹ <http://docs.gijon.es/sw/busgijon.asmx>

A continuación se presentan los distintos diagramas que modelan la estructura de nuestra base de datos. Diagramas entidad-relación, relacional y diagrama de integridad referencial.

5.2.2. Modelo entidad - relación

Un diagrama o modelo entidad-relación es una herramienta para el modelado de datos de un sistema de información. Estos modelos expresan entidades relevantes para un sistema de información, así como sus interrelaciones y propiedades.

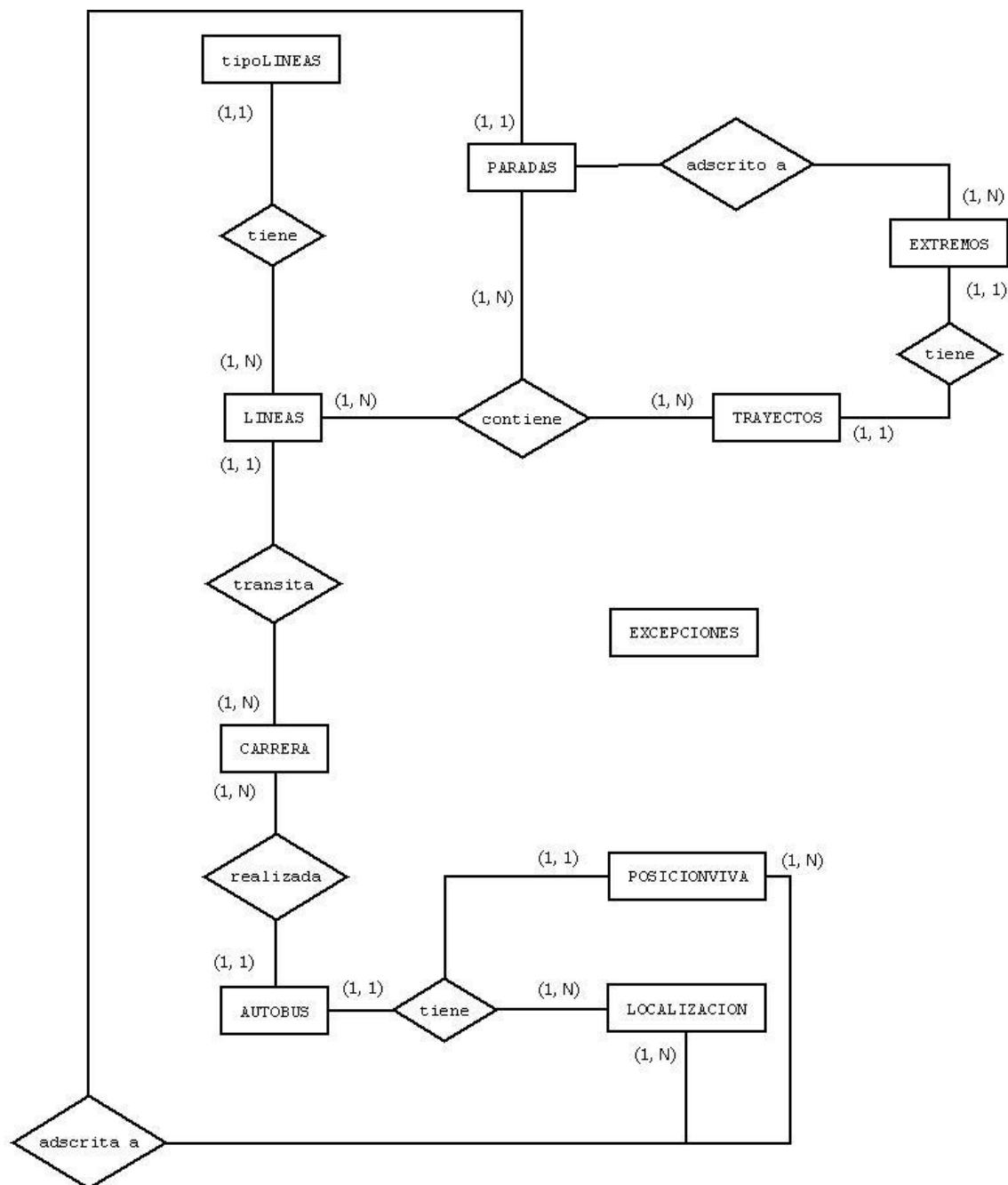


Figura 29: Diagrama entidad-relacion

A primera vista se observan 10 tablas. Sin embargo existe una relación ternaria de ‘muchos a muchos’ entre las tablas ‘Lineas’, ‘paradas’ y ‘Trayectos’.

Este tipo de relación se transforma en una tabla cuya clave primaria es la concatenación de las claves primarias de las tablas surgidas al transformar las entidades que forman parte de la relación.

Cada uno de los atributos que forman la clave primaria de esta tabla son clave ajena respecto a cada una de las tablas donde dicho atributo es clave primaria.

5.2.3. Diagrama relacional

El modelo entidad-relación se considera un modelo conceptual, ya que permite a un alto nivel ver con claridad la información utilizada en el problema que nos aborda.

El modelado relacional se encarga de desarrollar un buen modelo ‘lógico’, también conocido como ‘esquema de la base de datos’, a partir del cual se podrá realizar el modelado físico en el sistema gestor de la base de datos.

A continuación se muestra el diagrama relacional de las distintas tablas citadas anteriormente y que contiene los distintos atributos relativos a cada una de ellas:

LINEAS (idLinea, idTipoLinea, Descripcion) CP: idLinea VNN: idTipoLinea, Descripcion CAj: idTipoLinea -> tipoLINEAS(idTipoLinea)	
PARADAS (idParada, Descripcion, Lat, Lng) CP: idParada VNN: Descripcion, Lat, Lng	tipoLINEAS (idTipoLinea, Tipo) CP: idTipoLinea VNN: Tipo
TRAYECTOS (idTrayecto, idLinea, Descripcion, Sentido, idExtremos) CP: (idTrayecto, idLinea) VNN: Descripcion, Sentido, Extremos	AUTOBUS (idAutobus, Matricula, Modelo) CP: idAutobus VNN: Matricula, Modelo
EXTREMOS (idExtremos, idOrigenParada, idDestinoParada) CP: idExtremos CAj: idOrigenParada -> PARADAS (idParada) idDestinoParada -> PARADAS (idParada) VNN: idOrigenParada , idDestinoParada	EXCEPCIONES (idExcepcion, Fecha , Hora, Tipo, Descripcion) CP: idExcepcion VNN: Fecha, Hora, Tipo, Descripcion

TRAMOS (idParada, idLinea, idTrayecto, Descripcion, Orden)

CP: (idParada, idLinea, idTrayecto)

CAj: idParada -> PARADA(idParada)

idLinea -> LINEAS(idLinea)

idTrayecto -> TRAYECTOS(idTrayecto, idLinea)

LOCALIZACION (idAutobus, FechaActualizacion,

HoraActualizacion, Lat, Lng, idSiguienteParada)

CP: (idAutobus, FechaActualizacion, HoraActualizacion)

CAj: idAutobus -> AUTOBUS(idAutobus)

idSiguienteParada -> PARADA(idParada)

VNN: Lat, Lng, idSiguienteParada

CARRERA (idLoc, idAutobus, FechaActualizacion, HoraActualizacion, idParada, idLinea, idTrayecto, Minutos, Distancia)

CP: idLoc

CAj: idAutobus -> AUTOBUS(idAutobus)

idLinea -> LINEAS(idLinea)

VNN: Descripcion, Lat, Lng

POSICIONVIVA (idAutobus, idLinea, idTrayecto, FechaActualizacion, HoraActualizacion, Lat, Lng, idSiguienteParada)

CP: idAutobus

CAj: idAutobus -> AUTOBUS(idAutobus)

idSiguienteParada -> PARADA(idParada)

(idTrayecto, idLinea) -> TRAYECTOS(idTrayecto, idLinea)

VNN: Lat, Lng, idSiguienteParada

5.2.4. Diagrama integridad referencial

La integridad referencial es un sistema de reglas que utilizan la mayoría de las bases de datos relacionales para asegurarse que los registros de tablas relacionadas son válidos y que no se borren o cambien datos relacionados de forma accidental produciendo errores de integridad.

Es decir, los datos que referencian otros (claves foráneas) deben ser correctos. La integridad referencial hace que el sistema gestor de la base de datos se asegure de que no haya en las claves foráneas valores que no estén en la tabla principal.

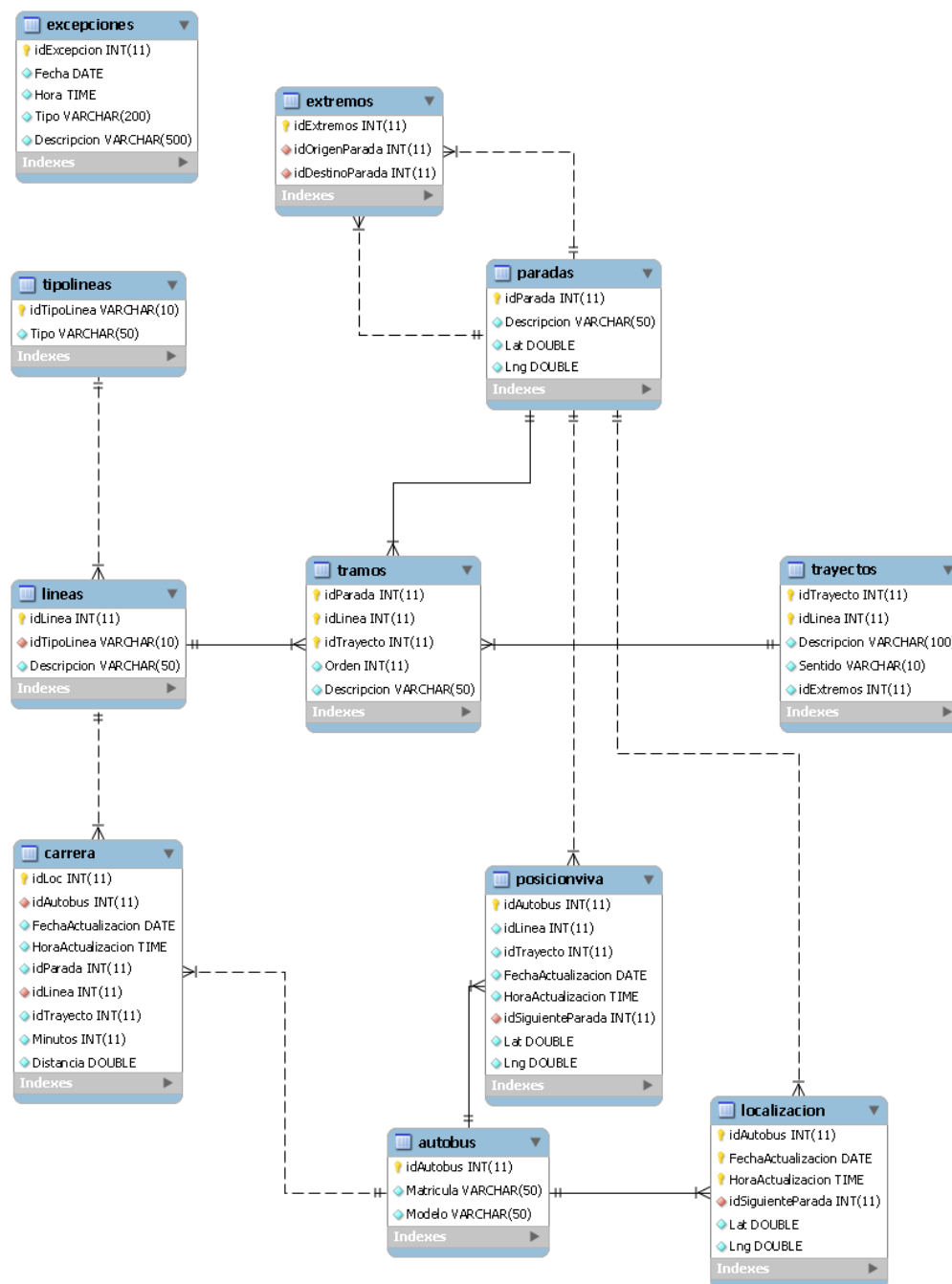


Figura 30: Diagrama de integridad referencial

5.2.5. Aspectos generales

Una vez hemos definido las distintas estructuras lógicas sobre las que se asentará nuestro sistema de información, resta crear un script SQL¹ que se encargue de la definición de la estructura de los objetos/tablas que integrarán nuestra base de datos.

¹ Lenguaje de consulta estructurado. Es un lenguaje declarativo de acceso a base de datos relacionales que permite especificar diversos tipos de operaciones en éstas, entre ellas, la creación e inserción de tablas que aquí tratamos.

6. Recuperación de datos

La recuperación de datos aúna la depuración o análisis de los datos procedentes del servicio web *busGijón*¹ más la persistencia² de esos datos ya depurados. Es decir, esas dos acciones se encuentran integradas en una aplicación cliente creada en lenguaje de programación Java. Así pues, en líneas generales, el programa cliente será un consumidor de los datos procedentes del servicio web.

Ésta, es una parte básica de este proyecto ya que alimenta a la aplicación web que da soporte a la presentación de estos datos además de nutrir la funcionalidad planificadora de rutas de autobuses.

A continuación damos paso a una descripción un poco más detallada acerca de esta aplicación consumidora de datos además de ver la estructura lógica que la implementa.

6.1. Descripción

La idea básica consiste en la creación de un programa, escrito en Java, que implemente la depuración y persistencia de datos vistos en el capítulo anterior. Esta consumición de datos será realizada de forma programada.

El flujo a seguir por este programa, o aplicación cliente del servicio web, constará de las siguientes acciones una vez arrancado y programada la fecha de finalización. Fecha en la que finalizará la consumición:

1. Creación de la estructura de datos, en caso de que no esté creada, a partir de un script en lenguaje *SQL* denominado '*fichero.sql*'.
2. Consumo de los datos del servicio web *busGijón*, depurado de dichos datos en base a las peculiaridades vistas en el capítulo anterior y finalmente, almacenamiento de éstos en la estructura creada en el punto #1. Distinguimos dos grupos de datos a consumir:
 - Estáticos: Estos datos únicamente se consumirán y almacenarán una vez por arranque de la aplicación. Es decir, si en posteriores puestas en funcionamiento de la aplicación, estos datos ya hubiese sido consumidos y almacenados, no se realizaría acción alguna.

Estos son los datos a insertar en las tablas:

Lineas, '*Paradas*', '*Trayectos*', '*Tramos*', '*tipoLineas*' y '*Extremos*'.

¹ <http://docs.gijon.es/sw/busgijon.asmx>

² Propiedad de los datos para que estos perduren de alguna forma.

- Dinámicos: Estos datos se consumirán y almacenarán, a intervalos de 3 minutos, de forma ininterrumpida hasta que:
 - Un despertador que extiende de la clase *Timertask*, irrumpa en el bucle en el que se encuentra y finalice la aplicación consumidora de datos.
 - Cualquier tipo de excepción, ya sea, un error de la base de datos, error HTTP,...etc., irrumpa en la ejecución normal del programa y este programa se quedará en un estado latente, durante 2 minutos, a la espera de que el error haya sido subsanado y vuelta al punto #2.

El nombre que recibe esta aplicación en nuestro proyecto es '*ClienteWebServices_Gijon*'. En el servidor¹ web, cedido de forma temporal por el tutor del proyecto, es donde se ha venido realizando la ejecución del .Jar de esta aplicación, además de ser donde se encuentra alojada la base de datos donde grabar nuestros datos. El programa se encuentra alojado en la carpeta '*bin*' de la carpeta personal que me asignó en su día el tutor. Su forma de ejecución, a través de un cliente *SSH* es la que sigue a continuación:

```
ssh      sebastian@sanfernando.cs.us.es      "cd      bin/dist;      nohup      java      -jar
ClienteWebServices_Gijon.jar 10 > salida.txt 2>salida-error.txt &"
```

Hemos de pasarle el número de días que queremos tener corriendo la aplicación en el servidor. En este caso puede apreciarse que son 10 días. Las días que se le pueden pasar por argumentos son: 1,2,3,4,5,10,15,20 días.

El motivo de alojar la aplicación cliente en un servidor web ha sido la disponibilidad de tenerlo las 24 horas del día, 7 días a la semana en funcionamiento.

Nuestra aplicación cliente, se comunicará con el servicio web (*BusGijón*) a través de su interfaz. La interfaz de un web service se describe en un documento XML utilizando el estándar WSDL. En este fichero, que recibe el mismo nombre que el servicio web seguido de '?wsdl', se definen los detalles de cómo usar el web service, las operaciones, el formato de los mensajes, el tipo y el protocolo de transporte del web service. Así pues, la interfaz de *BusGijón* será:

<http://docs.gijon.es/sw/busgijon.asmx?wsdl>

¹ <http://sanfernando.cs.us.es/>

6.2. Arquitectura lógica

Esta aplicación forma una de las partes más importantes de nuestro proyecto. Es decir, nos encontramos ante el motor que hace posible la inserción y consumición de datos del servicio web '*BusGijón*', que junto con la estructura que conforma la base de datos, forman el núcleo del proyecto. Así pues, a continuación pasaremos a observar un diagrama de clases simplificados de nuestro web service client, el cual muestra las relaciones existentes entre ellas.

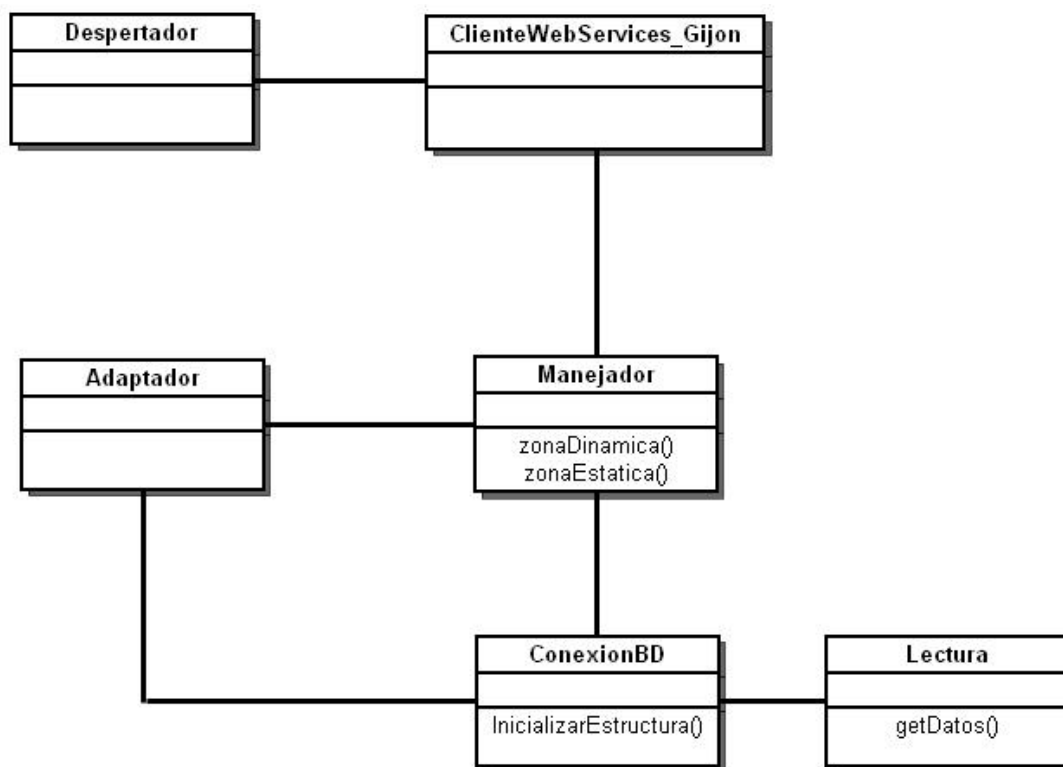


Figura 31: Consumidor de datos - Relaciones entre sus principales clases

Lectura

Clase responsable de la lectura del script '*sql*' que contiene la sintaxis que define la creación de las tablas que hacen posible la persistencia en nuestra base de datos.

ConexionBD

Clase encargada de la definición de los datos básicos de conexión a nuestra base de datos, así como de la creación de las tablas que integran dicha base de datos. Ésta clase define además, los métodos encargados de realizar las consultas y actualizaciones, así como el cierre de la conexión. Obviamente es una clase dependiente de la clase *'Lectura'*, puesto que le proporciona las sentencias pertinentes para la creación de las tablas en la base de datos.

Manejador

Clase encargada de realizar las solicitudes de consulta o inserción, la cual tiene dos métodos principales que vienen a ser llamados desde la clase principal. Estos métodos diferencian entre las tablas que no cambian su información con el transcurrir del tiempo, tablas con información estática, y tablas que van siendo actualizadas con el paso del tiempo, puesto que la información es dinámica, tablas con información dinámica.

Así pues, los métodos son los siguientes:

- **Void zonaDinamica()**

Este método es el encargado de insertar y/o actualizar la información procedente del servicio web en las tablas:

'Carrera', *'Localizacion'*, *'PosicionViva'* y *'Autobus'*.

- **Void zonaEstatica()**

En este método, se procede a insertar y/o actualizar la información procedente del servicio web en las siguientes tablas:

'Lineas', *'Paradas'*, *'Trayectos'*, *'Tramos'*, *'tipoLineas'* y *'Extremos'*.

Ahora bien, he de hacer notar que este último método será llamado una única vez desde la clase principal. Esto es así, puesto que las tablas tienen información que no cambia nunca y con un único volcado de datos sobre ellas es suficiente. Suponiendo que las tablas no hayan sido cargadas ya, de lo contrario, cualquier inserción será ignorada.

Aclarar que cuando nos referimos a datos estáticos, estamos hablando de paradas de autobús (elementos que nunca cambian en el tiempo y por lo tanto fijos), líneas que contienen la red de autobuses, trayectos de éstas, tramos de cada trayecto, tipos de líneas y extremos de trayectos. Conceptos ya tratados en el capítulo 3.

Por el contrario, el método que será llamado cada x minutos será *'zonaDinamica'*. Esto es así por motivos obvios. Cada vez que es llamada la función, se insertan nuevos valores generados por el servicio web en sus tablas correspondientes. Valores como las distintas localizaciones que puede tener un determinado autobús a lo largo de un día entero... etc.

Aunque, la tabla *'Autobuses'*, al tener como clave primaria el identificador del autobús únicamente recibirá nuevos autobuses. Es decir, su función es la de registrar los autobuses que componen la red de autobuses de Gijón. De modo que cuando se va a proceder a la inserción de un autobús ya existente, como es lógico la base de datos provoca una excepción, pero mediante la directiva *'INSERT IGNORE INTO '*, nuestra inserción fallida no provocará ningún tipo de excepción.

Adaptador

Clase destinada a proceder a la colocación de extremos en la tabla *'Extremos'*. Es decir, esta clase colocará los identificadores de las paradas correspondientes a *Origen* y *Destino* de la tabla *'Trayectos'*.

Despertador

Clase que extiende de *'TimerTask'* cuya misión es sacar del bucle principal, a la fecha programada, la consumición de datos del servicio web.

ClienteWebServices_Gijon

Clase principal responsable de fijar la fecha en la que la aplicación cliente dejará de realizar consultas al servicio web, abandonando la aplicación. En esta clase, además, se define el intervalo de tiempo empleado en acceder al servicio web. Y finalmente, es en esta clase desde donde hago uso de una de las clases base *'Manejador'* encargada de la captación de información por parte del web services.

7. Aplicación web

Llegados a este punto, tenemos una arquitectura de datos solida y consistente de la que hacer uso en nuestra aplicación web. Aplicación web, que como de su propio nombre se deduce, está disponible en formato web, además de formato móvil desde cualquier tipo de dispositivo. El principal objetivo de esta aplicación web es proporcionar una interfaz agradable, sencilla y de fácil acceso para poder consultar los aspectos más relevantes que se desprenden del servicio de autobuses de la ciudad de Gijón. Todos estos servicios de consulta serán detallados en la sección dedicada a relatar la arquitectura lógica/estructura de la aplicación web¹.

Por último, destacar, de entre las opciones de consulta de las que dispone la aplicación web de nuestro proyecto, el servicio de consulta destinado a encontrar la ruta entre una parada origen y otra parada destino. Este servicio de consulta será nuestro elemento diferenciador frente a cualquier otro sitio web de transportes públicos, de los cuales, la mayoría, únicamente ofertan servicios de consulta sobre la red de transporte que tratan.

Así pues, nuestra aplicación web tiene por objetivos:

- Proporcionar acceso fácil y ordenado a las distintas componentes que conforman la red de autobuses de Gijón (paradas, trayectos...etc.), es decir, servir de consulta al usuario.
- Proporcionar una descripción exacta del camino a seguir entre dos paradas, esto es, ofrecer al usuario una ruta a seguir que le sirva de guía en su viaje urbano. En nuestro proyecto venimos llamando a esta funcionalidad '*enrutaBus*'.
- Ofrecerse disponible a través de cualquier tipo de dispositivo móvil, o fijo.

Realmente, nuestro propósito para con la funcionalidad de ver las rutas radica en la obtención de una ruta detallada con indicaciones puntualizadas de paradas, líneas y transbordos a realizar entre las dos paradas dadas (origen y destino).

La forma en la que podemos conseguir hacer posible dar salida a esa funcionalidad, es, trasladando la información leída del servicio web a formato de grafo. Entendiendo por esto que, cada parada leída será un nuevo vértice, y cada par de vértices (dentro de un mismo trayecto y próximos entre sí) formarán una arista. Además, habremos de combinar la información proporcionada por el grafo con algún algoritmo de búsqueda que nos proporcione nuestra ruta deseada.

Todo lo relativo a la formación del grafo para su uso en algoritmos de búsqueda que nos proporcione la ruta a seguir, será detallado en el capítulo '*Importancia de los grafos*'.

¹ <http://sanfernando.cs.us.es/~sebastian>

7.1. Descripción

En la ingeniería del software denominamos **aplicación web** a aquellas aplicaciones que los usuarios pueden utilizar accediendo a un servidor web a través de internet o de una intranet mediante un navegador. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores web en la que se confía la ejecución al navegador.

Las aplicaciones web son populares debido a lo práctico del navegador web como cliente ligero, a la independencia del sistema operativo, así como a la facilidad para actualizar y mantener aplicaciones web sin distribuir e instalar software a miles de usuarios potenciales. Existen aplicaciones como los webmails, wikis, weblogs, tiendas en línea que son ejemplos bien conocidos de aplicaciones web.

Éstas, están formadas dependiendo de la estructura que se utilice, por un conjunto de páginas HTML, clases java, páginas XML, páginas JSP, etc..., así como otro tipo de recursos, ya sean ficheros de imágenes, sonidos, texto, etc.

Al igual que ocurría con la aplicación consumidora de datos, la estructura de nuestra aplicación web también se encuentra alojada en nuestra carpeta personal del servidor¹ web (proporcionado por nuestro tutor). Podrá accederse a la aplicación web (nuestro proyecto) a través de la siguiente dirección: <http://sanfernando.cs.us.es/~sebastian>.

El aspecto inicial que presenta la aplicación web es el que mostramos en la siguiente figura:

Proyecto fin de carrera 2010-2011



Figura 32: Pantalla de inicio de la aplicación web

¹ <http://sanfernando.cs.us.es>

A continuación pasaremos a describir brevemente el glosario de opciones disponibles en dichos menú:

- **Proyecto.** En esta opción usted podrá acceder a toda la información referente al proyecto. Memoria del proyecto, overview¹ sobre los opendata y acceso al blog² de proyecto. Mucha información para que se pueda comprender todo lo referente a este proyecto.



Figura 33: Menú proyecto - aplicación web

- **Web Services.** Esta segunda opción sigue siendo para conocer mejor la formación del proyecto. Concretamente se podrá acceder a información sobre lo que es un servicio web y qué servicio web utiliza la aplicación web. Además podemos acceder a información a tiempo real (opción '*consumo ordenado*') sobre las distintas características de los datos almacenados en la base de datos:
 - Configuración: Opción donde poder consultar a tiempo real datos sobre líneas, paradas, trayectos, tramos y autobuses.
 - Estado de los autobuses: Opción donde consultar a tiempo real datos sobre últimas localizaciones de los autobuses y carreras de estos.

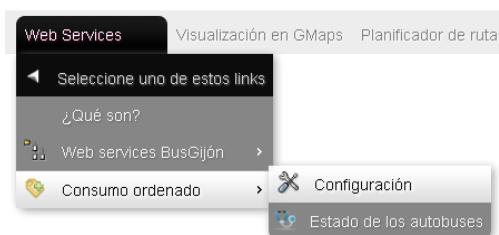


Figura 34: Menú WebServices - aplicación web

- **Visualización en GMaps.** Aquí, además de obtener enlace directo a información sobre el concepto de Google Maps, se podrán visualizar (haciendo uso de la API de GMaps):

¹ <https://datosenabierto.wordpress.com/estado-del-arte/>

² <https://datosenabierto.wordpress.com/>

- Líneas y sus trayectos: Sobre el mapa de Google Maps se podrán observar aquellas líneas y trayectos, a demanda del usuario.
- Rutas de los autobuses: Muestreo de las últimas ‘n’ localizaciones de un autobús seleccionado. Estas posiciones se unirán desde la primera hasta la última formando un grafo. Todo ello sobre el mapa de Google Maps haciendo uso de su API.

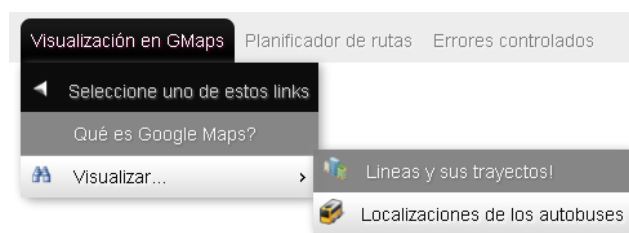


Figura 35: Menú visualizar en Gmaps - aplicación web

- **Planificador de rutas.** Es una de las principales funcionalidades que posee la aplicación web. En ella podremos visualizar, nuevamente sobre un mapa de Google Maps, de forma gráfica los tramos a seguir entre dos paradas de autobús dadas. Esta información será ampliada en la siguiente sección.

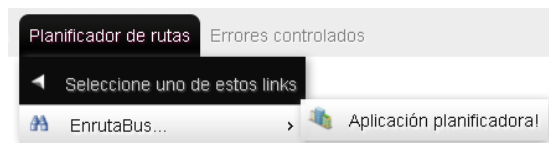


Figura 36: Menú planificador - aplicación web

- **Errores controlados.** Ofrece a los usuarios información acerca de las excepciones provocadas por caídas en nuestro servidor¹, caídas del servidor de Gijón (*busGijón*²), errores de transporte...etc. Es decir, ofrece las excepciones provocadas en la consumición del recuperador de datos mostrando por cada error: identificador, fecha, hora, tipo y descripción de la falla. Decir, que pueden ordenarse las excepciones por cada uno de las características mostradas.

idExcepción	Fecha	Hora	Tipo	Excepción
55	2011-09-22	06:52:28	com. click para ordenar tipo ws.client.ClientTransportException	HTTP transport error: java.net.ConnectEx timed out
54	2011-09-21	11:00:40	com.sun.xml.internal.ws.client.ClientTransportException	HTTP transport error: java.net.ConnectEx timed out
45	2011-09-20	06:21:34	com.sun.xml.internal.ws.client.ClientTransportException	HTTP transport error: java.net.ConnectEx timed out
46	2011-09-20	11:02:50	com.sun.xml.internal.ws.client.ClientTransportException	HTTP transport error: java.net.ConnectEx timed out

Figura 37: Consulta de excepciones en la aplicación web

¹ <http://sanfernando.cs.us.es/>

² <http://docs.gijon.es/sw/busgijon.asmx>

7.2. Arquitectura lógica

De entre todos los ficheros *PHP* que actualmente componen la estructura de nuestra aplicación web, podemos seleccionar algunos como los realmente importantes, siendo el resto mucho más específicos y menos utilizados dentro del funcionamiento habitual. Es por eso, que a continuación paso a mostrar dos diagramas simplificados de la estructura lógica que sigue nuestra aplicación web:

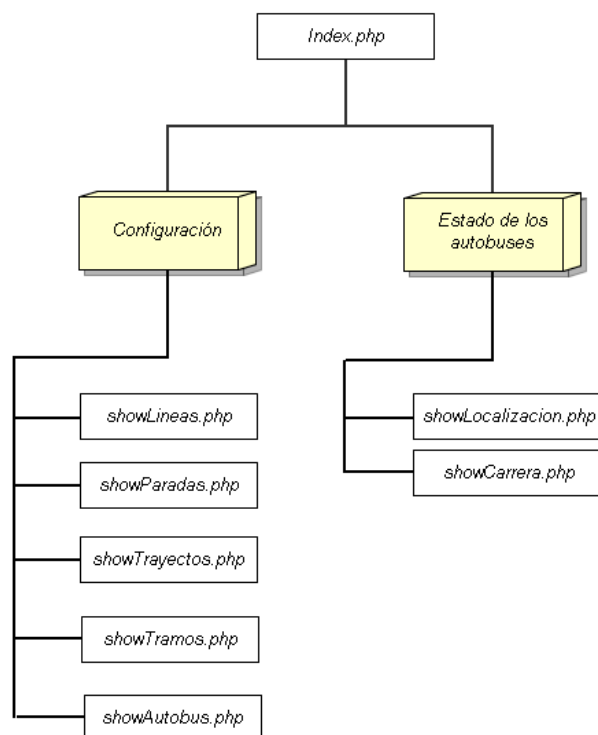


Figura 38: Diagrama aplicación web 1

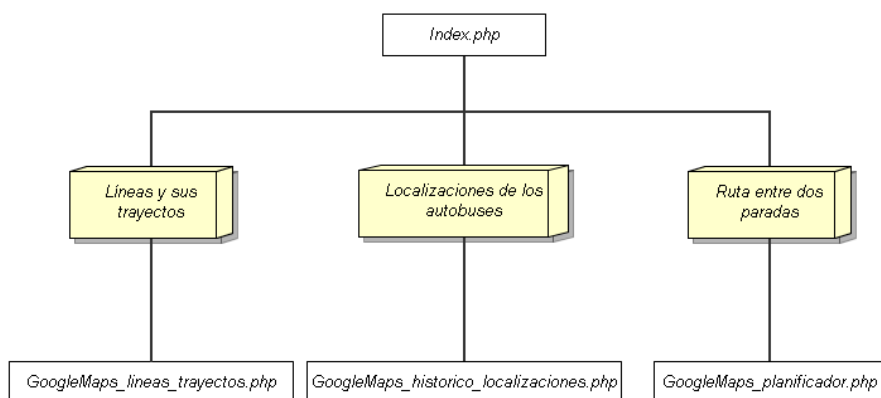


Figura 39: Diagrama aplicación web 2

Los cubos amarillos representan las distintas secciones donde englobar las opciones disponibles a ser consultadas en nuestra aplicación web. De esta forma es más perceptible el tipo de relación existente entre los ficheros más importantes.

Una vez aclarado lo anterior, pasemos a describir cada uno de los ficheros mostrados en las figuras:

Fichero index.php

Responsable de mostrar el menú de navegación (ver figura 32) a través del cual se puede acceder a las distintas secciones representadas en las figuras anteriores:

- Configuración
- Estado de los autobuses
- Líneas y sus trayectos
- Ruta de los autobuses
- Ruta entre dos paradas

Fichero showLineas.php

Es el responsable de mostrar las 33 líneas existentes en la red de autobuses, mostrando por cada una de ellas identificador de línea, tipo y descripción. Gracias a él, podremos filtrar nuestra consulta atendiendo a los distintos tipos de líneas existentes (normal, especial o todos), además de poder realizar una búsqueda de línea en función de su descripción.

También podremos ordenar nuestra consulta por línea y descripción.

Tipo de líneas:

Descripción a buscar:

Línea	Tipo	Descripción
1	NORMAL	CERILLERO - HOSPITAL <input data-bbox="877 1456 1149 1489" type="button" value="click para ordenar Descripción!"/>
2	NORMAL	EL CORTE INGLES - HOSPITAL DE CABUEÑES
4	NORMAL	LAUREDAL - CAMPUS (MESTAS)
5	NORMAL	PLAZA DEL HUMEDAL - HOSPITAL DE JOVE - MUSEL

Figura 40: Vista de la tabla 'Lineas'

Fichero showParadas.php

Este fichero, tiene por función el mostrarnos las 605 paradas de las que se compone la red de autobuses, mostrando por cada una de las paradas su descripción, localización en un mapa (valiéndonos de la Api de GoogleMaps) y listado de líneas adscritas a dicha parada. Además, nos ofrece la opción de buscar una parada en base a su descripción, apoyándose en las sugerencias que aparecerán al realizar cualquier búsqueda. También es posible la ordenación de nuestra consulta atendiendo a la descripción.

Inserte la parada a buscar:

 || Paradas

Descripción	Ver en mapa	Líneas que pasan por aquí!	
ZORRILLA		Linea 15	<input type="button" value="Ver"/>
VIÑAO		Linea 2	<input type="button" value="Ver"/>
VILLAR		Linea 2 Linea 4 Linea 18	<input type="button" value="Ver"/>
VILLAR		Linea 2	<input type="button" value="Ver"/>
VIESQUES		Linea 2	<input type="button" value="Ver"/>
VIESQUES		Linea 2	<input type="button" value="Ver"/>

Figura 41: Vista de la tabla 'Paradas'

Fichero showTrayectos.php

Gracias a este fichero, tenemos la posibilidad de consultar todas y cada una de los 157 trayectos de los que se compone la red de autobuses. Cada trayecto viene identificado por el número de línea al que pertenece, identificador de la parada de origen, identificador de la parada de destino, sentido y su descripción.

Además, podremos realizar una búsqueda de trayecto, atendiendo a la descripción de éste. Nuevamente, se nos ofrecerán una serie de sugerencias en la realización de nuestra búsqueda para que ésta sea más cómoda. Pudiendo también, realizar ordenaciones en nuestra consulta según línea, origen, destino, sentido y descripción (ascendente y descendientemente por cada uno de los atributos citados).

Inserte el trayecto a buscar:

Línea	Origen	Destino	Sentido	Descripción
1	180	34	Ida	CERILLERO - HOSPITAL DE CABUEÑES
1	34	180	Vuelta	HOSPITAL DE CABUEÑES - CERILLERO
1	34	180	Vuelta	HOSPITAL CABUEÑES-CERILLERO (POR SEM. NEGRA)
1	34	180	Vuelta	HOSPITAL DE CABUEÑES - CERILLERO
2	538	34	Ida	EL CORTE INGLES - HOSPITAL DE CABUEÑES

Figura 42: Vista de la tabla 'Trayectos'

Fichero showTramos.php

Responsable de mostrar de forma ordenada cada uno de los 4052 tramos de que se compone la red de autobuses de Gijón. Cada tramo está compuesto por identificador de línea, identificador de trayecto, orden y descripción de la parada.

Nuevamente tenemos opción de realizar una búsqueda de tramo atendiendo a la parada. Esto más bien es una criba, ya que por cualquier parada pertenece a varios tramos.

Inserte la parada a buscar:

Línea	Trayecto	Orden	Parada
4	14	14	ACUARIO
12	9	17	ACUARIO
16	26	33	ACUARIO
16	24	24	ACUARIO

Click para ordenar parada!

Figura 43: Vista de la tabla 'Tramos'

Fichero showAutobus.php

Este fichero es responsable de mostrarnos los autobuses que hayan circulado por la ciudad de Gijón durante el tiempo de monitorización de nuestra aplicación cliente. Recordemos que esta aplicación cliente se encarga de consumir y registrar los datos del servicio de autobuses en nuestra base de datos, y que esta consumición de datos se realiza de forma programada actualizando las últimas localizaciones de los autobuses.

Cada autobús será mostrado en función de su modelo y la última localización que se haya registrado de él. Por lo que podremos visualizar su última posición en el mapa de Google Maps, haciendo uso de su API. Como viene siendo habitual, podremos realizar una búsqueda de autobús en función de su modelo, ayudándonos de las sugerencias de búsqueda, además de poder ordenarlos por modelo.

Inserte la parada a buscar:

Línea	Trayecto	Orden	Parada
4	14	14	ACUARIO
12	9	17	ACUARIO
16	26	33	ACUARIO
16	24	24	ACUARIO

Click para ordenar parada!

Figura 44: Vista de la tabla 'Autobus'

Fichero showLocalizacion.php

ShowLocalización es el fichero responsable de mostrarnos las localizaciones de los autobuses existentes en nuestra base de datos. Estas serán las últimas localizaciones registradas por dichos autobuses. Cada una de ellas va acompañada del modelo de autobús, identificador de la línea sobre la que se encontraba dicho autobús en ese momento, identificador de la siguiente parada, fecha, hora en la que se registro la localización y visualización de ella sobre el mapa de Google Maps.

Todos y cada uno de los atributos característicos de una localización son ordenables, además, tenemos la posibilidad de buscar una localización concreta atendiendo al modelo de autobús.

Modelo a buscar:





Autobus	Línea	Siguiente parada	Fecha	Hora	Ver en mapa
NO94UB4X2	34	10		23:41	
NO94UB4X2	12	110	2011-09-21	23:39:49	
NL 263 F	10	109	2011-09-21	23:39:49	
MAN NL 263 F	16	663	2011-09-21	23:21:20	

Figura 45: Vista de la tabla 'Localizacion'

Fichero showCarrera.php

Responsable de mostrarnos las carreras registradas en la red de autobuses de Gijón por nuestra aplicación cliente. Cuando hablamos de '*Carrera*' nos referimos a las distintas llegadas o aproximaciones a una parada que de un autobús se desprende. Estas aproximaciones o llegadas son definidas por el identificador del autobús, número de la parada hacia la que se aproxima, número de línea en la que se encuentra dicho autobús, número de trayecto, fecha, hora en la que se registró, minutos estimados para la llegada de dicho autobús a la parada y por último, distancia a la que se encuentra el autobús de la parada (en metros).

Todos y cada uno de los atributos que caracterizan a una carrera son ordenables. Además disponemos de dos filtros que nos ayudarán a una mejor visualización de las carreras, '*líneas a filtrar*', que nos permitirá filtrar las carreras atendiendo a la línea elegida, y '*búsqueda por fecha*', que nos ayudará a filtrar las carreras en función de la fecha elegida.

Por último, remarcar, que el número de carreras mostradas está limitado a 100. El motivo es debido a la gran cantidad de registros que contiene esta tabla en nuestra base de datos y que hacen que cada consulta se prolongue en el tiempo una barbaridad. Así pues, nos vimos en la necesidad de limitar la consulta a 100 registros.

Líneas a filtrar:

Búsqueda por fecha:

Autobus	Parada	Línea	Trayecto	Fecha	Hora	Minutos	Distancia
280	138	34	2	2011-09-21	23:43:10		399.891312
280	214	34	2	2011-09-21	23:43:10	5	708.640337
280	219	34	2	2011-09-21	23:43:10	23	4795.814136
280	16	34	2	2011-09-21	23:43:09	9	2226.347737

Figura 46: Vista de la tabla 'Carrera'

Fichero GoogleMaps_lineas_trayectos.php

Responsable de mostrar las líneas y sus trayectos a demanda sobre el mapa de Google Maps. La representación gráfica sobre el mapa de la ciudad de Gijón constará de polilíneas (línea recta) formadas por la unión de las paradas que integran dicha línea.

Cada polilínea representa un trayecto de la línea seleccionada, mostrándose de un color pastel que permita su rápida identificación.

Como ya se ha comentado, al seleccionar una línea concreta se podrá observar la totalidad de trayectos que esa línea contiene. De lo contrario, si se desea contemplar un trayecto concreto, este aparecerá dibujado sobre el mapa con el comienzo y final de dicho trayecto remarcado con un marcador (verde y rojo respectivamente). Además, si se desea, gracias al botón 'ver más', podremos visualizar la totalidad de paradas que integra dicho trayecto.



Figura 47: Imagen descriptiva para la visualización de líneas

Fichero GoogleMaps_historico_localizaciones.php

Este fichero es el encargado de mostrar las últimas localizaciones (de un autobús determinado) en forma de grafo sobre un mapa Google Maps de la ciudad de Gijón. Este grafo constará de polilíneas (líneas rectas) formadas por la unión de las últimas localizaciones del autobús seleccionado. Se tomarán tantas localizaciones como se indique en la selección de muestreo [30 - 10000]. Por lo que, digamos que la representación que puede observarse es un seguimiento de las últimas 'n' localizaciones muestreadas de un autobús determinado.

El grafo aparecerá dibujado sobre el mapa con el comienzo y final del total de localizaciones muestreadas remarcados con un marcador (color verde y rojo respectivamente). Además, si se desea, gracias al botón 'ver más', podremos visualizar la totalidad de paradas que integran dicho grafo.

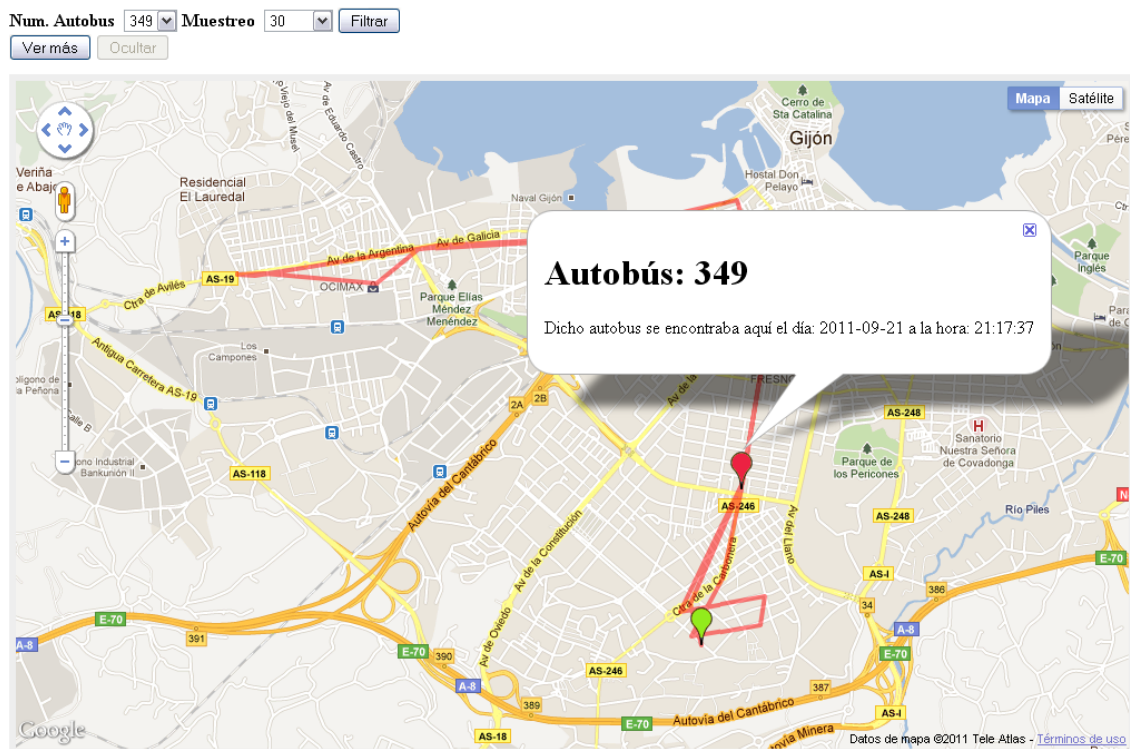


Figura 48: Imagen descriptiva del histórico de localización de un bus

Fichero GoogleMaps_planificador.php

Nos encontramos ante uno de los ficheros con más peso e importancia de entre todos los ya citados. Este fichero es el responsable de proporcionarnos una ruta detallada con las indicaciones para ir desde una parada de origen hasta una de destino. Es decir, dadas dos paradas, si existe ruta posible, nos ofrece las paradas de las que se compondrá nuestro camino, la línea/líneas a las que pertenecen, y los transbordos o cambios de línea si los hubiese.

Como venimos haciendo hasta ahora, la representación de la ruta vendrá de la mano de las polilíneas (líneas rectas) usadas por la API de Google Maps, quedando remarcados el comienzo y final de la ruta por un marcador (color verde y rojo respectivamente). Si existiesen varios transbordos o cambios de autobús en nuestra ruta, esto sería representado por cambios de color en la polilínea representada sobre el mapa. De esta forma se visualiza perfectamente los distintos transbordos y los cambios que hemos de hacer gracias a esa transición de color.

Siguiendo con la parte de representación gráfica, podremos visualizar la totalidad de paradas que integran nuestra ruta gracias al botón '*ver más*', estas paradas serán representadas por un marcador color azul. Por el contrario, podremos visualizar aquellos autobuses que se encuentren actualmente circulando por las líneas que integren nuestra ruta, gracias al uso del botón '*busLive!*'. Estos autobuses serán señalados con un marcador color rosa sobre el mapa de la ciudad de Gijón, representada gracias a la API

de Google Maps. De esta forma tendremos constancia de la posición del resto de autobuses que circulan por nuestra línea por si necesitásemos de su servicio. Aparte de las marcas color rosa, nos encontraremos con marcas color azul que nos indicarán las paradas de las que se compone nuestro trayecto.

Añadir, que tanto las paradas origen y destino vienen en formato autocompletado, mostrando su descripción más identificador. De esta forma, en la búsqueda de nuestra parada nos apoyaremos en la ayuda proporcionada por las sugerencias de éste.

Parada Origen	<input type="text" value="CAMINO DE RUBÍN - 180"/>
Parada Destino	<input type="text" value="PRAO DEL CUBANO - 136"/>
<input type="button" value="Filtrar"/>	<input type="button" value="Ver más"/> <input type="button" value="Ocultar"/> <input type="button" value="busLive!"/>



Figura 49: Imagen descriptiva del planificador de rutas

A continuación paso a mostrar un esquema de relaciones entre los anteriores ficheros citados (excluyendo al index). En él, observamos las conexiones que resultan de 'clickar' sobre determinados atributos que por lógica uno interrelaciona con tablas ya vistas. Es decir, en 'Trayectos', el atributo 'línea' se encuentra relaciona con la vista 'Líneas'. De manera que si pulsamos sobre dicho atributo nos muestras las características aplicables a dicha línea sobre la vista 'Líneas'. Ésta explicación es aplicable para todas y cada una de las vistas que generan los ficheros '.php' antes vistos:

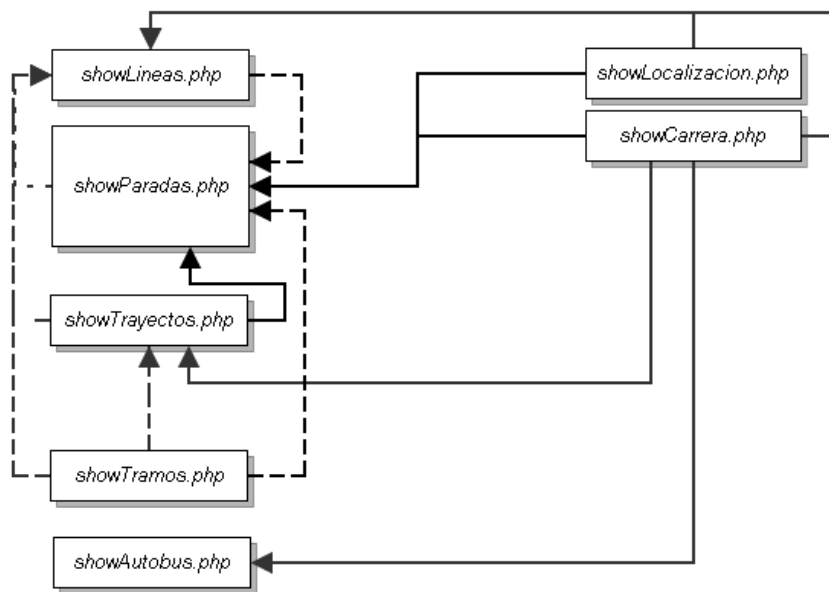


Figura 50: Esquema de relaciones entre ficheros PHP de la web

Así pues, este esquema podría considerarse como el flujo de navegación entre los elementos de cada una de las páginas o vistas que generan los '.php'.

7.3. Funcionalidad móvil

Actualmente, los teléfonos móviles forman parte de nuestro estilo de vida. Hoy en día los dispositivos móviles pueden conectarse a internet y ejecutar aplicaciones web. Es por eso que en nuestro proyecto nos hemos visto en la obligación de hacer accesible ésta aplicación web vía móvil, aparte, claro está, de estar accesible vía web.

Para hacer esto posible, en nuestro servidor, <http://sanfernando.cs.us.es/~sebastian>, debíamos albergar dos carpetas donde estructurar el contenido de nuestra aplicación web en función del dispositivo. 'Web' y 'Movil'. En la carpeta 'Web' tendremos el 95% de los ficheros que dan soporte al funcionamiento de la aplicación, a excepción de ese 5% que reside en la carpeta 'Movil'. En la carpeta 'Movil' únicamente tendremos un fichero (a excepción de los css...etc dependientes), 'index.php', en el que haciendo uso del plugin de jQuery (jQuery¹) para desarrollo web móvil para iPhone...etc, damos un aspecto acondicionado a estos dispositivos de ancho 480px.

No obstante, cuando un usuario accede a la dirección que da acceso a nuestra aplicación web, <http://sanfernando.cs.us.es/~sebastian>, se ha de elegir entre acceder a la carpeta 'Web' que contiene el contenido que permite visualizar nuestra web en dispositivos fijos o acceder a la carpeta 'Movil' que adapta la visualización de nuestra web a dispositivos móviles (nos cercioramos de que estos no tengan más de 480px de ancho). Para hacer posible esta elección, el fichero 'index.php' hace uso de la función php ('esMovil.php') que contempla todas las marcas y modelos de dispositivos móviles actuales, para de esta forma redirigir según sea el caso al fichero 'index.php' de la carpeta 'Web' o al fichero 'index.php' de la carpeta 'Movil'.

Así pues, la estructura que nos queda en la carpeta personal del servidor, <http://sanfernando.cs.us.es>, es la siguiente:

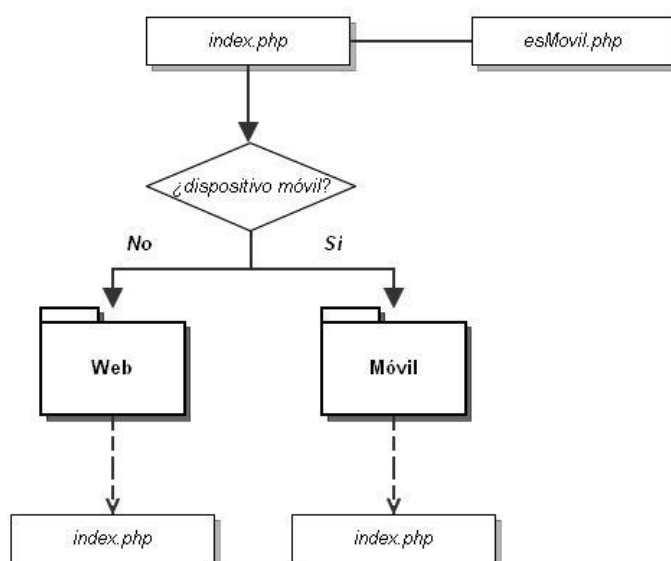


Figura 51: Estructura 'esMovil.php' en el servidor

¹ <http://jqtouch.com/>

Opera mobile

Llegados a este punto lo único que resta es probar o testear como se verá nuestra aplicación en cualquier dispositivo móvil. ¿Cómo hacer esto?. Haciendo uso del emulador *Opera Mobile*. Esta herramienta cumple perfectamente su misión y nos muestra como se verá nuestra aplicación web en cualquier dispositivo móvil. Aunque como ya hemos comentado, la resolución que le hemos dado durante el desarrollo de nuestra aplicación ha sido de un ancho de 480px y 800px de alto, por defecto es la resolución del *HTC Desire HD* (móvil en el que se ha testeado). Configuraciones menores mostraran vistas distorsionadas de la aplicación web.

¡Ojo!, si usted prueba la aplicación web con un iphone, deberá apaisarlo, es decir, ponerlo en modo horizontal. Ya que un iphone tiene la siguiente configuración 320x480 px.



Figura 52: Vista de Opera mobile

Por todo esto, en los bloques de estilo de cada uno de los ficheros *‘.php’* más importantes, hemos usado la propiedad “media” de CSS. Con esto, podemos especificar reglas y estilos que sean específicos para por ejemplo el **iphone** o cualquier otro dispositivo móvil del mercado.

Así podemos usar la expresión “max-device-width: 480px” para indicar que una hoja de estilos se muestre únicamente en dispositivos de pantalla cuya anchura máxima sea 480 pixeles. Utilizamos 480 pixeles porque es la anchura máxima en pixeles que muestra un iphone en posición horizontal. Aparte de la *HTC Desire HD* cuyas dimensiones son 480x800px, dimensiones que hemos seguido como modelo.

De la misma forma, hemos de especificar que una hoja de estilos se muestre en dispositivos cuya pantalla sea anchura superior a 480 pixeles usando la expresión “min-device-width: 480px”. De esta forma contemplamos el uso que alguien le pueda a nuestra aplicación desde un dispositivo fijo y de grandes dimensiones.

Así pues, algunas de las vistas tomadas de nuestra aplicación web a través de *Opera mobile* pueden verse en las siguientes figuras:



Figura 53: Página de inicio - Vista móvil



Figura 54: Página de consulta - Vista móvil

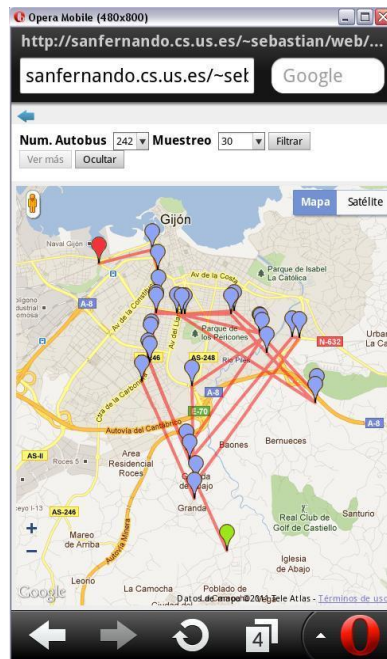


Figura 55: Página de consulta 2 - Vista móvil

8. Planificador de rutas

La importancia de los grafos en el proyecto radica en la necesidad de realizar búsquedas entre dos paradas cualesquiera, para así obtener la totalidad de paradas existentes entre ellas y poder determinar la ruta (funcionalidad de nuestra aplicación web). Estas búsquedas son realizadas en base a la información que ya tenemos en nuestra base de datos.

Las búsquedas únicamente son realizables sobre grafos. Hemos de trasladar la información leída de la red de autobuses de Gijón a formato grafo, entendiendo que cada parada leída será un nuevo vértice, y cada par de vértices (dentro de un mismo trayecto y próximos entre sí) formaran una arista.

En una primera instancia, realizaremos una comparativa de eficiencia entre unos algoritmos de búsqueda con el fin de encontrar el más eficiente sobre el grafo formado. Dicho algoritmo proporcionará un listado de las paradas que más tarde formará la ruta con sus indicaciones. En la siguiente figura puede observarse lo que pretendemos obtener, a partir de dos paradas (marcas verde y roja), conseguir el listado de paradas intermedias (marcas violetas):



Figura 56: Listado de paradas entre dos paradas dadas.

En segunda instancia, y tras obtener el listado de paradas entre origen y destino, hemos de transcribir ese listado de paradas a un conjunto de indicaciones de ruta. Dichas indicaciones son los pasos guiados a seguir para realizar nuestro viaje entre las paradas de origen y destino.

Hasta el momento, todas estas acciones son realizadas por clases Java y lo que aquí se pretende es conseguir, exactamente lo mismo, pero bajo el script *‘.php’* de búsquedas de rutas que el usuario final visualiza en nuestra aplicación web. Para ello habremos de ejecutar código Java bajo código PHP haciendo uso de una herramienta que posibilita esta acción.

Por lo que, llegados a este punto, nos encontraríamos en disposición de consultar cualquier tipo de ruta entre dos paradas de la red de autobuses de Gijón.

8.1. Creación del grafo

La información pertinente a las líneas, paradas, trayectos y tramos que forman parte de nuestra base de dato, son la fuente de la que partir para crear un grafo. Este grafo pasará a ser la representación formal de la red de autobuses que aquí estamos tratando. Donde, cada parada representará un vértice y cada par de vértices (dentro de un mismo trayecto y próximos entre sí) formarán una arista.

Por lo que, de entre las tablas existentes en nuestra base de datos, para formar el grafo, se harán uso de las siguientes:

- **Líneas**
- **Paradas**
- **Trayectos**
- **Tramos**

De forma general y para que el concepto de lo que se pretende hacer quede más claro, se han de comentar un par de cosas:

- Cada vértice del grafo queda representado por una o varias paradas (si estas se solapasen según el trayecto).
- Una arista, estará formada por dos paradas/vértices.

#	idParada	Descripcion	Lat	Lng
1	1	MUSEL	43.5654592752136	-5.70221994767325
2	2	MUSELÍN	43.5586662598484	-5.70392199164054

Figura 57: Consulta de paradas a la base de datos

- Cada línea queda identificada por un conjunto de trayectos, los cuales quedan definidos por un conjunto de paradas que siguen un orden determinado.

#	idLinea	idTipoLinea	Descripcion
1	1	1	CERILLERO - HOSPITAL DE CABUEÑES

Figura 58: Consulta de líneas a la base de datos

#	idTrayecto	idLinea	Descripcion	Sentido	idExtremos
1	1	1	CERILLERO - HOSPITAL DE CABUEÑES	Ida	1
2	2	1	HOSPITAL DE CABUEÑES - CERILLERO	Vuelta	2
3	6	1	HOSPITAL DE CABUEÑES - CERILLERO	Vuelta	3
4	7	1	HOSPITAL CABUEÑES-CERILLERO (POR SE...	Vuelta	4

Figura 59: Consulta de trayectos a la base de datos

- El conjunto de paradas ordenadas que definen un trayecto queda reflejado en la tabla *'Tramos'*.

#	idParada	idLinea	idTrayecto	Orden	Descripción
1	180	1	1	0	CAMINO DE RUBÍN
2	181	1	1	1	INEM
3	141	1	1	2	GRAN CAPITÁN
4	136	1	1	3	PRAO DEL CUBANO
5	137	1	1	4	URUGUAY
6	8	1	1	5	CASA DEL MAR
7	9	1	1	6	CUATRO CAMINOS
8	10	1	1	7	SANTA OLAYA
9	138	1	1	8	DOS DE MAYO
10	214	1	1	9	PLAZA DE LA HABANA
11	13	1	1	10	ESTACIÓN DEL NORTE

Figura 60: Consulta de tramos a la base de datos

8.1.1. Arquitectura lógica

Tras conocer las tablas que intervendrán en la formación del grafo, pasaremos a detallar la estructura lógica que contendrá el proyecto Java que posibilitará la creación del grafo. Este grafo contendrá la estructura de la red de autobuses de Gijón.

Para conseguir esto, usaremos nuestro proyecto de nombre *'Grafo'*, haciendo uso del paquete *'grafo'*. A continuación pasaremos a observar un diagrama de clases simplificado de las clases contenidas en este paquete, a excepción de la clase *'Puente'* que será descrita más adelante.

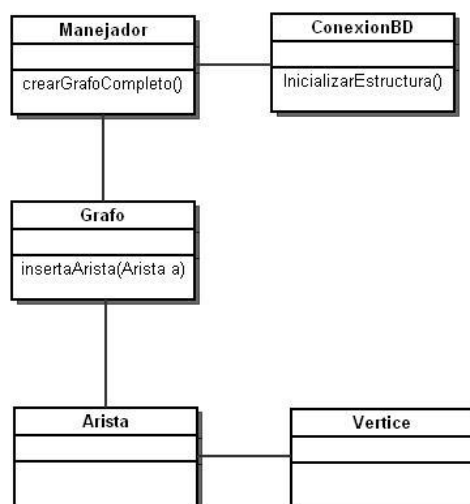


Figura 61: Diagrama de clases del paquete Grafo

Así pues, pasamos a describir el cometido de cada una de las clases de la figura anterior:

Vertice

Esta clase representa la unidad mínima del grafo. Contiene los atributos característicos que permiten diferenciarlo del resto de vértices; Algunos de los más importantes son: identificador de la parada de autobús, descripción y localización de ésta parada (Latitud, longitud).

Arista

Esta clase contendrá tres atributos, dos de los cuales serán los *vértices* que forman esta arista, y un tercero, correspondiente a un atributo '*Etiqueta*' encargado de almacenar la distancia real medida en kilómetros entre dichos vértices.

ConexionBD

Clase encargada de la definición de los datos básicos de conexión a nuestra base de datos, así como de la creación de las tablas que integran dicha base de datos. Esta clase define además, los métodos encargados de realizar las consultas y actualizaciones, así como el cierre de la conexión.

Grafo

La clase '*Grafo*' será la encargada de crear la estructura que relacione las distintas aristas que se irán insertando mediante el método '*insertarArista*'. Estas relaciones se mantienen gracias a una lista de adyacencias '*lista_adyacencias_idparada*' que mantiene enlazadas las paradas/vértices por su identificador de parada.

Manejador

Digamos que ésta, es la clase principal que hace uso de las anteriores clases ya descritas. Es decir, a través de su método '*crearGrafoCompleto()*' estaremos permitiendo la creación de un grafo que incluirá las 33 líneas de la red de autobuses además de todos los trayectos pertenecientes a cada una de ellas.

Aunque si lo deseamos, tenemos la posibilidad de crear un grafo que solo contemple una línea y todos sus trayectos (en lugar de la red de transportes al completo). Esta opción nos servirá para más adelante, comparar la eficiencia de los algoritmos de búsqueda teniendo en cuenta si buscan una ruta en 1 sola línea, o en las 33 líneas que componen la red de autobuses.

Una vez hemos visto la estructura que rodea a la formación de nuestro '*Grafo*', hemos de implementar un algoritmo capaz de proporcionarnos el camino a seguir entre dos paradas (ver figura 53). Este algoritmo de búsqueda ha de proporcionarme un camino óptimo entre dos paradas.

Se han de tener en cuenta varios algoritmos de búsqueda, para escoger el más eficiente y el que mejor se adapte a la búsqueda en nuestro grafo. Es decir, vamos a tener un espectro de 3 algoritmos de búsqueda, entre los que posteriormente, estableceremos una comparativa de eficiencia que nos permita decantarnos por uno de ellos:

1. **Algoritmo A***
2. **Búsqueda en profundidad**
3. **Algoritmo de Floyd**

El rendimiento será medido sobre 2 variables fundamentales:

- Número de paradas en que se realiza la ruta/camino.
- Cantidad de memoria empleada por éste.

Además estableceremos dos versiones sobre las que medir estas variables:

- **Versión de una línea**

Esta versión contempla una línea elegida de entre las 33 existentes. Por lo que contendrá la formación de un grafo creado gracias a los trayectos que contiene dicha línea.

- **Versión de todas las líneas**

Versión que contempla un grafo formado por todas las líneas de las que consta el servicio de autobús de Gijón. 33 líneas con sus correspondientes trayectos.

8.2. Búsquedas empleadas

Para posibilitar la obtención de una ruta entre dos paradas de la red de autobuses, hemos de implementar un algoritmo de búsqueda que permita capturar el conjunto de paradas de las que se compone dicho camino/ruta.

Para conseguirlo, pasaremos a implementar un conjunto de algoritmos búsquedas, más concretamente tres. De entre los que, más adelante, se establecerá una comparativa de eficiencia para saber cual aplicaremos a la funcionalidad de nuestra aplicación web. Funcionalidad que nos permite obtener la ruta entre dos paradas.

8.2.1. Algoritmo A*

DESCRIPCIÓN

El **algoritmo de búsqueda A*** (pronunciado ‘A asterisco’ o ‘A estrella’) se clasifica dentro de los algoritmos de búsqueda informados. Presentado por primera vez en 1968 por Peter E. Hart, Nils J. Nilsson y Bertram Raphael, el algoritmo A* encuentra, siempre y cuando se cumplan unas determinadas condiciones, el camino de menor coste entre un vértice origen y uno destino.

El problema de algunos algoritmos de búsqueda en grafos informados, es que se guían en exclusiva por la función heurística, la cual puede no indicar el camino de coste más bajo, o por el coste real de desplazarse de un nodo a otro. Pudiéndose dar el caso de que sea necesario realizar un movimiento de coste mayor para alcanzar la solución. Se da por ello el hecho de que un buen algoritmo de búsqueda informada debería tener en cuenta ambos factores, el valor heurístico de los nodos y el coste real recorrido.

Así, el algoritmo A* utiliza una función de evaluación $F = G + H$, donde H representa el **valor heurístico** del nodo a evaluar desde el actual hasta el nodo de destino. Y G el **coste real** del camino recorrido para llegar a dicho nodo destino.

A* mantiene dos estructuras de datos auxiliares o listas que almacenarán nodos, la lista *Abiertos*, que contiene los nodos que se tienen que expandir, y la lista *Cerrados* que contiene los que ya han sido expandidos. El nodo *Origen* se inserta en la lista de *Abiertos* para ser expandido. En este punto se inicia la propagación de la búsqueda, se crea un bucle que acabará en dos posibles opciones: cuando la lista *Abiertos* este vacía, o cuando se encuentre el nodo *Destino*.

El bucle consiste en varios pasos. Primero se obtiene el primer nodo de la lista *Abiertos*. Se tomará aquel nodo con menor F . Si esta lista estuviese vacía, significaría que el algoritmo no ha encontrado solución alguna y por tanto no se halla el *Destino*, luego el algoritmo ha fallado. Por otra parte, si el nodo que cogimos de la lista *Abiertos* es el *Destino*, el algoritmo habrá acabado ya que hemos encontrado el *Destino*.

El camino entre el origen y el destino es un conjunto de nodos que se obtendrán guardando cada predecesor del nodo destino. En otro caso, se seguirá expandiendo la lista, obteniendo una lista de sucesores del nodo actual, y guardando en la lista *Abiertos*, aquellos nodos que no estuvieran antes en una lista, es decir, aquellos nodos a los que aun no habíamos llegado. De esta forma, el algoritmo acabará encontrando el *Destino* o recorriendo todos los nodos.

IMPLEMENTACIÓN

A continuación pasaremos a explicar nuestra implementación del *algoritmo A** en el lenguaje Java. Nos encontramos ante una búsqueda de soluciones con información del dominio.

Para tomar contacto con la estructura de nuestro programa adjunto una imagen descriptiva de los métodos y atributos empleados en el algoritmo, junto a su visibilidad.

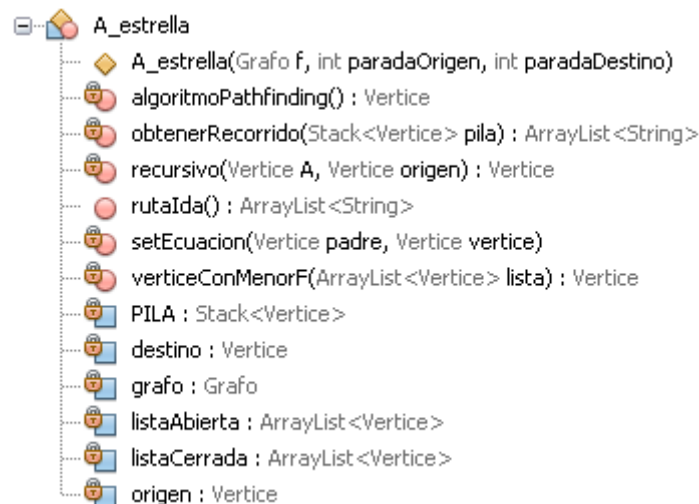


Figura 62: Atributos y métodos del algoritmo A*

Vamos a analizar más detenidamente cada uno de los miembros contenidos en esta implementación. Analizándolos de forma agrupada según su categoría, estas son: atributos y métodos.

■ *Atributos*

1. **grafo : Grafo**
Objeto que contendrá cargado el grafo sobre el cual aplicar el algoritmo.
2. **origen : Vertice**
Vertice desde el cual iniciamos el camino.
3. **destino : Vertice**
Vertice al cual llegar.
4. **listaAbierta : ArrayList<Vertice>**
Lista de vértices que contendrá aquellos vértices por tratar.
5. **listaCerrada : ArrayList<Vertice>**
Lista de vértices que contendrá aquellos vértices que ya han sido tratados.

6. PILA : *Stack<Vertice>*

Pila donde almacenaremos el recorrido a seguir entre los dos puntos dados.

■ *Métodos*

1. *A_estrella(Grafo g, int paradaOrigen, int paradaDestino)*

Constructor que inicializa los atributos privados de la clase gracias a los atributos pasados por parámetros.

Nuestros vértices son inicializados gracias al identificador de las paradas obtenidas por parámetro.

2. *algoritmoPathfinding() : Vertice*

Método principal sobre el que se sustenta la implementación del A*.

Dicho método, añade el vértice inicial a la lista *Abierta* y repite lo siguiente mientras la lista *Abierta* no sea vacía o el vértice actual sea distinto del vértice destino.

Busca de entre la lista *Abierta* aquel vértice con menor valor *F*, para acto seguido hacerlo vértice actual. Con esta acción el vértice es eliminado de la lista *Abierta* y pasado a la lista *Cerrada*.

Tras comprobar que el vértice actual es distinto del vértice de destino, tomamos todos los vértices adyacentes al actual y por cada vértice adyacente comprobamos que dicho adyacente o bien no se encuentre en la lista *Abierta* y *Cerrada* o por el contrario que se encuentre en la lista *Abierta*.

Si el vértice adyacente a tratar no figura en la lista *Abierta*, ni en la lista *Cerrada*, asignamos a dicho vértice los costes *F*, *G* y *H*. Hacemos que el vértice adyacente apunte a su vértice padre, es decir, al vértice actual y finalmente añadimos el vértice adyacente a la lista *Abierta*.

Si por el contrario nos encontramos en la situación de que el vértice adyacente ya figuraba en la lista *Abierta*, hemos de comprobar si el camino para este vértice es mejor usando el coste *G* como baremo. Un coste *G* menor significa que este será un mejor camino. Si es así, cambiamos el padre del vértice adyacente (al estar en la lista *Abierta* es de suponer que ya tendría asignado un *padre*) tomando como padre el vértice actual y recalculamos los costes *F*, *G* y *H*.

Por tanto, y recordando lo ya dicho, nuestro algoritmo finalizará cuando sea añadido a la lista *Abierta* el vértice *destino* en cuyo caso el camino habrá sido encontrado, o por

el contrario se falle en encontrar el vértice *destino* y la lista *Abierta* esté vacía. En este caso no hay camino.

Para finalizar, hemos de decir que el método devolverá el vértice *destino* el cual se encontrará apuntando a su vértice *padre*, el cual a su vez también estará apuntando a su vértice *padre* y así sucesivamente hasta que lleguemos al vértice *origen* de forma que tenemos nuestro camino.

Los métodos de los que depende son:

- *Vertice verticeConMenorF(ArrayList<Vertice> lista)*
- *Void setEcuacion(Vertice padre, Vertice hijo)*

```

METODO

    vertice actual <- null
    listaAbierta <- vertice origen

    REPETIR HASTA( tamaño listaAbierta == 0)
        actual <- verticeConMenorF(listaAbierta)
        listaCerrada <- actual

        SI (actual == destino) ENTONCES
            Devolver actual //FIN
        SI NO
            adyacentes[] <- getAdyacentes(actual)
            /*Recorro todos los adyacentes*/
            REPETIR HASTA (visitar todos los adyacentes)
                SI (listaAbierta no contiene adyacente ^
                    listaCerrada no contiene adyacente) ENTONCES

                    /*Establezco los costes F, G y H*/
                    setEcuacion(actual, adyacente)
                    /*ady. apunta a su padre*/
                    adyacente->setPadre(actual)
                    /*Añado a la listaAbierta el vertice ady.*/
                    listaAbierta <- adyacente

                SI NO (listaAbierta contiene adyacente) ENTONCES
                    SI (adyacente->getG() < actual->getG()) ENTONCES
                        /*Establezco los costes F, G y H*/
                        setEcuacion(actual, adyacente)
                        /*Hago que apunte al padre actual*/
                        adyacente -> setPadre(actual)
                    FIN SI
                FIN SI
            FIN REPETIR
        FIN SI
    FIN REPETIR
FIN METODO

```

Figura 63:Pseudocódigo de método 'algoritmoPathfinding'

3. **verticeConMenorF(ArrayList<Vertice> lista) : Vertice**

Método encargado de devolver el vértice con menor valor F . Recordemos que F es la heurística empleada para calcular el camino óptimo hasta el destino y siempre hemos de tomar el vértice con valor para asegurarnos dicha acometida.

Su implementación consiste en recorrer el listado de vértices, siempre que éste tenga más de 1 elemento, y devolver el vértice que tenga menor valor F .

Antes de la devolución del vértice, éste es eliminado de la lista *Abierta*.

```
COMIENZO(lista de Vertices)

    resultado <- null //Vertice resultado a devolver

    /*Insertamos en resultado el primer vertice*/
    resultado <- lista[0]

    SI (tamaño lista > 1) ENTONCES
        PARA i<-1 HASTA i< tamaño lista HACER
            SI ( lista[i]->getF() < resultado.getF() ) ENTONCES
                resultado <- lista[i] //añado vertice a resultado
            FIN SI
        FIN PARA
    FIN SI

    Devolver resultado

FIN
```

Figura 64: Pseudocódigo del método 'verticeConMenorF'

4. **setEcuacion(Vertice padre, Vertice hijo) : void**

Este método es el encargado de establecer los costes F , G y H necesarios para el cálculo del mínimo camino. En dicha función de evaluación G representa el coste real del camino recorrido para llegar al nodo *destino* y H representa el valor heurístico del vértice a evaluar -hijo- desde el actual hasta el vértice de *destino*.

Por lo tanto y teniendo en cuenta lo anterior, nuestro valor G será la distancia en kilómetros desde el vértice padre hasta el vértice hijo. Es decir, dejamos constancia del camino real que nos llevará al vértice *destino*.

Asimismo, el valor H ha de ser un valor de estimación de la distancia existente hasta el vértice *destino*. Por lo tanto, vamos a tomar la distancia en kilómetros desde el vértice hijo hasta el vértice *destino*.

Para el cálculo de estas distancias nos valdremos de la librería que nos permite realizar conversiones entre distintos sistemas de representación cartográficos, la librería *UTM*.

De esta forma, solo nos queda asignar al vértice *hijo* el coste G que ya tuviese su vértice *padre* más el coste G aquí calculado. De igual forma para asignar el coste H al vértice *hijo* hemos de tener en cuenta la acumulación que arrastra el vértice *padre* al que le hemos de sumar el nuevo valor H aquí calculado.

Finalmente a nuestro vértice hijo hemos de asignarle el coste F , el cual recordemos que es la suma de G y H .

Luego, llegados a este punto, el vértice hijo pasado por parámetros tendrá asignados correctamente los costes necesarios para el cálculo del menor camino.

```
METODO(Vertex padre, Vertex hijo)

    /*Calculo H*/
    distancia_H <- getDistanciaH(Vertex hijo, Vertex destino)

    /*Calculo G*/
    distancia_G <- getDistanciaG(Vertex padre, Vertex hijo)

    /*Asigno valores*/
    hijo->setG(padre->getG() + distancia_G)
    hijo->setH(padre->getG() + distancia_H)
    hijo->setF(hijo->getG() + hijo->getH())

FIN METODO
```

Figura 65: Pseudocódigo del método 'setEcuacion'

5. recursivo(*Vertex destino*, *Vertex origen*) : *Vertex*

Método recursivo empleado para recorrer todos los vértices padres conectados desde el vértice *destino* —el cual es devuelto en la función ‘*algoritmoPathfinding*’—. Dentro de esta recursión iremos almacenando los vértices que componen el camino en nuestra variable PILA, esta inserción puesto que se comienza por el final del camino —*destino*— estará en orden inverso.

Luego el objetivo del método es dejar constancia del camino en la variable PILA, la cual es una pila del tipo ‘*Stack*’.

```

METODO(Vertex destino, Vertex origen)

    SI (destino == origen) ENTONCES
        PILA push <- destino
        Devolver destino

    SI NO
        PILA push <- destino
        Devolver METODO(destino->getPadre(), origen)

    FIN SI

FIN METODO

```

Figura 66: Pseudocódigo del método 'recursivo'

6. obtenerRecorrido(*Stack<Vertex> pila*) : *ArrayList<String>*

Método que recibe una pila con todos los vértices que describen el recorrido a tomar para ir desde el vértice *origen* hasta el vértice *destino*.

El método devolverá una lista de *Strings* descriptivos de cada una de las paradas que conforman el camino.

Finalmente nos encontramos con el último método que agrupa las llamadas a los métodos anteriores para facilitar la llamada al algoritmo.

```

COMIENZO(pila de vertices)

    resultado //Lista descriptiva de los vertices

    REPETIR
        /*Vamos tomando los monticulos de la pila
        e insertando en la lista resultado*/
        resultado <- getDescripcion( pila pop() )

    HASTA(pila este vacía)

    Devolver resultado

FIN

```

Figura 67: Pseudocódigo del método 'obtenerRecorrido'

7. **rutaIda() : ArrayList<String>**

Método en el que organizamos las distintas llamadas a los métodos ya definidos con el fin de proporcionarnos un listado de *Strings* descriptivos de las paradas que conforman el camino a seguir entre los vértices *origen* y *destino*.

Primero llamaremos al método '*algoritmoPathfinding*' el cual nos devolverá el vértice de *destino* apuntando al vértice *padre* el cual a su vez apuntará al resto de vértices *padres* hasta llegar al vértice *origen*.

Posteriormente a esa primera llamada, llamamos al método '*recursivo*' el cual transcribirá el camino a seguir almacenándolo en nuestra PILA los vértices que lo conforman.

Finalmente hacemos uso del método '*obtenerRecorrido*' el cual tomará la PILA de vértices y devolverá una lista de *Strings* descriptivos de las paradas que forman el camino.

8.2.2. Búsqueda en profundidad

DESCRIPCIÓN

Una **Búsqueda en profundidad** (en inglés DFS o Depth First Search) es un algoritmo que permite recorrer todos los nodos de un grafo o árbol de manera ordenada, pero no uniforme. Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto. Cuando ya no quedan más nodos que visitar en dicho camino, regresa, de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.

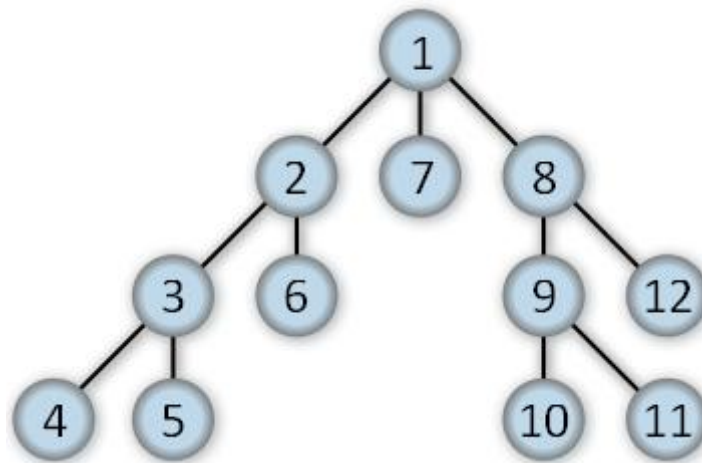


Figura 68: Ilustración de la búsqueda en profundidad

■ Características:

- Expande un camino al máximo partiendo de la raíz.
- Cuando no puede expandir más una rama, retrocede al nodo más cercano para continuar por ahí la expansión.
- Utiliza una estructura de datos temporal de tipo **PILA**(LIFO).
- Puede emplear *límite* de exploración o longitud máxima de exploración para no perpetuar un camino en profundidad. Por debajo de ese límite nunca se explorarían los nodos.
- Sin límite de exploración, en grafos infinitos no tendrá fin.
- En cada descenso a un nuevo nivel, almacena todos los sucesores de ese nivel(y los pone en la pila).

A continuación paso a mostrar el **pseudocódigo** para implementar con éxito este algoritmo:

1. Crear una lista de nodos llamada *ABIERTA* y asignarle el nodo raíz, que representa el estado inicial del problema planteado.
2. Hasta que se encuentre una meta o se devuelva fallo, realizar las siguientes acciones:
 - a. Si *ABIERTA* está vacía, terminar con fallo; en caso contrario, continuar.
 - b. Extraer el primer nodo de *ABIERTA* y denominarlo '*m*'.
 - c. Si la profundidad de '*m*' es igual a *lp*(límite de profundidad), regresar a *b*; en caso contrario, continuar.
 - d. Expandir '*m*' creando punteros hacia este nodo desde todos sus sucesores, de forma que pueda saberse cuál es su predecesor. Introducir dichos sucesores al principio de *ABIERTA* siguiendo un orden arbitrario (*La 'falta de orden' refleja el carácter no informado de este procedimiento*).
 - i. Si algún sucesor de '*m*' es meta, abandonar el proceso iterativo señalado **2**, devolviendo el camino de la solución, que se obtiene recorriendo los punteros de sus antepasados.
 - ii. Si algún sucesor de '*m*' se encuentra en un 'callejón sin salida', eliminarlo de *ABIERTA*. (Se continua el proceso iterativo en el paso *b*).

IMPLEMENTACIÓN

En este apartado será explicada con detalle nuestra implementación en lenguaje Java para el algoritmo *búsqueda en profundidad*, que no es más que una búsqueda de soluciones sin información del dominio.

Para tomar contacto con la estructura de nuestro programa adjunto una imagen descriptiva de los métodos y atributos empleados en el algoritmo, junto a su visibilidad.

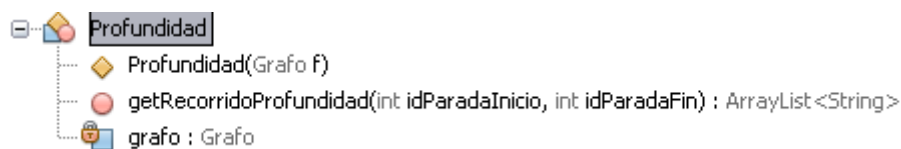


Figura 69: Atributos y métodos de la búsqueda en profundidad

Como puede observarse en la figura 67, únicamente disponemos de un atributo privado (*grafo*), el constructor de la clase y el método público principal que nos devolverá una lista de cadenas (*Strings*) descriptivas de las paradas visitadas en el camino, '*getRecorridoProfundidad*'.

Vamos a analizar más detenidamente cada uno de los miembros contenidos en esta implementación. Analizándolos de forma agrupada según su categoría, estas son:

▪ ***Atributos***

1. **grafo : Grafo**

Atributo privado de la clase '*Grafo*' que contiene cargado el grafo pertinente sobre el cual realizar nuestra búsqueda en profundidad.

▪ ***Métodos***

1. **Profundidad(Grafo f)**

Constructor de nuestra clase que recibe por parámetros el grafo e inicializa nuestro atributo.

2. **getRecorridoProfundidad(int idParadaInicio, int idParadaFin) : ArrayList<String>**

Método principal a consultar que nos proporcionará una lista descriptiva de las paradas que componen el camino entre las dos paradas que pasamos por parámetros *idParadaInicio* e *idParadaFin*.

Los pasos a seguir para su implementación son los siguientes:

1. Crear una pila de vértices y asignarle el vértice inicial, que representa el estado inicial del problema planteado.
2. Crear una lista booleana de procesados y marcar el vértice inicial como procesado.
3. Hasta que nuestra pila se encuentra vacía o encontremos la meta, realizar las siguientes acciones:
 - a. Extraer y eliminar de nuestra pila el vértice que se encuentra en la cima. Acto seguido lo añadimos a nuestra lista *Resultado* para indicar que ya ha sido visitado.
 - b. Si el vértice extraído de la pila es igual al vértice *meta*; nos salimos del bucle y devolvemos la lista *Resultado* que contiene los vértices visitados.
 - c. Si por el contrario el vértice extraído es distinto del vértice final o *meta*; obtenemos los vértices adyacentes del vértice sacado de la pila y por cada uno de ellos hacemos lo siguiente:
 - i. Si el vértice adyacente a tratar no ha sido procesado; almacenamos el vértice en nuestra pila y lo marcamos

como procesado. *Continuamos el proceso iterativo en el paso 3.*

Con objeto de facilitar la comprensión y visualización de la implementación, se adjunta un pseudocódigo descriptivo del método:

```
COMIENZO(vertex inicio, vertex fin)
    resultado //Lista de vertices a devolver al final
    encontrado <- false
    procesados[] <- false

    pila <- inserto vertex inicio
    procesados[inicio] <- true

    MIENTRAS (pila no vacia ^ encontrado == false) HACER
        //Saco de la pila un vértice
        u <- pila pop()
        //Añado a la lista resultado el vertex visitado 'u'
        resultado <- u

        SI (vertex u != vertex fin) ENTONCES
            adyacentes <- getAdyacentes(vertex u)

            REPETIR
                SI (procesados[adyacente] == false) ENTONCES
                    pila push <- adyacente
                    procesados[adyacente] <- true
                FIN SI

            HASTA(visitar todos los vértices adyacentes)

        SI NO
            encontrado <- true
        FIN SI

    FIN MIENTRAS

    Devolver resultado
FIN
```

Figura 70: Pseudocódigo del método 'getRecorridoProfundidad'

8.2.3. Algoritmo de Floyd

DESCRIPCIÓN

En informática, el algoritmo de Floyd-Warshall, descrito en 1959 por Bernard Roy, es un algoritmo de análisis sobre grafos para encontrar el camino mínimo en grafos dirigidos ponderados. El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución. El algoritmo de Floyd-Warshall es un ejemplo de programación dinámica.

El algoritmo de Floyd-Warshall compara todos los posibles caminos a través del grafo entre cada par de vértices. El algoritmo es capaz de hacer esto con sólo v^3 comparaciones (esto es notable considerando que puede haber hasta v^2 aristas en el grafo, y que cada combinación de aristas se prueba). Lo hace mejorando paulatinamente una estimación del camino más corto entre dos vértices, hasta que se sabe que la estimación es óptima.

Como ya se ha mencionado, se basa en el esquema de ‘programación dinámica’. Este método utiliza una tabla para ir almacenando los resultados correspondientes a instancias más sencillas del problema a resolver.

Sea un G un grafo dirigido y ponderado. Suponemos que los vértices están numerados de 1 a n . En este caso, utilizamos una matriz con los pesos de las aristas, de tal forma que cada elemento de la matriz representa el peso c_{ij} asociado a la arista (v_i, v_j) . Utilizaremos $c_{ij} = \infty$ para representar que no existe arista entre los vértices v_i y v_j .

La idea principal consiste en encontrar una matriz D de $n \times n$ elementos, de tal forma que cada elemento D_{ij} sea el coste mínimo de los caminos entre v_i y v_j .

Para calcular el camino de coste mínimo entre los vértices i y j podemos considerar dos posibilidades: no pasar por el vértice k , en cuyo caso tendremos que calcular el mejor camino con el resto de los vértices ($D_{k-1}[i,j]$), o bien pasar por el vértice k , en cuyo caso tendremos que obtener caminos que vayan de i a k y de k a j .

El algoritmo comienza con una inicialización natural de D (D_0) y se genera iterativamente la secuencia de matrices D_1, D_2, \dots, D_n , cuyos elementos tienen el siguiente significado:

- $D_0[i,j] = c_{ij}$, peso asociado a la arista desde el vértice i al vértice j .
- $D_1[i,j] = \min(D_0[i,j], D_0[i,1] + D_0[1,j])$. Menor de los costes entre el anterior camino desde i hasta j y la suma de los costes de caminos desde i hasta 1 y 1 hasta j .
- $D_2[i,j] = \min(D_1[i,j], D_1[i,2] + D_1[2,j])$. Menor de los costes entre el anterior camino desde i hasta j y la suma de los costes de caminos desde i hasta 2 y 2 hasta j .

- $D_n[i,j] = \min(D_{n-1}[i,j], D_{n-1}[i,n] + D_{n-1}[n,j])$. Menor de los costes entre el anterior camino desde i hasta j y la suma de los costes de caminos desde i hasta n y n hasta j .

Para cada vértice sería conveniente almacenar el índice del último vértice que ha conseguido que el camino sea mínimo desde v_i hasta v_j . Para ello, se utiliza una matriz de vértices según el siguiente criterio:

- $A(v_i, v_j) = 0$, si no hay camino de v_i a v_j , o hay una arista entre los vértices.
- $A(v_i, v_j) = v_k$, si v_j es accesible desde v_i a través de v_k en el camino mínimo entre ambos vértices.

IMPLEMENTACIÓN

En este apartado será explicada con detalle nuestra implementación en lenguaje Java para el algoritmo *de Floyd*.

Para tomar contacto con la estructura de nuestro programa adjunto una imagen descriptiva de los métodos y atributos empleados en el algoritmo, junto a su visibilidad.

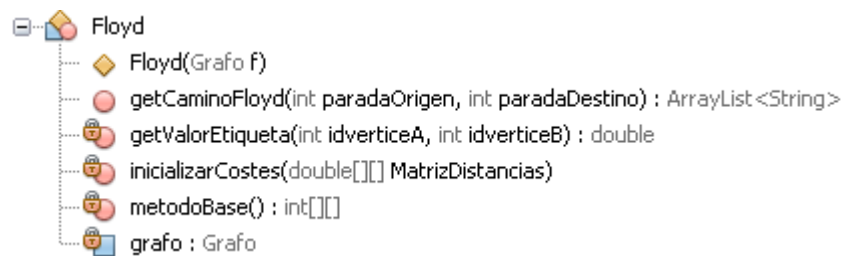


Figura 71: Atributos y métodos del algoritmo de Floyd

Podemos observar en la figura anterior que nuestra clase se compone de 5 métodos, 3 de los cuales son privados y los restantes públicos, y un atributo privado que contendrá cargada la estructura de nuestro grafo.

Vamos a analizar más detenidamente cada uno de los miembros contenidos en esta implementación. Analizándolos de forma agrupada según su categoría, estas son:

▪ *Atributos*

1. `grafo : Grafo`

Atributo privado de la clase '*Grafo*' que contiene cargado el grafo pertinente sobre el cual realizar nuestra búsqueda en profundidad.

▪ **Métodos**

1. **Floyd(Grafo f)**

Constructor de nuestra clase que recibe por parámetros el grafo e inicializa nuestro atributo.

2. **metodoBase() : int[][]**

Método privado que implementa el pseudocódigo proporcionado en la definición dada sobre el algoritmo Floyd y que devuelve una matriz bidimensional que contiene el camino mínimo entre cada par de vértices. De tal forma que, si obtuviésemos una matriz tal que:

	1	2	3	4	5
1	-	1	1	2	4
2	2	-	4	2	4
3	3	4	-	3	4
4	2	4	4	-	4
5	2	4	4	5	-

Estamos obteniendo toda la información necesaria para determinar la ruta más corta entre dos vértices cualesquiera del grafo.

Para determinar la ruta asociada del camino mínimo entre el nodo 1 y el nodo 5 haremos lo siguiente:

- Consultamos $D[1,5]=4$, por tanto el nodo predecesor al 5 es el 4, es decir, $4 \rightarrow 5$.
- Consultamos $D[1,4]=2$, por tanto el nodo predecesor al 4 es el 2, es decir, $2 \rightarrow 4 \rightarrow 5$.
- Consultamos $D[1,2]=1$, por tanto el nodo predecesor al 2 es el 1, es decir, $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$, y así ya tenemos la ruta completa.

La determinación del camino es una tarea de la que no se encarga el método aquí definido, es una tarea delegada en el método '*getCaminoFloyd*' que se detallará más adelante.

A continuación pasamos a dejar el pseudocódigo del método:

```

constante n = G.vertices().cardinalidad()
tipo int [n] [n] MatrizDistancias
    int [n] [n] MatrizCamino

COMIENZO(Grafo G, MatrizCamino A)

    var int i, j, k; Matrizdistancias D fvar

    /*Almacena en D0 los pesos asociados a las aristas de G.
    El camino mínimo de un vértice a sí mismo se considera 0.*/
    inicializarCostes(G, D)

    para i=1 hasta n hacer
        para j=1 hasta n hacer
            A[i,j] = 0
        fpara
    fpara

    //El índice k indica el subíndice de la matriz D
    //que se está generando
    para k=1 hasta n hacer
        para i=1 hasta n hacer //El índice i indica la fila
            para j=1 hasta n hacer // El índice j indica la columna
                si (D[i,k] + D[k,j] < D[i,j]) entonces
                    D[i,j] = D[i,k] + D[k,j];
                    A[i,j] = k;
                fsi
            fpara
        fpara
    fpara
    devuelve D
FIN

```

Figura 72: Pseudocódigo del método 'metodoBase'

El único método del cual depende este '*metodoBase*' es '*inicializarCostes*'.

3. *inicializarCostes (double[][] MatrizDistancias) : void*

Método privado encargado de inicializar la matriz bidimensional '*MatrizDistancias*' con las distancias entre cada par de vértices.

De esta forma, el valor C_{ij} representa el coste de ir desde el vértice i al vértice j . En caso de no existir un arco entre ambos, el valor C_{ij} será infinito. Y si el vértice i es igual al vértice j el valor C_{ij} será 0.

Cada uno de los vértices que forman parte del grafo ha sido numerado de 0 hasta *número de vértices - 1*. Por lo que no existe ningún tipo de problema en

la identificación de cada uno de estos y la localización de los pesos entre cada par de vértices que forman una arista.

A continuación se expone el pseudocódigo pertinente a la implementación de dicho método.

```
COMIENZO( Double[][] MatrizDistancias)

var
    n = grafo.getNvertices()
    Vertice [] adyacentes
    Vertice temp
fvar

para i=0 hasta n hacer
    adyacentes = grafo.getAdyacentes(i)

    para j=0 hasta n hacer
        temp = grafo.getVertice(j)

        si (i==j) entonces
            MatrizDistancias[i][j] = 0
        si no si (adyacentes contiene vertice temp) entonces
            MatrizDistancias[i][j] = getValorEtiqueta(i,j)
        si no
            MatrizDistancias[i][j] = Infinito
        fsi
    fpara
fpara

FIN
```

Figura 73: Pseudocódigo del método 'inicializarCostes'

El único método del cual depende es '*getValorEtiqueta*'

4. *getValorEtiqueta(int idverticeA, int idverticeB) : double*

Método privado cuya cometido es obtener el peso/valor/etiquetado de la arista formada por los identificadores de los vértices pasados por parámetro (*idverticeA*, *idverticeB*).

Como ya se ha comentado, en la creación del grafo, los vértices han sido etiquetados desde 0 hasta *número vértices* - 1. Así pues, en esta función hemos de:

1. Obtener los vértices que forman la arista, en función de esos id.
2. Buscar entre las aristas que tiene el grafo, aquella arista cuyo origen y destino sean los vértices ya indicados. Una vez encontrada la arista, devolver su peso.

A continuación pasamos a poner un pseudocódigo descriptivo del método:

```

COMIENZO(idverticeA, idverticeB)
    var
        Vertice verticeorigen, verticedestino
        double valor = 0.0
        int Naristas
    fvar

    verticeorigen = grafo.getVertice(idverticeA)
    verticedestino = grafo.getVertice(idverticeB)

    para i=0 hasta Naristas ^ valor=0.0 hacer
        si (grafo.getDestino() == verticedestino ^
            grafo.getOrigen() == verticeorigen) entonces
            valor = grafo.getEtiqueta()
        fsi
    fpara
FIN

```

Figura 74: Pseudocódigo del método 'getValorEtiqueta'

5. **getCaminoFloyd(int paradaOrigen, int paradaDestino) : ArrayList<String>**

Método público básico encargado de devolver el camino formado entre los dos identificadores pasados por parámetros. Identificadores que representan a una parada/vértice.

Inicialmente obtendremos los vértices correspondientes a los identificadores pasados por parámetros para pasar a llamar al método principal y que contiene el algoritmo de Floyd. Llamamos a 'metodoBase' que nos devolverá una matriz de distancias con los mínimos caminos ya definidos entre cada par de vértices.

El siguiente paso a realizar consiste en recorrer la matriz de distancias obtenida gracias a 'metodoBase' en orden inverso, esto es, desde la columna final (*destino*) hasta llegar al vértice destino. Algo tal que:

	1	2	3	4	5
1	-	1	1	2	4
2	2	-	4	2	4
3	3	4	-	3	4
4	2	4	4	-	4
5	2	4	4	5	-

Estamos obteniendo toda la información necesaria para determinar la ruta más corta entre dos vértices cualesquiera del grafo.

Para determinar la ruta asociada del camino mínimo entre el nodo 1 y el nodo 5 haremos lo siguiente:

- Consultamos $D[1,5]=4$, por tanto el nodo predecesor al 5 es el 4, es decir, $4 \rightarrow 5$.

- Consultamos $D[1,4]=2$, por tanto el nodo predecesor al 4 es el 2, es decir, $2 \rightarrow 4 \rightarrow 5$.
- Consultamos $D[1,2]=1$, por tanto el nodo predecesor al 2 es el 1, es decir, $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$, y así ya tenemos la ruta completa.

Es decir, habremos ido introduciendo en una pila cada nodo/vértice visitado para después recorrer dicha pila y reconstruir finalmente el camino que será devuelto.

A continuación se indica el pseudocódigo perteneciente a este método:

```

COMIENZO(paradaOrigen, paradaDestino)
var
    int a,b,pos
    int[][] estrategia
    Stack<Integer> pila
    Lista<String> resultado
fvar

a = grafo.getVertice(paradaOrigen)
b = grafo.getVertice(paradaDestino)
estrategia = metodoBase()

/*Me situo en la fila correspondiente*/
pos = estrategia[a][b]
pila push() = b
pila push() = pos

hacer
    pos = estrategia[a][pos]
    pila push() = pos
mientras( pos != a)

hacer
    resultado add = getDescripcion(pila pop())
mientras(pila no vacia)

Devolver resultado
FIN

```

Figura 75: Pseudocódigo del método 'getCaminoFloyd'

8.3. Rendimiento

La capacidad de lograr el mejor camino deseado, con el mínimo de recursos (paradas/vértices) posibles, entre dos paradas cualesquiera es un factor determinante a la hora de decantarnos por la elección de uno, u otro algoritmo.

Por lo que, después de haber detallado los algoritmos de búsqueda a emplear en nuestro proyecto, hemos de establecer una comparativa que nos permita decantarnos por uno de ellos. Hemos de medir el rendimiento de los 3 algoritmos ya vistos:

1. **Algoritmo A***
2. **Búsqueda en profundidad**
3. **Algoritmo de Floyd**

El rendimiento será medido sobre 2 variables fundamentales:

- Número de paradas en que se realiza la ruta/camino.
- Cantidad de memoria empleada por éste.

La **cantidad de memoria empleada** por la ejecución de cada algoritmo será medida gracias a NetBeans Profiler. Es una característica opcional del entorno de desarrollo NetBeans integrado(IDE). NetBeans Profiler es una herramienta poderosa que proporciona información importante sobre el comportamiento en tiempo de ejecución de una aplicación, rendimiento de la CPU y uso de la memoria.

A la hora de la medición de la cantidad de memoria usada, obtendremos una imagen descriptiva que nos mostrará la cantidad de memoria utilizada por nuestro algoritmo de búsqueda. Por lo que en los gráficos que hacen alusión a la cantidad de memoria utilizada podremos apreciar una zona del gráfico color violeta, este color nos estará indicando la cantidad de espacio de almacenamiento dinámico en uso. Por lo que el color violeta será el único en el que deberemos de prestar atención.

Mencionar, que la zona del gráfico rosada nos indica el tamaño asignado por la pila de la máquina virtual de Java. Pero no influye para nada en nuestro estudio de rendimiento.

Estableceremos dos versiones sobre las que medir estas variables:

- **Versión de una línea**

Esta versión contempla una línea elegida de entre las 33 existentes. Por lo que contendrá la formación de un grafo creado gracias a los trayectos que contiene dicha línea.

- **Versión de todas las líneas**

Versión que contempla un grafo formado por todas las líneas de las que consta el servicio de autobús de Gijón. 33 líneas con sus correspondientes trayectos.

Realmente no existe diferencia alguna entre versiones. Es decir, establecer una división entre cargar un grafo con una sola línea y cargar un grafo con todas las líneas no es una característica diferenciadora que nos haga decantarnos por un algoritmo de búsqueda u otro. Pero ayuda a ver el comportamiento de ellos al tener más o menos carga de trabajo (más líneas vs menos líneas a analizar).

8.3.1. Versión de 1 línea

La versión de una línea implica la generación de un grafo que contenga todos los trayectos pertenecientes a dicha línea. Recordar que cada trayecto está formado por una serie de paradas dispuestas en un orden específico que lo conforma.

Para ejemplificar lo explicado acerca de esta versión vamos a tomar la línea número 1:

#	idLinea	idTipoLinea	Descripcion
1	1	1	CERILLERO - HOSPITAL DE CABUEÑES

Figura 76: Consulta de línea - version 1 línea

Esta línea contiene un total de 4 trayectos, tal que:

#	idTrayecto	idLinea	Descripcion	Sentido	idExtremos
1	1	1	CERILLERO - HOSPITAL DE CABUEÑES	Ida	1
2	2	1	HOSPITAL DE CABUEÑES - CERILLERO	Vuelta	2
3	6	1	HOSPITAL DE CABUEÑES - CERILLERO	Vuelta	3
4	7	1	HOSPITAL CABUEÑES-CERILLERO (POR SE...	Vuelta	4

Figura 77: Consulta de trayecto - versión 1 línea

A su vez, cada trayecto posee un conjunto de paradas dispuestas de un cierto orden que conforma el camino:

#	idParada	idLinea	idTrayecto	Orden	Descripcion
1		180	1	1	0 CAMINO DE RUBÍN
2		181	1	1	1 INEM
3		141	1	1	2 GRAN CAPITÁN
4		136	1	1	3 PRAO DEL CUBANO
5		137	1	1	4 URUGUAY

Figura 78: Consulta de parada - versión 1 línea

En resumen, el grafo sobre el cual aplicaremos cada uno de los 3 algoritmos es el grafo de la línea 1.

El camino a encontrar será el que va desde la parada #34(Hospital de Cabueñes) hasta la parada #707(Rastro).

A continuación será mostrado el camino tomado en cada algoritmo, n° de paradas y cantidad memoria utilizada.

1. Algoritmo A*

Al realizar la ejecución del algoritmo, el camino a seguir que nos arroja es el siguiente camino óptimo:

[HOSPITAL DE CABUEÑES, TANATORIO, UNIVERSIDAD LABORAL, JARDÍN BOTÁNICO, INTRA, CHALETs, RASTRO]

El camino a seguir está compuesto por 7 paradas y la máxima cantidad de memoria utilizada es la siguiente:

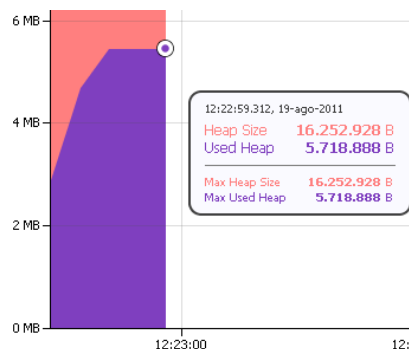


Figura 79: Medición de memoria - A* - ver 1 línea

La memoria utilizada es de 5.718.888 Bytes o lo que es lo mismo 5.45396 Megabytes

2. Búsqueda en profundidad

Al realizar la ejecución de nuestro algoritmo obtenemos el siguiente camino:

[HOSPITAL DE CABUEÑES, TANATORIO, UNIVERSIDAD LABORAL, JARDÍN BOTÁNICO, INTRA, ESCUELA DE MARINA, EL MOLINON, LAS MESTAS, EL PARQUE, PLAZA DE TOROS, PARQUE COCHERAS, CONTINENTAL, LOS CAMPOS, BEGOÑA, PLAZA EUROPA, HUMEDAL (LADO SEMÁFORO), PEDRO DURO, PLAYA DE PONIENTE, ACUARIO, COMISARIA, PLAZA DE LA HABANA, CORTES DE CÁDIZ, MÓSTOLES, SANTA OLAYA, CUATRO CAMINOS, ATENEO, URUGUAY, FÁTIMA, MARTÍN, BARROS, CAMINO DE RUBÍN, INEM, GRAN CAPITÁN, PRAO DEL CUBANO, URUGUAY, CASA DEL MAR, CUATRO CAMINOS, SANTA OLAYA, DOS DE MAYO, PLAZA DE LA HABANA, ESTACIÓN DEL NORTE, MUSEO DEL FERROCARRIL, PEDRO DURO, F.E.V.E., GOTA DE LECHE (LADO MAGNUS BLIKSTAD), BEGOÑA, CABRALES, RECONQUISTA, RAMÓN Y CAJAL, CODEMA, PLAZA DE TOROS, LA ASUNCIÓN, LAS MESTAS, LA GUÍA, CHALETs, CTRA. VILLAVICIOSA (ESCUELA DE LA MARINA), INTRA, JARDÍN BOTÁNICO, PARQUE TECNOLÓGICO, TANATORIO, CHALETs, **RASTRO**]

En total son 62 paradas de las que se compone nuestro camino y la cantidad máxima de memoria utilizada es:

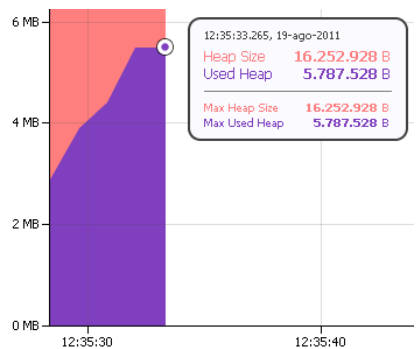


Figura 80: Medición de memoria - Profundidad - ver 1 línea

La memoria utilizada es de 5.787.528 Bytes o lo que es lo mismo 5.51942 Megabytes

3. Algoritmo de Floyd

Al poner en ejecución nuestro algoritmo obtenemos el siguiente camino óptimo:

[CAMINO DE RUBÍN, INEM, GRAN CAPITÁN, PRAO DEL CUBANO, URUGUAY, CASA DEL MAR, CUATRO CAMINOS]

En total consta de 7 paradas, donde la memoria utilizada es la siguiente:

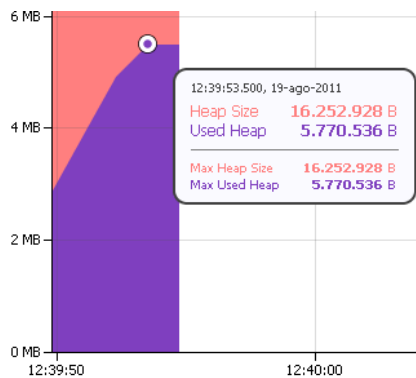


Figura 81: Medición de memoria - Floyd - ver 1 línea

La memoria usada es de 5.770.536 Bytes o lo que es lo mismo 5.50321 Megabytes

A continuación podremos observar dos gráficas en la que se encuentra cada uno de los algoritmos y el valor de las variables en base a las cuales realizar la comparativa. En el podremos apreciar una comparativa entre nº de paradas y memoria utilizada:

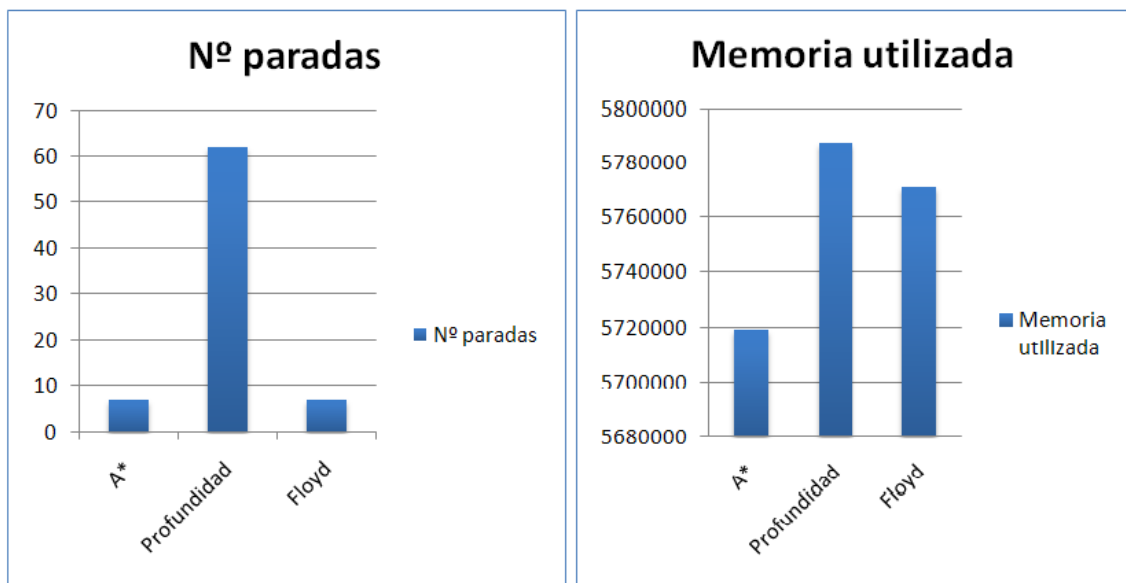


Figura 82: Paradas vs Memoria - versión 1 línea

Podemos observar que tanto el algoritmo A* como el algoritmo de Floyd son los más eficientes en cuanto a número de paradas recorridas a lo largo del camino. Ambos algoritmos tienen un total de 7 paradas, lo que supone un 88,7% de diferencia con respecto al algoritmo de profundidad que pasa por 62 paradas.

Con respecto a la memoria utilizada por estos algoritmos, puede observarse que el algoritmo A* es el que menos recursos utiliza, seguido del algoritmo de Floyd y finalizando nuevamente en el algoritmo de profundidad como algoritmo menos eficiente.

Luego, teniendo considerando nº de paradas y memoria utilizada, el algoritmo más eficiente es el A*, seguido muy a la par por el algoritmo de Floyd.

El algoritmo más deficiente es por supuesto el algoritmo en profundidad.

8.3.2. Versión de todas las líneas

En esta versión el grafo sobre el cual se pretende hallar el camino, integra todas las líneas de las que consta la red de autobuses de Gijón (33 líneas).

Hemos de recordar que cada línea lleva implícita una serie de trayectos, los cuales a su vez están compuestos por una serie de paradas dispuestas en un orden determinado.

Al cargar nuestro grafo completo, dispondremos de 605 vértices/paradas y 824 aristas (par de vértices).

El camino a buscar para esta explicación será el mismo que para la versión de una línea. El camino a encontrar será el que va desde la parada #34(Hospital de Cabueñes) hasta la parada #707(Rastro).

Por lo tanto, hemos de buscar un camino –lo más optimo posible- entre las paradas #34 y #707 ya citadas. Para esto, haremos uso de los 3 algoritmos de búsquedas donde compararemos el rendimiento de cada uno de estos en función de las variables de rendimiento:

- Nº de paradas
- Cantidad de memoria utilizada

A continuación, comenzaremos el pequeño análisis de cada uno de los algoritmos:

1. Algoritmo A*

En este algoritmo, al realizar su ejecución para obtener un camino entre las dos paradas #34(Hospital de Cabueñes) y #707(Rastro), obtenemos el siguiente camino:

[*HOSPITAL DE CABUEÑES*, TANATORIO, CANDENAL, INTRA, ESCUELA DE MARINA, *RASTRO*]

Luego, el nº de paradas en que se realiza nuestro camino es de 6. Además la cantidad de memoria empleada en la ejecución de este algoritmo es la siguiente:

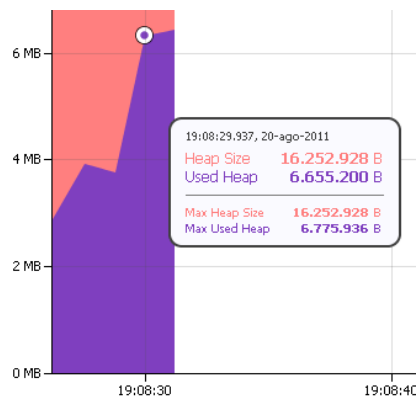


Figura 83: Medición de memoria – A* - ver todas

La cantidad de memoria utilizada es de 6.655.200 Bytes.

2. Búsqueda en profundidad

En la búsqueda en profundidad, el camino a buscar entre las dos paradas ya citadas es el siguiente:

[*HOSPITAL DE CABUEÑES*, TANATORIO, CANDENAL, INTRA, ESCUELA DE MARINA, *RASTRO*]

Luego, el n° de paradas en que se realiza nuestro camino vuelve a ser de 6, al igual que en el algoritmo A*. Además la cantidad de memoria empleada en la ejecución de este algoritmo es la siguiente:

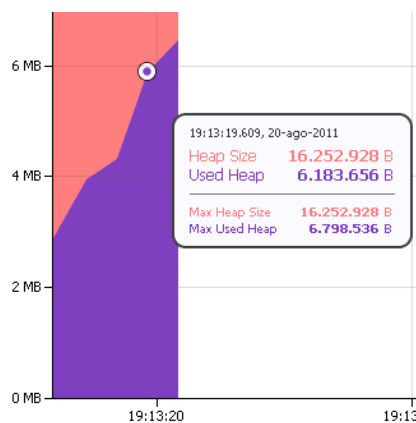


Figura 84: Medición de memoria - Profundidad - ver todas

Puede observarse que la cantidad de memoria utilizada es de 6.138.656 Bytes

3. Algoritmo de Floyd

Para nuestro algoritmo de Floyd, el camino que arroja tras realizar la búsqueda en el grafo al completo es el siguiente:

[*HOSPITAL DE CABUEÑES*, TANATORIO, CANDENAL, INTRA, ESCUELA DE MARINA, *RASTRO*]

Volvemos a obtener el mejor camino posible con un total de 6 paradas a recorrer. Por último destacar la cantidad de memoria utilizada en este algoritmo:

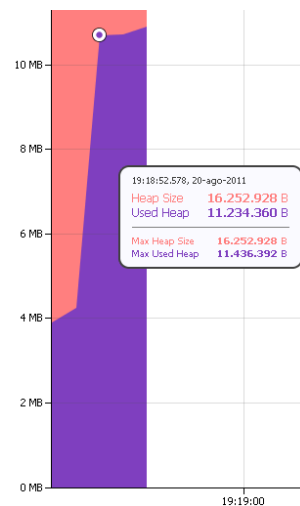


Figura 85: Medición de memoria - Floyd - ver todas

Este algoritmo es con diferencia el que más memoria consume, llegando casi al doble de memoria utilizada por los dos algoritmos anteriores. 11.234.360 Bytes.

A continuación podemos observar una tabla comparativa en la que apreciar las diferencias entre cada uno de los algoritmos para el grafo al completo.

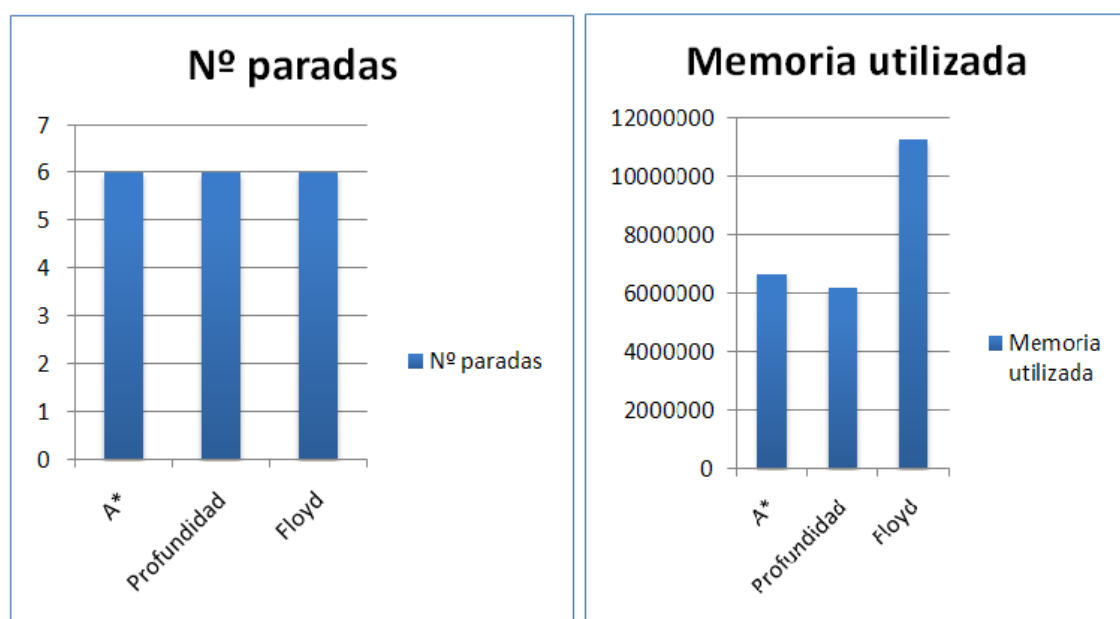


Figura 86: Parada vs Memoria - versión todas las líneas

La única diferencia significativa es la cantidad de memoria utilizada por cada uno de los algoritmos. En la comparativa, podemos apreciar como es la búsqueda en profundidad la más eficiente, seguida de cerca por el algoritmo A*.

Puede destacarse el algoritmo de Floyd como el peor a utilizar en la utilización de grafos al completo.

8.3.3. Conclusión

En términos generales y atendiendo a los resultados obtenidos tanto en la versión de 1 línea, como en la versión de todas las líneas, el algoritmo A* es el más eficiente tanto por nº de paradas como por memoria. Así pues, será este algoritmo el que usemos para buscar las líneas existentes entre dos dadas. Además habremos de aplicar dicho algoritmo sobre la creación de un grafo que contemple todas las líneas de la red de autobuses.

Por el contrario, el peor algoritmo atendiendo a las dos versiones, es el de búsqueda en profundidad ya que la gran cantidad de paradas que obtiene en la versión de 1 línea, respecto del resto de búsquedas, es motivo suficiente para no tomarla en cuenta.

Tanto con el algoritmo A*, como el algoritmo de Floyd, las paradas que obtenemos son consecutivas y existe una arista entre cada par de paradas/vértices que arrojan estos algoritmos. Con la búsqueda en profundidad, esto no se da. La lista de paradas/vértices que arroja no tiene porque ser consecutiva ya que es una búsqueda que cuando explora una rama (llegando a su final) en la que no se encuentra la parada de destino automáticamente salta a otra rama en la que seguir explorando. Este salto dado provoca que exista una ruptura entre vértices y no refleje ninguna arista ya que realmente no existirá ningún tipo de conexión entre esos vértices no consecutivos.

La búsqueda en profundidad nos proporcionará una ruta (refiriéndonos a la lista de paradas) a seguir en los grafos que tengan cargados una única línea. Para los grafos completos (todas las líneas cargadas), funciona bien un pequeño porcentaje de los casos. Puesto que al saberse como una búsqueda exploratoria sin información alguna, el algoritmo de profundidad proporcionará vértices sin seguir el orden que realmente siguen las paradas/vértices en el grafo.

8.4. Ruta entre dos paradas

Con anterioridad hemos tratado el registro de paradas de que se compone el camino formado entre las paradas de origen y destino proporcionadas por un usuario. Recordemos que dichas paradas, origen y destino, son insertadas en el formulario del fichero php '*GoogleMaps_planificador.php*' haciendo uso de las sugerencias (descripciones de paradas) para minimizar posibles equivocaciones a la hora de escribir y todo gracias a la biblioteca de Javascript, jQuery.

Ahora, nos encontramos ante la necesidad de indicar las líneas/trayectos que ha de tomar el usuario entre cada par de paradas que conforman su camino, de esta forma el usuario se encuentra ante un camino guiado.

8.4.1. Descripción

Con objeto de mantener un registro de las líneas/trayectos que ha de tomar el usuario a lo largo de su camino, cada una de las aristas que forman parte del grafo – representativo de la red de autobuses de Gijón- ha de llevar etiquetadas las líneas/trayectos a las que pertenecen.

Sin embargo, una arista puede pertenecer a varias líneas/trayectos a la vez. Esto podemos observarlo en el siguiente camino, en el cual, al preguntar por el camino entre la parada #34 y la #38 obtenemos lo siguiente:

[*HOSPITAL DE CABUEÑES*, [1 | 2, 1 | 6, 1 | 7], *TANATORIO*, [1 | 2, 1 | 6, 1 | 7], *UNIVERSIDAD LABORAL*, [1 | 2, 1 | 6, 1 | 7], *JARDÍN BOTÁNICO*]

La ruta para ir desde la parada 'Hospital de cabueñes' hasta la parada 'Jardín botánico' es el siguiente:

Hospital – Tanatorio – Universidad Laboral – Jardín botánico.

Entre cada par de paradas, se nos presentan varias opciones –validas todas ellas – para poder realizar nuestro trayecto. Estas opciones vienen remarcadas por un identificador tal que '*a|b*' donde '*a*' es la línea a tomar y '*b*' el trayecto dentro de dicha línea.

Luego, el hecho de que una arista quede etiquetada por varias líneas/trayectos nos está indicando que disponemos de varias alternativas/combinaciones para realizar el tramo del que se compone dicha arista.

Sí de una arista a otra cambiamos de línea/trayecto, estaremos realizando un transbordo. Lo ideal en todo camino es realizar el mínimo número de transbordos posibles hasta llegar a nuestro destino. Para ello, por cada arista hemos de intentar mantenernos en la misma línea/trayecto –siempre que exista elección posible-,

realizando transbordos en las aristas que no sigan la misma línea/trayecto en la que hasta el momento nos encontrábamos.

Con el propósito de entender mejor las transiciones a realizar entre cada par de paradas, imaginemos el siguiente ejemplo, en el cual nos encontramos con 5 paradas.

Numeradas del 1 al 5. Entre cada par de paradas encontramos uno o varias alternativas para realizar la transición de una parada a otra:

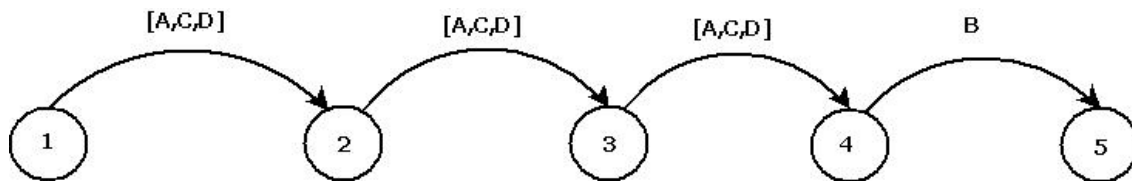


Figura 87: Múltiples opciones - transiciones

Hemos de conseguir que entre cada par de aristas las transiciones sean iguales. Cuando entre dos aristas encontremos que sus transiciones son distintas nos encontraremos ante un transbordo.

Las rutas posibles en la anterior imagen son las siguientes:

1. Desde la parada #1, hasta la parada #4 elegimos como elemento de transición A. Una vez en la parada #4, realizamos un transbordo hacia la transición B para llegar así a la parada #5.
2. Desde la parada #1, hasta la parada #4 elegimos como elemento de transición C. Una vez en la parada #4, realizamos un transbordo hacia la transición B para llegar así a la parada #5.
3. Desde la parada #1, hasta la parada #4 elegimos como elemento de transición D. Una vez en la parada #4, realizamos un transbordo hacia la transición B para llegar así a la parada #5.

En la siguiente imagen podemos observar que la transición elegida es la A y en el momento de realizar el cambio/transbordo, tomamos la transición B.

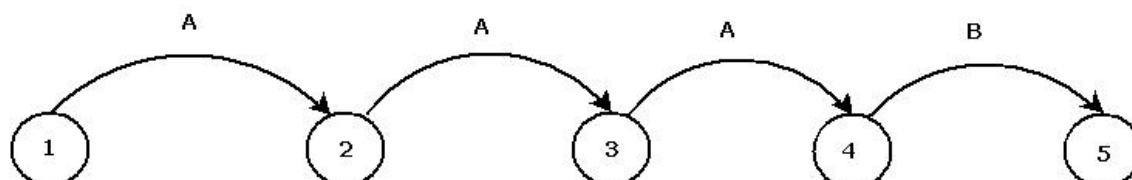


Figura 88: Opción única - transiciones

8.4.2. Arquitectura lógica

Para hacer posible la obtención de una ruta en la que obtengamos las paradas, líneas y transbordos que definen el camino a buscar entre dos puntos, después de haber conseguido la lista de paradas/vértices que nos proporciona los algoritmos de búsqueda hacemos uso de las siguientes clases, pertenecientes a nuestro proyecto ‘Grafo’:

- **Bloque**

Clase que agrupa cada una de las paradas resultantes de una búsqueda (no importa el algoritmo de búsqueda a usar, es aplicable a todos). Éste agrupamiento constituye un bloque para reunir paradas pertenecientes a una misma línea de un mismo trayecto. Por lo que cada bloque constituirá un estado (línea + trayecto) y el cambio entre éstos supondrá un transbordo o cambio de línea.

- **Transicion**

Ésta clase hace uso directo de la clase ‘Bloque’. Su tarea principal es generar una serie de ‘Bloques’ a partir de una búsqueda determinada (A*, Profundidad o Floyd), para posteriormente transcribir éstos bloques a cadena de texto entendible por el usuario.

Al preguntar por la ruta entre las paradas #34 (Hospital de Cabueñes) y #30(Jardín Botánico), nuestro algoritmo A*, nos devolvería la siguiente lista de paradas que forman parte de nuestra ruta:

Hospital – Tanatorio – ... – Intra– Jardín botánico.

Después de aplicar las transiciones vistas en este apartado, valiéndonos de las clases mencionadas, obtendremos la siguiente cadena que nos proporciona información más comprensible:

Línea 15 | 10 : **HOSPITAL DE CABUEÑES**, TANATORIO, CANDENAL, INTRA,
ESCUELA DE MARINA. Km’s: 1

[TRANSBORDO]

Línea 4 | 12 : ESCUELA DE MARINA, VIÑAO, VIESQUES, FEJOO, ESTACIÓN
DE SERVICIO VIESQUES. Km’s: 9

[TRANSBORDO]

Línea 612 | 2 : ESTACIÓN DE SERVICIO VIESQUES, LA ASUNCIÓN, LAS
MESTAS. Km’s: 13

[TRANSBORDO]

Línea 90 | 1 : LAS MESTAS, CHALETS, CTRA. VILLAVICIOSA (ESCUELA DE LA MARINA). Km's: 15

[TRANSBORDO]

Línea 1 | 1 : CTRA. VILLAVICIOSA (ESCUELA DE LA MARINA), INTRA, **JARDÍN BOTÁNICO**. Km's: 15

Es decir, para ir desde la parada *Hospital de Cabueñes* hasta la parada *Jardín Botánico*, hemos de montarnos en la línea número 15, trayecto 10, para cambiarnos en la parada *Escuela de Marina* a la línea 4, trayecto 12, donde realizamos transbordo en la parada *Estación de Servicio Viesques ...etc.* Y así hasta llegar a *Jardín Botánico*. Durante estos cambios de líneas o transbordos hemos recorrido una serie de paradas por cada una de las líneas, paradas que se encuentran citadas junto a cada una de dichas líneas.

De esta forma obtenemos las indicaciones precisas para ir desde una parada de origen, a una parada de destino.

8.5. Puente PHP/Java

La consecución de la cadena de texto que nos muestra la ruta a seguir entre dos paradas es obtenida gracias al uso de nuestro proyecto java, del que ya hemos hablado anteriormente. Pero lo que se pretende en éste proyecto es conseguir que esa misma cadena de texto sea accesible desde el fichero '*GoogleMaps_GRAFOS_RUTA.php*'.

Dicho fichero '*.php*' es el responsable de establecer conexión con la información proporcionada por las clases java ya citadas. Pero, desde un '*.php*' no se puede ejecutar código Java. O al menos, eso creíamos hasta descubrir PHP/Java Bridge.

PHP/Java Bridge es una potente herramienta que nos permite instanciar clases Java y ejecutar su código desde PHP, y viceversa. Esto lo hace de manera muy sencilla. Por lo que, desde nuestro script '*.php*' instanciamos las clases Java necesarias para encontrar la ruta entre dos paradas de la red de autobuses de Gijón.

Para hacer operativa esta herramienta en nuestro proyecto, hemos de bajarnos el .war desde la web de PHP/Java Bridge. Una vez descargado este .war, lo descomprimos/desplegamos en el directorio '*webapps*' del servidor Apache, generando una carpeta '*JavaBridge*'. Dentro de este .war ya descomprimido podemos encontrar una carpeta '*WEB-INF*', dentro de ella debemos crear una carpeta llamada '*classes*' y en su interior colocar los '*.class*' de las clases intervinientes en la generación de la ruta entre las dos paradas de la red de autobuses.

Básicamente, esta sería la estructura que presentan el puente PHP/Java. A un extremo situamos los '*.class*' de las clases Java que deseamos ejecutar en un script php, y a otro extremo situamos nuestro script php desde el cual realizaremos las instanciaciones necesarias para hacer uso de esas clases Java a través de sus '*.class*'.

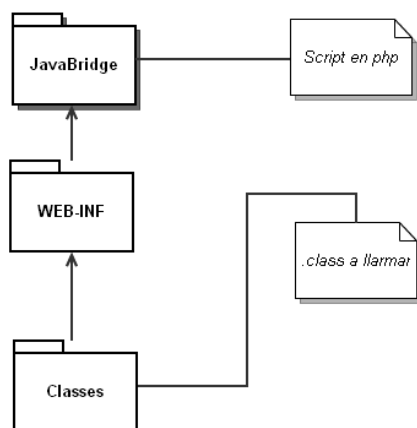


Figura 89: Esquema PHP/JavaBridge

9. Conclusiones y trabajos futuros

A lo largo de los capítulos de esta memoria hemos descrito el desarrollo de las 3 componentes básicas que posibilitan la reutilización de la información procedente del servicio web *busGijón*. Estas 3 componentes, elementos u objetivos básicos que integran nuestro proyecto, se encuentran enmarcados en:

- **Programa recuperador de datos** cuya principal misión es la de consumir, depurar y hacer persistir la información leída de *busGijón* en una base de datos.
- **Presentación de estos datos** a través de la creación de una página web. Ésta será accesible desde cualquier dispositivo fijo o móvil. Para la construcción de la web nos valdremos, principalmente, del lenguaje de programación PHP.
- **Creación de un módulo planificador de rutas** que nos dará las indicaciones necesarias para ir desde una parada hasta otra.

En este último capítulo pasamos a describir las posibles líneas de continuación y trabajo futuro a largo plazo. Estas líneas de trabajo futuro estarán enmarcadas en uno de las 3 componentes antes citadas. Finalmente relataremos nuestra valoración personal del trabajo desarrollado.

9.1. Trabajo futuro

Con vistas a una futura extensión de este proyecto, además de alguna que otra posible mejora, a continuación enumero algunas opciones que podían haber representado una posible mejora a la hora de la realización de este trabajo.

9.1.1. Mejoras gráficas

Esto es, realizar las modificaciones visuales que se estimen oportunas a la aplicación web. Dichas modificaciones están orientadas a la mejora de la experiencia de consulta del usuario.

Una de las principales mejoras gráficas a aplicar en este proyecto, sería mejorar la interfaz ofrecida por la parte móvil que ofrece el proyecto. Es decir, profundizar más en

el uso de JQtouch¹, plugin para el desarrollo web móvil basado en JQuery², con el fin de conseguir perfeccionar el aspecto visual de formularios, tablas, paginación...etc.

Además, también es posible optimizar la experiencia visual de la aplicación web general. Esta mejora se ha de realizar programando, en función de nuestra necesidad, el apartado CSS de cada uno de los ficheros *PHP* que integran este proyecto.

Esta mejora se encuentra posicionada en la componente de presentación de datos.

9.1.2. Obtención de conocimiento

Es decir, añadir una ontología *OWL*³ a la información contenida en nuestra base de datos, de manera que los datos ofrecidos por la base de datos son cargados de semántica, pasando de ofrecer información a conocimiento. De esta manera se procesa el contenido de la información, obteniendo conocimiento, en lugar de representar información.

La aplicación consumidora de datos es el marco en el cual posicionar la futura línea de trabajo aquí descrita.

9.1.3. Creación de un sistema multiagente

Creación de un sistema multiagente, basada en la plataforma de agentes JADE, que permita la caracterización de los principales elementos que intervienen en el servicio de autobuses aquí estudiado. Elementos tales como: autobuses, paradas, trayectos...etc.

Gracias a estos agentes, se posibilita la obtención de información más pormenorizada. Información, tal como:

- Información acerca del estado real de una línea.
- Retrasos de tiempo que se dan en una parada determinada, localizando el autobús que los provoca.
- Notificación de atascos en determinados trayectos, así como localizar el tramo en el que se haya podido originar...etc.

¹ <http://jqtouch.com>

² <http://jquery.com>

³ Lenguaje de Ontologías Web

Esta creación del sistema multiagente se encuentra localizado tanto en el marco de la aplicación recuperadora de datos, como en el marco de la componente de presentación de datos (página web).

9.1.4. Enrutador con aproximaciones

Esta mejora estaría remarcada en la componente planificadora de rutas, o creación del planificador de rutas. La línea de trabajo futuro a desarrollar en este apartado, constaría de la implementación de una funcionalidad que en función de la parada de origen marcada por el usuario, sumado a los tiempos de llegada de los autobuses a dicha parada, se pueda elegir entre una ruta u otra.

Por lo que, el algoritmo no solo tendría en cuenta las paradas de autobús, sino que se tomaría en cuenta los tiempos de llegada de un autobús cualquiera. Es decir, tendríamos varias rutas alternativas a elegir dependiendo de los autobuses que se aproximasen a la parada.

9.2. Valoración final

Por una parte, y entrando a valorar la filosofía de reutilización y accesibilidad que se desprende de los *Open Data*, puede decirse que ésta filosofía, o más bien, movimiento social de liberalización de datos, me ha parecido bastante interesante / me ha causado bastante impresión.

Este interés está fundamentado en el carácter social que implícitamente lleva ésta forma de liberar, o de poner a disposición del público, numerosos tipos de datos. Ya que, el hecho de crear una aplicación, reutilizando datos procedentes de un servicio público, y que además sea de utilidad para los usuarios de dicho servicio, es bastante gratificante.

En general, todas aquellas aplicaciones con utilidades sociales desarrolladas a partir de información pública, a partir de campos como pueden ser movilidad urbana, transporte, tráfico rodado, sanidad y educación, son las que justifican la existencia de este tipo de apertura de datos.

Aplicaciones como puedan ser: mapa de predicciones meteorológicas, mapa de incidencias de tráfico, callejeros, planificadores de visita...etc.

Además, existe otro tipo de aplicaciones aparecidas en el panorama nacional e internacional en torno a otro de los conceptos tradicionalmente asociados al potencial del *Open Data*: el de la transparencia y la rendición de cuentas. Estas aplicaciones, a nivel general, están bien, puesto que son de interés un interés más político-social.

Estas, son aplicaciones del tipo: ¿Cuánto dinero público se dedica a la gestión de los residuos en tu ayuntamiento?, ¿cuándo se publican las convocatorias de oposiciones?, ¿A qué dedican su tiempo los políticos?, y un largo etcétera.

Pero, el carácter redentor de cara al pueblo/ciudadanía, con el que muchos partidos políticos, hacen uso de este tipo de datos, se encuentra fuera de lugar. Además se aleja bastante del fin último de conseguir aplicaciones de valor añadido que proporcionen un bien a la sociedad. Es decir, se liberan datos de los sueldos de los líderes de estos partidos políticos, ¿y qué?, ¿acaso con ellos podemos construir algo de utilidad que sirva a terceros/sociedad?... no.

Volviendo a los *Open Data*, pero esta vez, aplicados al proyecto que aquí se presenta, puede calificarse de bastante positiva la experiencia. Experiencia que comenzó buscando datos de transporte público, por lo que desde un principio íbamos por la línea que queríamos transitar. Aunque, al principio, no fue nada fácil encontrar un buen conjunto de datos de ésta índole, ¿por qué?.

El motivo radicaba en que, inicialmente, comenzamos tratando el servicio web de *TUSSAM*, servicio ofrecido por la red de transportes públicos de Sevilla, el cual, aparentemente, ofrecía fácil accesibilidad a los datos. Pero, tras comprobar que el

servidor permanecía más tiempo fuera de servicio que operativo y no ser capaz de comprender los argumentos requeridos por algunas de las operaciones publicadas, tras varias semanas de intentos se decidió dejarlo y continuar la búsqueda en otro sitio.

Tras unos días de búsqueda, encontramos el servicio web '*BusGijón*'. Éste, ofrecía todas sus operaciones sin ningún tipo de ambigüedad y con facilidad a la hora de realizar las distintas consultas. Además de lo estable de su servidor y del servicio que ofrecen. Así pues, en Gijón nos quedamos e implementamos nuestro planificador de rutas, ofreciendo así un servicio de valor añadido a lo que en un principio nos proporcionaban los datos del servicio web.

Comentar, que me hubiese gustado aplicar todo lo creado con el servicio de Gijón, a mi ciudad, Huelva. Debido a que en el momento de ponerse con el proyecto, no se encontró ninguna fuente de datos relativa al servicio de autobuses de Huelva, decidimos darlo por perdido.

En resumidas cuentas, la elaboración y estudio de este proyecto, ha supuesto un reto enriquecedor en todos los aspectos, a la par que duro. Por lo que me llevo una buena sensación tras su realización.

Anexos

Condiciones de uso – Aviso legal

En cada una de las modalidades citadas y que se encuentran especificadas en cada conjunto de datos del catálogo, las condiciones de uso son las siguientes:

1. La Ley 37/2007, de 16 de noviembre, sobre reutilización de la información del sector público, que transpone la Directiva 2003/98/CE del Parlamento Europeo y del Consejo, regula la reutilización de la información pública de la que disponen las administraciones y los organismos en los que participan mayoritariamente, es decir, el derecho de todos los agentes potenciales del mercado a la reutilización de la información de las instancias públicas. Asimismo, la Ley 29/2010, de 3 de agosto, del uso de los medios electrónicos en el sector público de Cataluña, establece que hay que difundir electrónicamente la información de interés general para la ciudadanía (tráfico, tiempo, medio ambiente, cultura, salud, educación...).

De acuerdo con esta normativa, siempre hay que citar el titular, la fuente de la información. A cambio, la Administración permite la reproducción (copia), la distribución (en los diferentes formatos: CD, libro, vídeo...) y la comunicación pública de la obra (la difusión, por lo tanto, vía Internet y soportes digitales) y, además, la transformación de la obra conllevará obras derivadas, siempre que no se contradiga con la licencia o aviso que pueda tener una obra y que es la que prevalece, y tener finalidad comercial (es decir, obtener ganancias económicas).

En este caso, para reutilizar la información, hay que seguir las condiciones siguientes:

- No desnaturalizar el sentido de la información.
- Citar siempre la fuente de la información.
- Mencionar la fecha de la última actualización de la información.

2. La reutilización se puede limitar por la tutela de otros bienes jurídicos prioritarios, como por ejemplo la protección de los datos personales, la intimidad o los derechos de protección intelectual de terceros. La reutilización de obras protegidas por la propiedad intelectual se formaliza mediante el uso de la licencia de Reconocimiento Creative Commons CC BY 3.0.

En la práctica se permite la reproducción, la distribución, la comunicación pública y la transformación para generar una obra derivada, sin ninguna restricción, siempre que se cite al autor (Generalitat de Cataluña). La licencia completa se puede consultar en

<http://creativecommons.org/licenses/by/3.0/es/legalcode.es>



Reconocimiento (by): Se permite cualquier explotación de la obra, incluyendo una finalidad comercial, así como la creación de obras derivadas, la distribución de las cuales también está permitida sin ninguna restricción.

3. En el caso de las fotografías del Banc Iconogràfic de la Generalitat de Catalunya (BIG) y otros tipos de datos en los cuales hay participación de terceros, la reutilización se vehicula a través de alguna de la licencia Creative Commons Reconocimiento – SinObraDerivada CC BY-ND 3.0



Reconocimiento – SinObraDerivada (by-nd): Se permite el uso comercial de la obra pero no la generación de obras derivadas.

4. En determinados casos, la reutilización sólo será posible con solicitud previa en el ente generador de los datos.

Así pues, por lo tanto, la utilización, reproducción, modificación o distribución de los conjuntos de datos supone siempre la obligación de reconocer/citar el portal de datos en cuestión como la fuente de los conjuntos de datos de la forma siguiente:

Fuente de los datos: **Portal de Datos Abiertos. [Nombre del organismo o ente autónomo]**

Si se incluye esta cita en formato HTML, puede utilizar el marcado siguiente, o similar:

```
<p>Fuente de los datos: <a href="http:// ..." title="Nombre del Portal de Datos Abiertos">Secretaria general del Estado;a</a>.</p>
```

En cualquier caso, la aceptación de los términos de uso no supone la concesión de los derechos de autor ni la propiedad intelectual sobre los conjuntos de datos.

Fórmulas para la citación de la fuente

Como se ha dicho anteriormente, en cualquiera de las dos modalidades de reutilización (sin ningún tipo de condición vía Ley 37/2007 y con condiciones por medio de licencias Creative Commons BY o BY-ND), la fórmula exacta que se tiene que citar en los datos por parte de las empresas o usuarios reutilizadores es ésta:

Fuente: **Portal de Datos Abiertos. [Nombre del organismo o ente autónomo]**

Si se incluye esta cita en formato HTML, puede utilizar el marcado siguiente, o similar:

```
<p>Fuente de los datos: <a href="http:// ..." title="Nombre del Portal de Datos Abiertos">Secretaria general del Estado;a</a>.</p>
```

Bibliografía

¿Datos abiertos? sí, pero de forma sostenible. 06 de 2011.
<http://www.slideshare.net/mgarrigap/datos-abiertos-s-pero-de-forma-sostenible>.

9 de 05 de 2011. <http://www.uwgb.edu/dutchs/usefuldata/utmformulas.htm>.

2011. <http://econfianza.wordpress.com/2011/03/15/open-data/>.

Aranda, Sebastián González. */* Datos en abierto ~ */ | Bitácora de un proyecto fin de carrera...* 2011. <https://datosenabierto.wordpress.com/>.

Colón, Abimael Rodríguez. *Maestros del web.* 13 de 04 de 2011.
<http://www.maestrosdelweb.com/editorial/guia-mapas-marcadores/>.

comercio, Ministerio de industria y. <http://www.aporta.es/web/guest/index>.

Como crear relaciones y lograr integridad referencial en tablas InnoDB usando phpMyAdmin. <http://blog.rogertm.com/tutoriales/como-crear-relaciones-y-lograr-integridad-referencial-en-tablas-innodb-usando-phpmyadmin/77.html>.

CSS3 - Media Queries. 18 de 08 de 2010. <http://webdesignerwall.com/tutorials/css3-media-queries>.

Desafío Abredatos . 2011. <http://live.abredatos.es/teams>.

Ejecutar código php en java y viceversa | Blog-etil. 28 de 07 de 2008.
<http://gonetil.wordpress.com/2008/07/28/instalar-phpjava-bridge-en-linux/>.

Estréllate y arde. 26 de 04 de 2010. <http://www.estrellateyarde.org/discover/como-hacer-un-mashup>.

Gijón, Ayuntamiento de. *Reutilización de la información de Sector Público en Gijón.* 2011. <http://datos.gijon.es/>.

Java Bridge - Como unir PHP y Java. 9 de 02 de 2011.
<http://www.yavaris.com/blog/2011/02/09/java-bridge-un-puente-hacia-lo-desconocido/>.

Manual de PHP - Foros del web.
http://www.forosdelweb.com/wiki/Manual_de_PHP?utm_source=FDW&utm_medium=Avisos&utm_content=Registrados&utm_campaign=Wiki-fdw.

MEDIALAB PRADO. *Cartografía, descubrimiento, uso y representación.* 2011.
http://medialab-prado.es/article/cartografia_descubrimiento_uso_y_representacion.

Open Data CTIC. 2011. <http://datos.fundacionctic.org/>.

Open Data Euskadi. *Open Data Euskadi, el portal de Apertura de Datos Públicos del Gobierno Vasco*. 2011. <http://opendata.euskadi.net/w79-home/es>.

Wikipedia. 2011. <http://es.wikipedia.org/wiki/Wiki>.