WPS:GeoCoder basado en CartoCiudad y en tecnología OpenSource

Servicio de geocodificación basado en CartoCiudad y en libpostal, PostgreSQL y 52N-WPS

MOYA HONDUVILLA, Iván; MANSO-CALLEJO, Miguel Ángel; MOYA HONDUVILLA, Javier

La geocodificación es el proceso de traducción de direcciones o intersecciones en coordenadas -localización directa- (ISO 19133:2005), lo que constituye un proceso clave para el tratamiento de información georreferenciada mediante identificadores y direcciones postales y es una necesidad básica para muchas personas y empresas en su actividad. La geocodificación se basa en la existencia de bases de datos que almacenan la información disponible sobre viales, portales y puntos kilométricos, así como las divisiones administrativas y postales.

Sin embargo, la primera dificultad a la que se enfrentan los geocodificadores es la pluralidad de formas de escribir direcciones, seguida de las referencias ambiguas, abreviaturas y contextos locales. Existen numerosos servicios de geocodificación proporcionados por empresas tecnológicas: Google, Bing, Yahoo, otros basados en cartografía colaborativa (OpenStreetMap) y finalmente otros basados en la información oficial de los organismos cartográficos.

En consecuencia, se plantea la necesidad de trabajar en la implementación y desarrollo de un geocodificador que alcance una alta tasa de aciertos. Este propósito se ha materializado con el uso de la biblioteca Libpostal como normalizador y *parser* de direcciones mundiales de calles, utilizando procesamiento de lenguajes naturales y datos de libre disposición, como OpenStreetMap. Debido a que Libpostal no es un geocodificador completo, éste se complementa con una serie de algoritmos escritos en PL/PgSQL para PostgreSQL, implementados específicamente para el modelo y la información de la base de datos CartoCiudad. Si bien el diseño modular de las funciones PL/PgSQL permite adaptar este trabajo a otras bases de datos, sin importar el país o el idioma en el que se basen. A partir de la base de datos CartoCiudad, publicada en el centro de descargas del Centro Nacional de Información Geográfica, se aborda la optimización de la velocidad de respuesta en las búsquedas.

El algoritmo de geocodificación implementado proporciona tolerancia a errores tipográficos en los valores de entrada y está dotado de capacidad para resolver ambigüedades en ciertas búsquedas. Para ello, además de utilizar el normalizador Libpostal en la geocodificación, se emplea búsqueda difusa mediante un algoritmo fonético con una función heurística de calidad. En caso de no poder satisfacer una búsqueda exacta, se proporcionen alternativas, ordenadas decrecientemente por grado de coincidencia, junto a la georreferencia exacta tomada de la base de datos, o aproximada por un algoritmo de interpolación.

Para potenciar la interoperabilidad, en un entorno de Infraestructura de Datos Espaciales, se ha integrado su desarrollo en un Web Processing Service (WPS) conforme con la especificación *(05-007r7)* del *Open Geospatial Consortium* (OGC) y para su implementación se ha utilizado el *framework* WPS de 52°North.

Finalmente, se ha adaptado el geoportal GET SDI Portal diseñado conforme a la Directiva INSPIRE (2007/2/EC) con el fin de mostrar la integración del WPS y los resultados de las geocodificaciones en un mapa, intercambiando datos en formato JSON entre cliente y servidor.

PALABRAS CLAVE

Geocoder, CartoCiudad, libpostal, parser, PostgreSQL, PostGIS, PL/PgSQL, 52N-WPS, JSON, Geoportal

INTRODUCCIÓN

La geocodificación directa es el proceso que relaciona direcciones postales con coordenadas geográficas específicas [1] que son susceptibles de ser utilizadas para ubicar cosas en un mapa o Servicio de Mapas Web (WMS). El direccionamiento de calles es una cuestión común para la vida cotidiana de los ciudadanos, pero también para administraciones y empresas: el uso de direcciones en el ámbito digital facilita extraordinariamente la gestión eficiente de servicios, impuestos y actividades económicas. A pesar de su presencia cada vez más habitual en la sociedad, los geocodificadores siguen planteando el reto de hacer frente a la inevitable informalidad que los usuarios muestran a la hora de buscar direcciones; cabe citar las referencias ambiguas, las abreviaturas y el contexto cultural y local, entre otros.

Por todo lo anterior, se plantea la necesidad de trabajar en la implementación y desarrollo de un geocodificador mejorado cuyo objetivo principal sea alcanzar una elevada tasa de aciertos. Este es el propósito se pretende materializar con una serie de algoritmos implementados específicamente para el sistema gestor de base de datos PostgreSQL, utilizando la información oficial y de libre disposición ciudadana de la base de datos CartoCiudad [2]. Adicionalmente, para posibilitar un procesamiento masivo de direcciones postales a geocodificar, así como garantizar una razonable velocidad de respuesta, también se abordará la optimización de las consultas realizadas contra la base de datos.

CASUISTICAS DE LA NORMALIZACIÓN DE DIRECCIONES POSTALES

Las direcciones son los identificadores que las personas utilizamos para describir la localización de lugares y se dan en prácticamente todas las facetas de la vida moderna conectada a Internet: búsqueda de mapas, rutas, envíos, empresas de transporte de mercancías, servicios de paquetería, viajes y alojamiento en hoteles, etc. [3]

El primer obstáculo al que debe enfrentarse una máquina para geocodificar una dirección reside en su normalización, para poder ser procesada de forma automatizada. La tarea de normalización consiste en transformar las direcciones de lenguaje natural que usamos las personas para describir lugares, en una estructura de datos bien definida compuesta por los elementos clave necesarios para la geocodificación. En los servicios de búsqueda de direcciones, es habitual que la falta de normalización en los valores de entrada dificulte la ejecución de consultas en las bases de datos o que éstas sean satisfactorias. Este hecho se convierte en una de las principales limitaciones de cualquier buscador, cuyo fundamento básico son las consultas contra una base de datos relacional, como es el caso del geocodificador propuesto en este trabajo.

Las direcciones buscadas de forma confusa son típicamente cadenas cortas, ambiguas y, con frecuencia, repletas de abreviaturas y diferentes variedades lingüísticas de origen local, incluso dentro de un mismo idioma. Este contexto de cosas puede llegar a confundir en mayor o menor grado al algoritmo de normalización. Por ejemplo, las abreviaturas como 'Plz' (plaza) y 'St.' (santa) deberían ser tenidas en cuenta. Por otra parte, desde la perspectiva de un único usuario, normalmente sólo hay una única respuesta correcta a una consulta (con la excepción de búsquedas más amplias del tipo 'restaurantes en una determinada plaza'). En ocasiones, puede que ni siquiera deba considerarse la interacción con el usuario, como por ejemplo en casos donde sea necesario geocodificar por lotes (procesos de tipo *batch*) [4] miles de direcciones contenidas en, por ejemplo, un archivo de texto CSV [5], una página Web o una API de terceros.

Una vez normalizada una dirección postal, el problema de la geocodificación se convierte en un problema de comparación de cadenas de texto que, a priori, resulta más sencillo de abordar. Sin embargo, puede ocurrir que, por un error tipográfico o por error de la propia normalización, se intente geocodificar una dirección que, simplemente, no existe bien en la realidad, o bien en la base

de datos. Para minimizar en la medida de lo posible la tasa de búsquedas fallidas, es recomendable proporcionar al algoritmo de geocodificación una cierta tolerancia a errores, que permita a este indagar alternativamente direcciones de fonética similar.

NORMALIZACIÓN DE DIRECCIONES POSTALES. LIBPOSTAL

El objetivo del proyecto Libpostal [6] es la elaboración cooperativa de una librería o biblioteca especializada en la normalización (*parsing*) de direcciones de calles que puedan estar ubicadas en cualquier rincón del planeta. La librería, escrita en el lenguaje de programación ANSI C, basa su fundamento en el Procesamiento de Lenguaje Natural (PLN). Este término hace referencia a una gama de técnicas informáticas ideadas para analizar y representar textos naturales en uno o más niveles de análisis lingüístico, con el propósito de acercarse lo más posible a un estilo humano de procesamiento de lenguaje, con vistas a aplicarlo a una amplia variedad de tareas o aplicaciones [7], [8].

Aprovechando la filosofía del procesamiento del lenguaje natural y el uso de datos de libre disposición pública como OpenStreetMap, el proyecto Libpostal trata de discernir la mejor forma de normalizar cualquier tipo de dirección, de cualquier país y escrito en cualquier idioma [9], [10]. Otra de las características es el uso de licencia MIT [11], un tipo de licencia de software libre permisiva y con pocas limitaciones de uso, por lo que resulta adecuada para fines académicos o comerciales. Dada la relativamente extensa comunidad de Libpostal, en este trabajo se ha apostado por esta biblioteca para perseguir una óptima tasa de aciertos a la hora de geolocalizar una dirección.

En este trabajo, Libpostal es utilizado como módulo o biblioteca que acompaña a un geocodificador para la base de datos CartoCiudad [12]. Para su integración se ha utilizado una extensión para PostgreSQL llamada pgpsql-postal, escrita también en ANSI C igualmente bajo licencia MIT.

GEOCODIFICACIÓN DE DIRECCIONES POSTALES

Una vez obtenida la dirección normalizada o estructurada en el paso anterior, el siguiente objetivo es definir un algoritmo de búsqueda en función de la información disponible y, en su caso, en función del grado de éxito que se obtenga al realizar la búsqueda contra la base de datos.

Es posible que no se obtenga ningún resultado con la información dada, debido bien a un error tipográfico, o simplemente porque el nombre referido en la búsqueda no coincida exactamente con el contenido en la base de datos. En este caso, el algoritmo explora diferentes tentativas con búsqueda exacta (que coincida exactamente la palabra o palabras buscadas con la base de datos), y en caso de no lograr un resultado adecuado, opta por la búsqueda difusa, tomando en consideración palabras similares desde el punto de vista fonético. Además, en caso de localizar un vial coincidente sin posibilidad de asociarlo con un portal concreto almacenado en la base de datos, como forma alternativa el algoritmo, aplica un interpolado del tramo para tratar de ubicar un portal ficticio, lo más aproximado posible a la ubicación real.

Como punto de partida para el desarrollo del servicio de geocodificación, se ha adoptado el equivalente ya desarrollado para la base de datos TIGER de la Oficina del Censo de los Estados Unidos [13]. El geocodificador de TIGER, está desarrollado en el lenguaje procedural *(PL)* para el Sistema Gestor de Base de Datos PostgreSQL, y utiliza funciones espaciales a través de la extensión PostGIS.

Para reutilizar sobre CartoCiudad los desarrollos disponibles de geocodificación del proyecto TIGER, 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 la definen como *'el análisis y modificación de un sistema, para reconstituirlo en una nueva forma'* [14]. 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.

La fase de ingeniería inversa para Geocoder TIGER, se realiza con la finalidad de identificar el uso los atributos 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 en el modelo de datos de CartoCiudad.

En síntesis, la labor realizada en este punto ha consistido en analizar el modelo de datos de TIGER, buscando posibles similitudes para lograr adaptar el algoritmo de geocodificación para la base de datos CartoCiudad. En dicho análisis se ha constatado que la tabla *'tiger_geocode_roads'* definida en Geocoder-TIGER y representada en la Figura 1 con algunas tuplas o filas de ejemplo, no satisface la tercera forma normal (3NF) definida por E.F.Codd [15], ya que existen atributos que dependen transitivamente de la clave, como se puede comprobar estudiando el Esquema Relacional de la tabla.

Data Output | Explain | Messages | History

id integer	tlid integer	fedirp charact...	fename character varying ...	fetype character v...	fedirs character v...	zip integer	state character var...	county character varying (90)	cousub character varying (90)	place character vary
1365732	142664522	W	7th	St	S	90813	CA	Los Angeles	Long Beach-Lakewood	Long Beach
1367479	142667528	E	Garvey	Ave	S	91791	CA	Los Angeles	East San Gabriel Valley	West Covina
1367657	142667761	E	Walnut	Dr	N	91748	CA	Los Angeles	East San Gabriel Valley	Industry
1367658	142667761	E	Walnut	Dr	N	91789	CA	Los Angeles	East San Gabriel Valley	Industry
1371227	142674427	E	Walnut	Dr	N	91748	CA	Los Angeles	East San Gabriel Valley	Diamond Bar
1371228	142674427	E	Walnut	Dr	N	91789	CA	Los Angeles	East San Gabriel Valley	Industry
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
1247381	141692990	W	7th	St	S	90813	CA	Los Angeles	Long Beach-Lakewood	Long Beach
1247382	141692991	W	7th	St	N	90813	CA	Los Angeles	Long Beach-Lakewood	Long Beach
1248343	141694035	N	Pacific	Ave	W	90802	CA	Los Angeles	Long Beach-Lakewood	Long Beach
1248344	141694037	N	Pacific	Ave	W	90802	CA	Los Angeles	Long Beach-Lakewood	Long Beach
1261362	141709492	E	Garvey	Ave	S	91791	CA	Los Angeles	East San Gabriel Valley	West Covina
1263052	141711394	W	Garvey	Ave	S	91706	CA	Los Angeles	East San Gabriel Valley	Baldwin Park
1263053	141711394	W	Garvey	Ave	S	91790	CA	Los Angeles	East San Gabriel Valley	Baldwin Park

Figura 1: Tabla denominada *'tiger_geocode_roads'*. Extracto.

A continuación, se ejemplifica un subconjunto de dependencias funcionales donde se da lugar la dependencia transitiva.

T = {id,tlid,fedirp,fename,fetype,fedirs,zip,state,country,cousub,place}
L1 = {id-> place; place –> state; place –> county; place –> cousub}

("state", "country" y "cousub" dependen transitivamente de la clave "id")

Siendo:

T *= Conjunto de atributos de R.*
L1 *= Subconjunto de dependencias funcionales de R.*
R *= Esquema Relacional deducido de la tabla "tiger_geocode_roads"*
$L1 \subset L / R(T,L) \notin 3FN$

Figura 2: Subconjunto de dependencias funcionales.

Esta dependencia transitiva puede provocar anomalías en la base de datos si, además de estar diseñada para su explotación con el servicio Geocoder, se pretende realizar operaciones de 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.

El esquema relacional extraído de la tabla *'codpostal_vial'* está ideado para el Geocoder de CartoCiudad de forma análoga a Geocoder-TIGER. Inspirado a partir de la tabla *'tiger_geocode_roads'*, este esquema no satisface la tercera forma normal (3FN), por dependencia transitiva entre clave y atributo, ni tampoco la segunda forma normal (2FN). Sin embargo, tanto la tabla *'tiger_geocode_roads'* como la tabla *'codpostal_vial' (esta última generada a partir de los datos originales de CartoCiudad)* están pensadas para mantener en el tiempo sus datos sin cambios durante semanas e incluso meses, debiendo de ser generadas previamente a partir de la base de datos TIGER y CartoCiudad, respectivamente, cuando se añadan o modifiquen viales.

En concreto, *'codpostal_vial'* tiene como valor añadido integrar una relación implícita N:M entre *'codigo_postal'* y *'tramo_vial'* por un lado, además de incluir municipio y provincia, algo de lo que carecía CartoCiudad cuando se concibió este trabajo, tanto en sus especificaciones oficiales como en los datos disponibles al público. No se han tenido en cuenta cambios posteriores que pudieran obligar a modificar la metodología de este trabajo. La principal justificación para desnormalizar la base de datos reside en incluir la relación directa N:M entre *'tramo_vial'* y *'codigo_postal'*, que no está incluida directamente en la base de datos proporcionada por CartoCiudad. De esta manera, se puede optimizar una búsqueda, acotando su rango si se proporciona el código postal. Además, se contaría con la ventaja añadida de no aumentar el coste de acceso a la información con el uso del operador de álgebra relacional Unión Natural *(JOIN)*.

En la Figura 3 se muestra un extracto de la tabla *'codpostal_vial'* propuesta para este trabajo, derivada de la base de datos CartoCiudad.

gid [PK] integer	id_vial numeric	id_cp numeric	id_mun numeric	cod_postal character v...	tipo... sma...	tip_via_in character var...	nom_via character varying (100)	nom_altern character varying ...	nom_municipio character (100)	provinc charact
667338	010000063	010000002	010000001	01240	1	CUSTA	La Fortaleza	-998	Alegría-Dulantzi	Alava
667339	010000064	010000002	010000001	01240	1	CUSTA	Las Cabras	-998	Alegría-Dulantzi	Alava
667340	010000065	010000002	010000001	01240	1	PARKE	Solandia	-998	Alegría-Dulantzi	Alava
667341	010000066	010000001	010000001	01193	1	PLAZA	Herriko	-998	Alegría-Dulantzi	Alava
667342	010000066	010000002	010000001	01240	1	PLAZA	Herriko	-998	Alegría-Dulantzi	Alava
667343	020000001	020000002	020000001	01470	1	ALAM	Salbador	-998	Amurrio	Alava
667344	020000002	020000002	020000001	01470	1	AVDA	Done Erroke	-998	Amurrio	Alava
667345	020000003	020000002	020000001	01470	1	BARRI	San Jose	-998	Amurrio	Alava
667346	020000004	020000002	020000001	01470	1	CALLE	Abiaga	-998	Amurrio	Alava
667347	020000005	020000002	020000001	01470	1	CALLE	Abiagabarri	-998	Amurrio	Alava
667348	020000006	020000002	020000001	01470	1	CALLE	Adarraga	-998	Amurrio	Alava
667349	020000007	020000002	020000001	01470	1	CALLE	Aiara	-998	Amurrio	Alava
667350	020000008	020000002	020000001	01470	1	CALLE	Aldai	-998	Amurrio	Alava
667351	020000009	020000002	020000001	01470	1	CALLE	Aldaiondo	-998	Amurrio	Alava

Figura 3: Tabla denominada *'codpostal_vial'*. Extracto.

Para generar la tabla de búsqueda desnormalizada *'codpostal_vial'*, se analiza la situación inicial resumida en el modelo Entidad-Relación mostrado en la Figura 4, donde puede observarse la relación N:1 entre *'portal_pk'* y *'tramo_vial'*, y a su vez, otra relación N:1 entre *'portal_pk'* y *'codigo_postal'*, de la siguiente forma:

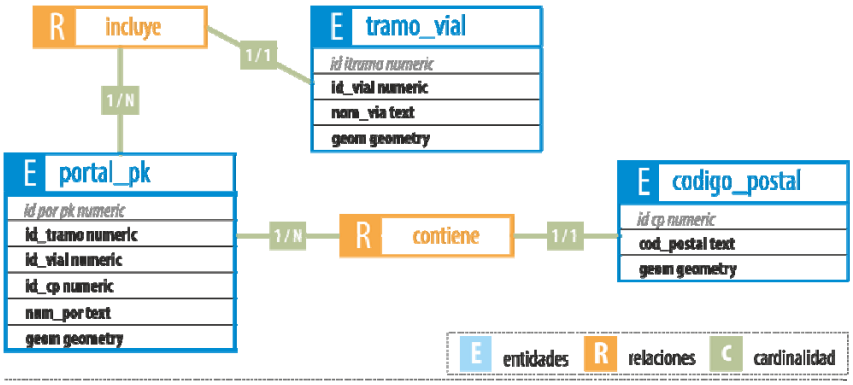

Figura 4: Modelo Entidad-Relación. Extracto.

De dicha relación se deduce que:

- *'id_tramo'* contenido en *'portal_pk'*, es clave foránea de *'tramo_vial'*.
- *'id_cp'* contenido en *'portal_pk'*, es clave foránea de *'codigo_postal'*.

Sin embargo, se ha detectado que existen viales sin portales asociados en CartoCiudad que, a pesar de ello, disponen de direcciones para geocodificar. Aunque no existe una relación en el modelo E-R de tramos asociados a códigos postales, existe la posibilidad de hacer una correlación directa utilizando la función espacial *'ST_Within (geometry1 st_geometry, geometry2 st_geometry)'* de PostGIS, que devuelve TRUE si el primer objeto *'ST_Geometry'* está completamente inscrito en el segundo; de lo contrario, devuelve FALSE. Por ello, se desecha la idea de crear la nueva tabla a partir de *'portal_pk'*.

Por todo lo anterior, la solución propuesta para esta problemática consiste en crear una nueva tabla denominada *'codpostal_vial'* que cumpla con las siguientes premisas:

- Agrupar todos viales de la tabla *'tramo_vial'*, dado que un vial contiene uno o varios tramos.
- Establecer una relación entre *'tramo_vial'* y *'codigo_postal'*, utilizando para ello la relación espacial de *'ST_Within (tramo_vial.geom, codigo_postal.geom)'*.
- Incluir una relación directa entre vial y municipio, además de provincia, de manera que, para realizar una búsqueda acotada por municipio o provincia, no sea necesario aumentar el coste de la consulta con álgebra relacional Unión Natural *(JOIN)*.

La declaración SQL de la nueva tabla será la siguiente:

```
CREATE TABLE codpostal_vial (
        gid numeric,
        id_vial numeric,
        id_cp numeric,
        id_num numeric,
        cod_postal character varying(5),
        tipo_via smallint,
        tip_via_in character varying(25),
        nom_via character varying(100),
        nom_altern character varying(100),
        nom_municipio character varying(100),
        provincia character varying(100),
CONSTRAINT codpostal_vial_pkey PRIMARY KEY(gid)
);
```

Figura 5: Declaración SQL de la tabla *'codpostal_vial'*.

Para generar los datos contenidos en ella, primero se modificará la tabla *'tramo_vial'* mediante las primeras sentencias de la consulta SQL de la figura 6, con el objetivo de añadir los atributos *'id_cp'* y *'cod_postal'*, y así lograr relacionar tramos con códigos postales. Aunque el atributo *'cod_postal'* sea información redundante, se incluye para facilitar la tarea de creación de la tabla *'codpostal_vial'*. Posteriormente, dichos atributos se completarán con la función espacial *'ST_Within'* de la siguiente forma:

```
ALTER TABLE tramo_vial ADD COLUMN cod_postal VARCHAR(5);
ALTER TABLE tramo_vial ADD COLUMN id_cp numeric;

UPDATE tramo_vial tv
SET (id_cp,cod_postal) = (SELECT id_cp,cod_postal FROM código_postal cp WHERE ST_Within (tv.geom, cp.geom) limit 1);
```

Figura 6: Encaje de la función espacial *'ST_Within'*.

Con el objetivo de optimizar las búsquedas y evitar en la medida de lo posible el uso de JOIN, se añade el atributo *'cod_postal'* relacionado con el atributo *'id_cp'*, y este último en la tabla resultante *'codpostal_vial'*. Seguidamente, se insertarán todos los viales contenidos en la tabla *'tramo_vial'* en la nueva tabla *'codpostal_vial'*, existiendo viales redundantes si alguno de ellos está incluido varios códigos postales. Inmediatamente después, se purgarán los viales sin *'id_vial'* vinculado a un código válido, como resultado de la adicción de portales a la base de datos. Dada la importancia que tienen para las búsquedas que serán necesarias en la geocodificación, el siguiente paso es importar los atributos *'tipo_via'*, *'tip_via_in'*, *'nom_via'* y *'nom_altern'* contenidos en la tabla *'tramo_vial'*, purgando las tuplas vacías, dada su irrelevancia para las búsquedas.

A continuación, se creará la clave foránea *'id_mun'*, que vincula viales con municipios. Si bien es cierto que en la copia a libre disposición al público de CartoCiudad no está incluida ninguna tabla de municipios, en su documentación [16] aparece reseñada una correlación directa entre viales y municipios. Se deduce así que la codificación de *'id_vial'* se crea para viales urbanos (incluidos rotondas y enlaces urbanos), a través de la siguiente fórmula: INE_MUN = 10.000.000 + secuencial, siendo INE_MUN el código de provincia y municipio del Instituto Nacional de Estadística (INE).

Como regla general, en CartoCiudad el código de todos los identificadores (id_*) será un número entero de doce cifras, que se forma como INE_MUN * 10.000.000 + secuencial, excepto en los casos del fenómeno vial para viales interurbanos, Provincia y Comunidad Autónoma, por no circunscribirse a un único municipio. Tenido esto en cuenta, se generará una nueva columna o atributo: id_mun; de forma análoga a nom_via, se crea el atributo nom_municipio, a partir de datos oficiales del INE.

En 1974 Raymond F. Boyce y Codd definieron una importante Forma Normal, denominada de Boyce-Codd (FNBC) [17], que además de cumplir la tercera forma normal (3FN), hace frente a ciertos tipos de anomalías en el resultado de consultas con JOIN, como la aparición de tuplas extrañas. Para un futuro cumplimiento de la misma, sería posible descomponer los esquemas ya contemplados y rehacer la aplicación PL/pgSQL mediante reingeniería, y en su caso, reescritura de las sentencias SQL implicadas en las consultas.

En la figura 7 se muestra el modelo Entidad-Relación resultante para la implementación de la base de datos necesaria para el geocodificador.

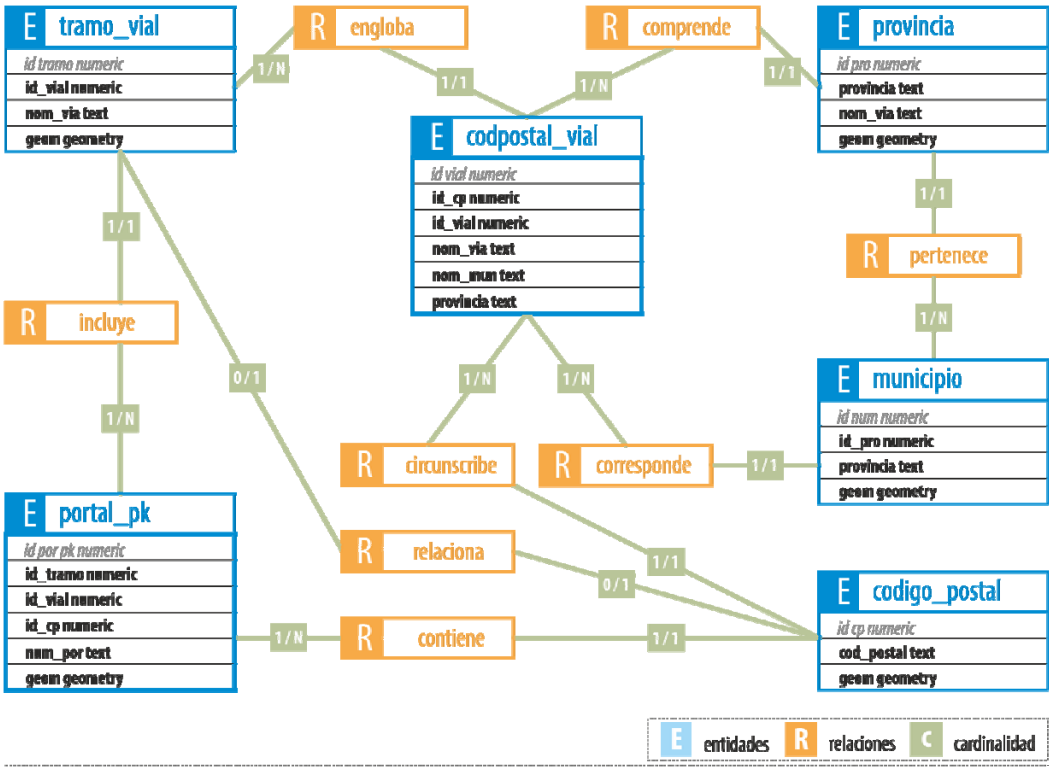

Figura 7: Modelo Entidad-Relación.

ALGORITMO

Cuando se llama a la función PL/pgSQL de geocodificación principal *'geocode_address'* el algoritmo de búsqueda, cuya implementación parcial se muestra en la figura 8, es el siguiente:

1. Si proporciona ninguna dirección, se termina el algoritmo sin devolver ninguna dirección, ya que éste es un campo obligatorio y no tiene sentido geocodificar nada sin él.

2. Si se proporciona un código postal, se llama a la función *'geocode_address_zip'*. En caso de éxito, se devuelve el resultado de esta consulta como dirección o direcciones geocodificada/s.

3. Si la anterior consulta no devuelve ninguna dirección geocodificada, y se ha proporcionado un nombre de municipio, se llama a la función *'geocode_address_place_exact'*. En caso de éxito, se devuelve el resultado de esta consulta.

4. Si la anterior consulta no devuelve ninguna dirección geocodificada, y se ha proporcionado un nombre de municipio, se llama a la función *'geocode_address_place_fuzzy'*. En caso de éxito, se devuelve el resultado de esta consulta.

5. Si la anterior consulta devuelve no devuelve ninguna dirección geocodificada, y se ha proporcionado un nombre de provincia, se llama a la función *'geocode_address_state'*. En caso de éxito, se devuelve el resultado de esta consulta.

```
OPEN result FOR
    SELECT
     cv.nom_via,
     cv.tip_via_in,
     consulta.portal as portal,
     cv.nom_municipio,
     cv.provincia,
     cv.cod_postal,
     rate_attributes(consulta.direccion, normaliza(cv.nom_via),
       consulta.tipoDireccion, normaliza(cv.tip_via_in),
       consulta.municipio,normaliza(cv.nom_municipio)) as rating,
     ST_X(interpolate_from_address(consulta.portal,CAST (tv.imp_bajo AS integer),CAST (tv.imp_alto AS integer),
                  CAST (tv.par_bajo AS integer),CAST (tv.par_alto AS integer),tv.geom)) as lon,
     ST_Y(interpolate_from_address(consulta.portal,CAST (tv.imp_bajo AS integer),CAST (tv.imp_alto AS integer),
                  CAST (tv.par_bajo AS integer),CAST (tv.par_alto AS integer),tv.geom)) as lat,
     'si' as interpolado,
     'si' as preciso
     FROM
      codpostal_vial cv, tramo_vial tv
     WHERE
     CASE WHEN consulta.provincia IS NOT NULL THEN
      (soundex(normaliza(cv.provincia)) = soundex(consulta.provincia)) ELSE TRUE
     END AND
      soundex(normaliza(cv.nom_municipio)) = soundex(consulta.municipio) AND
      soundex(normaliza(cv.nom_via)) = soundex(consulta.direccion) AND
      cv.id_vial = tv.id_vial AND
      cv.id_cp = tv.id_cp AND
      CASE WHEN consulta.portal % 2 = 0 THEN
       (tv.par_bajo <= consulta.portal AND tv.par_alto >= consulta.portal)
      ELSE
       (tv.imp_bajo <= consulta.portal AND tv.imp_alto >= consulta.portal)
      END
    ORDER BY rating;
    RETURN result;
```

Figura 8: Consulta SQL para geocodificar una dirección normalizada. Extracto.

Si todo lo anterior ha fracasado, no se devuelve ninguna dirección geocodificada. Adicionalmente si no se proporciona código postal, ni municipio, ni provincia, la búsqueda resultaría demasiado ambigua, por lo que este geocodificador no realizaría ninguna tentativa de búsqueda. En todo caso, cabe señalar que existen geocodificadores como el de Google Maps que devuelven una respuesta sin necesidad de introducir la localidad y/o provincia utilizando la ubicación del propio usuario en el mapa, eliminando así la ambigüedad de localización.

PROBLEMAS EN LA GEOCODIFICACION

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 [18], [19]. 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 los efectos cartográficos oficiales), no podría ser correctamente geocodificada hasta que sea dada de alta en la base de datos.
- Fallos en la propia fuente de la base de datos CartoCiudad. Puede existir información geográfica incorrecta o desactualizada (direcciones de reciente creación o en desarrollo, así como calles que por diversos motivos experimentan un cambio de nombre). Sin embargo, la dirección sí será normalizada, pues como se ha visto anteriormente, en dicho proceso no se verifica si existe realmente la calle.
- También ocurren diversos problemas con la normalización. Por ejemplo, si se introducen elementos ajenos al formato establecido (p. e. escalera, planta, puerta…), estos elementos pueden ser erróneamente interpretados como si fueran un portal.
- En otros casos, si se introduce una dirección en otro idioma co-oficial no contemplado en CartoCiudad, el normalizador de direcciones no será capaz de extraer el nombre de municipio ni provincia. En el caso de 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
- En direcciones cuya denominación está abreviada, y que no resulte siquiera similar fonéticamente a la denominación original introducida en la base de datos, no será correctamente geocodificada.

En definitiva, son múltiples las circunstancias por las que el geocodificador puede fallar en su función, sin que exista ninguna implementación conocida que solvente satisfactoriamente todas las circunstancias adversas anteriormente descritas.

OPTIMIZACION E INDEXADO

Conseguida la correcta funcionalidad del Geocoder, se aplican técnicas para el optimizado del 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. PostgreSQL utiliza para el indexado una implementación de árbol B+ (b-link tree) de alta concurrencia de Lehman-Yao. 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.

Se utiliza a continuación un índice para las columnas *'nom_via'*, utilizando las función *'soundex()'* (figura 9). 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.

```
CREATE INDEX codpostal_vial_soundex_idx
  ON codpostal_vial (sondex(normaliza(nom_via)));
```

Figura 9: Índice SQL con uso de función fonética.

Para complementar este índice simple, se crea otro índice compuesto (figura 10), formado por las tres columnas implicadas en la consulta SQL y aplicando las funciones de normalización y búsqueda difusa.

```
CREATE INDEX codpostal_vial_place_fuzzy_provincia_idx
  ON codpostal_vial
  (soundex(normaliza(provincia)),
  soundex(normaliza(nom_municipio)),
  soundex(normaliza(nom_via)),
  id_cp);
```

Figura 10: Índice SQL compuesto con distintos atributos.

Mediante la consulta EXPLAIN ANALYZE de PostgreSQL [20], se observa cómo su planificador de consultas SQL utiliza el índice compuesto recién creado, lo que significa acceder del modo más eficiente posible. En este caso, la consulta se realiza en tan sólo 0.202 ms, incrementándose en 1.079 ms adicionales para la planificación.

QUERY PLAN text
Aggregate (cost=468.15..468.16 rows=1 width=8) (actual time=0.151..0.151 rows=1 loops=1)
-> Nested Loop (cost=0.99..468.13 rows=8 width=0) (actual time=0.090..0.147 rows=3 loops=1)
-> Index Scan using codpostal_vial_place_fuzzy_provincia_idx on codpostal_vial cv (cost=0.42..8.45 rows=1 width=8) (actual time=0.061..0.064 rows=6 loops=1)
Index Cond: (('M363'::text = soundex(normaliza((provincia)::text))) AND ('A426'::text = soundex(normaliza((nom_municipio)::text))) AND ('M600'::text = soundex(...
-> Index Scan using portal_pk_vial_idx on portal_pk pp (cost=0.56..459.65 rows=3 width=8) (actual time=0.011..0.013 rows=1 loops=6)
Index Cond: (id_vial = cv.id_vial)
Filter: (num_por = 16)
Rows Removed by Filter: 34
Planning time: 1.079 ms
Execution time: 0.202 ms

Figura 11: Planificación de consulta para PostgreSQL

INTERFACE WPS

Una vez desarrolladas y compiladas las funciones de Geocoder en PL/pgSQL en PostgreSQL, el uso del Geocoder se reduce a invocar la función *'geocoder'*, siendo este 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/rol y su contraseña para acceder al SGBD Oracle o 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) basada en el estándar Web Processing Service (WPS) conforme con el Open Geospatial Consortium (OGC), 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 protocolos de comunicación HTTP GET, HTTP POST, y SOAP.

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 4.6 (Neon). En esta herramienta se declara e implementa la clase pública Geocoder como clase hija de la clase abstracta *'AbstractAlgorithm'*. Esta hereda sus atributos y métodos disponibles en el *framework*, lo que permite 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.

Parámetro	Uso	Definición
service = WPS	Obligatorio	Nombre del servicio OGC
request = Execute	Obligatorio	Nombre de la operación
version = 1.0.0	Obligatorio	Versión de la especificación
lenguaje = en-ES	Opcional	Idioma preferido de respuesta
identifier = es.upm.GeoCoder	Obligatorio	Identificador del proceso
datainputs = direccion=C/Mayor 23 Madrid; calidad = 9999; exacto = 0; maximo = 10	Opcional, cuando se especifica uno o más parámetros de entrada	Lista de identificadores, valores y atributos de entrada del proceso

Figura 12: Definición y uso de la operación *Execute* de WPS.

```
http://127.0.0.1/ows?Request=Execute&Service=WPS&version=1.0.0&identifier=es.upm.GeoCoder
&DataInputs=direccion=C/Mayor 23 Madrid; calidad=9999;exacto=0;maximo=10
```

Figura 13: URL con paso de parámetros GET para invocar la operación Execute.

Esta clase escrita en Java, que mediante el uso de 'extends' hereda métodos y objetos de la clase pública *'AbstractAlgorithm'*, recibe 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'*, o bien llamando a *'geocode_address_place_exact'* si a través de los parámetros de entrada se ha solicitado 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 conjunto de parámetros de salida, la clase GeoCoder envía una colección de datos alfanuméricos que debe transformarse 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 los siguientes elementos:

- La dirección buscada, incluida la información del tipo de vial
- El municipio
- La provincia
- El código postal
- La localización geográfica definida en un sistema de referencia espacial identificado por su código EPSG
- Un estimador numérico de la calidad de la respuesta cuando se ha solicitado una búsqueda difusa. El indicador de calidad adoptará valores próximos a "cero" (0) cuando la calidad es máxima.

Finalmente se genera una respuesta en forma de documento XML conteniendo la gramática definida en la especificación. Un ejemplo de información espacial devuelta por la clase Geocoder declarada en el *framework* de 52ºNorth, se puede ver en la figura 14.

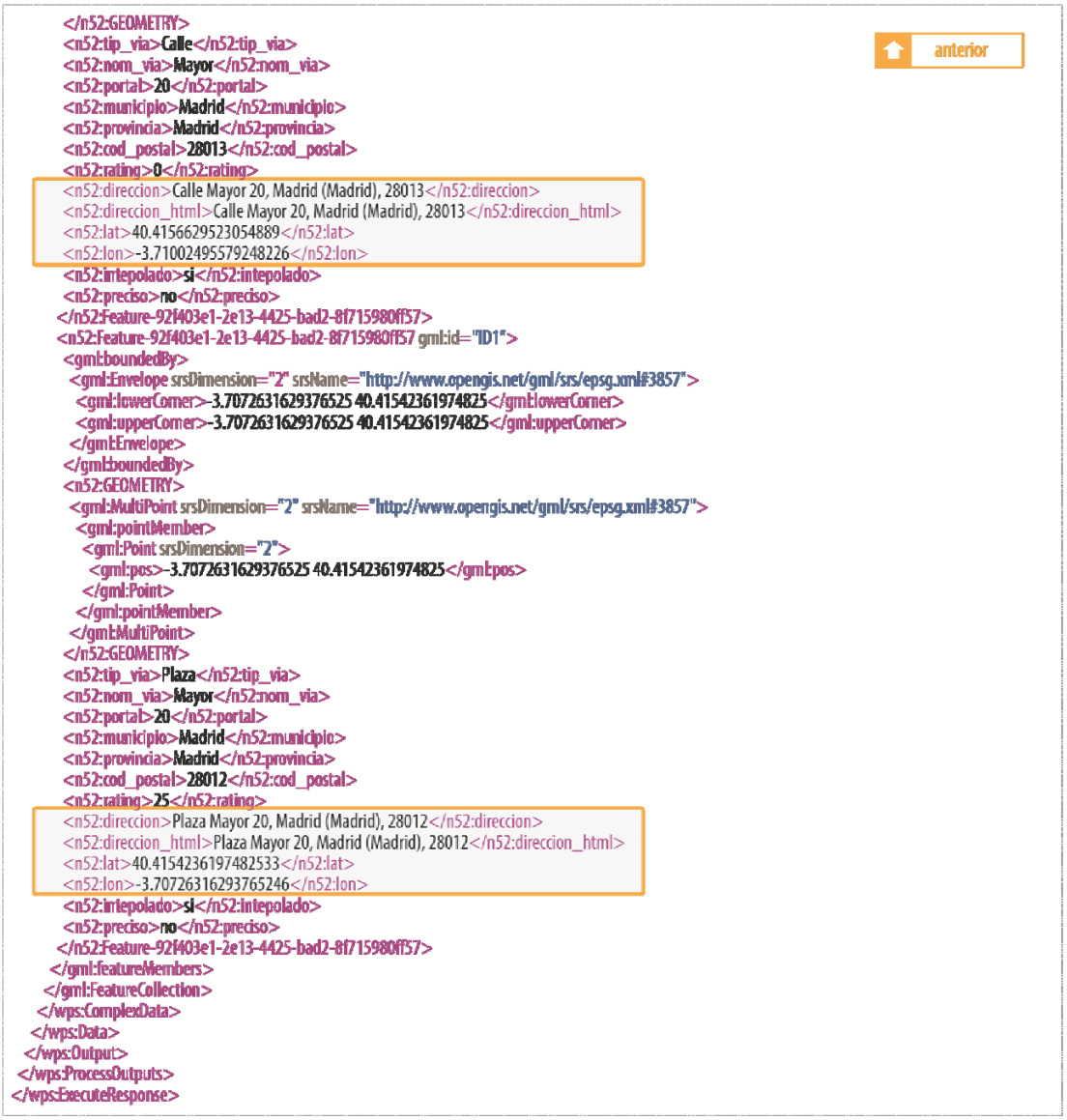

Figura 14: Respuesta de la operación Execute, en formato XML.

Como resultado se devuelve unas coordenadas geográficas longitud y latitud, junto con la dirección asociada contenida en la base de datos, ya que puede ocurrir, al utilizar métodos de búsqueda basados en lógica difusa, que la dirección no coincida con la que hemos proporcionado. También pueden suceder errores propios del algoritmo de normalización. Por todo lo anterior, y por el propio diseño del geocodificador, éste puede arrojar múltiples resultados ordenados de mayor a menos coincidencia con la dirección introducida, por lo que no está de más contrastar los campos que describen la dirección, con la dirección que se haya introducido.

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, así como las bibliotecas adecuadas para conectarse como cliente para PostgreSQL.

INTEGRACIÓN EN EL GEOPORTAL

Una vez implementada una interfaz de aplicación, surge la necesidad natural de dotar al geocodificador de un interfaz de usuario amigable e intuitivo, para facilitar así la interacción entre usuarios y aplicación (en este caso, geocodificador).

Por ello, se opta por utilizar un Geoportal, que basado en Get-SDI Portal [21], podría definirse como un sitio web cuya finalidad es ofrecer a los usuarios el acceso a una serie de recursos y servicios basados en la información geográfica, permitiendo el descubrimiento, el acceso y la visualización de los datos geoespaciales, utilizando un navegador estándar, y posibilitando la integración, la interoperabilidad y el intercambio de información entre instituciones privadas, públicas, así como a los ciudadanos [22].

En síntesis, la modificación de Get-SDI Portal ha consistido en reemplazar la llamada a la API de Google Maps, por una llamada al geoproceso *'es.upm.GeoCoder'* implementado en el servicio WPS. Para ello, se introducirá un nuevo parámetro a la operación Execute, con el que se especificará el tipo de datos que desea como respuesta:

```
http://HOST:PORT/wps/WebProcessingService?Request=Execute&Service=WPS&version=1.0.0&Identifier=es.upm.GeoCoder&RawDataOutput=resultado@mimeType=application/json&DataInputs=direccion=calle%20Mayor%2020%20Madrid
```

Figura 15: URL con paso de parámetros para invocar la operación Execute.

La petición devuelve una respuesta en forma de estructura de datos del tipo JSON, como se puede observar en la figura:

Figura 16: Respuesta de la operación Execute, en formato JSON. Extracto.

Ya que puede existir más de una dirección candidata, ésta se extrae del array *'features[]'*, cuyo orden viene predefinido, según un índice *rating* que determina el grado de aproximación o similitud de la dirección insertada en la consulta, con la dirección encontrada en la base de datos.

Si bien se puede observar que existen algunas redundancias como las coordenadas geográficas, esto es debido a que el servicio WPS lo facilita como atributo del tipo Geography Markup Language (GML). Sin embargo, para facilitar la integración, se han añadido los atributos *'lat'* y *'log'* como del tipo *'string'* (aunque asimilables a numéricos).

Como resultado de la implementación realizada, la figura X se muestra el Geoportal adaptado como cliente geoproceso *'es.upm.GeoCoder'* que facilita el servicio WPS. Como se puede observar, al introducir una dirección "calle mayor 30 Madrid", ésta se geocodifica y se devuelve un listado de dos direcciones resultado, ordenadas de mayor a menor coincidencia.

Haciendo uso de la interfaz web implementada, mediante el uso del botón "lupa", situado a la derecha de cada dirección, el Geoportal se posiciona automáticamente sobre la dirección georreferenciada en el mapa:

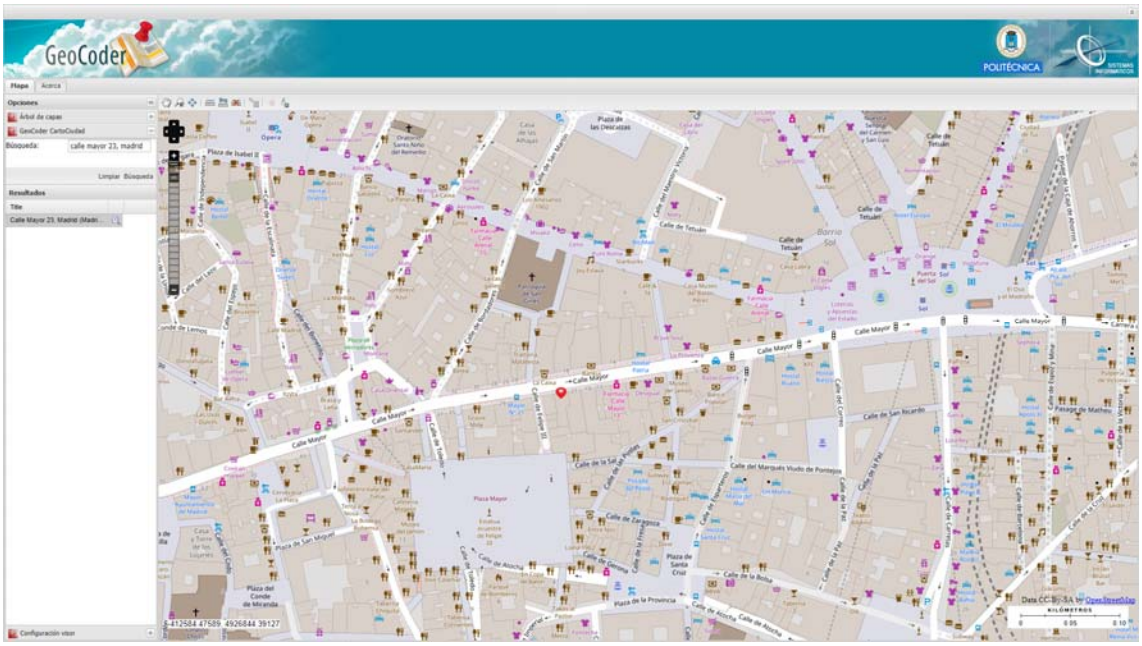

Figura 17: Interfaz del geoportal basado en GetSDI Portal, con la integración del GeoCode.

CONCLUSIONES

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

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. Para la normalización, se propone el uso de la biblioteca Libpostal, un normalizador de direcciones especializado en dicha tarea, y basado en Procesamiento de Lenguaje Natural, y desarrollado con licencia MIT por una extensa comunidad de desarrolladores.

Una especificación de requisito fundamental para un buscador, en este caso un Geocoder, es minimizar 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 si fuera preciso la planificación que el SGBD realiza para las consultas. Una adecuada elección y uso de los índices, produce una gran 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 de la base de datos que consideren la relación 1:N entre *"código postal"* y *"tramo_vial"* – con el fin de evitar la necesidad de un procesado previo-, así como para garantizar la preservación de la integridad en actualizaciones, modificaciones y borrados de tuplas en tiempo real, se recomienda encarecidamente el cumplimiento de FNBC para la base de datos adaptada para Geocoder. No obstante, como solución intermedia, se puede mantener la tabla des-normalizada *'codpostal_vial'* 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.

Además de la normalización de direcciones como paso previo a la geocodificación, es necesaria una buena estrategia para proporcionar a la aplicación cierta tolerancia a errores, así como dotar al geocodificador de flexibilidad en la búsqueda. Para ello, se propone 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.

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 y formatos de intercambio de datos diferentes, por ejemplo JSON, que facilita la integración del servicio WPS en clientes web ligeros. El desarrollo también está basado en la SGBD PostgreSQL, que puede funcionar en sistemas operativos diferentes.

Con la capa de aplicación -utilizando el estándar WPS-, el servicio de geocodificación es integrable en un entorno de Infraestructura de Datos Espaciales (IDE) que persigue la interoperabilidad de los sistemas. Gracias a la utilización de estándares, es muy sencilla la implementación de la integración del geocodificador en el geoportal GET SDI Portal, pudiendo geocodificar direcciones de manera fácil e intuitiva para los usuarios finales.

REFERENCIAS

[1] International Organization for Standardization (ISO). Geographic information - Location based services (ISO 19133:2005). Location based services (Tracking and navigation) (Vol. 19133:2005, pp. 139). Geneva, Switzerland ISO/TC 211 Geographic information/Geomatics. (2005).

[2] Centro Nacional de Información Geográfica (CNIG). Descripción del proyecto Cartociudad. (2017). Último acceso el 15/10/2017 en: *www.cartociudad.es/portal/web/guest/que-es-cartociudad*

[3] Farvacque-Vitkovic, C., Godin, L., Leroux, H., Verdet, F., y Chavez, R. Street Addressing and the Management of Cities. Washington, DC: The World Bank. (2005).

[4] International Business Machines Corporation (IBM). z/OS Basic Skills Information Center Mainframe concepts (pp. 58). USA. (2005).

[5] The Internet Engineering Task Force (IETF). Common Format and MIME Type for CSV Files (Vol. RFC 4180, pp. 8). California, USA. (2005).

[6] Openvenues. Libpostal Library. The Git Hub Inc. (2017). Último acceso el 15/10/2017 en: *https://github.com/openvenues/libpostal*

[7] IJSMI, Ed. Natural Language Processing concepts and methods revisited. International Journal of statistics and Medical Informatics, 4(1), 1-6. (2017).

[8] Liddy, E. D. Natural Language Processing. In S. o. I. S. (iSchool) (Ed.), Encyclopedia of Library and Information Science (2 ed., pp. 15). Marcel Decker, Inc. Syracuse, NY. (2001).

[9] Barretine, A. Statistical NLP on OpenStreetMap. Toward a machine-interpretable understanding of place. (2016). Último acceso el 15/10/2017 en: *https://medium.com/@albarrentine/statistical-nlp-on-openstreetmap-b9d573e6cc86*

[10] Barretine, A. Statistical NLP on OpenStreetMap. Training Conditional Random Fields on 1 billion street addresses. (2016). Último acceso el 15/10/2017 en: *https://medium.com/@albarrentine/statistical-nlp-on-openstreetmap-part-2-80405b988718*

[11] Massachusetts Institute of Technology (MIT). The MIT License. (2017). Último acceso el 15/10/2017 en: *https://opensource.org/licenses/MIT*

[12] Centro Nacional de Información Geográfica (CNIG). Servicios Web de Cartociudad. (2017). Último acceso el 15/10/2017 en: *www.cartociudad.es/recursos/Documentacion_tecnica/CARTOCIUDAD_ServiciosWeb.pdf*

[13] Bureau, U. S. C. Topologically Integrated Geographic Encoding and Referencing (TIGER). Geography. (2017). Último acceso el 15/10/2017 en: *www.census.gov/geo/maps-data/data/tiger-geodatabases.html*

[14] Chikofsky, E. J., & Cross II, J. H. Reverse Engineering and Design Recovery: A Taxonomy. IEEE Software. 7(1), 15. doi:10.1109/52.43044. (1990).

[15] Codd, E. F. Further Normalization of the Data Base Relational Model. Courant Computer Science Symposia Series 6, "Data Base Systems". IBM Research Report RJ909. Republicado en 'Randall J. Rustin, (ed.). Prentice-Hall. New York. (1972).

[16] Centro Nacional de Información Geográfica (CNIG). Especificaciones del Producto CartoCiudad. (2013). Último acceso el 30/10/2017 en: *www.cartociudad.es/recursos/ Documentacion_tecnica/CARTOCIUDAD_Especificaciones.pdf*

[17] Codd, E. F., Recent Investigations into Relational Data Base Systems. IBM Research Report RJ1385. Republicado en 'Proc. 1974 Congress' (Stockholm, Sweden). New York. (1974).

[18] Harries, K., Mapping Crime: Principles and Practice. US Department of Justice, Office of Justice Programs. pp 206. Washington. (1999).

[19] Ratcliffe, J.H., On the accuracy of TIGER-type geocoded address data in relation to cadastral and census areal units. Int. J. Geogr Inf Sci. (2001).

[20] PostgreSQL 9.6.5 Documentation. The PostgreSQL Global Development Group (2017). EXPLAIN, show the execution plan of a statement. Último acceso el 30/10/2017 en: *www.postgresql.org/ docs/9.6/static/sql-explain.html*

[21] G.E. Technologies. Get SDI Portal. The Git Hub Inc. (2017). Último acceso el 30/10/2017 en: *https://github.com/GeospatialEnablingTechnologies/GET-SDI-Portal*

[22] Bernabé, M. A., & López Vázquez, C. M. Fundamentos para las Infraestructuras de Datos Espaciales (IDE) (1ª ed.). pp. 596. UPM Press, Madrid. (2012).

AUTORES

Iván MOYA HONDUVILLA
ivan81k@gmail.com
Universidad Politécnica de Madrid
Departamento de Ingeniería Topográfica y Cartografía

Miguel Ángel MANSO-CALLEJO
m.manso@upm.es
Universidad Politécnica de Madrid
Departamento de Ingeniería Topográfica y Cartografía

Javier MOYA HONDUVILLA
j.moya@geoimagine.es
Geoimagine S.L.
CEO