Servicio web de Geocodificación para Cartociudad

Adaptación del GEOCODER de TIGER-PostGIS a Cartociudad-Oracle

Moya-Honduvilla, Iván; Manso-Callejo, Miguel A.

El trabajo presenta las adaptaciones y los desarrollos realizados para implementar un Servicio Web de Procesamiento (WPS) que ofrezca un algoritmo de geocodificación para CartoCiudad basándose en proyectos Open Source (52North WPS y Geocoder-TIGER PostGIS).

CartoCiudad se materializa en una Base de Datos Oficial de la red viaria, con estructura topológica de SIG, que permite la navegación continua por todo el territorio español que además contiene la división administrativa censal y postal en polígonos. Un Geocoder es una aplicación que utiliza su base de datos con la finalidad de geocodificar es decir devolver uno o varios juegos de coordenadas a partir de una dirección.

El desarrollo se basa en el código abierto, escrito en *PL/PgSQL* para el Sistema Gestor de Bases de Datos (SGBD) PostgreSQL que utiliza las funciones del módulo PostGIS, para geocodificar la base de datos TIGER del U.S. Bureau. La metodología empleada en la adaptación de TIGER a CartoCiudad, parte del análisis del modelo de datos de ambas bases de datos, la identificación de las entidades equivalentes y, la traducción de dichas equivalencias al lenguaje *procedural PL/PgSQL*. La siguiente fase, consiste en la traducción de las funciones en *PLpg/SQL* para PostGIS, al lenguaje de programación PL/SQL para Oracle. En este paso, se identifican las diferencias entre ambos lenguajes, y se aplica su traducción. Asimismo, se identifican las funciones espaciales para PostGIS y se busca su equivalencia para Oracle *spatial*. En todo el proceso se toma como referencia la documentación aportada por ambos SGBD (PostgreSQL y Oracle). Una vez adaptados los desarrollos se ha abordado la optimización de la velocidad de respuesta, indexando adecuadamente los atributos utilizados en la geocodificación. También se ha asociado los viales con los códigos postales mediante una relación N:M, dado que el modelo de datos CartoCiudad no proporciona ésta relación y las funciones espaciales son costosas en tiempo de ejecución. Con la inclusión de la relación N:M entre código postal y vial, se propone un modelo de datos sin normalizar, justificado debidamente, para optimizar las búsquedas.

El algoritmo de geocodificación, busca maximizar el porcentaje de acierto en las respuestas, proporcionando tolerancia a errores tipográficos en los valores de entrada y dotando de capacidad para resolver ambigüedades en ciertas búsquedas. Para ello, se emplea una búsqueda difusa con función heurística de calidad, de tal modo que en caso de no poder satisfacer una búsqueda exacta, proporciona al usuario una lista de posiciones asociadas a una dirección, ordenadas decrecientemente por el grado de coincidencia informado por la función heurística, junto a la correspondiente dirección normalizada. En la búsqueda difusa se emplea un algoritmo fonético, de modo que alternativamente se buscarán direcciones similares por su pronunciación.

Para facilitar la utilización en un entorno IDE (interoperabilidad), se ha integrado el desarrollo basado fundamentalmente en los SGBD, en un WPS conforme con OGC. En la implementación de esta capa se utiliza el *framework* desarrollado en Java por 52°North.

PALABRAS CLAVE
WPS, Geocoder, CartoCiudad, PostgreSQL, Oracle, TIGER

1. INTRODUCCIÓN

CartoCiudad es un proyecto resultado de la integración y armonización de datos aportados por diferentes organismos públicos (principalmente Dirección General del Catastro, Instituto Nacional de Estadística, Sociedad Estatal de Correos y Telégrafos e Instituto Geográfico Nacional)[1], que consiste en generar cartografía digital oficial de ciudades y núcleos de población españoles con viales e información textual. Constituye una Base de Datos Oficial de red viaria, con estructura topológica de SIG (Sistema de Información Geográfica), que asegura la continuidad geográfica en todo el territorio nacional.

El detalle sobre la fuente de los organismos públicos que tienen la responsabilidad de su mantenimiento, son los siguientes:

- Las capas de fondo urbano son proporcionadas por la Dirección General del Catastro
- Los nombres de calles son recopilados de los Ayuntamientos por el Instituto Nacional de Estadística (INE).
- Las secciones y distritos censales también son definidos por el Instituto Nacional de Estadística.
- Los Códigos Postales se generan con la información proveniente de la Sociedad Estatal de Correos y Telégrafos.
- El trazado y nombre de las carreteras se han obtenido a partir del Instituto Geográfico Nacional y del Ministerio de Fomento.

La producción de CartoCiudad está planteada en varios años. De momento, a fecha de Septiembre de 2010, no se encontraban disponibles todos los municipios en la base de datos, aunque está prevista su incorporación próximamente. Según los datos facilitados por el Proyecto CartoCiudad, se estima que en la actualidad, al menos un 88% de la población empadronada en municipios de España, su residencia habitual está incluida en esta base de datos.

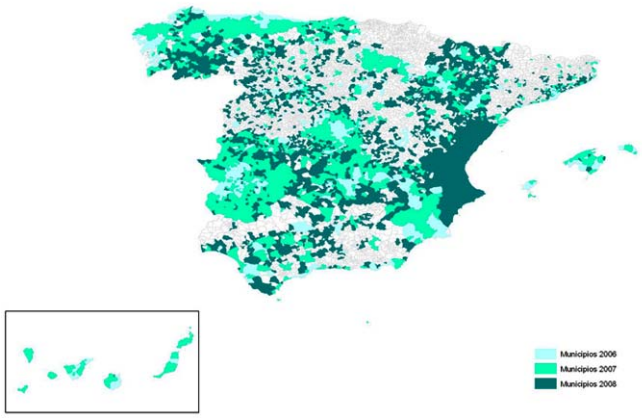

Figura 1: Cobertura de CartoCiudad dada por su página oficial [1]

El proyecto CartoCiudad, además de integrar la cartografía, tiene como meta desarrollar servicios Web geográficos útiles para la administración y los ciudadanos. Entre otros, pueden implementarse, basándose en la información geográfica almacenada en CartoCiudad, servicio de mapas, servicio de nomenclátor, de procesamiento (hallar rutas óptimas, áreas de influencia, etc…) y otras más que aún no se han previsto ni desplegado. Uno de los servicios que se pueden crear dentro del proyecto CartoCiudad, es el es el proceso de asignar coordenadas geográficas (latitud-longitud) a puntos del mapa (direcciones, puntos de interés, etc…), lo que se conoce como geocodificación.

La meta de este trabajo es desarrollar un servicio de geocodificación basado en la información de los viales contenidos en la cartografía de CartoCiudad, de modo que ante una búsqueda textual de una dirección ya sea postal o punto kilométrico se obtenga como respuesta el conjunto de resultados que más se aproximan a la consulta. Este conjunto de localizaciones priorizadas por un indicador de calidad contendrá además de la dirección completa normalizada sus coordenadas.

2. Metodología

Como punto de partida para el desarrollo del servicio de geocodificación, se ha adoptado el equivalente ya desarrollado y disponible de forma abierta para la base de datos TIGER de la Oficina del Censo de los Estados Unidos (del inglés *United States Census Bureau*). El geocodificador de TIGER, está desarrollado en el lenguaje procedural (PL) para el Sistema Gestor de Base de Datos PostgreSQL, y utiliza las funciones espaciales de la extensión espacial PostGIS.

Para reutilizar los desarrollos disponibles de geocodificación del proyecto TIGER sobre el proyecto CartoCiudad se han aplicado técnicas de ingeniería inversa, consistentes en averiguar cómo funcionan los procedimientos almacenados, entender el modelo de datos, etc. Siendo más precisos el trabajo realizado puede considerarse reingeniería del software ya que en este caso sí se dispone del código fuente, con el propósito de modificarlo y adaptarlo para que cumpla las mismas funciones con el modelo de datos de CartoCiudad. Como aclaración al concepto de reingeniería de software antes mencionado, Elliot J. Chikofsky y James H. Cross [2] la definen como "*El análisis y modificación de un sistema, para reconstituirlo en una nueva forma*". La modificación de un *software* como proceso de reingeniería, generalmente requiere un proceso de ingeniería inversa para el análisis del sistema, y de ingeniería directa para la reconstitución en un nuevo software. En la figura 2 se resume esquemáticamente esta metodología de reingeniería y modificación.

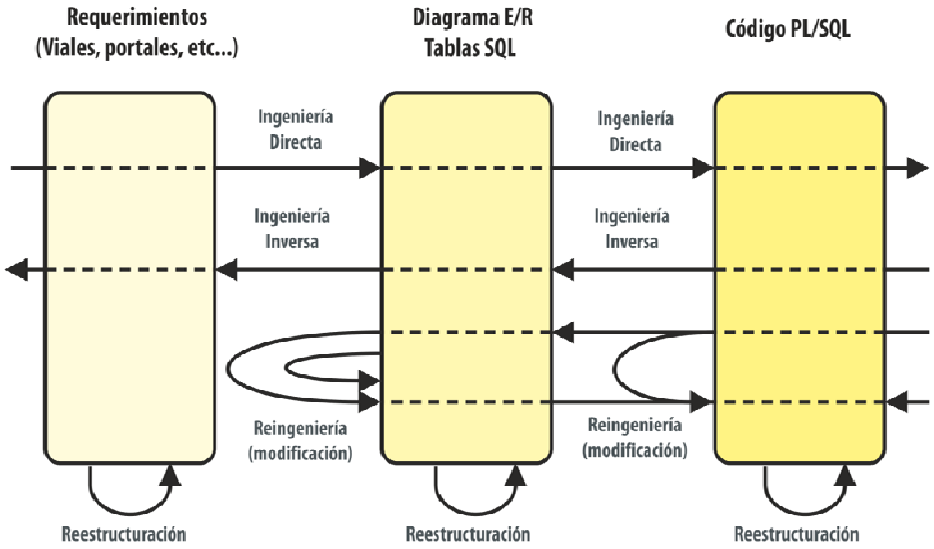

Figura 2: Reingeniería de software propuesta por Elliot J. Chikofsky y James H. Cross

La fase de ingeniería inversa para Geocoder-TIGER, se realiza con la finalidad de identificar el uso las columnas de las tablas implicadas en consultas SQL. En cuanto a la fase de ingeniería directa, se ha utilizado para expresar en el lenguaje SQL las consultas a en el modelo de datos de CartoCiudad y en la base de datos Oracle.

Parte del trabajo de reingeniería ha consistido en analizar los modelos de datos de ambas cartografías, analizando los esquemas relacionales de las tablas definidas en cada modelo y estudiando ambos casos desde el punto de vista de la normalización de las bases de datos. En esta etapa se ha constatado que la tabla "*tiger_geocode_roads*" definida en Geocoder-TIGER y representada en la Figura 3 con algunas tuplas o filas de ejemplo, no satisface la tercera forma normal (3NF) definida por E.F.Codd [3], ya que existen atributos que dependen transitivamente de la clave, como se puede comprobar en el siguiente ejemplo, o caso de estudio, del Esquema Relacional de la tabla, analizando un subconjunto de dependencias funcionales donde se da lugar la dependencia transitiva.

T_1 = {id,tlid,fedirp,fename,fetype,fedirs,zip,state,country,cousub,place}
L_1 = {id-> place; place –> state; place –> county; place –> cousub}
("*state*", "*country*" y "*cousub*" dependen transitivamente de la clave "*id*")

Siendo:

T_1 = Conjunto de atributos de R.
L_1 = Subconjunto de dependencias funcionales de R.
R = Esquema Relacional deducido de la tabla "*tiger_geocode_roads*"

$L_1 \subset L_1$ / R(T_1,L_1) $\notin$ 3FN

Este hecho puede provocar anomalías en la base de datos si además de estar diseñada para ser explotada con el servicio Geocoder se pretende realizar actualización, inserción y borrado de datos en los procesos propios de mantenimiento de la cartografía. De este modo tan sólo se podría garantizar la integridad de la información al actualizarla mediante la reconstrucción completa de la base de datos.

	id integer	tlid integer	fedirp character	fename character	fetype character	fedirs character	zip integer	state character	county character	cousub character varying(9(	place character v.
	677712	122907228	N	Bayshore	Rd	W	95112	CA	Santa Clara	San Jose	San Jose
	788872	125053865	S	Shore	Ctr	W	94501	CA	Alameda	Alameda	Alameda
	1089163	141515181	W	Garvey	Ave	N	91790	CA	Los Angeles	East San Gabriel Valley	West Covina
	1247380	141692989	W	7th	St	N	90813	CA	Los Angeles	Long Beach-Lakewood	Long Beach

Figura 3: Tabla "*tiger_geocode_roads*" de Geocoder-TIGER

El esquema relacional extraído de la tabla "*vial_lookup*" propuesta para el Geocoder de CartoCiudad de forma análoga a Geocoder-TIGER, tampoco satisface la tercera forma normal (3FN), por dependencia transitiva entre clave y atributo, ni la segunda forma normal (2FN). Sin embargo, tanto la tabla "*tiger_geocode_roads*" como la tabla "*vial_lookup*" generada a partir de CartoCiudad, están pensadas para mantener sin cambios su contenido debiendo de ser generadas previamente a partir de la base de datos TIGER y CartoCiudad, respectivamente. En concreto, "*vial_lookup*" tiene como valor añadido el incluir la relación N:M entre código postal y vial, algo de lo que carecía CartoCiudad en sus especificaciones hasta Enero de 2010. Por ello, la principal justificación para desnormalizar la base de datos, reside en incluir la relación directa N:M entre vial y código postal, que no está soportada en la base de datos proporcionada por CartoCiudad a fecha de Diciembre de 2009. No obstante, en las especificaciones del producto CartoCiudad a fecha de 13/01/2010 (versión 8), comprende la inclusión del código postal en el *portal_pk* (relación N:1 entre las entidades *codigo_postal* y *portal_pk*).

Además, se contaría con la ventaja añadida de no aumentar el coste de acceso a información con el uso del operador de álgebra relacional Unión Natural (JOIN).

En la figura 4 se muestra la tabla "*vial_lookup*" propuesta para este trabajo, derivada de la base de datos CartoCiudad.

id_cp numeric(12,0)	id_vial numeric(12,0)	id_mun numeric(12,0)	cod_posta character(	tip_via character(	nom_via character(100	municipio character(1	provincia character
280740000002	280740000006	280740000001	28912	CALLE	LOPE DE VEGA	Leganés	Madrid
280740000002	280740000008	280740000001	28912	CALLE	MADRID	Leganés	Madrid
280740000002	280740000009	280740000001	28912	TRVA	MADRID	Leganés	Madrid

Figura 4: Tabla "*vial_lookup*" de GeoCoder-CartoCiudad

En cualquier caso, para una futuro cumplimiento de la Forma Normal de Boyce-Codd (FNBC), definida en 1974 por Raymond F. Boyce y Codd[4]-que además de cumplir 3FN, hace frente a ciertos tipos de anomalías en el resultado de consultas con JOIN, como la aparición de tuplas extrañas- sería posible descomponer los esquemas ya contemplados y rehacer la aplicación PL/SQL mediante reingeniería.

Otra parte del trabajo ha consistido en adaptar el lenguaje *procedural PLpg/SQL* para PostgreSQL, en su equivalente para PL/SQL de Oracle. Si bien las similitudes entre ambos son notorias, cada uno cuenta con sus propias peculiaridades. Como elemento fundamental en esta fase, se ha precisado la documentación de ambos lenguajes. Otro punto a destacar es la búsqueda de equivalencias entre las funciones espaciales proporcionadas por ambos lenguajes y sus respectivos SGBD. Un ejemplo podría ser la función "*line_interpolate_point(geometry, double precision)"* para PostGIS, que interpola una línea. Su equivalencia para Oracle Spatial no es directa, necesitándose invocar de manera secuencial varios procedimientos, pertenecientes al paquete "MDSYS.SDO_LRS", que define, modifica, consulta y convierte geometrías lineales de Oracle en elementos de tipo "*Linear Referencing System*" dedicadas a interpolar un número de portal/PK a lo largo de una poli línea acotada por los valores máximos y mínimos de portales.

Como parte del conjunto de funciones resultantes, tanto para *PLpg/SQL* como para PL/SQL, en la figura 5 se muestra a modo de ejemplo, un Diagrama de Componentes UML donde se representa un subconjunto de funciones pertenecientes a la aplicación PL/SQL completa. Por razones de espacio, no se detalla en esta publicación el esquema completo. Cada componente representa una función PL/SQL, en el caso de uso se describe la llamada, y el interface representa el objetivo de dicha llamada. Desde el punto de vista estructural, este diagrama muestra un acoplamiento a nivel de datos, donde éstos se envían a través de parámetros, y los resultados se devuelven como retorno de la función. Este tipo de acoplamiento se caracteriza por un buen grado de independencia, facilidad para modificar, depurar errores y reusabilidad [5]. La cohesión de esta estructura es funcional, ya que cada componente se encarga de ejecutar una sola función o tarea concreta. Este grado de cohesión aporta un buen nivel de independencia entre componentes, comprensión y limpieza de código [5].

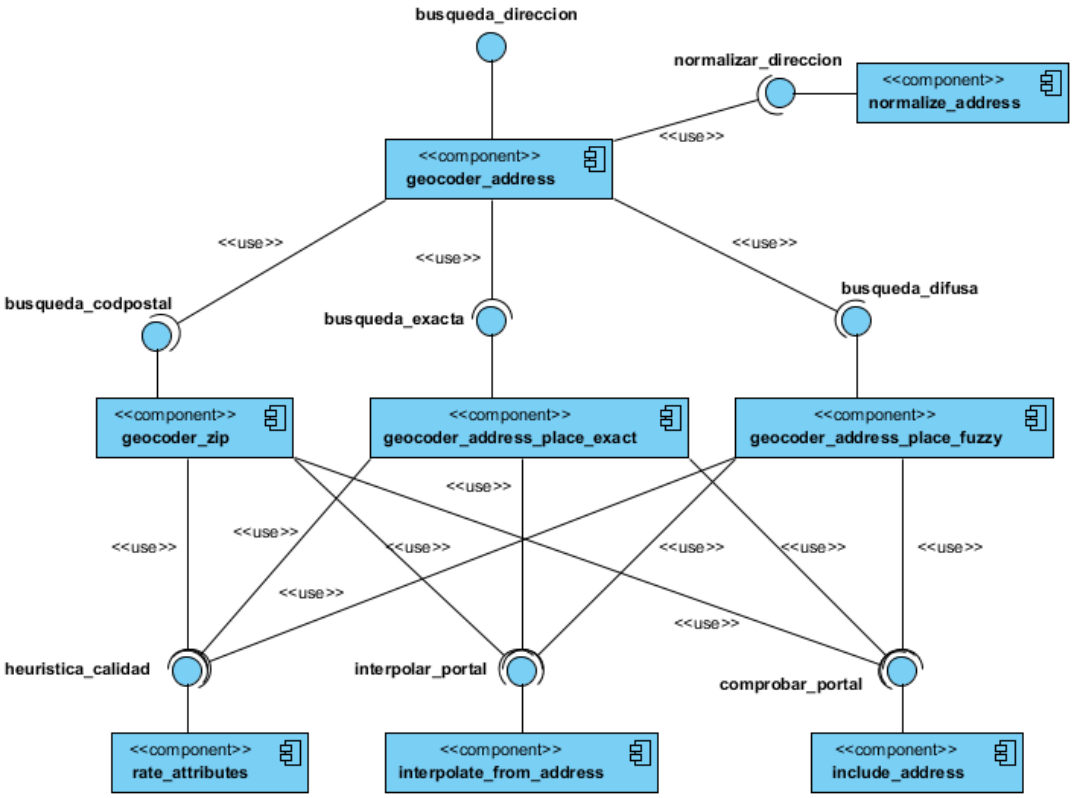

Figura 5: Definición de subconjunto de funciones PL/SQL en Diagrama de Componentes UML

2.1 NORMALIZACIÓN

La falta de normalización en los valores de entrada de una aplicación de búsqueda, dificulta la realización de consultas en las bases de datos [6], por lo que se convierte en uno de los principales obstáculos de un buscador cuya fuente de información sean consultas contra una base de datos relacional, como es el caso del geocodificador propuesto en este trabajo. Debido a este importante inconveniente, es necesario normalizar en la medida de lo posible el formato de las direcciones que se pretenden buscar.

Se propone para ello una pequeña metodología de normalización, distribuida en seis pasos que se describen a continuación:

1. Estandarización de tipos de vía. Tanto "*calle*" como "*C/*" se refieren indistintamente al mismo tipo de vial. Se identifican estas variaciones y se elige una palabra estándar que identifique inequívocamente a un sólo tipo de vial. TIGET-Geocoder prevé una estandarización equivalente, donde en unas tablas auxiliares, se almacenan abreviaturas equivalentes que definen un tipo de vial dado, como es el caso de "*AVENUE*", "*AVE*" ó "*AV*".

2. Capitalización de letras. Muchos motores de búsqueda, así como las consultas SQL, distinguen entre minúscula y mayúscula. Arbitrariamente, se elige capitalizar todas las letras a minúsculas.

3. Eliminar tildes, diéresis y otros tipos de caracteres especiales (Ej: à,é,ö,º,/,&). Para una consulta SQL, la palabra "*Móstoles*" sería distinta a "*Mostoles*" por la omisión de la tilde en esta última. Asimismo uso de diéresis, tildes y otros caracteres especiales no son correctamente procesados con las implementaciones para PostgreSQL y Oracle de los algoritmos fonéticos como *Soundex* o *Double Metaphone*, debida cuenta a que las tildes o diéresis no se utilizan en inglés.

4. Eliminar palabras vacías *(stop words)*. Son palabras vacías los artículos, pronombres y preposiciones (de, la, los, y, etc.). La eliminación de estas palabras en la indexación permite resolver con mayor grado de acierto las búsquedas que se escriben de distinta forma. Por ejemplo "*Calle de las Huertas*" en Madrid, también se suele escribir como "*Calle Huertas*" o "*Calle de Huertas".*
5. Interpretación de las direcciones a geocodificar. El objetivo fundamental de este paso es extraer los componentes diferenciados de una dirección, a partir de una cadena de texto dada, siempre que ésta cumpla con un formato establecido. Se busca un compromiso entre flexibilidad, precisión y tiempo de respuesta. Se ha optado por el siguiente algoritmo de normalización de direcciones para extraer los campos fundamentales para localizar una dirección. El algoritmo no comprueba si la dirección realmente existe, simplemente se extrae una provincia y un municipio (si aparece en la dirección) y comprueba si existen. Para la normalización de la dirección no tienen por qué existir el vial, el código postal o el portal introducidos. La estrategia seguida por el algoritmo es la que se expone a continuación:

   - ✓ Extraer código postal, determinado como número de cinco cifras. No importa su posición.
   - ➔ Si se ha encontrado código postal:
     - ✓ Anotarlo y eliminarlo de la cadena de texto.

   - ✓ Extraer tipo de vial, al principio de la cadena de texto.
   - ➔ Si se ha encontrado un tipo de vía:
     - ✓ Anotarlo y eliminarlo de la cadena de texto.

   - ✓ Extraer provincia del final de la cadena de texto, usando búsqueda difusa si falla la búsqueda exacta.
   - ➔ Si se ha encontrado una provincia:
     - ✓ Anotarlo y eliminarlo de la cadena de texto.

   - ✓ Extraer municipio del final de la cadena de texto, usando búsqueda difusa si falla la búsqueda exacta. Se acota la búsqueda a la provincia encontrada anteriormente, si la hay.
   - ➔ Si se ha encontrado un municipio:
     - ✓ Anotarlo y eliminarlo de la cadena de texto.
   - ➔ En caso contrario, si se ha anotado una provincia, se asume que el municipio es también la provincia.
     - ✓ Anotar el municipio igual que la provincia

   - ✓ Extraer el portal por la derecha
   - ➔ Si se ha encontrado un portal:
     - ✓ Anotarlo y eliminarlo de la cadena de texto.

   - ✓ El nombre del vial (calle, avenida, plaza...) es el contenido restante, se anota y finaliza el algoritmo.

2.2 ALGORITMO FONÉTICO

Soundex es un algoritmo fonético inicialmente pensado para indexar nombres según su pronunciación en inglés. Este algoritmo es muy utilizado para la investigación de genealogía en el ámbito anglosajón. Sin embargo, *Soundex* tiene deficiencias. Por ejemplo, devuelve un número alto de falsos resultados, debido a que ante dos nombres de viales muy diferentes la función *Soundex* devuelve el mismo valor, de modo que la búsqueda devolverá resultados irrelevantes que pertenecen a otra dirección diferente a la buscada. Por ejemplo, "*Jaca*" y "*Joyosa*" tienen el código de *Soundex* J200. Por otra parte, algunas variantes del nombre de vial realmente similares reciben códigos de *Soundex* diferentes, perdiendo resultados relevantes y creando un problema de falsos negativos. Por ejemplo, "*Higuera*", "*Higueral*", "*Higueras*" e "*Higuerilla*" cada uno tiene un código de *Soundex* diferente, por lo que buscando cualquiera de ellos nos devolverán resultados para los demás. Además, *Soundex* no clasifica los resultados del más al menos probable. Esto último obliga a buscar una función de calidad, que determine el grado de aproximación o similitud con la palabra buscada respecto a la base de datos. Finalmente, *Soundex* está está generalmente implementado según la fonética inglesa, por lo que disminuye su eficacia en su utilización con otras lenguas como el castellano. Si bien existen adaptaciones de *Soundex* para la fonética castellana, el algoritmo definido para la función *Soundex* implementado por defecto en Oracle y PostgreSQL está definido de acuerdo con la fonética inglesa.

Se deja abierta la posibilidad de utilizar otros algoritmos fonéticos como *Double Metaphone*, para posteriores revisiones del Geocoder. Doble Metaphone utiliza un conjunto de reglas mucho más complejas que Soundex, como por ejemplo, pruebas de detección de aproximadamente 100 diferentes contextos del uso de la letra “C”. Sin embargo, cuenta como desventaja no estar implementado por defecto en Oracle, por lo que en este trabajo se opta por usar Soundex por razones de compatibilidad.

En la Figura 6 se puede observar una tabla con ejemplos de utilización de Soundex. Se observa cómo "*rebelion*" y "*revelion*" tiene un código Soundex idéntico, por lo que serían equivalentes a la hora de realizar una búsqueda. De ese modo, si por error tipográfico u ortográfico se equivocase una consonante con pronunciación similar, la búsqueda se realizaría de todos modos con satisfactorio resultado, por lo que un Geocoder con búsqueda difusa como el planteado en este trabajo, gozaría de cierta tolerancia a errores. Al contrario que algoritmos criptográficos como *md5()* donde las colisiones hash para dos palabras distintas son inaceptables, las colisiones en *Soundex()* son deseables cuando se trata de buscar palabras equivalentes por su fonética.

	soundex()	metaphone(,4)	dmetaphone()	md5()
rebelion	R145	RBLN	RPLN	58e8cac1c898b28c9a97404a1576166c
revelion	R145	RFLN	RFLN	d3e5b9eb55245724d9f4da5c9e0ddf5d
revelador	R143	RFLT	RFLT	7ae0276ab4fbbba85e589600804cfad
revelacion	R142	RFLS	RFLS	d995b28f1114f190ae2a7201979866fe
reverencia	R165	RFRN	RFRN	f443a0104d1f0488806e0f3a8ea42ba9
reversible	R162	RFRS	RFRS	7927be390e743cc4cee581d12d029078
rebocar	R126	RBKR	RPKR	7c372117575888817451 55f7739f3ab1

Figura 6: Comparación

2.3 PROBLEMAS EN LA GEOCODIFICACIÓN

Es importante destacar que en cualquier técnica de geocodificación conocida, nunca se llega a la tasa óptima del 100% de aciertos o respuestas válidas, debido a diferentes tipos de error conocidos [7][8]. Algunos de ellos se detallan a continuación.

Una dirección que recientemente haya sido físicamente creada, pero que a efectos burocráticos o administrativos no esté dada de alta, y por tanto sea una calle inexistente a efectos cartográficos oficiales, no podría ser codificada geográficamente hasta que sea dada de alta a efectos oficiales.

Sin embargo la dirección sí será normalizada, pues no se verifica si existe realmente la calle en el proceso de normalización. Este caso puede darse en barrios de reciente construcción o en desarrollo, como por ejemplo el barrio de Las Tablas en el norte de Madrid:

- "Calle de María Tudor 11, Madrid" (De reciente construcción)

  Normalización
  Dirección: 11
  Tipo vial: "Calle"
  Vial: "Maria Tudor"
  Localidad: "Madrid"
  Provincia: "Madrid"
  Código Postal: NULL

  Geocodificación
  Coordenadas: 40.4949570824775,-3.3613466339956 (EPSG:4326)
  Tipo vial: "Calle"
  Vial: "Morata de Tajuña"
  Localidad: "Alcalá de Henares"
  Provincia: "Madrid"
  Código Postal: "28806"

  Problema
  La normalización es correcta, pero no es la dirección buscada, ya que no existe en la Base de Datos.

Si se introducen elementos ajenos al formato establecido (p.e.: escalera, planta, puerta…), estos elementos pueden ser erróneamente interpretados como si fueran un portal:

- "Calle Alameda 3, 5ºA esc 6, Madrid"

  Normalización
  Dirección: 6
  Tipo vial: "Calle"
  Vial: "Alameda 3"
  Localidad: "Madrid"
  Provincia: "Madrid"
  Código Postal: NULL

  Geocodificación
  Coordenadas: 40.4122046920416, -3.69393231384188 (EPSG:4326)
  Tipo vial: "Calle"
  Vial: "Alameda"
  Localidad: "Madrid"
  Provincia: "Madrid"
  Código Postal: "28014"

Problema
La normalización es incorrecta puesto que no se extrae correctamente el número de portal que se quiere buscar. El error se propaga a la geocodificación, que localiza el portal nº6 ubicado en la calle Alameda, en lugar del portal inicialmente requerido (nº3).

Si se introduce una dirección en otro idioma co-oficial, sin estar contemplado en CartoCiudad, el normalizador de direcciones no será capaz de extraer el nombre de municipio ni provincia.

- "Kalea Bizkaia 27, Donosti"

  Normalización
  Dirección: 27
  Tipo vial: NULL
  Vial: "Kalea Bizkaia"
  Localidad: "Doneztebe/Santesteban"
  Provincia: NULL
  Código Postal: NULL

  Geocodificación
  No se devuelve ningún resultado.

  Problema
  El error se produce al normalizar la dirección.

Si se introduce un nombre de vial idéntico al nombre de un tipo de vial, sin especificar primero el tipo de vial, el normalizador confundirá nombre de vial con tipo de vial.

- "Avenida 45, Alcobendas 28001"

  Normalización
  Dirección: 45
  Tipo vial: "Avenida"
  Vial: NULL
  Localidad: "Alcobendas"
  Provincia: NULL
  Código Postal: 28001

  Geocodificación
  No se devuelve ningún resultado.

  Problema
  Error en la normalización, al confundir tipo con vial.

Ruido en la dirección introducida. Elementos ajenos al formato preestablecido, como introducir el nombre de un negocio o institución al principio de la cadena de texto, confundirían al localizador.

- "Escuela de Idiomas Omega, Calle Paz 14, Almeria"

  Normalización
  Dirección: 14
  Tipo vial: NULL
  Vial: "Escuela Idiomas Omega"
  Localidad: "Almería"
  Provincia: "Almería"
  Código Postal: NULL

  Geocodificación
  Coordenadas: 40.4122046920416, -3.69393231384188 (EPSG:4326)
  Tipo vial: NULL
  Vial: "Escuelas Miramar (Las)"
  Localidad: "Almería"
  Provincia: "Almería"
  Código Postal: "04071"

Direcciones ambiguas o imprecisas. Si se asocia erróneamente una provincia y un municipio que no se corresponden entre sí, el normalizador no distinguirá el municipio. No ocurre lo mismo si el código postal introducido no existe, o no coincide con la localidad

- 'Plaza de Portugal 14 Arroyomolinos Badajoz 28921'

  Normalización
  Dirección: 14
  Tipo vial: "Plaza"
  Vial: "Portugal"
  Localidad: "Badajoz"
  Provincia: "Badajoz"
  Código Postal: 28921

  Geocodificación
  Coordenadas: 38.8787000294574, -6.97634728286225 (EPSG:4326)
  Tipo vial: "Plaza"
  Vial: "Portugal"
  Localidad: "Badajoz"
  Provincia: "Badajoz"
  Código Postal: 06001

  Problema
  Existe ambigüedad, al ser Arroyomolinos municipio de la provincia de Cáceres (y también de Madrid), en lugar de Badajoz.

Direcciones cuyo portal no existe, pudiéndose haber introducido por error.

- 'Calle de Dulcinea 50 Alcorcón Madrid'

  Normalización
  Dirección: 30
  Tipo vial: "Calle"
  Vial: "Dulcinea"
  Localidad: "Alcorcón"
  Provincia: "Madrid"
  Código Postal: NULL

  Geocodificación
  Coordenadas: 40.4507300593543, -3.6997249606957 (EPSG:4326)
  Tipo vial: "Calle"
  Vial: "Dulcinea"
  Localidad: "Madrid"
  Provincia: "Madrid"
  Código Postal: 28020

  Problema
  No existe el portal en la calle del municipio buscado, por tanto el Geocoder devuelve una alternativa dentro de la provincia. Se encuentra una localización, pero no es la buscada.

Direcciones cuya denominación está abreviada, que no resulta siquiera similar fonéticamente a la denominación original introducida en la Base de Datos.

- 'Calle 2 de Mayo 4 Móstoles Madrid'

  Normalización
  Dirección: 4
  Tipo vial: "Calle"
  Vial: "2 Mayo"
  Localidad: "Móstoles"
  Provincia: "NULL"
  Código Postal: NULL

  Geocodificación
  No se devuelve ningún resultado.

Problema
En la Base de Datos CartoCiudad existe una "Avenida Dos de Mayo (Del)"

Como conclusión a estos problemas, observar que el algoritmo de normalización es mejorable, pero la subsanación de los errores descritos es compleja y podría llevar a otros errores nuevos y colaterales.

2.4 OPTIMIZACIÓN E INDEXADO

Una vez conseguida la correcta funcionalidad del GeoCoder, se aplican técnicas para el optimizado en tiempo de respuesta, como una especificación de requisitos deseable y asociada a cualquier buscador. El objetivo es minimizar el tiempo de respuesta al usuario o proceso *batch*, sin necesidad de incrementar los recursos hardware para ello. Las herramientas básicas para conseguirlo son, la adecuada utilización de índices

PostgreSQL utiliza para el indexado una implementación de árbol B+ (*b-link tree*) de alta concurrencia de Lehman-Yao [9][10]. Análogamente, Oracle utiliza una implementación de árbol B*, una variante del *btree*. En éste tipo de índices, el coste de acceso a un registro es de orden logarítmico O($\log_b N$), mientras que el coste de acceso a un rango K es de orden logarítmico O($\log_b N + K$), siendo B el tamaño de página utilizado tanto en PostgreSQL como en Oracle, N el número de tuplas o filas insertadas en la tabla a indexar, y finalmente K es el tamaño de la secuencia a buscar. Este coste resultante será en el peor de los casos, y mejorará en la práctica cuanto mayor sea el índice de llenado de cada página.

```
EXPLAIN SELECT * FROM codpostal_vial
WHERE normaliza(provincia) = normaliza('Madrid')
AND normaliza(municipio) = normaliza('Alcorcón')
AND soundex(normaliza(nom_via)) = soundex(normaliza('Mayor'));

"Seq Scan on codpostal_vial  (cost=100000000.00..100140241.35
rows=1 width=293) (actual time=482.323..3062.305 rows=5 loops=1)"
"  Filter: ((normaliza((provincia)::text) = 'madrid'::text) AND
(normaliza((municipio)::text) = 'alcorcon'::text)
          AND (soundex(normaliza((nom_via)::text)) =
'M600'::text))"
"Total runtime: 3062.332 ms"
```

Figura 7: Cobertura de CartoCiudad

El plan de ejecución que el planificador de PostgreSQL genera, es una búsqueda secuencial (*Seq Scan*), que en la práctica significa un barrido completo de la tabla, lo que resulta muy ineficiente en tiempo de acceso. A este inconveniente se le suma la ejecución de las funciones *normaliza()* y *soundex()*, que penalizan en gran medida el tiempo total de ejecución de la consulta SQL. En este caso, la consulta se demora en 3.062 milisegundos (ms). Dado lo ineficiente de la búsqueda secuencial, surge la necesidad de aprovechar los beneficios de los índices implementados en árbol B.

Se utiliza a continuación un índice para la columna "*nom_via*", utilizando las funciones *soundex() y normaliza()* de forma anidada. De este modo, se almacenará en el índice el resultado de estas funciones aplicada a la columna indexada para toda la tabla, y no será necesario realizar un barrido completo de la columna seleccionada. Además se aplicará un factor de llenado (fillfactor) de 100, con el objetivo de maximizar el llenado de página, y minimizar el número de páginas accedidas en cada

```
CREATE INDEX nom_via_soundex_lookup_idx
  ON vial_lookup
  USING btree
  (soundex(normaliza(nom_via)))
  WITH (fillfactor=100);
```

Figura 8: Índice para nom_via

Aplicando a la misma consulta, se observa una mejora espectacular en el tiempo de acceso.

```
"Bitmap Heap Scan on vial_lookup  (cost=18.96..3086.27 rows=1
width=293) (actual time=1.042..8.582 rows=5 loops=1)"
"  Recheck Cond: (soundex(normaliza((nom_via)::text)) =
'M600'::text)"
"  Filter: ((normaliza((provincia)::text) = 'madrid'::text) AND
(normaliza((municipio)::text) = 'alcorcon'::text))"
"  ->  Bitmap Index Scan on nom_via_soundex_lookup_idx  (cost=
0.00..18.96 rows=858 width=0) (actual time=0.149..0.149 rows=455
loops=1)"
"        Index Cond: (soundex(normaliza((nom_via)::text)) =
'M600'::text)"
"Total runtime: 8.619 ms"
```

Figura 9: Consulta con índice para nom_via

Se observa el uso del índice creado (*Bitmap Index Scan*), que encuentra 455 filas coincidentes en sólo 0.149ms. Sin embargo, el planificador debe de filtrar el resultado anterior, seleccionando únicamente los viales coincidentes con un municipio y provincia dado. A pesar de realizarse una búsqueda secuencial (*Bitmap Heap Scan*), el conjunto se ha reducido a 455 filas, con la ventaja de que este sub-resultado se encuentra almacenado en memoria RAM. A pesar de la doble consulta (índice y secuencia) el tiempo de respuesta obtenido es de 8.6ms

Para ahorrar la anterior búsqueda secuencial, se crea un índice compuesto, formado por las tres columnas implicadas en la consulta SQL y aplicando las funciones de normalización y búsqueda difusa.

```
CREATE INDEX provincia_soundex_lookup_idx
  ON vial_lookup
  USING btree
  (normaliza(provincia),normaliza(municipio),soundex(normaliza
(nom_via)))
  WITH (fillfactor=100);
```

Figura 10: Indexado compuesto

```
"Index Scan using provincia_double_soundex_lookup_idx on
vial_lookup  (cost=0.76..9.07 rows=1 width=293) (actual time=
0.101..0.108 rows=5 loops=1)"
"  Index Cond: ((normaliza((provincia)::text) = 'madrid'::text)
AND (normaliza((municipio)::text) = 'alcorcon'::text)
              AND (soundex(normaliza((nom_via)::text)) =
'M600'::text))"
"Total runtime: 0.135 ms"
```

Figura 11: Consulta con índice compuesto

En esta ocasión, el planificador utiliza el índice compuesto antes creado, que significa acceder del modo más eficiente posible. En este caso la consulta se realiza en tan sólo 0.135ms

Otra manera de optimizar la aplicación basada en PL/SQL es optimizar la clausuras de las consultas, reduciendo en la medida de lo posible el uso de JOIN, y prestando especial atención a cómo se accede a los atributos en cada una de las selecciones. Como ejemplo, en Tiger Geocoder se utiliza la siguiente consulta SQL para localizar un lugar o "place" al final de una cadena de texto:

```
SELECT INTO tempInt count(*) FROM place_lookup
    WHERE texticregexeq (fullStreet, '(?i)' || name || '$');
```

Figura 12: Consulta SQL para TIGER

Esta consulta es ineficiente en PostgreSQL, pues no aprovecha el indexado de la columna "name", por usar la función "*texticregexeq()*" que no puede ser indexada debido a que su resultado depende de un valor de entrada, forzando así a una búsqueda secuencial y desaprovecha los índices.

Geocoder para CartoCiudad utiliza una consulta equivalente que sí aprovecha el uso de indexado, tanto con PostgreSQL como con Oracle, mediante el operador LIKE:

```
SELECT INTO tempInt count(*) FROM municipio_lookup
    WHERE direccion LIKE '%' || municipio;
```

Figura 13: Consulta SQL optimizada para CartoCiudad

De esta manera, se ha comprobado que se puede llegar a obtener una mejora en el tiempo de consulta de 10 a 1 (90ms contra 900ms que se obtenía en el TIGER Geocoder original), para un número equivalente de tuplas en ambos casos.

2.5 INTERFACE DE APLICACIÓN WPS

Una vez desarrolladas y compiladas las funciones de Geocoder, tanto *PLpg/SQL* en PostgreSQL, como PL/SQL en Oracle, el uso del Geocoder se reduce a invocar la función "*geocoder(dirección)"*, siendo el parámetro de entrada la dirección a buscar. Esta función devuelve un puntero o cursor, donde se almacena el resultado. En cualquier caso, el uso de una base de datos precisa disponer de un usuario/role y su contraseña para acceder al SGBD Oracle ó PostgreSQL, de modo que dicho usuario disponga de los pertinentes permisos de acceso a los datos y la ejecución de la función que realiza la consulta.

Para estandarizar su uso, y facilitar que terceras aplicaciones lo utilicen, se implementa una capa de aplicación (séptimo nivel del modelo OSI [11]) basada en el estándar WPS (*Web Processing Service*) conforme con el *Open Geospatial Consortium* (OGC) [12], para la ejecución de procesos en un entorno cliente-servidor. De esta manera, Geocoder para CartoCiudad puede estar disponible como un servicio en Internet, y cualquier cliente WPS puede efectuar consultas y recibir respuestas estandarizadas mediante los protocolo de comunicación HTTP GET, HTTP POST, y SOAP.

Esta capa habilita una comunicación cliente-servidor mediante la llamada a tres métodos u operaciones diferentes, cuya función brevemente se describe a continuación:

- **getCapabilities():** el cliente obtiene un listado de los diferentes servicios proporcionados por el servidor WPS.

- **describeProcess():** el cliente solicita información acerca de un servicio en concreto, cuyo identificador se pasa como parámetro en la llamada. El servidor envía como respuesta un identificador del servicio, su descripción, así como los parámetros tanto de entrada como de salida que utiliza este servicio.

- **execute():** el cliente invoca a un servicio concreto, pasando los parámetros adecuados, comos se muestra en la figura 14. El servidor procesa la petición y envía una respuesta al cliente.

Para la implementación de la capa de aplicación se ha seleccionado el *framework* desarrollado en Java por 52°North y se ha utilizado el entorno de desarrollo integrado de código abierto multiplataforma Eclipse 3.4 (Galileo). En esta herramienta se declara e implementa la clase pública *Geocoder* como clase hija de la clase abstracta *AbstractAlgorithm*, heredando sus atributos y métodos disponibles en el frameworky que nos permitirá recibir los valores de los parámetros de entrada y enviar los resultados una vez se haya procesado la consulta. La clase *GeoCoder* tendrá definidos como atributos privados, los parámetros de conexión a la base de datos (host, usuario, contraseña, etc…), así como el nombre de las columnas utilizadas en la respuesta dada por la base de datos, que serán declarados a partir de un fichero de configuración en formato XML.

Name and example	**Mandatory/ Optional**	**Definition and format**
service=WPS	M	Service type identifier
request= Execute	M	Operation name
version=1.0.0	M	Specification and schema version for this operation
identifier= org.n52.wps.server.algorithm.GeoCoder	M	Process identifier
DataInputs= direccion=C/Mayor 23 Madrid, calidad=9999, exacto=0, maximo=10	O	List of titles and URL references to values of inputs to this process execution, comma separated. Each input shall be recorded as a pair of DataInputs values, with the first value being the input Identifier, and the second value being either the input value, or the URL of that (complex) input value.

Figura 14: Parámetros URL para la operación Execute

Esta clase recibe, invocando al método *run()* que pertene a la clase pública *DefaultTransactionalAlgorithm,* unos parámetros de entrada, como la dirección a buscar, y otros parámetros opcionales como el número máximo de direcciones devueltas, un valor numérico que representa el mínimo de calidad o precisión esperado en la consulta difusa, así como un valor booleano que indique si se inhabilita o no la búsqueda difusa.
Con los parámetros de entrada convenientemente procesados, se realiza una conexión al SGBD donde se lanza la consulta SQL, llamando por defecto a la función de PL/SQL *geocode_address(),* bien llamando a *geocode_address_place_exact()* si el cliente solicita sólo una búsqueda exacta. El trabajo de geocodificación por tanto se realiza dentro del SGBD, ya que WPS únicamente cumple la tarea de capa intermedia entre cliente y servicio de geocodificación.

Una vez efectuada la consulta en la base de datos, como parámetros de salida, la clase *GeoCoder* envía una colección de datos alfanuméricos que hay que transformar al formato de salida seleccionado por el usuario entre los ofrecidos por el servicio. Para ello se define una colección de entidades espaciales (*FeatureCollection) cuyo tipo contiene* el nombre de los viales coincidentes con la dirección buscada, incluida la información del tipo de vial, municipio, provincia, código postal, la localización geográfica definida en un sistema de referencia espacial identificado por su código EPSG y finalmenteun estimador numérico de la calidad de la respuesta cuando se ha solicitado una búsqueda difusa. El indicador de calidad adopta valores próximos a "cero" (0) cuando la calidad es máxima.

Finalmente el método *run()* de la clase geocoder genera la respuesta en forma de documento XML conteniendo la gramática definida en la especificación.

Un ejemplo de información espacial devuelta por la clase pública *Geocoder* declarada en el framework de 52ºNorth, a través del método *execute*() de la interface WPService, se puede ver en la figura 15.

```
- <gml:boundedBy>
  - <gml:Box srsName="EPSG:4258">
      <gml:coordinates cs="," decimal="." ts="">-3.713963452502963,40.381576242907855 -
        3.62395247178935,40.488858696846364</gml:coordinates>
    </gml:Box>
  </gml:boundedBy>
- <gml:featureMember>
  - <GeoCoder:GeoCoder fid="0">
      <GeoCoder:tip_via>Calle</GeoCoder:tip_via>
      <GeoCoder:nom_via>Princesa</GeoCoder:nom_via>
      <GeoCoder:municipio>Madrid</GeoCoder:municipio>
      <GeoCoder:provincia>Madrid</GeoCoder:provincia>
      <GeoCoder:zip>28008</GeoCoder:zip>
      <GeoCoder:rating>30</GeoCoder:rating>
    - <GeoCoder:address_geom>
      - <gml:Point srsName="EPSG:4258">
          <gml:coordinates cs="," decimal="." ts="">-3.713963452502963,40.427719583790754</gml:coordinates>
        </gml:Point>
      </GeoCoder:address_geom>
    </GeoCoder:GeoCoder>
  </gml:featureMember>
- <gml:featureMember>
  - <GeoCoder:GeoCoder fid="1">
      <GeoCoder:tip_via>Calle</GeoCoder:tip_via>
      <GeoCoder:nom_via>Princesa de eboli</GeoCoder:nom_via>
      <GeoCoder:municipio>Madrid</GeoCoder:municipio>
      <GeoCoder:provincia>Madrid</GeoCoder:provincia>
      <GeoCoder:zip>28050</GeoCoder:zip>
      <GeoCoder:rating>50</GeoCoder:rating>
    - <GeoCoder:address_geom>
      - <gml:Point srsName="EPSG:4258">
          <gml:coordinates cs="," decimal="." ts="">-3.65740387871663,40.4868432623547</gml:coordinates>
        </gml:Point>
      </GeoCoder:address_geom>
    </GeoCoder:GeoCoder>
  </gml:featureMember>
```

Figura 15: Fragmento de respuesta de WPS para execute()

El servicio WPS implementado, al tratarse de un *Java Servlet*, es multiplataforma, por lo que se puede desplegar como servicio en cualquier servidor web que maneje *servlets* (Ej: Apache Tomcat, Jetty, JBoss, etc…) independientemente del sistema operativo utilizado en la máquina que cumpla el rol de servidor (Microsoft Windows, GNU/Linux, MacOSX, FreeBSD, Solaris, etc…). Únicamente se requiere tener una máquina virtual Java (JVM) instalada en el sistema (versión 1.6.0_01-b0 ó superior), así como las bibliotecas adecuadas para conectarse como cliente al SGBD utilizado, *postgresql-*.jdbc4.jar* para PostgreSQL, o bien *ojdbc14.jar* en el caso de Oracle.

3. CONCLUSIONES

Mediante los métodos de la reingeniería del software, se pueden reutilizar aplicaciones *Open Source* como *TIGER-Geocoder* en otros desarrollos y por otros equipos ajenos a la aplicación original, pudiendo ser readaptado los algoritmos originales, y aplicarlos para la geocodificación de direcciones en otros proyectos y entornos de SGBD como el caso expuesto de CartoCiudad y Oracle.

Una especificación de requisito fundamental para un buscador, en este caso un Geocoder, es minimizar al máximo el tiempo de respuesta. Para ello, se deben de crear tantos índices, simples y compuestos como sean necesarios utilizando las mismas funciones que se utilicen en los algoritmos de búsqueda. Además se debe de prestar especial interés en buscar consultas SQL optimizadas, que aprovechen al máximo los índices previamente usados, analizando para ello la planificación del SGBD para dicha consulta. Una adecuada elección y uso de los índices, produce una espectacular mejora de rendimiento.
Otra alternativa para mejorar el rendimiento de una base de datos relacional, es romper la normalización de su esquema relacional, si bien esta práctica está totalmente desaconsejada en una base de datos que esté sujeta a constantes actualizaciones en tiempo real, dado que entonces no se podría garantizar la integridad de los datos almacenados. Para futuras versiones que consideren la relación 1:N entre código postal y portal_pk y evitar la necesidad de un procesado previo, así como que también se necesite garantizar la preservación de integridad en actualizaciones, modificaciones y borrados de tuplas en tiempo real, se recomienda encarecidamente el cumplimiento de FNBC para la base de datos Geocoder CartoCiudad. No obstante, como solución intermedia, se puede mantener la tabla des normalizada "*vial_lookup*" como una especie de caché utilizable en cada consulta, que se podría actualizar mediante un proceso en *background* de baja prioridad en el planificador del sistema operativo.

La geocodificación tiene como principal inconveniente la falta de normalización en los valores de entrada, por lo que la propia aplicación debe asumir esta tarea, con la dificultad añadida de tener que tratar una variabilidad de direcciones ambiguas o "ruido" que dan lugar a errores en la automatización de la extracción de la dirección. Dado que aún no se conoce la forma de eliminar completamente estos errores, se procurará minimizarlos en la medida de lo posible.

Una buena estrategia para proporcionar a la aplicación cierta tolerancia a errores, y flexibilidad en la búsqueda, es implementar una estrategia de búsqueda difusa basada en algoritmos fonéticos y función heurística de calidad. No obstante, esta estrategia de búsqueda tiene limitaciones y puede proporcionar falsos positivos. En estos casos se debe dejar abierta al usuario la posibilidad de inhabilitar este tipo de búsqueda, a favor de otras más conservadoras como la búsqueda exacta.

Para añadir compatibilidad a la aplicación, se precisa añadir una capa de aplicación como la especificada para el procesamiento por OGC (WPS), lo que facilita su interoperabilidad con terceras aplicaciones conformes con dicho estándar. Igualmente el desarrollo de esta capa en una tecnología multiplataforma como Java, facilita su despliegue en multitud de plataformas diferentes. El desarrollo también deja abierta la posibilidad de utilizar como SGBD tanto PostgreSQL como Oracle, que pueden funcionar en sistemas operativos diferentes.

4. AGRADECIMIENTOS

La tecnología desarrollada en este trabajo ha sido parcialmente soportada por el Instituto Geográfico Nacional a través de las colaboraciones enmarcadas en el proyecto CartoCiudad durante los años (2005-2007).

5. REFERENCIAS

[1] Página oficial del Proyecto CartoCiudad (http://www.cartociudad.es/portal/)

[2] Elliot J. Chikofsky, James H. Cross II, "Reverse Engineering and Design Recovery: A Taxonomy," IEEE Software, vol. 7, no. 1, pp. 15, Jan./Feb. 1990, doi:10.1109/52.43044

[3] Codd, E.F. "Further Normalization of the Data Base Relational Model." (Presented at Courant Computer Science Symposia Series 6, "Data Base Systems," New York City, May 24th-25th, 1971.) IBM Research Report RJ909 (August 31st, 1971). Republished in Randall J. Rustin (ed.), Data Base Systems: Courant Computer Science Symposia Series 6. Prentice-Hall, 1972

[4] Codd, E. F. "Recent Investigations into Relational Data Base Systems." IBM Research Report RJ1385 (April 23rd, 1974). Republished in "Proc. 1974 Congress" (Stockholm, Sweden, 1974). New York, N.Y.: North-Holland (1974).]

[5] Hurtado Melero, Tomás. "Ingeniería de Software". Departamento de Publicaciones de la Escuela Universitaria de Informática de Madrid. ISBN: 84-87238-32-7

[6] Spinak, E. "Errores Ortográficos en el ingreso de Bases de Datos". Revista española de documentación científica, 18, (3), 307-319, 1995.

[7] Harries, K., Mapping Crime: Principles and Practice (Washington: US Department of Justice). 1999.

[8] Ratcliffe JH: On the accuracy of TIGER-type geocoded address data in relation to cadastral and census areal units. Int J Geogr Inf Sci 2001

[9] Mark Sullivan, Michael A. Olson. "An Index Implementation Supporting Fast Recovery for the POSTGRES Storage System" ICDE 1992.

[10] P. Lehman, S. Yao. "Efficient Locking for Concurrent Operations on B-trees," ACM Trans. on Database Systems, 6(4), Diciembre 1981.

[11] ISO/IEC 7498-1 Second Edition 15/11/1994

[12] Peter Schut, Petter. (ed.): OpenGIS® Web Processing Service 1.0.0, OGC. Document OGC 05-007r7, http://www.opengeospatial.org/standards/wps (2007).

6. CONTACTOS

Iván Moya Honduvilla
Ivan81k@gmail.com
LatinGEO
Universidad Politécnica de Madrid

Miguel Ángel Manso Callejo
m.manso@upm.es
ETSI en Topografía, UPM
Dpto. Ing. Topográfica y Cartográfica