Universidad Internacional de La Rioja (UNIR)

Escuela de Ingeniería

Grado en Ingeniería Informática

Servicio Web para la Corrección Ortográfica Multilingüe

Página web del proyecto:
https://github.com/jmbeltran/SWCOM

Trabajo Fin de Grado

Presentado por: Beltrán Vicente, José María

Director/a: de la Fuente Valentín, Luis

Ciudad: Ponferrada

Fecha: 24/09/2015

Trabajo de Fin de Grado	Titulo del Grado	Alumno
	Curso de adaptación al Grado de Ingeniería Informática	José María Beltrán Vicente

Resumen

El presente trabajo realiza un estudio sobre la corrección ortográfica computarizada, analizando diferentes técnicas propuestas para este fin, y desarrolla un servicio Web para la evaluación ortográfica de textos, utilizando alguno de los métodos presentados. El servicio se ha bautizado con SWCOM (Servicio Web para la Corrección Ortográfica Multilingüe) y permite analizar ortográficamente textos en varios idiomas, utilizando como soporte un diccionario de palabras de ese idioma, y presentado como resultado un documento XML con los errores detectados, junto con una lista de sugerencias para la sustitución de las palabras erróneas. También se realiza una exposición de las diferentes arquitecturas y protocolos utilizados para la implementación de servicios Web, con especial atención a la tecnología REST, la cual se ha seleccionado para implementar el servicio.

Palabras Clave: corrección ortografía servicio web REST

Abstract

This paper makes a study on the computerized spell checking, analyzing different techniques proposed for this purpose, and develops a Web service for spell evaluation of texts, using any of the methods presented. The name of service is SWCOM (acronym in Spanish of Web Service for Multilingual Spell Checking) and allows orthographically analyze texts in several languages, using as support a dictionary of words in that language, and presented results in an XML document with the errors, along with a list of suggestions to replace the wrong words. An exhibition of the different architectures and protocols used to implement Web services, with special attention to the REST technology, which has been selected to implement the service is also performed.

Keywords: spell check service web REST

Contenido

1. Introducción.

La corrección ortográfica automática es un problema estudiado de forma profusa desde los inicios de la informática. Las primeras investigaciones en la materia se realizaron a finales de los años 50. En 1959 Bledsoe y Browning propusieron un método para la corrección de texto en la salida de un sistema de reconocimiento de caracteres que, con pequeñas adaptaciones, sigue utilizándose actualmente (Bledsoe, W. W. y Browning, I., 1959). En 1959 Les Earnest comienza a trabajar en el desarrollo del primer sistema para el reconocimiento de texto manuscrito. Durante la investigación identifica la necesidad de dotar al sistema con un corrector ortográfico y realiza grandes avances en la materia (Lindgren, 1965) (Earnest, Machine recognition of cursive writing, 1962). Earnest construye el primer corrector ortográfico, el cual utiliza un diccionario con las 10.000 palabras inglesas más comunes, implementado sobre siete rollos de cinta de papel perforado (Earnest, The first cursive handwriting recognizer needed a spelling checker, 2015). Pocos años después Earnest se convierte en profesor de la Universidad de Stanford (California, EEUU) donde continua, junto con varios de sus alumnos, la investigación sobre la corrección ortográfica. En 1971 Ralph Gorin, alumno de doctorado de Earnest, construye un corrector ortográfico más simple y rápido. El programa, denominado SPELL, se desarrolló para un ordenador DEC PDP-10 y fue escrito en lenguaje ensamblador (Gorin, 1973) . Algunos autores consideran a SPELL el primer corrector ortográfico computarizado sin embargo, como hemos visto, sólo es una evolución de los desarrollos realizados por Earnest. Este corrector inspiró el desarrollo de Ispell, el primer corrector ortográfico para UNIX y precursor de los correctores ortográficos modernos.

Desde estos primeros tiempos hasta nuestros días han surgido innumerables estudios sobre la corrección ortográfica de textos, poniendo en relieve el interés suscitado por el tema en investigadores de diferentes áreas de conocimiento. Este interés es debido, principalmente, a la aplicación práctica que la solución tiene en la industria. El uso más conocido de los correctores ortográficos se relaciona con los procesadores de texto, sin embargo se utilizan en muchos otros terrenos. Sistemas de reconocimiento óptico de caracteres, en los que resulta imprescindible procesar la salida para verificar la corrección del texto analizado, o sistemas de reconocimiento automático de la voz, donde un análisis del texto generado puede mejorar notablemente la fiabilidad del sistema, son algunos ejemplos de esta utilidad.

Por otra parte Internet y la Arquitectura Orientada a Servicios (SOA, del inglés Service Oriented Architecture) han propiciado nuevos modelos de computación que enfatizan la máxima “divide y vencerás” convirtiendo los grandes sistemas de información en un conjunto

de servicios especializados en la solución de problemas específicos. Los servicios se consideran actualmente la mejor opción para la integración de diferentes sistemas por su facilidad de implementación, su estandarización y su independencia de los sistemas a integrar. Los servicios web son uno de los mecanismos más utilizados para desplegar arquitecturas SOA. De forma básica podemos considerar un servicio web como una colección de métodos, independientes de la plataforma y del lenguaje de programación utilizado para su implementación, que podemos invocar desde Internet. Actualmente se consideran dos tecnologías para el desarrollo de servicios web, SOAP y REST. La primera, propuesta por W3C, define la arquitectura de los servicios web basándose en el uso de estándares tales como XML, SOAP, WSDL, etc. La segunda propone un modelo de arquitectura basada en el protocolo HTTP, simplificando enormemente el proceso de invocación de los servicios.

El presente trabajo comienza con un estudio teórico de diferentes técnicas propuestas para la corrección ortográfica automática, con objeto de ofrecer una visión precisa sobre la complejidad del problema. Se seleccionará alguno de los métodos propuestos para desarrollar un servicio web RESTful[1] que permita evaluar ortográficamente un texto, lo cual marca el carácter diferencial del trabajo, dado que la mayoría de herramientas de corrección ortográfica se suelen integrar en otros productos como utilidad de soporte a los mismos. Los servicios web constituyen verdaderas APIs de integración, lo cual permitirá dotar de esta funcionalidad a cualquier aplicación o servicio, independientemente de la plataforma en la que corran y de los lenguajes de programación utilizados para su desarrollo. Aunque existen librerías disponibles para el desarrollo de correctores ortográficos se ha querido implementar el servicio desde cero, poniendo el foco del proyecto sobre el problema de la corrección ortográfica.

A continuación, y dentro del mismo capítulo, se realizará una breve introducción a SOA y a los servicios web, describiendo las tecnologías más utilizadas para su desarrollo.

El siguiente apartado presenta el análisis de requisitos del sistema a construir. Se referenciará la metodología utilizada para la planificación y gestión del proyecto y se presentarán los diagramas de casos de uso y clases del sistema.

A continuación se realizará una descripción detallada de la solución aportada, desde un punto de vista funcional, detallando el comportamiento de todos los componentes del sistema. Al final de este apartado se explicará como desplegar el servicio en un servidor de aplicaciones Glassfish.

[1] Los servicios construidos bajo los principios de la arquitectura REST se conocen como RESTful.

En el siguiente capítulo se presentará la arquitectura del servicio, las herramientas utilizadas para su desarrollo y el código fuente. Se explicará, de forma detallada, cada una de las clases implementadas para solucionar el problema.

Por último se expondrán los resultados de la evaluación del servicio desarrollado con la finalidad de verificar el cumplimiento de los requerimientos establecidos.

El trabajo concluirá con una exposición sobre las posibles líneas de trabajo futuro.

2. Contexto y estudio preliminar.

El problema se centra en dos cuestiones principales, la corrección ortográfica y los servicios web. En este capítulo realizaremos una introducción teórica a ambas cuestiones con objeto de tener un conocimiento general de las mismas que sirva como entrada para el desarrollo de soluciones alternativas a las planteadas en este trabajo.

2.1. Corrección ortográfica automática.

Un corrector ortográfico es una aplicación informática que se utiliza para analizar textos con el fin de detectar y, de forma automática o interactiva, corregir faltas ortográficas.

La ortografía es un conjunto de normas que regulan la escritura de una lengua. Una norma ortográfica del castellano sería "*se escriben con b los verbos terminados en –bir, salvo hervir, servir y vivir y sus compuestos*" (Real Academia de la Lengua Española, 1999). Por consiguiente un corrector ortográfico analizaría los elementos de un texto, principalmente palabras, para determinar que cumplen las normas de la ortografía de una lengua.

Aplicando la regla anterior al texto *El agua hierbe a cien grados*, comprobamos que presenta un error ortográfico en la palabra *hierbe*, dado que esta debería escribirse, según dicha regla, con *v*. Supongamos ahora la frase *"El agua hervir a cien grado".* Desde el punto de vista ortográfico el texto sería correcto dado que todos los elementos cumplen las normas ortográficas de la lengua española (todas la palabras están bien escritas), sin embargo existe un error en el tiempo verbal empleado y en el número de la palabra *grado.* El primero no puede ser un infinitivo y la segunda debería estar en plural. Este tipo de errores estarían en el campo de la gramática y no de la ortografía. La gramática se encargaría de la forma de usar y organizar las palabras en una oración[2].

Por consiguiente, a la hora de analizar un texto podemos encontrar, básicamente, dos tipos de errores: ortográficos y gramaticales.

Aunque en la actualidad se han desarrollado herramientas que realizan la corrección ortográfica y gramatical de forma conjunta no deben confundirse ambas. Los algoritmos empleados en cada caso son diferentes.

Muchos autores incluyen en sus estudios sobre la corrección automática de textos ambos problemas, diferenciando entre aquellas palabras que no existen, a las que denominan *non-words*, y aquellas que si existen pero no son correctas en el contexto analizado, a las cuales

[2] *La Nueva Gramática de la Lengua Española* se articula en tres partes fundamentales: morfología, que analiza la estructura de las palabras, su constitución interna y sus variaciones; la sintaxis, que se ocupa de la forma en que se ordenan y combinan; y la fonética y fonología, que estudias los sonidos del habla y su organización lingüística.

se refieren como *real-words*. En la frase *Los murcilagos son los únicos mamífero voladores*, la palabra *murcilagos* sería un error non-word dado que es un término inexistente. Sin embargo el término *mamífero* sería un error real-word, dado que aunque la palabra existe no es correcta en el contexto de dicha frase. Según esto los elementos non-word serían errores ortográficos mientras que los real-word son errores gramaticales.

Dada extensión de cada campo, en este estudio nos centraremos exclusivamente en el problema de la corrección ortográfica en un sentido estricto, es decir errores non-word, exponiendo brevemente algún mecanismo para la corrección de errores gramaticales que pueda servir como introducción para la ampliación de los desarrollos realizados bajo el marco de este trabajo.

2.2. Funcionamiento básico de un corrector ortográfico.

El problema de la corrección ortográfica se puede dividir a su vez en tres subproblemas principales:

- **Tokenización.** Consiste en la división del texto a analizar en tokens. En el caso de un corrector ortográfico los tokens serían palabras.
- **Detección.** Consiste en determinar si una palabra está bien o mal escrita.
- **Corrección.** Consiste en sustituir la palabra mal escrita por otra correcta.

La tokenización es uno de los procesos más utilizados en el procesamiento del lenguaje natural. La división de un texto en unidades más elementales es fundamental para su posterior tratamiento. Dependiendo del tipo de análisis, los tokens pueden ser caracteres, sílabas, conjuntos de caracteres de un número determinado de letras (n-gramas), palabras, etc.

Actualmente, la mayoría de técnicas para la detección de errores ortográficos utilizan un diccionario para la búsqueda de palabras. Una palabra es considerada errónea sino se encuentra en el diccionario. Este método parece el más eficaz ya que reduciría el problema a una búsqueda sobre un conjunto limitado de elementos, trasladando la complejidad a la construcción del diccionario. Los correctores que utilizan diccionarios son independientes del lenguaje y para la detección de errores en textos de un determinado idioma solo tendríamos que utilizar un diccionario de este idioma.

La corrección consistirá, normalmente, en proponer una lista de palabras candidatas para sustituir la palabra errónea. La elección de una palabra entre la lista de candidatas puede ser automática o interactiva. En la corrección interactiva el usuario selecciona una palabra de la lista de candidatas, mientras que en la selección automática es el corrector el que selecciona dicha palabra.

Puede ocurrir que una palabra sea considerada error cuando no lo es, debido a no encontrarse en el diccionario. En este caso el usuario podría optar por añadir la palabra al diccionario.

2.3. Detección de errores ortográficos.

Las principales técnicas para la detección de errores ortográficos (non-word) son *análisis de n-gramas* y *búsqueda en diccionario*. Menos habitual es la utilización de técnicas basadas en el uso de reglas ortográficas, que se circunscribe al ámbito de la investigación, dado su total dependencia del lenguaje. A continuación introduciremos cada una de ellas.

2.3.1. Análisis de n-gramas.

Un n-grama es una subsecuencia de n elementos de una secuencia dada. Son muy utilizados en diversas áreas de conocimiento (lingüística, genética, química, etc.) para la determinación de la corrección de determinadas cadenas de elementos o la predicción de los próximos elementos de una determinada secuencia. En lingüística computacional los n-gramas pueden ser diferentes elementos, tales como fonemas, sílabas, letras, palabras, etc., dependiendo del área de estudio. En la corrección ortográfica un n-grama sería, normalmente, una secuencia de n caracteres de una determinada palabra. Frecuentemente se usan valores de n=1, n=2 o n=3 denominándose estos elementos unigramas, bigramas y trigramas respectivamente. En general, la técnica consiste en determinar la probabilidad de ocurrencia de determinados n-gramas en el lenguaje. Para ello se descompone cada palabra del texto a analizar en n-gramas y se compara cada uno de ellos con una matriz en la que se almacena la frecuencia de cada n-grama en el lenguaje. En función de esta frecuencia se calcula la probabilidad de cada n-grama. Por ejemplo, en español el trigrama *mas* tendrá una frecuencia alta mientras que *jxw* no se encontrará. Cuando una palabra tiene uno o varios n-gramas inexistentes (probabilidad 0) o poco frecuentes (probabilidad muy baja) se considera un posible error. Para determinar la frecuencia de cada n-grama se pueden utilizar uno o varios corpus de un determinado lenguaje dividiendo cada fuente de datos en n-gramas y anotando cada aparición de un determinado n-grama en una tabla.

También es posible construir tablas de n-gramas binarias donde se indica si el n-grama aparece (1) o no (0) en un determinado corpus.

Imaginemos que queremos construir una tabla de probabilidades para realizar un análisis ortográfico de textos utilizando digramas. Siguiendo esta técnica implementaríamos una matriz bidimensional donde cada fila y columna representarían un elemento del conjunto de caracteres del lenguaje. El elemento *[1,1]* representaría la probabilidad del digrama *aa,* el *[1,2]* de *ab,* el *[1,3]* de *ac,* y así sucesivamente. Con trigramas utilizaríamos una matriz tridimensional.

Podemos incrementar la precisión del método introduciendo la posición de los n-gramas dentro de cada palabra. De este modo un determinado n-grama puede ser frecuente en una determinada posición pero ser improbable en otra. Para ello las celdas de la matriz incluirían una lista donde cada elemento almacenaría la posición del n-grama y su frecuencia.

Diversos estudios señalan que esta técnica se utiliza principalmente para la corrección de textos producidos por sistemas OCR (del inglés Optical Character Recognition) demostrando menos efectividad en procesadores de texto, en los que se utilizan, normalmente, técnicas de búsqueda en diccionarios (Kukich, 1992).

Como veremos más adelante los n-gramas tienen también aplicación en la corrección automática de palabras y en los campos del reconocimiento de voz, traducción automática, texto predictivo, etc. Por este motivo importantes empresas han desarrollado soluciones que proporcionan acceso a grandes corpus de n-gramas. Entre ellas cabe destacar Yahoo! N-Grams, Google books Ngram Viewer y Microsoft Web N-gram Services.

2.3.2. Búsqueda en diccionarios.

Es la técnica más usada por la mayoría de correctores ortográficos y consiste en la búsqueda de una determinada palabra en una lista de palabras existentes en el lenguaje a analizar, que denominaremos diccionario, con objeto de determinar si es correcta o se trata de un error. En un diccionario perfecto (aquel que contuviese todas las palabras, y sus correspondientes derivaciones, de una determinada lengua) si la palabra no se encuentra podríamos considerarla un error cierto. En la práctica es enormemente complicado contar con un diccionario perfecto por lo que si una palabra no está contenida en él podría considerarse un posible error, o error ambiguo, el cual debería ser confirmado por un humano. Por consiguiente, la efectividad del método depende en gran medida del tamaño del diccionario lo que a su vez incide negativamente en el rendimiento del proceso (a mayor número de palabras mayor tiempo de procesamiento, espacio en memoria, etc.)

Un diccionario es, normalmente, un fichero de texto que contiene un conjunto de palabras de un determinado idioma (en el siguiente apartado veremos algunos métodos para la construcción de diccionarios). La búsqueda directa sobre el fichero no se utiliza en ningún caso, dado que sería un proceso secuencial con una complejidad muy alta [O(n)] a la que habría que añadir el bajo rendimiento de las unidades de almacenamiento frente a la memoría principal. Por este motivo es necesario cargar el fichero en una estructura de datos que nos permita optimizar el proceso de búsqueda. Dado el número de palabras que previsiblemente contendrá un diccionario, la elección de la estructura de datos y los algoritmos de búsqueda son determinantes para el obtener un buen rendimiento. Podemos utilizar diferentes estructuras de datos:

- **Árboles binarios de búsqueda**. La búsqueda tendría una complejidad de $O(\log_2 n)$. Para un diccionario con 1.000.000 de palabras el número de iteraciones para encontrar una palabra seria de 20.
- **Tablas hash**. La búsqueda tendría una complejidad de O(1). Con una buena función hash, que no produzca colisiones, el número iteraciones para encontrar una palabra sería siempre de 1.

Aunque la priori parece clara la elección de las tablas hash para implementar el diccionario presentan una dificultad derivada de la elección de la función hash. Con un número grande de elementos, la elección de esta función es esencial para evitar que se produzcan colisiones que puedan dar lugar a errores en el proceso de corrección. En nuestro caso se ha volcado esta responsabilidad en el lenguaje de programación utilizando la clase *HashTable* del paquete *java.util* que nos garantiza el almacenamiento y recuperación de datos sin colisiones.

El sistema de detección mediante el uso de diccionarios es independiente del idioma. Para la detección de errores en un texto de una determinada lengua, sólo tenemos que usar un diccionario de dicha lengua. El cambio de idioma no produciría ninguna variación en el código del programa. Por otra parte es posible utilizar diccionarios técnicos para el análisis de textos con terminología específica de un área de conocimiento (medicina, informática, etc.). Por todos estos motivos la técnica de búsqueda en diccionarios es la más utilizada por la mayoría de correctores existentes en el mercado para la detección de errores ortográficos y es la que usaremos para el desarrollo de SWCOM.

2.3.2.1. Formato de los diccionarios.

Los diccionarios utilizados por los correctores ortográficos son, normalmente, archivos de texto que contienen una lista de palabras correctas de un determinado idioma. Un problema que se plantea a priori es la inclusión de todas las palabras de un lenguaje junto con sus derivaciones. Para un verbo se incluirían todas la formas de la conjugación, para un sustantivo se incluirían las variaciones de género y número, etc. Este método genera grandes diccionarios que requieren mucho espacio en memoria. Por el contrario reducen la complejidad de la detección a una operación de búsqueda.

Para optimizar el espacio en memoria muchos correctores utilizan un diccionario que contiene las "formas básicas" o lexemas del lenguaje junto a un conjunto de reglas de derivación. Este método optimiza el uso de memoria pero aumenta la complejidad del proceso de detección, ya que es necesario ejecutar una rutina que interprete estas reglas de derivación[3]. Cada diccionario está formado por un archivo de formas básicas *(.dic*) y un archivo de afijos[4] (*.aff*).

Cada palabra de un archivo *.dic* puede estar seguida de una barra / y uno o más flags. Por ejemplo:

correcto/kSG
ortografía/S
trabajo/fS
fin/S
grado/LS

El archivo de afijos, *.aff*, contiene las reglas de derivación correspondientes a cada uno de los flags del diccionario. La mayoría de reglas de derivación están etiquetadas como PFX (prefijos) y SFX (sufijos). Cada regla tiene el siguiente encabezado:

[SFX|PFX] <identificador de regla> <combinable> <número de líneas de la regla>

Donde <identificador de regla> es una letra que identifica a la regla, *<combinable>* indica si se puede combinar con otras reglas y *<número de líneas de la regla>* especifica el número de líneas de la regla, que se colocan a continuación del encabezado.

[3] La mayoría de correctores ortográficos utilizan esta técnica.

[4] Un afijo es un conjunto de caracteres que se anteponen (prefijo) o se posponen (sufijo en un lexema para formar diferentes palabras.

Por ejemplo, *SFX S Y 32* sería el encabezado de una regla tipo sufijo cuyo identificador es la letra *S*. Sería combinable con otras reglas (*Y*, Yes) y tendría *32* líneas.
Cada una de las líneas que sigue al encabezado constituye una regla de derivación que se aplicaría a las palabras que contiengan un flag que coincida con el identificador del la regla. En el ejemplo anterior la regla se aplicaría a todas las palabras que tienen el flag *S*.
La sintaxis de una regla es:

[PFX|SFX] <identificador de regla> < letras a borrar> <letras a añadir> <condiciones>

Donde *<idenficador de regla>* es la letra que identifica a la regla, *<letras a borrar>* especifica el número de letras que se borran de la palabra original[5] y *<condiciones>* es una expresión regular que indica cuando se aplica la regla.
Veamos un ejemplo.

SFX S 0 s [aceéfgiíkoóptuúw]
SFX S 0 es [bdhíjlmnrúy]

La primera regla indica que cuando las palabras terminan en uno de los caracteres entre corchetes no se quitaría ninguna letra de la palabra original y se añadiría una *s*. La segunda regla se aplicaría a las palabras que terminen en las letras entre corchetes. En este caso se añadiría la cadena *es* a la palabra original.
Supongamos las palabras del ejemplo de más arriba:

trabajo/fS
fin/S

Si aplicamos las reglas anteriores obtendríamos las palabras *trabajos* y *fines*.
Veamos otro ejemplo tomando las reglas del diccionario de español del corrector Aspell. En el diccionario encontramos la entrada *correcto/kSG.* Las reglas correspondientes a los flag anteriores son:

[5] Puede ser 0, una o más letras. Cuando es 0 indica que no se borra ninguna letra. Cuando son letra, una o más, especifica las letras que se eliminan al principio, en el caso de prefijos (PFX) o al final en el caso de sufijos (SFX) de la palabra.

PFX k Y 4	*SFX S Y 32*	*SFX G Y 18*
PFX k 0 i l	*SFX S 0 s [aceéfgiíkoóptuúw]*	*SFX G e a [^u]e*
PFX k 0 im [bp]	*SFX S 0 es [bdhíjlmrúy]*	*SFX G que ca que*
PFX k 0 in [^blpr]	*SFX S á aes á*	*SFX G o a o*
PFX k 0 ir r	*SFX S 0 es [^áeéíóú]n*	*SFX G 0 a [dlrz]*
	SFX S 0 es [^áéíóú]s	*SFX G án ana án*
	...[6]	*...*[6]

La primera regla es un prefijo. En nuestro caso se podría aplicar la regla *PFX k 0 in [^blpr]* ya que la palabra *correcto* no comienza por ninguna de las letras entre corchetes[7]. De esta regla derivaríamos la palabra *incorrecto*. Para el segundo flag se aplicaría la regla *SFX S 0 s [aceéfgiíkoóptuúw]*, ya que la palabra termina por *o*, carácter que está incluido en el grupo entre corchetes. Por consiguiente obtendríamos la palabra *correctos*. Por último se aplicaría la regla *G*. Dado que nuestra palabra termina en *o*, se aplicaría la regla *SFX G o a o*, que indica que se elimina la letra *o* y se añada la letra *a*. Por tanto se obtiene la palabra *correcta*. Todas las reglas anteriores son combinables por tanto a las palabras derivadas se les podrían aplicar también dichas reglas. De esta forma obtendríamos las palabras *correctos, correctas, incorrecto, incorrecta, incorrectos* e *incorrectas.*

El proceso para reducir una palabra a su forma básica se denomina *stemming*. Los algoritmos de *stemming* tomarían una palabra y a partir de las reglas de derivación obtendrían su forma básica, Sería el proceso inverso a la derivación, que hemos visto en los ejemplos anteriores.

Algunos correctores incorporan algoritmos de steaming y/o derivación que aplican durante el proceso de detección de errores. Otros utilizan un diccionario con todas las formas que obtienen derivando todas las palabras del diccionario. En nuestro caso hemos optado por esta última opción.

2.3.3. Reglas ortográficas.

Aunque se han encontrado algunos estudios sobre el uso de reglas ortográficas para la detección de errores en la práctica no se suele utilizar por la dificultad de implementar las reglas y su dependencia del idioma. Otros estudios como “Corrector *ortográfico de libre*

[6] El archivo .aff contiene más reglas para este identificador pero se omiten por no ser de aplicación en este ejemplo.

[7] El carácter ^ expresa una negación por lo que la expresión [^blpr] se aplicaría a todas la palabras que NO comiencen por las letras b, l, p o r.

distribución basado en reglas de derivación", realizado por J. Carretero y S. Rodríguez, orientan el problema hacía la compilación de un diccionario, no hacia el desarrollo de un corrector ortográfico.

2.4. Corrección de errores ortográficos.

El proceso de corrección se inicia con la generación de sugerencias para una palabra errónea. Se podrían obtener ninguna, una o varias sugerencias. Si encontramos una sola sugerencia, esta sería la corrección propuesta. En caso de encontrar varias sugerencias podríamos aplicar un método interactivo para seleccionar la palabra adecuada, en el que el usuario del corrector seleccionaría la que considera más apropiada para la corrección, o automático, en el que el propio corrector seleccionaría una palabra entre todas las sugerencias generadas. En base a esto podríamos dividir el problema de la corrección en dos subproblemas:

- **Generación de sugerencias**. Genera una lista de posibles palabras correctas.
- **Corrección**. Entre todas las sugerencias selecciona la más adecuada. Puede ser interactiva o automática.

2.4.1. Técnicas para la generación de sugerencias.

En primer lugar estudiaremos algunas de las técnicas propuestas por diferentes autores para la generación de una lista de sugerencias, formada por palabras candidatas para la sustitución de una palabra errónea. Existen multitud de algoritmos de generación de sugerencias basados en las técnicas expuestas a continuación. En este trabajo se pretende introducir la técnica general haciendo mención, en algunos casos, a determinados algoritmos existentes o proponiendo, cuando estos fueran muy complejos, algún procedimiento simple que utilice la técnica descrita, con objeto de comprender de forma esencial el método. En la bibliografía descrita se puede profundizar en cada uno de las técnicas descritas.

2.4.1.1. Similitud de cadenas.

Aunque existen varias medidas para cuantificar la similitud entre dos cadenas de caracteres, la más utilizada en el ámbito de la corrección ortográfica es la distancia de edición, considerando que las cadenas son idénticas si su distancia de edición es 0.

La distancia de edición entre dos cadenas de caracteres a y b puede definirse como el número de operaciones básicas que hay que realizar sobre la cadena a para transformarla en b. Las operaciones básicas son:

- **Inserción**. Consiste en insertar una letra en la cadena.
- **Eliminación**. Consiste en eliminar una letra de la cadena.
- **Sustitución**. Consiste en sustituir una letra por otra.
- **Trasposición**. Consiste en el intercambio de dos caracteres adyacentes.

Los primeros en proponer esta medida fueron Damerau (Damerau, 1964) y Levenshtein (Levenshtein, 1966). La distancia de edición de Levenshtein contempla tres operaciones básicas (inserción, eliminación y sustitución) mientras que la distancia de Damerau incluye la transposición. Para entender mejor el concepto veamos algunos ejemplos:
La distancia de Levenshtein entre las cadenas $niño$ y $minas$ es 4.

1. Sustitución de n por m: $niño \rightarrow miño$
2. Sustitución de $ñ$ por n: $miño \rightarrow mino$
3. Sustitución de o por a: $mino \rightarrow mina$
4. Inserción de s: $mina \rightarrow minas$

Veamos ahora un ejemplo con un error ortográfico. La distancia de Levenshtein entre $grado$ y $grdao$ es 2:

1. Sustitución de d por a: $grdao \rightarrow graao$
2. Sustitución de a por d: $graao \rightarrow grado$

Si utilizamos la operación de transposición, propuesta por Damerau, la distancia sería 1:

1. Trasposición de los caracteres d y a: $grdao \rightarrow grado$

Existen diferentes algoritmos para calcular la distancia de Levenshtein entre dos cadenas de caracteres (García, 2007). Uno de ellos se basa en el llenado de una matriz d de tamaño $n\ x\ m$ mediante programación dinámica, donde n y m serían la longitud de las cadenas a

comparar, *s1* y *s2*. El algoritmo comienza asignando valores a la primera fila y columna de la matriz de la siguiente forma:

Para i=0 hasta longitud(s1) d[i,0]=i
Para i=0 hasta longitud(s2) d[0,j]=j

Supongamos que queremos comparar las cadenas *s1=niño y s2=minas*, después de aplicar estás formulas la matriz quedaría:

		n	*i*	*ñ*	*o*
	0	1	2	3	4
m	1				
i	2				
m	3				
a	4				
s	5				

Ahora tenemos que recorrer la matriz asignando valores a las celdas vacías de la siguiente forma:

Para i=0 hasta longitud(s1)
Para j=0 hasta longitu(s2)
Si s1[i]=s2[j] Entonces coste=0 sino coste=1
d[i,j]=minimo(d[i-1,j]+1,d[i,j-1]+1,d[i-1,j+1]+coste)

Al ejecutar el algoritmo tendríamos la siguiente tabla:

		n	*i*	*ñ*	*o*
	0	1	2	3	4
m	1	1	2	3	4
i	2	2	1	2	3
m	3	2	2	2	3
a	4	3	3	3	3
s	5	4	4	4	**4**

La distancia entre *s1* y *s2* sería el valor de la celda *d[longitud(s1),longitud(s2)]* en este caso 4.

Los algoritmos que utilizan este método para generar la lista de sugerencias de una palabra errónea, calculan la distancia de edición entre dicha palabra y todas las entradas de un diccionario, incorporando a la lista aquellas cuya distancia de edición sea mínima. En este caso para cada error necesitaríamos ejecutar el algoritmo que calcula la distancia de edición entre dos palabras un número de veces igual al número de entradas en el diccionario.

Para optimizar el proceso podemos generar todas las ediciones posibles con distancia 1 de la palabra errónea aplicando las operaciones básicas. En el caso de no encontrar ninguna sugerencia podríamos buscar ediciones con distancia 2, y así sucesivamente. Cada una de las formas generadas se busca en el diccionario, marcando como candidatas aquellas que se encuentren. Algunos autores hacen referencia a este algoritmo denominándolo cálculo inverso de la distancia mínima de edición (Kukich, 1992).

En este algoritmo las operaciones de inserción y sustitución utilizarán un alfabeto de caracteres válidos del idioma correspondiente. Sea C este conjunto y n el número de caracteres del mismo.

Veamos un ejemplo sencillo. Supongamos que necesitamos corregir la palabra *vre*. En primer lugar generamos todas las ediciones con distancia 1 de esta palabra:

Inserciones	Eliminación	Sustitución	Trasposición
$a_i+vre\ \forall a_i \in C$	re	$a_i+re\ \forall a_i \in C$	rve
$v+a_i+re\ \forall a_i \in C$	ve	$v+a_i+e\ \forall a_i \in C$	ver
$vr+a_i+e\ \forall a_i \in C$	vr	$vr+a_i\ \forall a_i \in C$	
$vre+a_i\ \forall a_i \in C$			

El número de ediciones con distancia 1 de una palabra con una longitud *l* sería:

- Inserciones: $n*(l+1)$
- Eliminaciones: l
- Sustituciones: $n*l$
- Trasposiciones: $l-1$

Numero de ediciones con distancia 1=$n*(l+1)+l+n*l+(l-1)=2nl+n+2l-1$

A modo de ejemplo, supongamos un idioma con un alfabeto de 32 caracteres. El número de ediciones para una palabra de 3 caracteres sería de 229. Para una palabra de 4 caracteres tendríamos 295 ediciones con distancia 1. Para una palabra de 10 caracteres sería 691.

Si no encontramos ninguna sugerencia con distancia 1 podríamos generar ediciones con distancia 2. Aquí el coste de computación se dispara dado que debemos aplicar de nuevo todas las operaciones a cada edición con distancia 1. Esto supondría que para una palabra de 3 caracteres tendríamos 52.441 ediciones con distancia 2. Para una palabra de 4 caracteres tendríamos 87.025 ediciones y para una palabra de 10 caracteres 477.481. Teniendo en cuenta que la mayoría de errores ortográficos están a una distancia 1, podemos considerar que este algoritmo es más eficiente que la opción de buscar todas las palabras con distancia 1 del diccionario.

Este método fue usado por Ralf Gorin para construir SPELL y por otros muchos investigadores con posterioridad (Kukich, 1992) lo que demuestra su efectividad. Incluso algunos lenguajes de programación, como PHP[8], incluyen en el propio lenguaje una función para calcular la distancia de edición entre dos cadenas de caracteres. Diversa literatura sobre corrección ortográfica afirma que un 80% y un 95% de los errores ortográficos están a una distancia de edición de 1 de la palabra correcta. El porcentaje de efectividad llegaría al 98,9% si incluimos las variaciones a una distancia de edición de 2 (Norvig, 2007).

Para implementar nuestro servicio hemos elegido el método del cálculo inverso de la distancia mínima de edición que en nuestro caso denominaremos *Damerau inverso*.

2.4.1.2. Similitud de claves.

Consiste en asignar a cada palabra una clave de forma que palabras similares tengan claves idénticas. Esta similitud puede expresarse en términos de fonética, donde dos palabras homófonas tendrían la misma clave. Está técnica fue patentada en 1918 por Odell y Rusell bajo en nombre de *Soundex*, y originalmente fue utilizado para agrupar nombres en inglés con una pronunciación similar. El algoritmo original de Soundex consistía en asignar a cada palabra una clave con un tamaño máximo de 4 caracteres alfanuméricos. El primer elemento es el carácter inicial de la palabra y el resto un máximo de 3 dígitos que se asignan siguiendo la siguiente tabla:

[8] PHP incorpora la función int levenshtein (string $str1 , string $str2) que calcula la distancia de edición entre las cadenas str1 y str2.

Letra de la palabra	Digito asignado
a, e, i, o, u, h, w, y	*0*
b, f, p, v	*1*
c, g, j, k, q, s, x, z	*2*
d, t	*3*
l	*4*
m, n	*5*
r	*6*

En el proceso de asignación se eliminan los ceros. Los números adyacentes repetidos se fusionan en uno solo. Dado que el algoritmo se diseño para el idioma inglés pondremos un ejemplo en este lenguaje.

Las palabras *mail* (correo) y *male* (varón) son homófonas. Veamos las claves que les corresponden:

- $mail: m004 \rightarrow m4$
- $male: m040 \rightarrow m4$

Para otros idiomas deberíamos construir una tabla de asignación diferente, siguiendo los principios de la fonética del lenguaje. El método es útil en idiomas con una fonética compleja, como el caso del inglés, sin embargo es poco utilizado en lenguas como el español. El algoritmo de generación de sugerencias consistiría en seleccionar todas las palabras con la misma clave que la palabra errónea. Para ello debemos aplicar Soundex a todas las entradas del diccionario. Dado que este valor es fijo puede almacenarse junto a la palabra en el diccionario. Soundex se perfeccionó para corregir algunas deficiencias encontradas dando lugar a Metaphone,

Otra técnica fue propuesta por Pollock y Zamora (Pollock, Joseph J. y Zamora, Antonio, 1984) y se denomina *Speedcop*. Este algoritmo asignaba a cada palabra una clave que se construía colocando todas las consonantes sin repetir juntas en orden de aparición seguidas de las vocales en orden de aparición. Cada consonante o vocal debe aparecer una sola vez en la clave. Por ejemplo a la palabra *murciélago* se le asignaría la clave *mrclguiéao*. Con este sistema se construía un diccionario ordenado alfabéticamente por la clave asignada por Speedcop junto a la palabra. Cuando se encontraba una palabra errónea se calculaba su clave speedcop y su posición en el diccionario. En algoritmo de corrección tomaba como

sugerencias n palabras hacía atrás y hacia delante desde la posición del error. Veamos un ejemplo tomando una parte de un diccionario procesado con speedcop:

Clave speedcop	Palabra
mscluia	*musical*
mscúio	*músico*
msguo	*musgo*
msluo	*muslo*
mstruia	*musitar*
msua	*musa*
msueo	*museo*

Supongamos que se detecta la palabra errónea *musta*. La clave de esta palabra es *mstua*. Esta palabra se situaría entre mstruia y *msua*. Si tomamos 2 palabras hacia arriba y hacia abajo tendríamos como sugerencias *musitar*, *muslo*, *musa* y *museo*.

Los creadores del algoritmo determinaron que una causa de fracaso importante del método era la omisión de letras consonantes al principio de palabra. Para solucionarlo introdujeron una segunda clave, que denominaron clave de omisión, que establece un orden para las consonantes de la clave en función de la probabilidad de error de omisión de cada consonante. Para ello analizaron un corpus y obtuvieron la siguiente secuencia:

j k q x z v w y b f m g p d h c l n t s r

donde *j* sería la letra que menos se omite y *r* la que más. Para construir la clave de omisión se toman las consonantes de la palabra, sin repetir, siguiendo el orden de la secuencia anterior, y continuación se colocan las vocales, sin repetir, en orden de aparición (Mitton, 1996). La clave de omisión para la palabra *murciélago* sería *mgclruiéao*.

2.4.1.3. Corpus de errores.

Esta técnica consiste, básicamente, en consultar un corpus con errores ortográficos habituales en el cual se asocia a cada palabra errónea una lista de sugerencias. Se puede utilizar como soporte adicional a otros métodos dado que el corpus de errores se construirá, normalmente, como resultado de las operaciones de un corrector ortográfico. Cuando éste detecta un error inserta una entrada en el corpus junto con la propuesta de corrección.

Existen diversos corpus de errores ortográficos en Internet. Partiendo de un corpus suficientemente bueno (que incluya muchos errores y sus posibles correcciones) el problema se reduce a una búsqueda, sin embargo en la práctica este método sólo se utiliza como soporte al resto de tecnologías ya que por muchos errores que contenga el corpus es imposible que incorpore todas las posibilidades y, por tanto, sería necesario aplicar otros métodos en el caso en el que la palabra errónea no se encuentre.

2.4.1.4. Reglas.

Se basan en la creación de una base de conocimiento con patrones de errores comunes que se representan mediante reglas de transformación de las palabras erróneas en correctas (Kukich, 1992). El proceso para la generación de candidatos consiste en usar todas las reglas aplicables con la palabra errónea buscando cada resultado en el diccionario y añadiendo a la lista de sugerencias las palabras encontradas. La dificultad del método radica en la codificación de la reglas. Es necesario realizar un estudio de los errores más comunes y establecer reglas de inferencia que a partir de los mismos nos permitan obtener las palabras correctas. Este método es muy dependiente del idioma.

2.4.1.5. N-gramas.

Las técnicas basadas en el uso de n-gramas para la corrección ortográfica son numerosas y, en algunos casos, complejas. Una de las técnicas, propuesta por Bordignon, Fernando R. A., Tolosa, G. H., Peri, J. y Barrientos, Diego (Bordignon, Fernando R. A., Tolosa, G. H., Peri, J. y Barrientos, Diego, 2005), consiste en la creación de un diccionario de n-gramas, donde cada n-grama tiene asociado una lista de palabras en las que aparece. Las palabras erróneas se dividen en n-gramas buscando cada uno de ellos en el diccionario y añadiendo a la lista de sugerencias aquellas palabras que tienen más n-gramas en común con la palabra errónea. Se podría filtrar el resultado midiendo la distancia de edición de cada sugerencia con la palabra errónea y seleccionando aquellas cuya distancia fuese menor.

2.4.2. Técnicas para la corrección automática.

Los métodos anteriores están orientados a generar un conjunto de sugerencias entre las cuales es posible seleccionar aquella que se considera más adecuada. Muchos correctores presentan al usuario una lista de sugerencias cuando encuentran un error, para que sea este quien determine cuál sería la palabra correcta. Este método es válido en entornos en los que el usuario interacciona con el corrector (por ejemplo, procesadores de texto, buscadores web, etc.) sin embargo no sería adecuado en aplicaciones donde la interacción

humana no existe o es mínima (por ejemplo reconocimiento del habla, OCR, etc.). En un entorno de edición de textos convencional es posible aplicar técnicas de selección interactiva de sugerencias, dado que no penalizarían en exceso el rendimiento global del sistema y podrían producir mejores resultados, sin embargo en otros entornos, como por ejemplo sistemas de reconocimiento del habla, serian inviables y es necesario incorporar la corrección automática, aun a riesgo de que los resultados no sean los más correctos. Imaginemos un sistema IVR (del inglés, Interactive Voice Response) que encueste al usuario sobre lo que ha querido decir en cada palabra que identifique como errónea.

En este apartado realizaremos un breve recorrido por los sistemas más utilizados para la corrección ortográfica automática los cuales se basan, principalmente, en modelos probabilísticos. En esencia, estas técnicas tratarían de seleccionar entre la lista de sugerencias aquella que fuese más probable de ser la palabra correcta. Para este estudio sólo se tendrá en cuenta la corrección de palabras aisladas, y no el contexto, siguiendo con los principios generales de la corrección ortográfica. En el caso de que dos o más sugerencias tuviesen la misma probabilidad debería realizarse una selección interactiva o bien elegir una de forma aleatoria, si se quiere que el proceso sea completamente automático.

Bledsoe y Brodwin (1959) fueron pioneros en usar modelos de probabilidad en técnicas de reconocimiento de caracteres (Kukich, 1992). Su método utilizaba el teorema de Bayes para la corrección de posibles errores en un proceso OCR donde W representaría una palabra correcta (existente en un diccionario) y M una cadena de salida del OCR. $P(W|M)$ sería la probabilidad condicional de que W sea la palabra correcta cuando el OCR produce como salida la palabra M. Según el teorema de Bayes:

$$P(W|M) = \frac{P(M|W) * P(W)}{P(M)}$$

$P(M|W)$ sería la probabilidad de que el OCR produzca la salida M cuando la palabra correcta es W. $P(W)$ es la probabilidad de ocurrencia de la palabra W y $P(M)$ la probabilidad de ocurrencia de la palabra M en la salida del OCR. Cada una de estas probabilidades se calcula durante el proceso de reconocimiento en base a la ocurrencia de cada carácter en el texto. La probabilidad de las palabras se calcula mediante la suma de probabilidades de sus caracteres. Esta técnica es la base de la mayoría de correctores ortográficos actúales.

Supongamos que tenemos una lista de sugerencias $W=\{W_1, W_2, ..., W_n\}$ para un determinado error M. La probabilidad de que W_i sea la corrección correcta para el error M, $P(W_i|M)$ vendría dado por la formula:

$$P(W_i|M) = \frac{P(M|W_i) * P(W_i)}{P(M)}$$

Según esto el problema de la corrección automática se limitaría a encontrar el $P(W_i|M)$ con mayor valor para una lista de sugerencias W. Dado que $P(M)$, probabilidad de encontrar el error M, tiene siempre el mismo valor, la palabra seleccionada de la lista W sería aquella donde $P(M|W_i)*P(W_i)$ sea máximo. Esto podría expresarse como:

$$argmax_{W_i} = P(M|W_i) * P(W_i)$$

$P(M|W_i)$ se denomina modelo de error, y contendría la probabilidad de que hayamos escrito M cuando en realidad queríamos escribir W_i. $P(W_i)$ se denomina modelo de lenguaje y contiene las probabilidades de ocurrencia de cada W_i en el lenguaje. El modelo del lenguaje se implementaría mediante un vector con las ocurrencias de cada palabra mientras que el modelo de error se representaría mediante matrices de confusión donde las filas serían las palabras correctas y las columnas las formas erróneas. El valor de la celda (a,b), donde a representa una fila y b una columna, sería la probabilidad de encontrar la forma errónea b cuando la forma correcta es a. Estas matrices se suelen alimentar por el propio corrector durante los procesos de análisis de diferentes textos.

Crear un modelo de error con palabras completas sería tremendamente costoso. La matriz de confusión tendría un número de filas igual al número de términos validos en el diccionario y tantas columnas como posibles errores encontrados para cada palabra valida. Para solucionar esto Mark D. Kernighan, Kenneth W. Church, y William A. Gale propusieron el método que se expone a continuación (Mark D. Kernighan, Kenneth W. Church, y William A. Gale., 1990). Construyeron un modelo de lenguaje analizando un corpus de 44 millones de palabras y calculando $P(W_i)$ en base al número de veces que la palabra W_i aparecía en el corpus. El modelo de lenguaje se completa con un vector de ocurrencia de cada carácter en el corpus y una matriz con la ocurrencia de cada digrama donde la celda (a,b) tomará el valor del número de veces que el digrama ab aparece en el corpus. Estas tablas se utilizarán para calcular las probabilidades en el modelo de error, como veremos a continuación.

Para crear el modelo de error propone emplear cuatro matrices de confusión, una por cada operación básica de edición, donde cada fila y columna representaría un carácter del lenguaje. En la matriz de inserciones la celda *(a,b)* contendría el número de veces que el carácter a fue escrito como ab. En la matriz de eliminaciones la celda *(a,b)* almacenaría el número de veces que los caracteres ab fueron escritos como a. En la matriz de sustituciones la celda *(a,b)* tendría el número de veces que escribimos a en lugar de b. Por último, la matriz de transposiciones guardaría en la celda *(a,b)* el número de veces que los caracteres ab fueron escritos como ba. Las probabilidades se calculan de la siguiente forma:

$$P(M|W) = \frac{add[w_{p-1,}m_p]}{chars[w_{p-1}]}$$

Está formula se aplica cuando la diferencia entre M y W es una inserción. add representa la matriz de confusión de la operación de inserción. La celda $add[w_{p-1,}m_p]$ contendría el número de veces que el carácter w_{p-1} se escribió como los caracteres $w_{p-1}m_p$, donde w_{p-1} sería el carácter que ocupa la posición *1* de la posible sustitución y m_p sería el carácter que ocupa la posición p de la palabra errónea.

De forma análoga las probabilidades para las operaciones de eliminación, sustituciones y transposiciones serían:

$$P(M|W) = \frac{del[w_{p-1,}w_p]}{chars[w_{p-1}, w_p]} \quad \text{Para eliminaciones}$$

$$P(M|W) = \frac{sub[m_{p,}w_p]}{chars[w_p]} \quad \text{Para sustituciones}$$

$$P(M|W) = \frac{rev[w_{p,}w_{p+1}]}{chars[w_p, w_{p+1}]} \quad \text{Para trasposiciones}$$

A modo de ejemplo supongamos que queremos calcular cual sería la mejor opción de corrección para la palabra *camion* habiendo obtenido como sugerencias las palabras *camión* y *camino*. Según el teorema de Bayes:

$$P(camión|camion) = \frac{P(camion|camión) * P(camión)}{P(camion)}$$

$$P(camino|camion) = \frac{P(camion|camino) * P(camino)}{P(camion)}$$

Ahora tendríamos que calcular cual de las probabilidades anteriores es mayor. $P(camión)$ y $P(camino)$ se calcularían a partir de modelo de lenguaje tomando como referencia el número de veces qua aparecen estas palabras en el corpus. $P(camion|camión)$ y $P(camion|camino)$ se calcularían utilizando el método anterior.

La palabra $camion$ se obtiene aplicando la operación de sustitución al carácter o, el cual se cambiaría por $ó$ para obtener la palabra correcta. Por consiguiente $P(camion|camión) = \frac{sub[o,ó]}{chars[ó]}$. $sub[o,ó]$ contiene el número de veces que $ó$ fue escrito como o. $chars[ó]$ contiene en número de veces que $ó$ aparece en el corpus. Para la otra sugerencia tendríamos $P(camion|camino) = \frac{rev[o,n]}{chars[o,n]}$. $rev[o,n]$ contiene el número de veces que los caracteres on fueron escritos como no y $chars[o,n]$ sería en número de ocurrencias de los caracteres on en el corpus.

Las matrices de confusión se alimentarían entrenando el corrector mediante técnicas de bootstrapping.

Este método es válido para errores non-word con distancia 1 de la palabra correcta.

2.5. Introducción a la detección y corrección de errores gramaticales.

La detección y corrección de errores gramaticales (real-word) está fuera del alcance de este estudio, sin embargo creemos conveniente realizar una pequeña introducción que, con las técnicas analizadas anteriormente, creemos resultará sencilla de entender. Para ello describiremos, de forma muy simple, un método basado en el estudio de la literatura sobre el tema basado en el análisis morfológico de las frases de un texto. Se desconoce la precisión y la eficacia del método expuesto ya que aunque, como se ha comentado, se basa en técnicas propuestas por diferentes autores, es de factura propia y totalmente teórico con la única finalidad de introducir la detección y corrección de errores real-word sirviendo como entrada para el impulso de nuevos estudios y desarrollos en esta área.

La técnica de búsqueda en diccionarios, tal y como se ha expuesto, para la detección de errores gramaticales no es válida, dado que el error no se produce en la palabra sino en el contexto de la misma. Por consiguiente se requiere realizar un análisis de dicho contexto para determinar si existen errores real-word. Supongamos el texto *La león son carnívoros.* Todas las palabras del texto son correctas desde un punto de vista ortográfico sin embargo existen varios errores gramaticales. En primer lugar realizamos un análisis morfológico

asignando a cada palabra una etiqueta que indicaría el tipo de palabra, el género, el número y la persona, en el caso de verbos. El proceso consistiría, de forma esencial, en verificar la correspondencia de estas etiquetas desde el punto de vista gramatical. Para ello es necesario que el diccionario utilizado en la fase de corrección ortográfica incorpore las etiquetas morfológicas de cada palabra, la cuales se pueden registrar durante la fase de búsqueda de errores non-word.

El primer paso para la detección de errores gramaticales consistiría en verificar la correspondencia de las etiquetas de género y número en palabras adyacentes. Para ello dividimos el texto el bigramas de palabras y comprobamos la correspondencia de cada etiqueta de género y número.

En la frase anterior tendríamos varios errores de concordancia:

La león error de género

león son error de género

son carnívoros correcto

En una segunda fase, y mediante la aplicación de reglas, generaríamos todas las posibilidades de concordancia para los pares en los que exista una palabra que esté incluida en un bigrama marcado como erróneo.

Opciones bigrama 1	Opciones bigrama 2	Opciones bigrama 3
La leona	*león son*	*son carnívoros*
El león	*león es*	*es carnívoro*

Aunque en el último bigrama no se había detectado un error debe incluirse en este proceso ya que la palabra *son* forma parte de otro bigrama en el que si se detectó un error.

Ahora debemos seleccionar las palabras con mayor ocurrencia y tomar los bigramas en los que aparecen para construir la frase correcta:

Palabra	Ocurrencias
La	1
leona	1
león	2
son	1

Palabra	Ocurrencias
es	2
carnívoro	1
carnívoros	1

Las palabras más frecuentes son *león* y *es*. Seleccionamos los bigramas donde aparecen y obtenemos la frase correcta, *El león es carnívoro.*

Este método detecta y corrige un tipo concreto de errores gramaticales. Para otro tipo de errores, en los que estén implicados errores sintácticos, signos de puntuación, etc. sería necesario aplicar otras técnicas.

Por ejemplo, para corregir errores sintácticos podríamos dividir el texto en trigramas de etiquetas morfológicas de tipo de palabra, con objeto de comprobar en un corpus la probabilidad de ocurrencia de determinadas combinaciones de etiquetas. Supongamos que tenemos la frase *La el león es carnívoro.* El análisis morfológico de género y número no detectaría ningún error por lo procederíamos a realizar un análisis de etiquetas de tipo de palabra. Obtendríamos los siguientes trigramas de etiquetas:

La el león: <artículo> <artículo> <sustantivo>
el león es: <artículo> <sustantivo> <verbo>
león es carnívor: <sustantivo> <verbo> <adjetivo>

Ahora debemos buscar en un corpus de etiquetas la frecuencia de cada una de estos trigramas con objeto de determinar la probabilidad de ocurrencia de cada uno en el lenguaje, señalando como posibles errores aquellos con poca probabilidad. Aparentemente el primer trigrama, *<artículo> <artículo> <sustantivo>*, tendrá poca probabilidad, lo que indicaría un posible error.

2.6. Correctores ortográficos.

Siendo la corrección ortográfica automática un problema tan antiguo como la informática, parecería lógico que existan innumerables desarrollos que implementen esta funcionalidad. En la actualidad cualquier aplicación o dispositivo que permita la redacción de textos suele incorporar un corrector ortográfico, además de otras herramientas tales como OCR, IVR, etc. Sin embargo en número de correctores ortográficos disponibles en el mercado no es tan amplio, además muchos de los correctores disponibles se basan en otros desarrollos

anteriores. Por ejemplo GNU Aspell es una evolución de Ispell y Hunspell está basado en Myspell. Estos correctores suelen ser de fuentes abiertas, lo que ha motivado que muchas otras aplicaciones los integren como herramientas de soporte para la corrección ortográfica. Hunspell es el corrector usado por Open Office, Libre Office, Thunderbird y Firefox, entre otros.

También existen correctores ortográficos online, que permiten el análisis de textos desde una página web, aunque como en el caso anterior suelen basarse en los desarrollos comentados. Existen también APIs para la corrección ortográfica, que podemos integrar con nuestros desarrollos, y servicios web para la corrección ortográfica, pero estos son todavía menos numerosos que las soluciones comentadas. Sólo hemos encontrado un API REST para la corrección orográfica de texto, que además es de pago (http://www.webspellchecker.net/web-services.html). Se ha validado una versión de prueba y comprobamos que sus funcionalidades son muy similares a nuestro desarrollo, devolviendo incluso un fichero XML, con los resultados del análisis ortográfico, con un formato similar a SWCOM. También existen otros servicios web desarrollados con SOAP (http://wsf.cdyne.com/SpellChecker/check.asmx?op=CheckTextBodyV2) con funcionalidades similares.

2.7. Servicios web.

El desarrollo de sistemas aislados con módulos especializados en la ejecución de determinadas funciones es un paradigma de la computación que cada vez tiene menos sentido en un mundo hiperconectado. Sería un completo error obviar las tecnologías que Internet pone a nuestro alcance y no integrarlas con nuestros desarrollos para obtener nuevas herramientas, más completas y eficaces. Lógicamente esto no siempre será posible ya que, para nuestra desgracia, en el juego también intervienen factores como la seguridad o la calidad de las comunicaciones que en muchas ocasiones darán al traste nuestras expectativas. La computación actual, sobre todo en el ámbito empresarial, es pragmática y se orienta por completo a la solución de problemas concretos. Lejos de aquellos enormes desarrollos orientados a cubrir todas las operaciones dentro de un área determinada, con módulos fuertemente acopados y funcionalidades que, en muchos casos, nunca se utilizaban, la demanda actual de soluciones TIC se centra en el soporte a los servicios.

En un sentido estricto un servicio es un conjunto de actividades destinadas a satisfacer una necesidad. Los servicios que presta una empresa serían la base de su negocio y estarían orientadas a la satisfacción de sus clientes. Una empresa TIC podría prestar servicios de

consultoría, desarrollo de aplicaciones, integración de sistemas, etc. Cada uno de estos servicios podría a su vez dividirse en otros, por ejemplo el desarrollo de aplicaciones podría englobar los servicios de análisis, diseño, construcción, implantación, etc., y estos a su vez en otros más simples hasta llegar a los servicios más esenciales. Pero una empresa no sólo presta servicios a sus clientes (externos) sino que también necesita dar soporte a determinados procesos internos tales como la gestión de personal, gestión económico-financiera, gestión de la producción, marketing, etc. Estos procesos también pueden considerarse servicios que la empresa debe prestar para satisfacer las necesidades de sus empleados, los cuales son considerados clientes (internos).

2.7.1. Arquitectura orientada a servicios.

La alineación de los requerimientos de las organizaciones y sus sistemas de información ha dado lugar a un nuevo modelo de arquitectura para la construcción de sistema distribuidos que se demonina SOA (del inglés Service Oriented Architecture, Arquitectura Orientada a Servicios). Un sistema desarrollado siguiendo este modelo es altamente escalable y sus módulos dan cobertura a necesidades concretas de la organización a la vez que son fácilmente integrables con otros sistemas. Para SOA un servicio es, básicamente, una función que acepta una llamada y devuelve una respuesta mediante una interfaz bien definida. Los servicios son consumidos por los clientes, que pueden ser personas, aplicaciones u otros servicios. Podemos caracterizar a los servicios por (Miko Matsumura, Bjoern Brauel y Jignesh Shah, 2009) :

- **Lo que el servicio hace.** Un servicio proporciona una capacidad para su consumidor, por ejemplo cambiar el email de un cliente, imprimir una factura, corregir un texto, etc.
- **Como se utiliza.** Un servicio cuenta con un método específico para poder usarlo, lo que se llama invocación, y presenta una interfaz bien definida para poder acceder a sus prestaciones.

En un servicio SOA no se define explícitamente dónde está ubicado ni cómo funciona. Los servicios se pueden acceder de forma remota, siguen el principio de caja negra, sabemos lo que hacen pero no cómo lo hacen y pueden acoplarse para construir nuevos servicios, o ensamblarse en secuencias para construir procesos (Miko Matsumura, Bjoern Brauel y Jignesh Shah, 2009).

Centrándonos en nuestro caso de estudio, un corrector ortográfico podría proveerse como un servicio. La principal ventaja de esta solución es que podríamos integrar la corrección ortográfica con cualquier aplicación, o servicio, de forma simple, sin preocuparnos de las tecnologías utilizadas por cada aplicación. Tradicionalmente los correctores ortográficos se limitaban a procesadores de texto, clientes de correo electrónico y otras pocas aplicaciones ofimáticas. Por ejemplo, en una arquitectura SOA podríamos incorporar la corrección ortográfica a los campos de texto de un programa de facturación. Esto no significa que en otros modelos de arquitectura de sistemas no sea posible hacer esta integración pero, con toda seguridad, será más complejo.

2.7.2. Servicios Web.

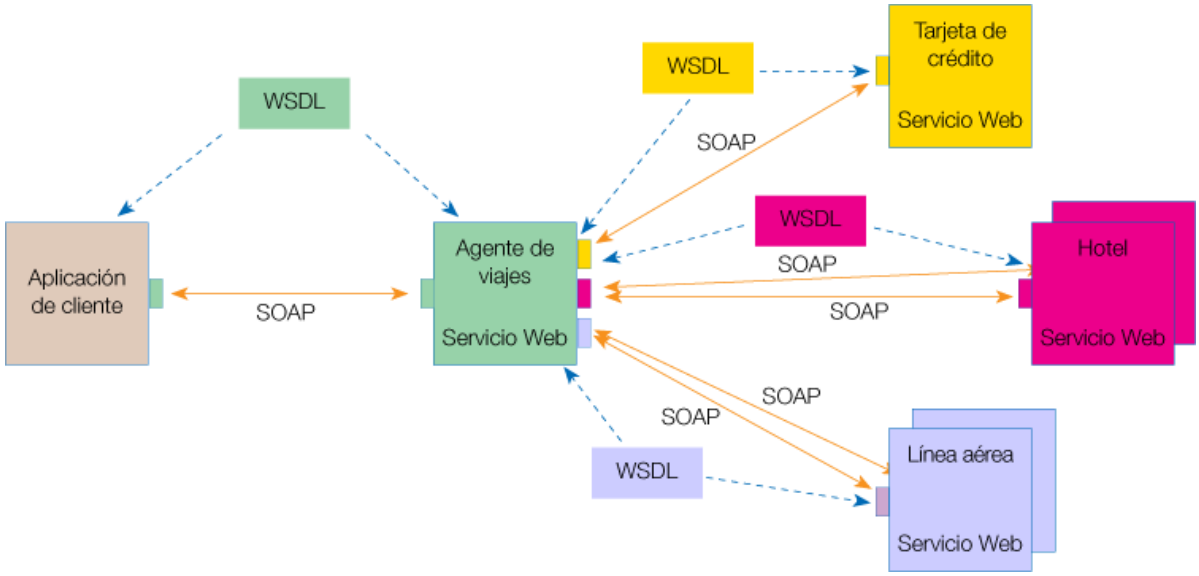
Los Servicios Web suelen ser APIs que pueden ser accedidas dentro de una red, principalmente Internet, y son ejecutados en el sistema que los aloja. Permiten la intercomunicación entre sistemas de cualquier plataforma y se utilizan en una gran variedad de escenarios de integración. La implantación de una solución basada en SOA no exige implantar servicios Web, no obstante los servicios Web son la forma más habitual de implementar SOA, pero no es suficiente tener un amplio repositorio de servicios web para tener una SOA. En cualquier caso los servicios web son actualmente una de las mejores soluciones para la integración de aplicaciones y la construcción de nuevas soluciones mediante la composición de otros servicios existentes.

No existe una definición formal de servicio web. De hecho la W3C (World Wide Web Consortium), responsable de definir la arquitectura para los servicios web, acuña diferentes definiciones:

- Software diseñado para soportar interoperabilidad entre máquinas sobre una red. (W3C, Web Service Glossary, 2004).
- Conjunto de aplicaciones o tecnologías con capacidad para interoperar en la Web. Estas aplicaciones o tecnologías pueden intercambiar datos con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la web. (W3C, Guía breve de servicios Web, 2004)
- Aplicación software identificada por un URI (Uniform Resource Identifier), cuyas interfaces públicas se pueden definir y describir mediante documentos XML. Su definición pude descubrirse por otros sistemas con objeto de que estos puedan interactuar de la forma prescrita por su definición, utilizando mensajes XML

intercambiados mediante protocolos de Internet. (W3C, Web Service Architecture Requirements, 2004).

Muchas de las aplicaciones actuales de Internet serían impensables sin la existencia de los servicios web. Plataformas para la compra de billetes de avión online, que consultan los horarios y tarifas de cientos de compañías aéreas, o comparadores de seguros, hipotecas, tarifas telefónicas, etc., utilizan servicios web para acceder a la información de varios sistemas. En la siguiente ilustración (W3C, Guía breve de servicios Web, 2004) se explica esta funcionalidad.

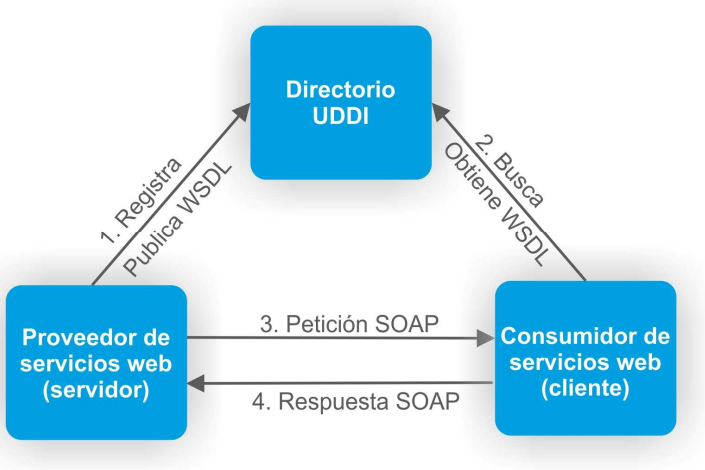


Supongamos un usuario que utiliza un navegador (aplicación de cliente) para contratar sus vacaciones accediendo a la web de una agencia de viajes. El viaje solicitado por el usuario puede requerir información de otros sistemas (líneas áreas, hoteles, etc.) que la aplicación de la agencia de viajes puede capturar invocando a los servicios web que proveen las aplicaciones de los diferentes sistemas. Una vez confirmado el viaje contratado se registrarán las reservas correspondientes en los diferentes sistemas y se procederá al pago, también mediante la invocación de servicios web.
En este proceso intervienen varias tecnologías que posibilitan el intercambio de información.

- **SOAP.** Protocolo Simple de Acceso a Objetos (del inglés *Simple Object Access Protocol*), está basado en XML y permite el intercambio de información entre los diferentes servicios. Tiene la capacidad de transmitir información compleja a través

de HTTP. SOAP especifica el formato de la información de los mensajes en la comunicación entre cliente y servidor.

- **WSDL.** Lenguaje de Descripción de Servicios Web *(*del inglés, *Web Services Description Language*). Permite tener una descripción de un servicio web: formato de los mensajes, requisitos del protocolo, forma de comunicación, etc.
- **UDDI.** (del inglés *Universal Description, Discovery and Integration*). Aunque no se refleja en el gráfico anterior, este estándar es de suma importancia en el uso de los servicios web. Es un directorio de servicios web. Se encarga de las funciones de descubrimiento y descripción de los servicios web. Se accede mediante mensajes SOAP a los cuales se responde con documentos WSDL. La siguiente imagen ilustra el funcionamiento de UDDI.



1. El proveedor registra el servicio web en directorio UDDI, almacenando el documento WSDL que describe dicho servicio.
2. El consumidor busca en el directorio UDDI un servicio web que pueda realizar las operaciones demandadas. El directorio UDDI responde con un documento WSDL que describe el servicio a usar.
3. Una vez seleccionado el servicio, el consumidor lo invoca mediante el envío de un mensaje SOAP que se construirá siguiendo la descripción del documento WSDL.
4. El proveedor recibe la petición, ejecuta los procesos correspondientes y envía un mensaje SOAP al consumidor con los resultados obtenidos.

2.7.3. REST.

Los servicios web descritos por la W3C, denominados comúnmente servicios web SOAP, pueden resultar complicados en determinados escenarios donde los requerimientos son muy simples. Los servicios web SOAP son difíciles de consumir y requieren el desarrollo de

clientes específicos que permitan acceder a los métodos publicados por los mismos. Esto ha provocado que muchos desarrolladores busquen alternativas para la creación de servicios Web. El término REST (del inglés Representational State Transfer) se originó en el año 2000, en una tesis doctoral sobre “Estilos arquitectónicos y diseño de arquitecturas de software basadas en la red” escrita por Roy Fielding, uno de los principales autores de la especificación del protocolo HTTP, quien define REST como un estilo de arquitectura para sistemas hipermedia distribuidos (Fielding, 2000), Si bien el término REST se refería originalmente a un conjunto de principios de arquitectura, actualmente se usa en un sentido más amplio para describir cualquier interfaz entre sistemas que utilice el protocolo HTTP directamente, para obtener datos u ordenar la ejecución de operaciones sobre los datos, sin necesidad de abstracciones adicionales en el intercambio de mensajes, como por ejemplo en SOAP.

Fielding parte de la arquitectura de la World Wide Web a la que aplica una serie de restricciones para definir REST que, como veremos, están implícitas el protocolo HTTP. Estas restricciones son:

- **Arquitectura cliente-servidor**. REST debe diferenciar de forma clara los componentes básicos en un intercambio de información; cliente (consumidor de servicios) y servidor (proveedor de servicios). Un componente de servidor, que ofrece un conjunto de servicios, escucha peticiones sobre esos servicios. Un componente de cliente que quiere consumir un servicio, envía una petición al servidor a través de un conector. El servidor puede rechazar o procesar la solicitud, enviando una respuesta al cliente. La separación de responsabilidades es el principio detrás de las restricciones de cliente-servidor. Esta separación permite evolucionar los dos componentes de forma independiente sin modificar la interfaz de usuario.
- **Comunicaciones sin estado (stateless).** La comunicación entre cliente y servidor debe ser sin estado. Esto significa que cada solicitud del cliente debe contener toda la información necesaria para comprender la petición no pudiendo almacenar ningún contexto en el servidor. Esto supone una desventaja en algunos entornos ya que puede disminuir el rendimiento de la red al aumentar la transmisión de datos repetidos enviados en una serie de peticiones, al no permitir que dichos datos se almacenen en un contexto compartido del servidor. Esta limitación es fuente de constantes debates en la industria, sobre todo desde que las API REST se han posicionado como una de las opciones más adecuadas para desplegar el Internet de

las cosas (IoT). Por este motivo se han desarrollado tecnologías basadas en REST que permiten conexiones con estado (stateful), como WebSockets.

- **Cacheable**. Esta restricción implica que el servidor debe definir algún modo de cachear las peticiones para aumentar el rendimiento. Las respuestas se almacenan en una memoria caché del cliente quien puede volver a utilizar los datos de respuesta para solicitudes equivalentes
- **Sistema por capas**. El sistema no debe forzar al cliente a saber por que capas se tramita la información, lo que permite que el cliente conserve su independencia con respecto a estas capas.
- **Interfaz uniforme**. La interfaz de comunicación no depende del servidor al que estamos haciendo las peticiones, ni del cliente que estamos usando para consumir los servicios, lo que garantiza que no importa quien haga las peticiones, ni quien las reciba, siempre y cuando ambos cumplan una interfaz definida de antemano.

2.7.3.1. Elementos de la arquitectura REST

Los elementos principales de una arquitectura REST son:

- **Recurso.** La clave de la abstracción de la información en REST es el recurso. Toda información en REST es un recurso. Posemos considerar un recurso como una entidad que representa un concepto que puede ser accedido públicamente. Por ejemplo, en una tienda online los artículos serían recursos. Esta es una de las principales diferencias entre REST y SOAP. Mientras SOAP publica métodos, REST publica recursos. En el ejemplo anterior, para mostrar los datos de un determinado artículo de la tienda online SOAP implementaría un método *VerArtículo* al que se pasaría como parámetro el código del artículo. En REST existiría un recurso para cada artículo. Por ejemplo, *Articulo0001* sería un recurso que representaría al artículo 0001 de la tienda.
- **Identificador de recurso.** Cada recurso en REST tiene un identificador único y global que se representa mediante URIs (del inglés *Uniform Resource Identifier*). Mientras que en SOAP utilizaríamos el método VerArtículo(0001) para acceder al artículo 0001 de la tienda online, en REST utilizaríamos únicamente su identificador, por ejemplo http://www.mitiendaonline.com/articulos/0001.
- **Representación.** La representación de un recurso es un formato de datos concreto usado para la transferencia de una copia del recurso entre el cliente y el servidor. En la implementación del servicio REST se establece que información de los recursos

es accesible desde el exterior y que representaciones se soportan. Por ejemplo, una representación de “Articulo0001” podría ser un documento XML con la información accesible de este. Podemos también implementar una representación en formato HTML, TXT, JSON, JPEG, etc.

2.7.3.2. REST y HTTP

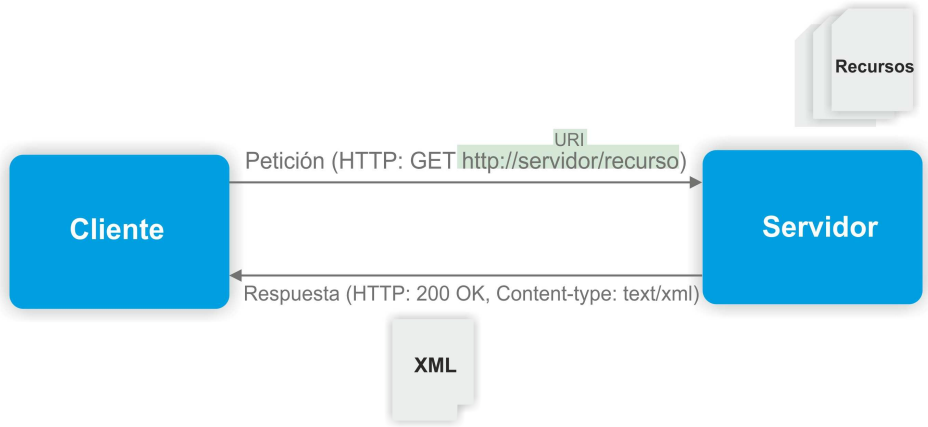
HTTP se adapta completamente a las restricciones impuestas por la arquitectura REST por tanto no es necesario definir nuevos estándares para implementar servicios RESTful. Sin embargo para que una aplicación sea considerada RESTful es necesario que siga determinadas normas, que pondremos en relación a como se implementan las mismas en HTTP (Fernández, 2013):

- **Identificación de recursos.** Toda aplicación REST debe poder identificar sus recursos de manera uniforme. HTTP implementa esto usando URIs (Uniform Resource Identifier). Cualquier recurso REST tendrá una identificación mediante una URI. La identificación de un recurso no incluye la representación del mismo.
- **Recursos y representaciones**. REST puede definir la manera en que podemos interactuar con la representación de los recursos. Sólo se permiten cuatro operaciones; CREATE, READ, UPDATE y DELETE. Las representaciones se dejan a instancias de la implementación final del programa. Dado que HTTP define distintas cabeceras de tipos, y un contenido en la respuesta, un servicio RESTful podrá enviar el contenido en el formato que quiera.
- **Mensajes autodescriptivos**. Cuando hacemos peticiones a un servidor, éste debería devolver una respuesta que nos permita entender, sin lugar a duda, cual ha sido el resultado de la operación, así como si dicha operación es cacheable, si ha habido algún error, etc. HTTP implementa esto a través del estado y una serie de cabeceras. El cómo se usan estos estados y cabeceras depende por entero de la implementación del servicio RESTful. REST no fuerza el contenido de dichos elementos, por lo que el programa que se ejecuta en el servidor, y que en última instancia accede a los recursos y opera con ellos, tiene la responsabilidad de devolver estados y cabeceras que se correspondan con el estado real de la operación realizada.

Veamos ahora un ejemplo sobre cómo se relacionan los servicios RESTful y HTTP. Un servicio RESTful contendrá lo siguiente:

- **URI del recurso**. Por ejemplo: http://www.mitiendaonline.com/articulos/0001, nos daría acceso al recurso Articulo0001)
- **El tipo de la representación de dicho recurso**. Por ejemplo, podemos devolver la cabera “Content-type: text/xml”, por lo que el cliente sabrá que el contenido de la respuesta es un texto con formato XML, que podrá procesar como prefiera. El tipo es arbitrario, siendo los más comunes JSON, XML y TXT.
- **Operaciones.** HTTP incorpora los métodos (POST, GET, PUT y DELETE) que dan soporte a cada una de las operaciones permitidas en REST, enunciadas más arriba.

El siguiente gráfico ilustra el funcionamiento de un servicio RESTful:



El cliente hace una petición, mediante una URI, al servidor para visualizar (GET) un recurso. El servidor recupera el recurso correspondiente y lo devuelve en el formato indicado (text/xml) al cliente.

3. Identificación de requisitos y metodología para la planificación y gestión del proyecto.

La finalidad de este capítulo es analizar y documentar las necesidades funcionales que deberán ser soportadas por la aplicación a desarrollar. Para ello se elaborará una Especificación de Requisitos de Software siguiendo el formato propuesto por el estándar IEEE 830-1998.

3.1. Propósito.

Se realizará un análisis con objeto identificar las necesidades funcionales y no funcionales para la implementación de una aplicación en la nube capaz de evaluar la ortografía de un texto y generar un informe con los errores encontrados. La particularidad de este servicio es que se accederá a él a través de llamadas remotas (SOAP o REST) de forma que el servicio sea integrable en aplicaciones de terceros. Por otra parte el validador ortográfico debería presentar un informe con las sugerencias para cada error encontrado, estadísticas, etc. Estamos acostumbrados a una evaluación ortográfica generada por software, como la integrada en procesadores de texto o navegador web. En ambos casos, la interfaz asume que el usuario está interactuando con el equipo. Sin embargo, una evaluación desatendida puede ser útil en entornos de procesado masivo de datos, en los que no es posible esta rápida interactividad con el usuario.

3.2. Alcance.

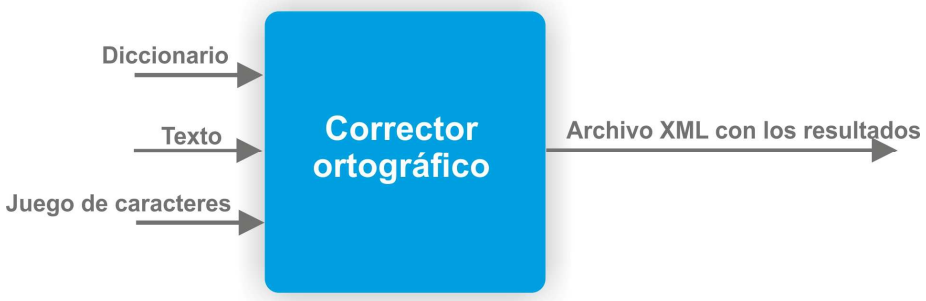
Se pretende desarrollar un servicio web RESTful, que denominaremos SWCOM (Servicio Web para la Corrección Ortográfica Multilingüe), orientado a evaluar la ortografía de un texto en cualquier idioma. Se limita el alcance al análisis de palabras non-word. Para realizar este análisis utilizará un diccionario del idioma del texto a analizar y el nombre de un juego de caracteres (ISO-8859-1, UTF-8, UFT-16, etc) que indicará al corrector el encoding utilizado por el texto a corregir y el diccionario, los cuales deberán utilizar el mismo encoding. Tanto el texto como el diccionario podrán estar en una máquina local o en Internet, para lo cual sus nombres deberán pasarse al servicio como URLs. Como resultado presentará documento XML que deberá contener la siguiente información:

- **Excepciones.** El servicio debe informar de todos aquellos errores en tiempo de ejecución que impidan que el proceso se ejecute de forma correcta. Dado que la

evaluación debe ser desatendida es necesario que esta información se incluya en el documento de salida.

- **Texto a analizar**. URL del texto a analizar y número de palabras encontradas en el mismo.
- **Diccionario a utilizar**. URL del diccionario a utilizar y el número de palabras existentes en el mismo. El diccionario será un fichero de texto con el vocabulario extendido de un idioma donde cada entrada ocupará una línea.
- **Palabras erróneas**. Para cada palabra errónea se especificará la línea del texto en la que aparece y una lista de sugerencias para la corrección de la misma.
- **Número de errores.** Número de palabras erróneas encontradas en el texto.
- **Tiempo de ejecución.** Tiempo de ejecución del proceso.

Desde el punto de vista del usuario el comportamiento del servicio sería el siguiente:



Las especificaciones generales señalaban la posibilidad de implementar el servicio utilizando REST o SOAP. Finalmente, y después de estudiar el problema, nos hemos decantado por REST, principalmente por su facilidad para consumir, que nos permitirá utilizar cualquier navegador para invocar el servicio, y su sencillez para desarrollarlo y desplegarlo, frente a los servicios SOAP.

Sin embargo la restricción stataless impuesta a los servicios REST hizo que, en un principio, nos cuestionásemos la elección. El principal motivo es el envío del diccionario en cada invocación al servicio, lo cual penalizaría el rendimiento global de la solución. En cualquier caso la W3C en su recomendación sobre SOAP versión 1.2. (W3C, SOAP Versión 1.2 Part 0: Primer (Second Edition), 2007) indica que SOAP es fundamentalmente stateless, aunque no supone una restricción como el caso de REST. Una posibilidad que se planteo para no tener que pasar el diccionario en cada invocación fue almacenar los diccionarios en una base de datos del servidor y pasar exclusivamente el idioma, o tipo de diccionario a utilizar. Esta solución obligaría a implementar un sistema de gestión de diccionarios e influiría

negativamente en la versatilidad del sistema a la hora de permitir al usuario decidir sobre que diccionario quiere utilizar en cada sesión de corrección.

Respecto al requerimiento de pasar el juego de caracteres, nos parece más interesante que limitar la funcionalidad un solo encoding, como hacen otros correctores. No obstante, y por cuestiones obvias, es necesario que texto y diccionario estén codificados con el mismo encoding y que además sea el que pasemos como parámetro.

3.3. Descripción general.

3.3.1. Perspectiva del producto.

El corrector ortográfico SWCOM será un servicio diseñado para operar en entornos WEB mediante llamadas remotas al mismo. Como cualquier servicio web su función principal será la integración con otros sistemas, la cual se conseguirá fácilmente mediante la implementación de clientes que entreguen al servicio los parámetros requeridos y procesen la salida en formato XML.

3.3.2. Funciones del producto.

Las funciones principales del producto son evaluar la ortografía de un texto y generar un informe con los errores encontrados y las sugerencias correspondientes a cada uno. Debe facilitar también una seria de estadísticas que nos ayuden a valorar la fiabilidad y el rendimiento del sistema (palabras analizadas, errores encontrados, tiempo de ejecución, etc.). El informe de salida se producirá en formato XML con objeto de que sea fácilmente interpretable por otras aplicaciones y servicios.

3.3.3. Características de los usuarios.

Aunque, a priori, el público objetivo del servicio son desarrolladores que decidan integrar el producto con otras soluciones, o implementar nuevas funcionalidades que mejoren el mismo, cualquier usuario podría usar este servicio directamente utilizando un navegador y escribiendo el URI del recurso correspondiente.

3.3.4. Restricciones.

3.3.4.1. Políticas reguladoras.

El servicio se desarrollará íntegramente con software de licencias abiertas y su código fuente deberá ser publicado bajo alguna licencia de código libre en algún repositorio online a elegir (GitHub, SourceForge, Google Code, etc.). Se podrá escoger la licencia concreta, según los términos del servicio del repositorio elegido, aunque se recomiendan las más populares: GPL, LGPL, Apache, MIT, AFL, etc.[9].

3.3.4.2. Interfaces con otras aplicaciones.

La interfaz con otras aplicaciones tendría las restricciones impuestas por la propia arquitectura REST y el protocolo HTTP.

3.4. Requisitos específicos.

Esta sección contiene los requisitos del servicio con un nivel de detalle suficiente para diseñar e implementar el mismo, así como realizar pruebas orientadas a comprobar que el servicio satisface todos los requerimientos.

3.4.1. Interfaces externas.

En este apartado describiremos todas las entradas y salidas del sistema a desarrollar.

3.4.1.1. Entradas.

Las entradas identificadas en las especificaciones iniciales y el alcance del sistema son:

- **URL del fichero de texto a analizar.** Admitirá cualquier formato de URI: http, ftp, file, etc.
- **URL del diccionario a usar para el análisis.** Al igual que para el fichero de texto admitirá cualquier formato de URL.
- **Juego de caracteres.** Es una cadena con el nombre de una norma de codificación de caracteres (encoding) como por ejemplo ISO-8859-1, UTF-8, UTF-16, etc.

[9] Esta restricción se toma de la Guía General del Trabajo de Fin de Grado.

3.4.1.2. Salidas.

Según las especificaciones iniciales la representación de los recursos de salida se realizará en formato XML y contendrá los siguientes datos, ya declarados en el alcance del sistema:

- **Excepciones.**
- **URL del texto a analizar y número de palabras en el mismo.**
- **URL del diccionario y número de palabras en el mismo.**
- **Palabras erróneas, líneas en la que aparecen y lista de sugerencias para cada una de ellas.**
- **Número de errores ortográficos encontrados.**
- **Tiempo de ejecución del proceso.**

3.4.2. Funciones.

En este apartado se describirán los requisitos funcionales del sistema, cada una de los cuales dará lugar a una función. Cada función aceptará unos datos de entrada, los procesará y producirá unos datos de salida. Una función podrá descomponerse en subfunciones. Para cada función se detallarán los siguientes datos:

Código función	*Cada requisito se identifica con un código único*	**Nombre**	*Nombre de la función*	
Descripción	*Descripción general de la función*			
Entradas	*Entradas que recibe la función*		Origen	*Origen de las entradas*
Salidas	*Salidas que produce la función*		Destino	*Destino de las salidas*
Restricciones	*Restricciones impuestas a la función*			
Descripción del proceso	*Descripción del proceso que ejecuta la función*			
Excepciones	*Errores que pueden producirse durante la ejecución de la función*			

Atendiendo a las especificaciones iniciales podemos identificar, de forma clara, dos funciones principales:

- Evaluar ortografía.
- Generar informe.

3.4.2.1. Evaluar ortografía.

Código función	SWCOM_R_01	Nombre	Evaluar ortografía	
Descripción	Evalúa la ortografía de un texto, detectando los errores y generando sugerencias para cada uno.			
Entradas	URL texto URL diccionario Juego de caracteres		Origen	Generar informe (SWCOM_R_02)
Salidas	Excepciones Nº palabras del texto Nº palabras del diccionario Palabras erróneas y líneas en las que aparecen Sugerencias Nº errores		Destino	Generar informe (SWCOM_R_02)
Restricciones	Corrección de errores non-word Texto y diccionario deben utilizar en mismo encoding que se indicará mediante la entrada juego de caracteres			
Descripción del proceso	El proceso recibirá como entradas una URL de un texto, la URL de un diccionario y un juego de caracteres. El sistema dividirá el texto en palabras para su posterior análisis y contará las palabras de mismo. Se cargará el diccionario en una estructura de datos que permita realizar búsquedas de forma óptima y se contarán las palabras existentes en el mismo. Para cada palabra del texto se comprobará si está en el diccionario. Si no se encuentra se entenderá que es un error, se almacenará la línea del texto en la que aparece y se generará un lista de sugerencias para su corrección.			
Excepciones	Es necesario controlar que las URL tengan una sintaxis correcta, que existan los ficheros correspondientes (texto y diccionario) y que puedan ser leídos, y que el juego de caracteres sea correcto. Por otra parte también deberá controlar que la llamada al servicio se produce según la interfaz definida por el mismo, y que el nombre de los recursos y sus parámetros sean correctos.			

Este requisito puede dividirse en los siguientes subrequisitos:

Dividir texto

Cargar diccionario

Buscar palabra

Generar sugerencias

3.4.2.2. Dividir texto en palabras.

Código función	SWCOM_R_01_01	Nombre	Dividir texto
Descripción	Divide un texto en palabras		
Entradas	URL texto Juego de caracteres	Origen	Evaluar ortografía (SWCOM_R_01)
Salidas	Nº de palabras Estructura de datos con palabras del texto	Destino	Evaluar ortografía (SWCOM_R_01)
Restricciones	La codificación de caracteres del texto deberá ser la especificada por la entrada juego de caracteres		
Descripción del proceso	Leerá el texto desde la URL especificada y lo dividirá en palabras. Cargará las palabras en una estructura de datos que permita su óptimo procesamiento. Dado que es necesario comprobar todas las palabras el acceso a la estructura de datos será secuencial. Deberá contar el número de palabras leídas.		
Excepciones	URL no válida, fichero inexistente o error de entrada al leer el mismo.		

3.4.2.3. Cargar diccionario.

Código función	SWCOM_R_01_02	Nombre	Cargar diccionario
Descripción	Carga el diccionario		
Entradas	URL diccionario Juego de caracteres	Origen	Evaluar ortografía (SWCOM_R_01)
Salidas	Nº de palabras Estructura de datos con palabras del diccionario	Destino	Evaluar ortografía (SWCOM_R_01)
Restricciones	La codificación de caracteres del diccionario deberá ser la especificada por la entrada juego de caracteres		
Descripción del proceso	Leerá el diccionario desde la URL especificada. Cargará las palabras en una estructura de datos que permita su óptimo procesamiento. En este caso debemos emplear una estructura de datos que minimice el número de accesos en cada búsqueda.		
Excepciones	URL no válida, fichero inexistente o error de entrada al leer el mismo.		

3.4.2.4. Buscar palabra.

Código función	SWCOM_R_01_03	Nombre	Buscar palabra	
Descripción	Busca una palabra en el diccionario			
Entradas	Palabra a buscar		Origen	Evaluar ortografía (SWCOM_R_01)
Salidas	¿Existe palabra?		Destino	Evaluar ortografía (SWCOM_R_01)
Restricciones	No existen			
Descripción del proceso	Busca una palabra en el diccionario y devuelve si existe o no			
Excepciones	No existen			

3.4.2.5. Generar sugerencias.

Código función	SWCOM_R_01_04	Nombre	Generar sugerencias	
Descripción	Genera una lista de sugerencias para una palabra errónea			
Entradas	Palabra errónea		Origen	Evaluar ortografía (SWCOM_R_01)
Salidas	Sugerencias		Destino	Evaluar ortografía (SWCOM_R_01)
Restricciones	No existen			
Descripción del proceso	Genera una lista de palabras candidatas para la sustitución de una palabra errónea.			
Excepciones	No existen			

3.4.2.6. Generar informe.

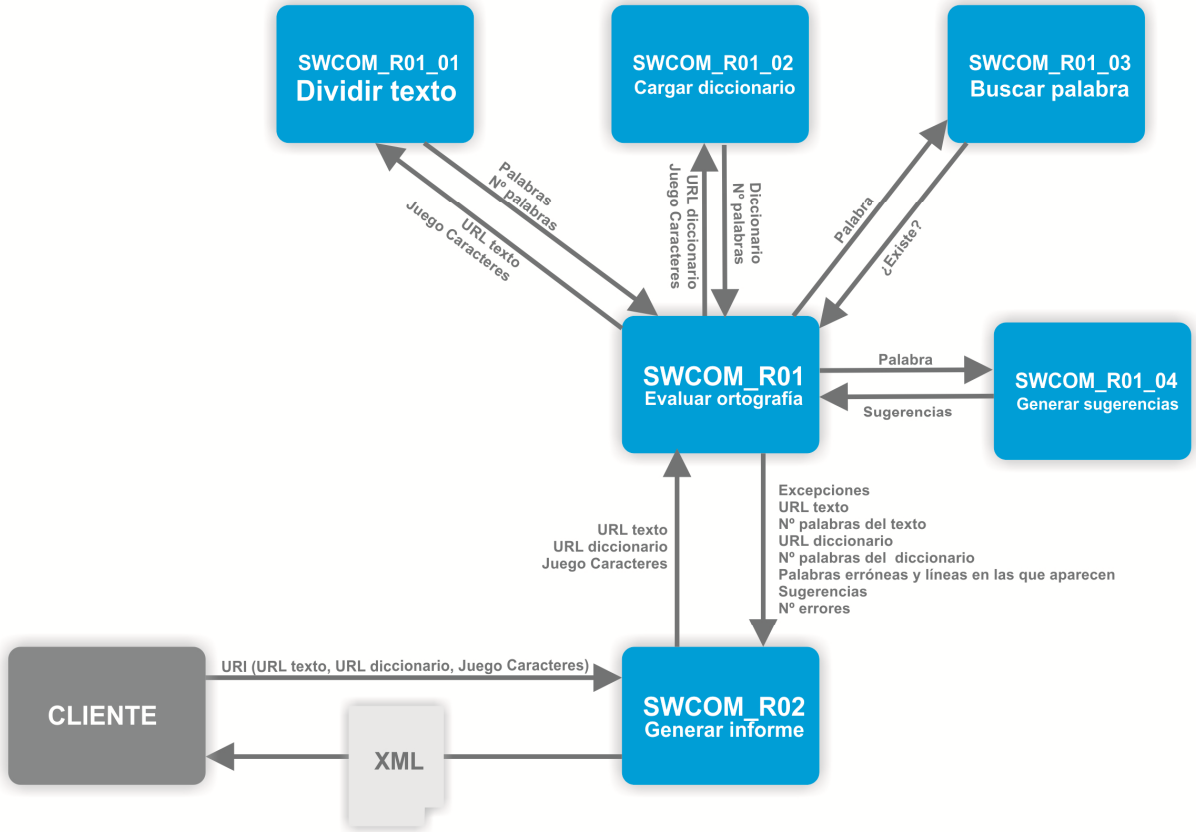
Código función	SWCOM_R_02	Nombre	Generar informe	
Descripción	Generará un informe XML con los datos de salida del proceso de evaluación.			
Entradas	URL texto URL diccionario Juego de caracteres		Origen	Cliente del servicio
	Excepciones URL texto Nº palabras texto URL diccionario Nº palabras diccionario, Palabras erróneas y línea en la que aparecen Sugerencias Nº errores Tiempo de ejecución			Evaluar ortografía (SWCOM_R_01)
Salidas	Fichero XML		Destino	Cliente del servicio
Restricciones	La codificación de caracteres del texto y del diccionario deberán ser iguales y coincidir con la especificada por la entrada juego de caracteres.			
Descripción del proceso	Iniciará el proceso de evaluación ortográfica del texto referenciado por la URL pasada como parámetro y generará un informe de salida en formato XML con los resultados de dicha evaluación. Computará el tiempo de ejecución de todo el proceso y lo incluirá en el informe de salida. En el apartado 3.4.2.7, al final de este capítulo, se especifica el formato del fichero XML de salida.			
Excepciones	No existen			

3.4.2.7. Formato del fichero XML de salida.

```
<?xml version=versión encoding=juego_de_caracteres?>
<swcom>
	<excepciones>Lista de errores resultado de la gestión de excepciones</excepciones>
	<texto>
		<url>URL del texto a analizar</url>
		<palabras>Número de palabras del texto</palabras>
	</texto>
	<diccionario>
		<url>URL del diccionario a utilizar</url>
		<palabras>Número de palabras del diccionario</palabras>
	</diccionario>
	<ortografía>
		<texto>URL del texto de entrada</texto>
```

```
                <diccionario>URL del diccionario</diccionario>
                <error>
                        <palabra>error1</palabra>
                        <línea>línea del texto en la que se encuentra el error1<línea>
                        <sugerencias>
                                <sugerencia>sugerencia1<sugerencia>
                                <sugerencia>sugerencia2 </sugerencia>
                                …
                                <sugerencia>sugerencian</sugerencia>
                        </sugerencias>
                </error>
                <error>
                        <palabra>error2</palabra>
                        <línea>línea del texto en la que se encuentra el error2<línea>
                        <sugerencias>
                                <sugerencia>sugerencia1<sugerencia>
                                <sugerencia>sugerencia2 </sugerencia>
                                …
                                <sugerencia>sugerencian</sugerencia>
                        </sugerencias>
                </error>
                …
                        <palabra>errorn</palabra>
                        <línea>línea del texto en la que se encuentra el errorn<línea>
                        <sugerencias>
                                <sugerencia>sugerencian<sugerencia>
                                <sugerencia>sugerencia2 </sugerencia>
                                …
                                <sugerencia>sugerencian</sugerencia>
                        </sugerencias>
                </error>
</ortografía>
<errores>Número de errores ortográficos encontrados</errores>
<tiempo>Tiempo de ejecución del proceso en ms</tiempo>
</swcom>
```

3.4.2.8. Representación de los flujos de información.



3.4.3. Requisitos de rendimiento.

La búsqueda en el diccionario es una operación crítica para el rendimiento. Dado el tamaño de los ficheros utilizados y el número de búsquedas realizadas (palabra propuesta y sustituciones) en cada corrección es necesario utilizar algoritmos y estructuras de datos que optimicen el rendimiento de las operaciones.

Debemos tener en cuenta la capacidad del servidor web y el número de usuarios concurrentes ya que cada sesión cargará un diccionario en la memoria.

Por otra parte la adopción de la tecnología REST puede limitar el rendimiento del sistema por la restricción stateless que nos exige pasar el diccionario en cada invocación al servicio.

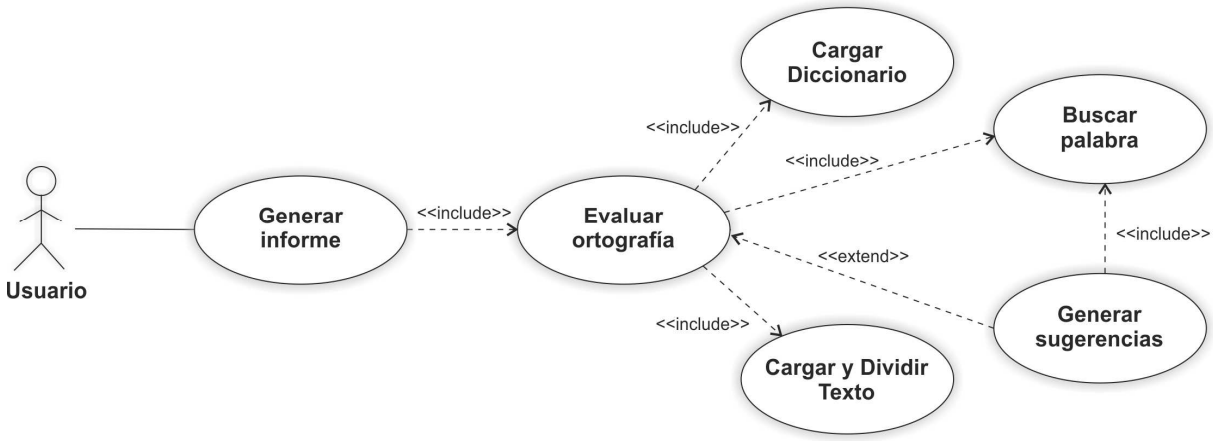
3.5. Metodología para la planificación y gestión del proyecto.

Se ha utilizado la metodología Métrica V3, del Ministerio de Hacienda y Administraciones Públicas (Ministerio de Hacienda y Administraciones Públicas, 2000), para el análisis y el diseño del sistema, utilizando un enfoque orientado a objetos. Dado el tamaño del sistema a

desarrollar muchas de las actividades se han obviado centrándonos principalmente en las actividades de establecimiento de requisitos y análisis y diseño de casos de uso y clases del sistema. Cuyo resultado se expone a continuación para una mejor comprensión del problema y soporte a los desarrollos.

3.5.1. Casos de uso.

La vista de casos de uso de un sistema comprende la descripción del comportamiento del mismo desde el punto de vista de los usuarios finales. Un caso de uso describe un conjunto de secuencias, donde cada una representa la interacción de los usuarios con el sistema (Grady Booch, James Rumbaugh e Ivar Jacobson, 2001). Estos comportamientos tienen correspondencia, normalmente, con las funciones declaradas en la especificación de requisitos. En nuestro caso los usuarios accederían al servicio a través de un cliente. Este cliente puede ser un navegador web u otra aplicación desarrollada por terceros que invocarán al servicio de corrección. El usuario pasará los parámetros correspondientes al servicio (texto, diccionario, juego de caracteres) esperando unos resultados (informe de corrección). El diagrama de casos de uso sería:

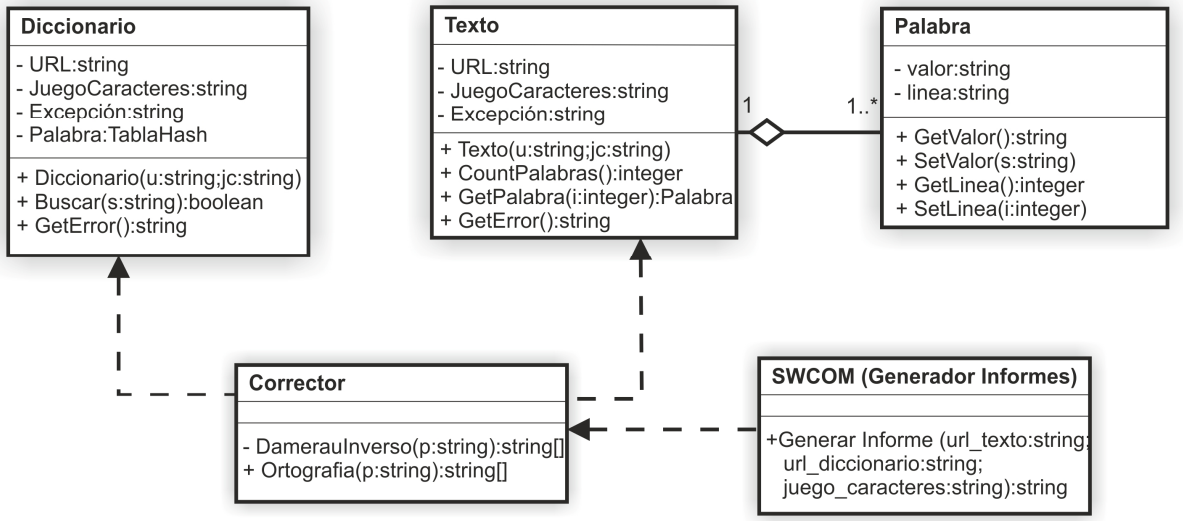


3.5.2. Clases.

El modelado de las clases puede resultar uno de los puntos más críticos en el diseño de la arquitectura ya que es posible construir diferentes modelos, todos ellos válidos, para un problema dado. Aunque la funcionalidad del sistema nunca debe verse comprometida por el conjunto de clases seleccionado, este puede dificultar la implementación del mismo.

Las clases se suelen utilizar para representar el vocabulario del sistema que se está desarrollando y se pueden utilizar para simbolizar software, hardware o entidades

puramente conceptuales (Grady Booch, James Rumbaugh e Ivar Jacobson, 2001). El diagrama de clases propuesto para el servicio web SWCOM es el siguiente:



4. Descripción del proyecto.

Siguiendo varias de las técnicas mencionadas anteriormente se ha desarrollado un servicio web con arquitectura REST orientado al análisis ortográfico de textos en cualquier idioma que utilice un alfabeto latino, aunque sería trivial modificar el código para que el servicio funcione con idiomas que utilicen otros conjuntos de caracteres. Señalaremos estos cambios en el siguiente capítulo de descripción técnica.

El servicio recibiría como parámetros el texto a corregir, un diccionario de palabras y el conjunto de caracteres utilizado para codificar el texto y el diccionario. El servicio retornaría un archivo XML con los errores encontrados, la línea del texto en la que se encuentra y una lista de sugerencias para la sustitución de cada error ortográfico.

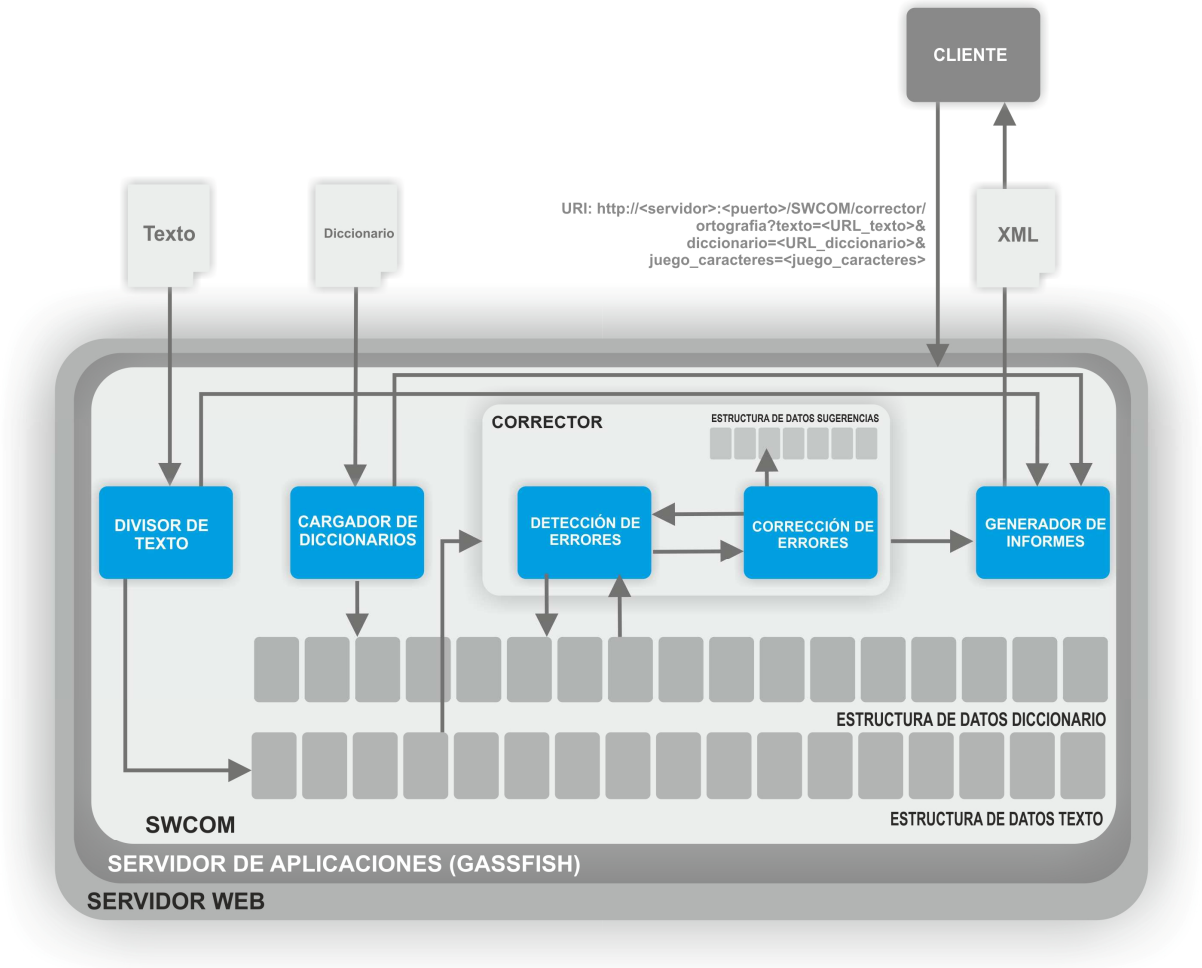
Tanto el texto como el diccionario se pasarían en formato URL, por lo que el servicio web podría realizar correcciones de cualquier texto de Internet utilizando como diccionario cualquier archivo de texto con el formato adecuado.

Podemos considerar el desarrollo como un servicio web puro donde nos existen interfaces para la carga de datos los cuales son tomados directamente desde archivos de Internet.

El problema se divide en varios subproblemas:

- **División del texto.** Es necesario dividir el texto a analizar en palabras o tokens para proceder al análisis individual de cada una de ellas. Las palabras se convertirán en minúsculas para compararlas con las almacenadas en el diccionario.
- **Cargar el diccionario.** A partir de un fichero de texto con palabras válidas de un lenguaje, es necesario generar una estructura de datos que permita realizar búsquedas optimizando el rendimiento.
- **Detección de errores.** Consistirá en realizar una búsqueda de cada palabra en el diccionario pasado como parámetro al servicio. En el caso de que la palabra no se encuentre en el diccionario se considerará un error.
- **Corrección de errores.** La corrección consistirá en generar una lista de sugerencias para las palabras erróneas.
- **Generar Informe.** Generará un informe en formato XML con los resultados del análisis.

La arquitectura del servicio, desde un punto de vista funcional, sería la siguiente:



El servicio web se consumirá desde cliente web que accederá al mismo mediante una URI con el siguiente formato:

```
http://<servidor>:<puerto>/SWCOM/corrector/ortografia?texto=<URL_texto>&diccionario=<URL_diccionario>&juego_caracteres=<juego_caracteres>
```

El archivo *Diccionario* se cargará en memoria sobre *ESTRUCTURA DE DATOS DICCIONARIO* desde la URL especificada en el URI. Dado que los accesos al diccionario son críticos para el rendimiento del sistema se considera que esta estructura debería ser una tabla hash. El módulo *DIVISOR DE TEXTO* cargará el archivo *Texto* desde la URL especificada en el URI y lo dividirá en palabras que se almacenarán en *ESTRUCTURA DE DATOS TEXTO*. Esta estructura puede ser un array convencional, dado que será necesario recorrerlo de forma secuencial pasando cada una de las palabras al módulo *CORRECTOR*. Este módulo estará compuesto por dos submódulos. El submódulo *DETECCIÓN DE ERRORES* comprobará si la palabra existe en la *ESTRUCTURA DE DATOS DICCIONARIO*. En caso negativo pasa la palabra actual al *SUBMÓDULO CORRECCIÓN*

DE ERRORES, que utilizará el algoritmo *Damerau Inverso* para generar variaciones de la palabra errónea con distancias de edición 1 y 2. Cada variación se buscará en *ESTRUCTURA DE DATOS DICCIONARIO*. Si se encuentra se considera una sugerencia y se almacena en la *ESTRUCTURA DE DATOS SUGERENCIAS*. Al finalizar cada corrección se pasan los datos correspondientes al módulo *GENERADOR DE INFORMES*, el cual se encargará de construir el archivo XML de salida. Este módulo recibirá datos de otros componentes. *DIVISOR DE TEXTO* le pasará el número de palabras leídas en *Texto* y *CARGADOR DE DICCIONARIO* le pasará el número de palabras contenidas en el *Diccionario*. El módulo CORRECTOR contará los errores encontrados y se los pasará a GENERADOR DE INFORMES después de procesar todas las palabras de ESTRUCTURA DE DATOS PALABRAS.

El servicio *SWCOM* correrá sobre un SERVIDOR *DE APLICACIONES (GLASSFISH)* instalado en un *SERVIDOR WEB*. Este último se encargará de medir el tiempo de ejecución y pasárselo al *GENERADOR DE INFORMES*. El tiempo de ejecución será la diferencia entre la hora del sistema al iniciar el proceso de corrección y la hora del sistema al finalizar dicho proceso. Todos los módulos incorporan rutinas para la gestión de excepciones. Si se produce un error en tiempo de ejecución su descripción se pasará al *GENERADOR DE INFORMES*.

Para una corrección precisa en se requiere que Diccionario y Texto utilicen el mismo enconding de texto que deberá coincidir con el parámetro `<juego_caracteres>` del URI de la llamada. No hay posibilidad de determinar de forma precisa en encoding de un archivo de texto, por lo que en caso de errores lo mejor es guardar los archivos con la misma codificación. La mayoría de editores de texto incorporan esta funcionalidad.

A continuación se detallará como se han abordado cada uno de estos problemas desde un punto de vista funcional profundizando, si fuera preciso, en el siguiente apartado, en algunas de las cuestiones planteadas.

4.1. División del texto.

La división de un texto en palabras es esencial en el tratamiento de cadenas, es por ello que muchos lenguajes de programación y herramientas incorporan rutinas que implementan esta funcionalidad. Los elementos obtenidos a partir de este proceso de segmentación suelen denominarse tokens. La obtención de los tokens, o palabras, se realiza a partir de un conjunto de delimitadores que indicarán donde comienza y termina cada palabra. Por

ejemplo, si definimos como delimitador el carácter espacio podríamos considerar tokens todos los conjuntos de caracteres que se encuentren entre dos espacios.

Se implementa una clase palabra en la cual se almacena el valor de cada token junto con la línea del texto en el que aparece. Todas las palabras se almacenan en una estructura de datos que se pasará como entrada al proceso de detección.

Para evitar conflictos se convierten todas las palabras a minúsculas, ya que de esta forma se almacenan en el diccionario.

4.2. Cargar el diccionario.

Los diccionarios que acepta el servicio son ficheros de texto en los que cada línea contiene una palabra. No se utilizan ficheros de afijos, por consiguiente el diccionario no contiene reglas de derivación sino una lista de palabras expandida. Existen multitud de listas de palabras en Internet que pueden utilizarse como vocabulario. También podríamos compilar la nuestra a partir de diferentes archivos.

Nuestro diccionario se implementa sobre una tabla hash lo que nos permitirá realizar búsquedas con una complejidad de O(1).

4.3. Detección de errores.

La detección de errores se reduce a buscar cada palabra almacenada en la estructura de datos en la que se almacena el diccionario. Habiendo seleccionado una tabla hash para implementar esta estructura la función es tan simple como determinar si existe alguna clave que coincida con la palabra buscada. En caso de no existir esta clave podemos considerar que la palabra es un error potencial con lo que tendremos que lanzar el proceso de corrección.

4.4. Corrección de errores.

Cuando una palabra no se encuentra en el diccionario se lanzará un proceso de corrección, que consistirá en generar una lista de palabras candidatas cuya distancia de edición a la palabra errónea sea 1 o 2. Como hemos visto en el capítulo 2, y según diversos estudios sobre la corrección ortográfica, entre un 80% y un 95% de los errores non-word estarían a una distancia de edición 1 por lo que la efectividad del servicio puede considerase alta. Para obtener la lista de sugerencias se parte de la palabra errónea y se generan todas las ediciones con distancia 1 de la misma. Cada edición generada se busca en el diccionario, si

se encuentra, se añade a una estructura de datos. En el caso de no hallar en el diccionario ninguna edición con distancia 1 se generarían ediciones con distancia 2.

4.5. Generar informe.

Es necesario construir un archivo XML, que será la salida del servicio que entregaremos al cliente. Cada uno de los módulos pasará la información correspondiente a este proceso que se encargará de construir cada una de las etiquetas XML, siguiendo el formato especificado en los requerimientos.

4.6. Invocando a SWCOM

Como se ha comentado anteriormente no se ha desarrollado ningún cliente para consumir el servicio, sin embargo al ser un servicio RESTful podemos probar su funcionalidad utilizando cualquier navegador para invocar la URI de los recursos correspondientes. En este caso los recursos son las correcciones de los textos de entrada, que se representan utilizando el formato XML. La representación de un recurso podría hacerse en otros formatos (JSON, TXT, etc.) con pequeñas modificaciones en el código.
La URL para la invocación del servicio tendrá la siguiente sintaxis:

```
http://<servidor>:<puerto>/SWCOM/corrector/ortografia?texto=<URL_texto>&diccionario=<URL_diccionario>&juego_caracteres=<juego_caracteres>
```

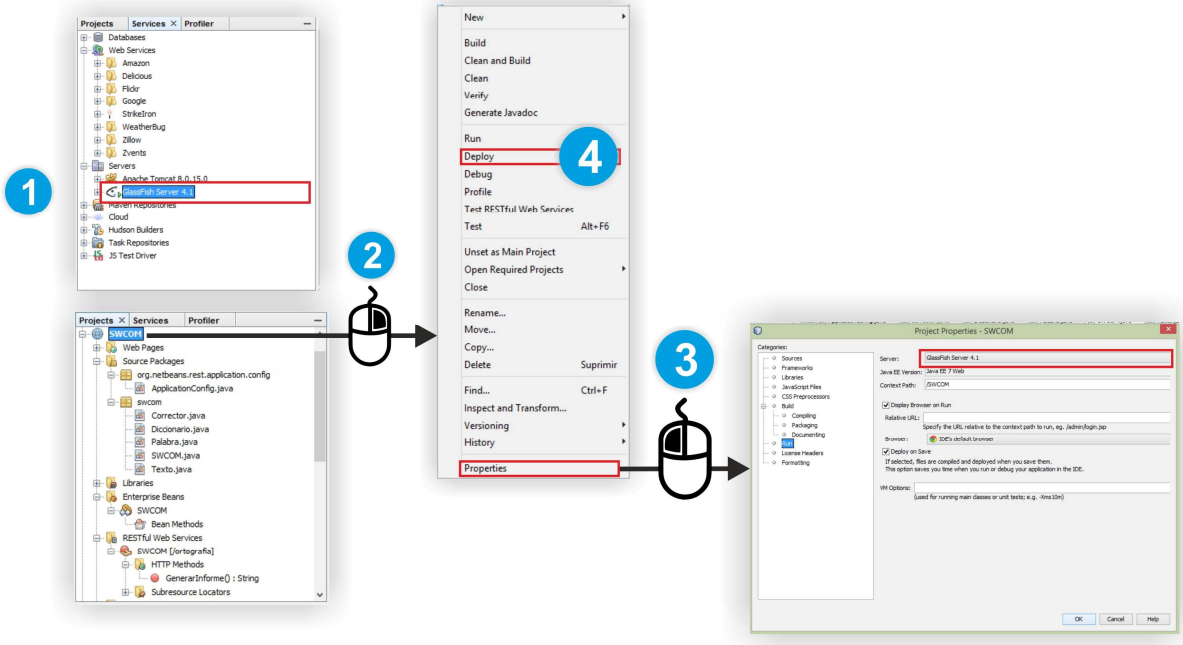
Como se ha introducido en capítulos anteriores la calidad del diccionario es determinante para obtener un buen resultado en la corrección.

4.7. Despliegue de SWCOM.

La versión actual de SWCOM se ha probado sobre un servidor Glassfish 4.1., pero se podría desplegar sobre cualquier otro servidor de aplicaciones compatible con Java EE y JAX-RS. Es posible desplegar el proyecto directamente desde el entorno de desarrollo Netbeans o iniciar la sesión en Glassfish y subir el archivo SWCOM.war correspondiente, que encontraremos en la carpeta /dist del proyecto Netbeans. Este archivo es el resultado de la compilación (Build) del proyecto.

4.7.1. Despliegue desde Netbeans.

Para desplegar el servicio desde Netbeans seguimos los siguientes pasos:



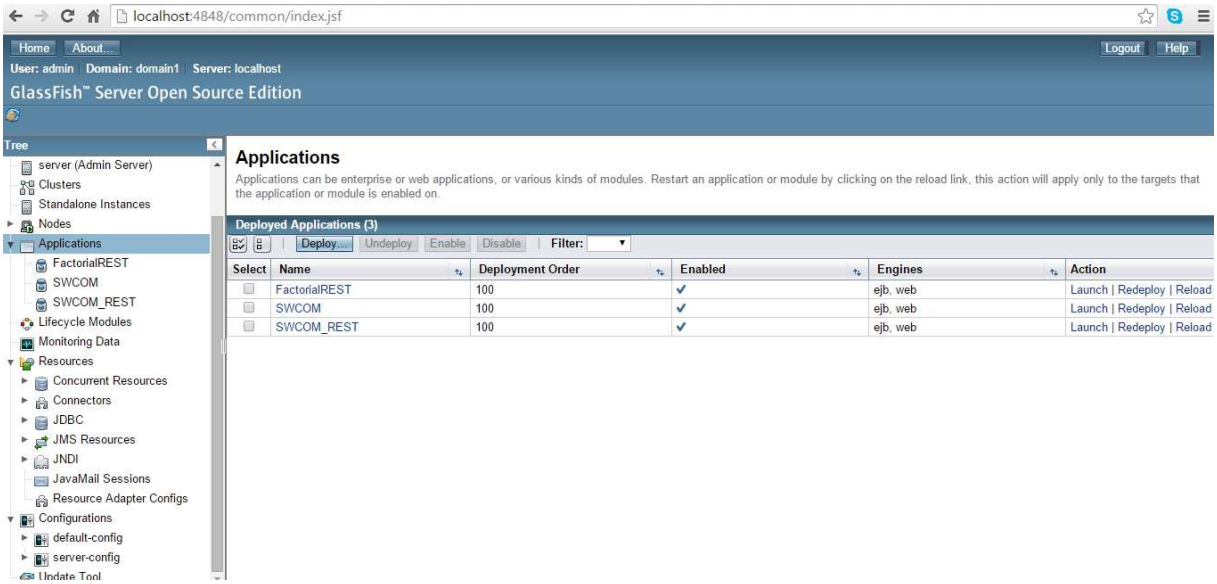
En primer lugar comprobaremos que Glassfish está instalado en Netbeans. Para ello seleccionamos la ficha *Services* y comprobamos que Glassfish esté dentro de la rama Servers[10] (1). Seleccionamos la ficha *Projects*, nos situamos sobre el proyecto SWCOM y hacemos clic sobre el botón derecho (2). Seleccionamos la opción *Properties* (3) para ver las propiedades del proyecto. Nos situamos en la categoría *Run* y comprobamos que en el campo Server figure el servidor Glassfish correspondiente[11]. Por último volvemos a mostrar el menú contextual del proyecto SWCOM (2) y hacemos clic en la opción Deploy (4). Si no se produce ningún error se instalará la aplicación en Glassfish.

4.7.2. Despliegue desde Glassfish.

Iniciamos la consola de administración de Glassfish (por defecto http://<dominio>:4848. Si tenemos un servidor local será http://localhost:4848). Iniciamos la sesión con el usuario admin y seleccionamos la rama Applications del árbol de opciones.

[10] Si tenemos ningún servidor Netbeans es necesario instalar uno.

[11] Podemos tener varios servidores Glassfish instalados en Netbeans.



Veremos todas las aplicaciones instaladas en el servidor. Hacemos clic en *Deploy*. Seleccionamos la opción *Packaged File to Be Uploaded to the Server* y hacemos clic en el botón *Seleccionar archivo*. Buscamos el archivo .war mencionado anteriormente en este apartado. Si no existe es necesario compilar (Build) el proyecto. Dejamos las opciones por defecto y hacemos clic en Ok para finalizar.

5. Descripción técnica.

5.1. Tecnologías utilizadas.

El proyecto se ha desarrollado íntegramente con tecnología Java. Para construir el servicio se ha utilizado el entorno de desarrollo Netbeans 8.0.2. y los siguientes componentes:

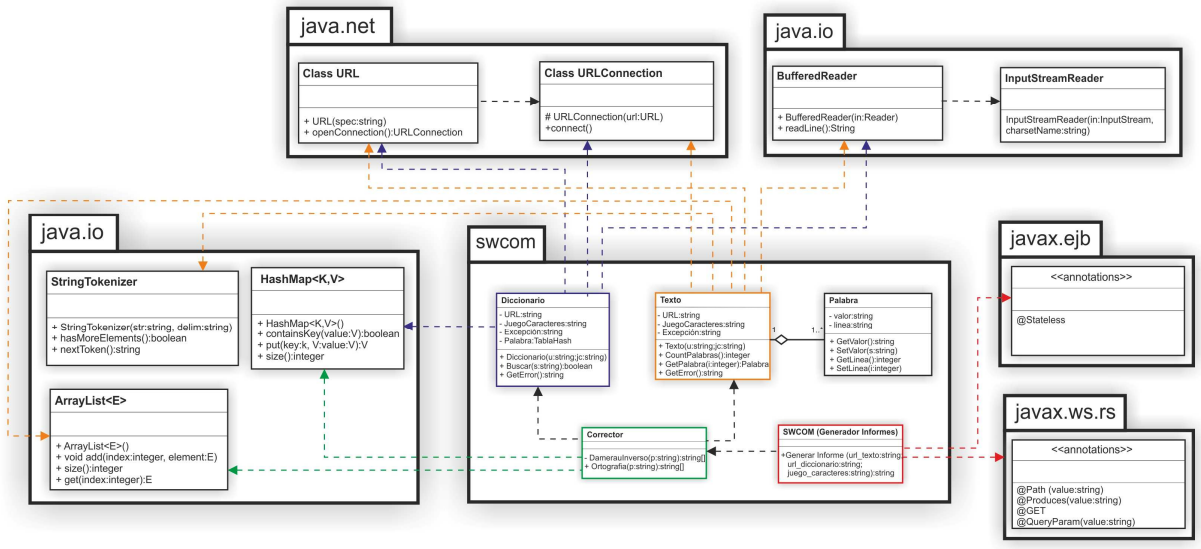
- JDK 1.8,
- Java EE 7
- Jersey 2.5.1. (incluido en Java EE 7)

Jersey es la implementación de referencia de Sun y Oracle para JAX-RS (Java API for RESTful Web Services). JAX-RS es un API orientado al desarrollo de servicios web siguiendo las especificaciones de la arquitectura REST, descrita en los documentos JSR-311 y JSR-339.

Las pruebas se han realizado sobre un servidor de aplicaciones Glassfish 4.1., de Sun y Oracle, corriendo sobre un sistema operativo Windows 8.1.

5.2. Arquitectura.

En el capitulo anterior presentamos la arquitectura funcional del servicio web que, dado el tamaño de este, podría considerarse suficientemente descriptiva para documentar la arquitectura del sistema. No obstante, y como guía para la implementación de la aplicación, se ha realizado un diagrama de paquetes en el que podemos ver todos los elementos que conforman el sistema y sus relaciones.



5.3. Análisis del código.

En este capítulo se realizará una revisión del código, destacando determinados aspectos que pueden resultar de interés para entender la complejidad del problema, al mismo tiempo que se demuestra cómo se aplican las técnicas descritas en capítulos anteriores.

5.3.1. Clase Palabra.

Representa una palabra del texto a corregir la cual se caracteriza por su valor, tipo String, y la línea que ocupa en el texto, tipo Int. Un objeto de la clase Texto, que veremos más adelante, estará compuesto por objetos de la clase Palabra. El código es simple e implementa métodos para establecer y recuperar (set y get) los valores de cada palabra:

Código de la clase Palabra

```
public class Palabra {
	private String valor;
	private int linea;
	public Palabra(){
	  //Contructor
	}
	public void SetValor(String v){
		this.valor=v;
	}
	public String getValor(){
		return this.valor;
	}
	public void SetLinea(int l){
		this.linea=l;
	}
	public int getLinea(){
		return this.linea;
	}
}
```

5.3.2. Clase Texto.

Esta clase lee un texto desde la URL de entrada y lo divide en palabras. Cada palabra se almacena en una estructura ArrayList de objetos de tipo palabra. Las propiedades de esta clase son:

- `String URL`. Representa la URL del texto a analizar.
- `String juego_caracteres`. Representa el juego de caracteres utilizado para procesar el texto.
- `private ArrayList<Palabra> Palabras`. ArrayList de tipo palabra que contiene cada palabra del texto y la línea en la que aparece.
- `String error`. Variable utilizada para capturar la descripción de las excepciones con objeto de mostrarlas en el recurso de salida.

Los métodos de la clase son:

- `Public Texto(string u,String jc).` Es el constructor de la clase. Se encarga de leer el fichero de texto desde la URL u, usando el juego de caracteres jc. Divide el texto en palabras y carga cada una en la estructura ArrayList Palabras. Cada línea leída del fichero es procesada por la clase StringTokenizer, encargada de dividir el texto en palabras según los delimitadores definidos. El corrector es muy sensible a la definición de estos limitadores ya que de ellos depende la selección de las palabras a analizar. Todas las palabras se convierten a minúsculas para facilitar la comparación con las entradas de diccionario, que también deberán estar en minúsculas.
- `public Palabra GetPalabra (int i).` Devuelve el objeto palabra del ArrayList Palabras que ocupa la posición i.
- `public int CountPalabras()`. Devuelve el número de palabras del texto (número de elementos del ArrayList Palabras)
- `public String GetError()`. Devuelve el valor de la propiedad error.

Código de la clase Texto

```
import java.io.*;
import java.util.ArrayList;
import java.util.StringTokenizer;
import java.net.URLConnection;
import java.net.URL;

public class Texto {
	private String URL;
	private String juego_caracteres;
	private ArrayList<Palabra> Palabras = new ArrayList<Palabra>();
	private String error;
	public Texto(String u, String jc) { //Constructor
	this.URL=u;
	this.juego_caracteres=jc;
	this.error="";
	try{
		URL url = new URL(this.URL);
		URLConnection uc = url.openConnection();
		uc.connect();
		BufferedReader bf = new BufferedReader(new
		InputStreamReader(uc.getInputStream(),this.juego_caracteres));
		String linea;
		int nlinea=1;
		String delimitadores= "0123456789!¡¿?-+*,;. \\\"\'[]{}()<>@#$%&=";
		while ((linea = bf.readLine()) != null){
			StringTokenizer palabrasSeparadas = new
			StringTokenizer(linea,delimitadores);
			while (palabrasSeparadas.hasMoreElements()){
				Palabra p = new Palabra();
				p.SetValor(palabrasSeparadas.nextToken().toLowerCase());
				p.SetLinea(nlinea);
				this.Palabras.add(p);
			}
			nlinea++;
		}
		bf.close();
	}catch(Exception e){
		this.error=e.getMessage();
	}
	}
```

```
        public int CountPalabras(){
                return this.Palabras.size();
        }
        public Palabra GetPalabra(int i){
                return this.Palabras.get(i);
        }
        public String GetError(){
                return this.error;
        }
}
```

5.3.3. Clase Diccionario

Representa el diccionario utilizado para la corrección. La clase lee un archivo de texto desde la URL indicada y la carga es una estructura HashMap. Las propiedades de la clase son:

- `String URL`. Representa la URL del texto a analizar.
- `String juego_caracteres`. Representa el juego de caracteres utilizado para procesar el diccionario.
- `private HashMap<String, Integer> Palabras`. Es un HashMap (tabla hash) que almacena cada palabra del diccionario.
- `String error`. Variable utilizada para capturar la descripción excepciones con objeto de mostrarlas en el recurso de salida.

Los métodos de la clase son:

- `public Diccionario(string u, string jc)`. Es el constructor de la clase. Se encarga de leer el fichero que contiene el diccionario desde la URL u, utilizando el juego de caracteres jc. Carga las palabras en la estructura HashMap Palabras.
- `public Boolean Buscar(String s)`. Esta función implementa la operación de búsqueda en el diccionario. Determina si el string s aparece en el HashMap Palabras.
- `public int CountPalabras()`. Devuelve el número de palabras del diccionario (número de elementos del HashMap Palabras)
- `public String GetError()`. Devuelve el valor de la propiedad error.

Código de la clase Diccionario

```
import java.io.*;
import java.net.URL;
import java.net.URLConnection;
import java.util.HashMap;

public class Diccionario {
```

```
        private String URL;
        private String juego_caracteres;
        private String error;
        private HashMap<String, Integer> Palabras = new HashMap<String, Integer>();
        public Diccionario(String u, String jc) {
        this.URL=u;
        this.juego_caracteres=jc;
        this.error="";
        try{
                URL url = new URL(this.URL);
                URLConnection uc = url.openConnection();
                uc.connect();
                BufferedReader bf = new BufferedReader(new
                InputStreamReader(uc.getInputStream(),this.juego_caracteres));
                String linea;
                int i=1;
                while ((linea = bf.readLine()) != null) {
                        this.Palabras.put(linea.toLowerCase(),i++);
                }
                bf.close();
        }catch(Exception e){
                        this.error=e.getMessage();
        }
        }
        public Boolean Buscar(String s) {
                if(this.Palabras.containsKey(s)) return true;
                return false;
        }
        public int CountPalabras(){
                return this.Palabras.size();
        }

        public String getError() {
                return this.error;
        }
}
```

5.3.4. Clase Corrector.

Genera una lista de sugerencias para una palabra. Para ello utiliza el algoritmo de cálculo de la distancia mínima de edición inverso, que en la implementación hemos denominado DamerauInverso. Las propiedades de la clase son:

- `Diccionario Dic`. Representa el diccionario utilizado para la corrección de la palabra.

. Los métodos son:

- `public Corrector(Diccionario d)`. Es el constructor de la clase. Recibe como entrada un objeto diccionario que asigna a la propiedad Dic.
- `Private ArrayList<string> DamerauInverso (String palabra)`. Genera todas las ediciones con distancia 1 del String palabra utilizando las operaciones de edición propuestas por Damerau.

Se establecen los caracteres que se utilizarán para las operaciones de sustitución e inserción.

```
char caracteres_validos[]= {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
'ñ', 'á', 'é', 'í', 'ó', 'ú', 'ü'};
```

El corrector se podría utilizar para generar ediciones en otro idioma modificando la lista de caracteres válidos.

El algoritmo recorre la palabra para cada operación almacenando cada nueva edición en la estructura ArrayList ediciones.

```
ArrayList<String> ediciones = new ArrayList<String>();
//Eliminaciones
for(int i=0; i < palabra.length(); ++i) ediciones.add(palabra.substring(0, i) +
palabra.substring(i+1));
//Trasposiciones
for(int i=0; i < palabra.length()-1; ++i) ediciones.add(palabra.substring(0, i)
+ palabra.substring(i+1, i+2) + palabra.substring(i, i+1) +
palabra.substring(i+2));
//Sustituciones
for(int i=0; i < palabra.length(); ++i) for(char c : caracteres_validos)
ediciones.add(palabra.substring(0, i) + String.valueOf(c) +
palabra.substring(i+1));
//Inserciones
for(int i=0; i <= palabra.length(); ++i) for(char c : caracteres_validos)
ediciones.add(palabra.substring(0, i) + String.valueOf(c) +
palabra.substring(i));
```

La función `palabra.substring (i,j)` devuelve la parte de la cadena `palabra` comprendido entre las posiciones i y j. En caso de utilizar un solo argumento, `palabra.substring(i),` se devuelve la cadena entre las posiciones i y el final de palabra.

No resultará difícil de entender con un ejemplo sencillo:

i	0	1	2	3
palabra	*m*	*e*	*s*	*a*

Eliminaciones. Si emulamos la iteración tendríamos:

A = `palabra.substring(0, i)`

B = `palabra.substring(i+1)`

i	A	B	A+B
0		*esa*	*esa*
1	*m*	*sa*	*msa*
2	*me*	*a*	*maa*
3	*mes*		*mes*

Trasposiciones.

```
A = palabra.substring(0, i)
B = palabra.substring(i+1, i+2)
C = palabra.substring(i, i+1)
D = palabra.substring(i+2)
```

i	A	B	C	D	A+B+C+D
0		*e*	*m*	*sa*	*emsa*
1	*m*	*s*	*e*	*a*	*msea*
2	*me*	*a*	*s*		*meas*

Sustituciones. En este caso tenemos dos bucles anidados. El primero recorre la palabra, for(int i=0; i < palabra.length(); ++i), y el segundo recorre el vector de caracteres validos, for(char c : caracteres_validos).

```
A = palabra.substring(0, i)
B = String.valueOf(c)
C = palabra.substring(i+1)
```

i	c	A	B	C	A+B+C
0	a		*a*	*esa*	*aesa*
	b		*b*	*esa*	*besa*
	c		*c*	*esa*	*cesa*
	…	…	…	…	…
1	a	*m*	*a*	*sa*	*masa*
	b	*m*	*b*	*sa*	*mbsa*
	c	*m*	*c*	*sa*	*mcsa*

	…	…	….	…	…
2	a	*me*	*a*	*a*	*meaa*
	b	*me*	*b*	*a*	*meba*
	c	*me*	*c*	*a*	*meca*
	…	…	…	…	…
3	a	*mes*	*a*		*mesa*
	b	*mes*	*b*		*mesb*
	c	*mes*	*c*		*mewc*
	…	…	…	…	…

Inserciones. Al igual que en caso anterior tenemos dos bucles anidados.

A = `palabra.substring(0, i)`

B = `String.valueOf(c)`

C = `palabra.substring(i)`

i	c	A	B	C	A+B+C
0	a		*a*	*mesa*	*amesa*
	b		*b*	*mesa*	*bmesa*
	c		*c*	*mesa*	*cmesa*
	…	…	…	…	…
1	a	*m*	*a*	*esa*	*maesa*
	b	*m*	*b*	*esa*	*mbesa*
	c	*m*	*c*	*esa*	*mcesa*
	…	…	…	…	…
2	a	*me*	*a*	*sa*	*measa*
	b	*me*	*b*	*sa*	*mebsa*
	c	*me*	*c*	*sa*	*mecsa*
	…	…	…	…	…
3	a	*mes*	*a*	*a*	*mesaa*
	b	*mes*	*b*	*a*	*mesba*
	c	*mes*	*c*	*a*	*mesca*
	…	…	…	…	…
4	a	*mesa*	*a*		*mesaa*

i	c	A	B	C	A+B+C
	b	*mesa*	*b*		*mesab*
	c	*mesa*	*c*		*mesac*
	…	…	…	…	…

- `public ArrayList<String> Ortografia(String palabra)`. Es el generador de la lista de sugerencias para una determinada palabra. La lista de sugerencias se almacena en una estructura ArrayList de String denominada candidatas.

```
ArrayList<String> candidatas = new ArrayList<String>();
```

Llama a la función *DamerauInverso* para generar todas las ediciones con distancia 1 de la palabra.

```
ArrayList<String> ediciones = DamerauInverso(palabra);
```

Busca cada elemento del ArrayList ediciones en el diccionario añadiendo a candidatas aquellas que se encuentre en el diccionario. Cuando una palabra se encuentra en el diccionario se comprueba que esta se haya incluido ya en el ArrayList candidatas como consecuencia de una edición anterior que produjese la misma palabra.

```
for(String s : ediciones) if(this.Dic.Buscar(s)) if (!candidatas.contains(s))
candidatas.add(s);
```

Si encontramos alguna palabra candidata la función devuelve el ArrayList candidatas.

```
if(candidatas.size() > 0) return candidatas;
```

En caso contrario generamos palabras con distancia de edición 2. Para ello tomamos cada una de las palabras de la lista de ediciones y llamamos a la función *DamarauInverso*. Esto genera ediciones con distancia 1 de cada edición con distancia 1 de la palabra original, lo que permite obtener las palabras con distancia 2 de la original. Al igual que en bucle anterior comprobamos que las nuevas ediciones no estén ya en la lista de candidatas.

```
for(String s : ediciones) for(String w : DamerauInverso(s))
if(this.Dic.Buscar(w)) if (!candidatas.contains(w))candidatas.add(w);
```

Código de la clase Corrector

```
import java.util.ArrayList;
import java.util.HashMap;
public class Corrector {
	private Diccionario Dic;
	public Corrector(Diccionario d){
		this.Dic=d;
	}
	private ArrayList<String> DamerauInverso(String palabra){
		char caracteres_validos[]= {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
		'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y',
		'z', 'ñ', 'á', 'é', 'í', 'ó', 'ú', 'ü'};
		ArrayList<String> ediciones = new ArrayList<String>();
		//Eliminaciones
		for(int i=0; i < palabra.length(); ++i) ediciones.add(palabra.substring(0, i)
		+ palabra.substring(i+1));
		//Trasposiciones
		for(int i=0; i < palabra.length()-1; ++i) ediciones.add(palabra.substring(0,
		i) + palabra.substring(i+1, i+2) + palabra.substring(i, i+1) +
		palabra.substring(i+2));
		//Sustituciones
		for(int i=0; i < palabra.length(); ++i) for(char c : caracteres_validos)
		ediciones.add(palabra.substring(0, i) + String.valueOf(c) +
		palabra.substring(i+1));
		//Inserciones
		for(int i=0; i <= palabra.length(); ++i) for(char c : caracteres_validos)
		ediciones.add(palabra.substring(0, i) + String.valueOf(c) +
		palabra.substring(i));
		return ediciones;
	}

	public ArrayList<String> Ortografia(String palabra) {
		ArrayList<String> candidatas = new ArrayList<String>();
		ArrayList<String> ediciones = DamerauInverso(palabra);
		for(String s : ediciones) if(this.Dic.Buscar(s)) if (!candidatas.contains(s))
		candidatas.add(s);
		if(candidatas.size() > 0) return candidatas;
		for(String s : ediciones) for(String w : DamerauInverso(s))
		if(this.Dic.Buscar(w)) if (!candidatas.contains(w))candidatas.add(w);
		return candidatas.size() > 0 ? candidatas : candidatas;
	}
}
```

5.3.5. Clase SWCOM.

Implementa el método HTTP del servicio RESful. Recibe los parámetros y lanza el proceso de corrección generando el archivo XML descrito en los requerimientos.

La anotación @Stateless convierte la clase corregir en un EJB de sesión sin estado, cumpliendo uno de los principios básicos de la arquitectura REST. La anotación @Path indica el path del URI del recurso dentro del servidor. En nuestro caso el recurso se accederá en el path /ortografía. Mediante la anotación @Produces indicamos el formato de salida de los recursos, en este caso text/xml.

La clase sólo tiene un método:

- `public String GenerarInforme(@QueryParam("texto") String tp,@QueryParam("diccionario") String dp, @QueryParam("juego_caracteres") String jcp)`. La función GenerarInforme está precedida por la anotación *@GET* lo cual indica que implementa un método GET del protocolo HTTP. La anotación *@QueryParam* establece una relación entre los argumentos de la función y los parámetros del método HTTP. Estos parámetros se inyectan en el URI a partir del path establecido en la anotación *@Path*. Devuelve un String que será un XML con el formato especificado en los requerimientos.
  Inicializamos las variables:

```
long ti= System.currentTimeMillis();
ArrayList<String> sugerencias = new ArrayList<String> ();
int errores=0;
StringBuilder br = new StringBuilder();
Texto t = new Texto(tp,jcp);
Diccionario d = new Diccionario(dp,jcp);
```

  `ti` recoge el tiempo del reloj del sistema con objeto de calcular el tiempo de ejecución del proceso. Al final de la función se toma de nuevo el tiempo. El tiempo de ejecución será la diferencia entre ambos tiempos. El ArrayList `sugerencias` almacenará las sugerencias para una palabra errónea. La variable `errores` almacena el número de errores del texto. `br` es una variable StringBuilder y almacenará la cadena de texto en formato XML de salida. `t` representa el texto de entrada y `d` el diccionario.
  Se recorre el texto buscando cada palabra en el diccionario. Si no se encuentra llama al método `Ortografia` de la clase corrector para producir una lista de sugerencias. La construcción del XML es trivial añadiendo a `br` las líneas correspondientes para ir construyendo en XML.

Código de la clase SWCOM

```
import java.util.ArrayList;
import javax.ejb.Stateless;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;

@Stateless
@Path ("/ortografia")
@Produces("text/xml")
public class SWCOM {
	@GET
		public String GenerarInforme(@QueryParam("texto") String
		tp,@QueryParam("diccionario") String dp, @QueryParam("juego_caracteres") String
		jcp){
		long ti= System.currentTimeMillis();
```

Código de la clase SWCOM

```
                ArrayList<String> sugerencias = new ArrayList<String> ();
                int errores=0;
                StringBuilder br = new StringBuilder();
                Texto t = new Texto(tp,jcp);
                Diccionario d = new Diccionario(dp,jcp);
                br.append("<?xml version='1.0' encoding='UTF-8'?>").append("\n");
                br.append("<swcom>").append("\n");
                if (t.GetError()!="" || d.getError()!="")
                {
                        br.append("<excepciones>");
                        br.append(t.GetError()).append("\n");
                        br.append(d.getError()).append("\n");
                        br.append("</excepciones>").append("\n");
                }else {
                        br.append("<texto>").append("\n");
                        br.append("<url>").append(tp).append("</url>").append("\n");
                        br.append("<palabras>").append(t.CountPalabras()).append("</palabras>").
                        append("\n");
                        br.append("</texto>").append("\n");
                        br.append("<diccionario>").append("\n");
                        br.append("<url>").append(dp).append("</url>").append("\n");
                        br.append("<palabras>").append(d.CountPalabras()).append("</palabras>").
                        append("\n");
                        br.append("</diccionario>").append("\n");
                        br.append("<ortografia>").append("\n");
                        Corrector c = new Corrector(d);
                        for (int i=0;i<t.CountPalabras();++i){
                                if (!d.Buscar(t.GetPalabra(i).getValor())){
                                        errores++;
                                        br.append("<error>").append("\n");
                                        br.append("<palabra>").append(t.GetPalabra(i).getValor())
                                        .append("</palabra>").append("\n");
                                        br.append("<linea>").append(t.GetPalabra(i).getLinea()).a
                                        ppend("</linea>").append("\n");
                                        br.append("<sugerencias>").append("\n");
                                        sugerencias=c.Ortografia(t.GetPalabra(i).getValor());
                                        for (int j=0;j<sugerencias.size();++j)
                                        br.append("<sugerencia>").append(sugerencias.get(j)).appe
                                        nd("</sugerencia>").append("\n");
                                        br.append("</sugerencias>").append("\n");
                                        br.append("</error>").append("\n");
                                }
                        }
                        br.append("<errores>").append(Integer.toString(errores)).append("</error
                        es>").append("\n");
                        br.append("</ortografia>").append("\n");
                }
                long tf= System.currentTimeMillis();
                br.append("<tiempo>").append(Long.toString(tf-
                ti)).append("</tiempo>").append("\n");
                br.append("</swcom>").append("\n");
                return br.toString();
        }
}
```

6. Evaluación.

Se han realizado múltiples pruebas funcionales verificando el compleobteniendo resultados satisfactorios, en comparación con otras herramientas de corrección. Las pruebas se han realizado sobre un ordenador personal con procesador Intel(R) Core(TM) i5 2520M 2.50GHz, 4 GB de memoria RAM y sistema operativo Windows 8.1. Se han empleado varios textos con errores y diccionarios de palabras de uso libre. La mayoría de las pruebas se han realizado con el diccionario de español para GNU Aspell, después de obtener una lista completa de palabras, a partir del diccionario de español es_ES.cwl, mediante la herramienta word-list-compress. Este diccionario es mantenido por Agustín Martin, Santiago Rodríguez y Jesús Carretero, y está disponible para descarga en la página http://aspell.net/win32/ bajo licencia GPLv2. La versión usada tiene un total de 595.946 palabras.

Después de probar el corrector de forma independiente y con objeto de verificar su adaptación a los requisitos establecidos se realizó un análisis comparativo con GNU Aspell en su versión para Windows, obteniendo para diferentes archivos tasas de detección y tiempos de ejecución similares.

A modo de ejemplo, empleando un diccionario con 247.000 terminos para corregir un texto con 439 palabras, de las cuales 108 son errores non word, SWCOM emplea menos de un segundo (750 ms) detectando 99 errores y realizando sugerencias correctas en todos los casos. Si utilizamos el diccionario de Aspell, mencionado arriba, se detectan la totalidad de los errores empleando un tiempo de ejecución de 1650 ms.

Para corregir un archivo con 17.000 palabras y 4212 errores emplea un tiempo de 22 segundos.

7. Conclusiones y trabajo futuro.

Ha quedado patente la importancia de la corrección ortográfica y las arquitecturas orientadas a servicios. La combinación de ambas aporta la posibilidad de integrar la corrección ortográfica en diferentes aplicaciones mediante pequeñas adaptaciones. Si bien es cierto que existen otros correctores más maduros y fiables, este trabajo abre las puertas a la investigación en este campo con la finalidad de fomentar el desarrollo de nuevas soluciones, en un escenario donde, como hemos podido constatar, la oferta de servicios web orientados a la corrección ortográfica es escasa. También hemos puesto en relieve el enorme interés que tienen en la actualidad este tipo de soluciones, cuyo ámbito no se restringe a las aplicaciones ofimáticas, siendo imprescindibles como soporte a procesos de reconocimiento óptico de caracteres, reconocimiento del habla, conversión de voz a texto, etc. Aplicaciones tan populares como Siri de Apple, deben parte de su fiabilidad a la integración de técnicas de corrección ortográfica.

Con SWCOM se ha iniciado un trabajo que tiene una clara continuidad. En primer lugar se podría ampliar el alcance desarrollando un módulo para la selección de la mejor sugerencia, utilizando las técnicas expuestas en este estudio, con el objetivo se integrarse ad hoc con sistemas que precisen una corrección desatendida. También se podrían implementar algoritmos para el análisis gramatical o aplicar métodos de stemming, para usar los diccionarios y ficheros de afijos que proveen otros correctores, entre otros.

8. Referencias y enlaces.

Miko Matsumura, Bjoern Brauel y Jignesh Shah. (2009). *SOA Adoption for Dummies.* Wiley Publishing.

Bledsoe, W. W. y Browning, I. (1959). Pattern recognition and reading by machine. *Proceedings of the Eastern Joint Computer Conference*, (págs. 225-232). Nueva York.

Bordignon, Fernando R. A., Tolosa, G. H., Peri, J. y Barrientos, Diego. (2005). *Método de Corrección Ortográfica Basado en Trigramas y Distancia de Edición.* Departamento de Ciencias Básicas - Universidad Nacional de Luján.

Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM* , 171-176.

Durham, I., Lamb, D.A. y Saxe, J.B. (1983). Spelling correction in user interfaces. *Communications of the ACM* , vol. 26, no. 10, pp. 764-773.

Earnest, L. (1962). Machine recognition of cursive writing. *IFIP Congress.* Munich.

Earnest, L. (07 de 07 de 2015). *The first cursive handwriting recognizer needed a spelling checker.* Obtenido de http://web.stanford.edu/~learnest/les/spelling.htm

Fernández, A. (2013). *Asociación de Desarrolladores Web de España.* Obtenido de http://www.adwe.es/general/colaboraciones/servicios-web-restful-con-http-parte-i-introduccion-y-bases-teoricas

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* Univerity of California, Irvine.

García, J. F. (2007). Métricas de Similitud para Búsqueda Aproxima. *Journal of Technology* , Volumen 6, No. 2.

Gorin, R. (1973). *Using the spelling checker, SAIL User Program Memo.*

Grady Booch, James Rumbaugh e Ivar Jacobson. (2001). *El Lenguaje Unificado de Modelado.* Addison Wesley.

K. Gray, M. (s.f.). *www.mit.edu.* Obtenido de Massachusetts Institute of Technology: http://web.mit.edu/~mkgray/jik/sipbsrc/src/ispell-3.1/Contributors

Keller, W.A. y Burkhard, R.M. (1973). Some approaches to best-match file searching. *Communications of the ACM* .

Kukich, K. (1992). Techniques for automatically correcting words in text. *Computing Surveys* , vol. 24, no. 4, pp. 377-439.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *ADS* , Vol. 10, p. 707.

Lindgren, N. (1965). Machine Recognition of Human Language, Part III – Cursive Script Recognition. *IEEE Spectrum* .

Mark D. Kernighan, Kenneth W. Church, y William A. Gale. (1990). A spelling correction program based on a noisy channel model. *Proceedings of the 13th conference on Computational linguistics* (págs. 205–210). Association for Computational Linguistics.

Ministerio de Hacienda y Administraciones Públicas. (2000). *Portal de Adminitración electrónica.* Obtenido de Metrica Versión 3: http://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html#.VgNtd8uvHIU

Mitton, R. (1996). Spellchecking by computer. *Journal of the Simplified Spelling Society* , Vol 20, No 1, pp 4-11.

Navarro, G. (1999). *A Guided Tour on Approximate String Matching.* Santiago de Chile: Departamento de Ciencias de la Computación, Universidad de Chile.

Norvig, P. (10 de Abril de 2007). *How to write a spelling corrector*. Recuperado el 12 de 06 de 2015, de http://norvig.com/spell-correct.html

Pollock, Joseph J. y Zamora, Antonio. (1984). Automatic spelling correction in scientific and scholarly text. *Communications of the ACM* , vol. 27, no. 4, pp. 358-368.

Real Academia de la Lengua Española. (1999). Letra b. En *Ortografía de la Lengua Española.*

W3C. (2 de 12 de 2004). *Guía breve de servicios Web*. Obtenido de http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb

W3C. (2007). *SOAP Versión 1.2 Part 0: Primer (Second Edition).* W3C.

W3C. (11 de 02 de 2004). *Web Service Architecture Requirements*. Obtenido de http://www.w3.org/TR/wsa-reqs/

W3C. (11 de 02 de 2004). *Web Service Glossary*. Obtenido de http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice