4. Desarrollo y trabajo sobre el sistema Ecode

Se propusieron algunas soluciones para optimizar el sistema Ecode que, después de analizar su viabilidad y conveniencia, fueron implementadas. En general se desarrollaron dos módulos adicionales dedicados a la entrada y salida de los datos, además de hacer una limpieza general del código y reescritura de las expresiones regulares.

El sistema original quedó asimilado por completo en el nuevo sistema y no cambió el flujo de los datos dentro de éste, a pesar de haber sido reescrito parcialmente y optimizado. El modelo final del Ecode se muestra en la siguiente figura:

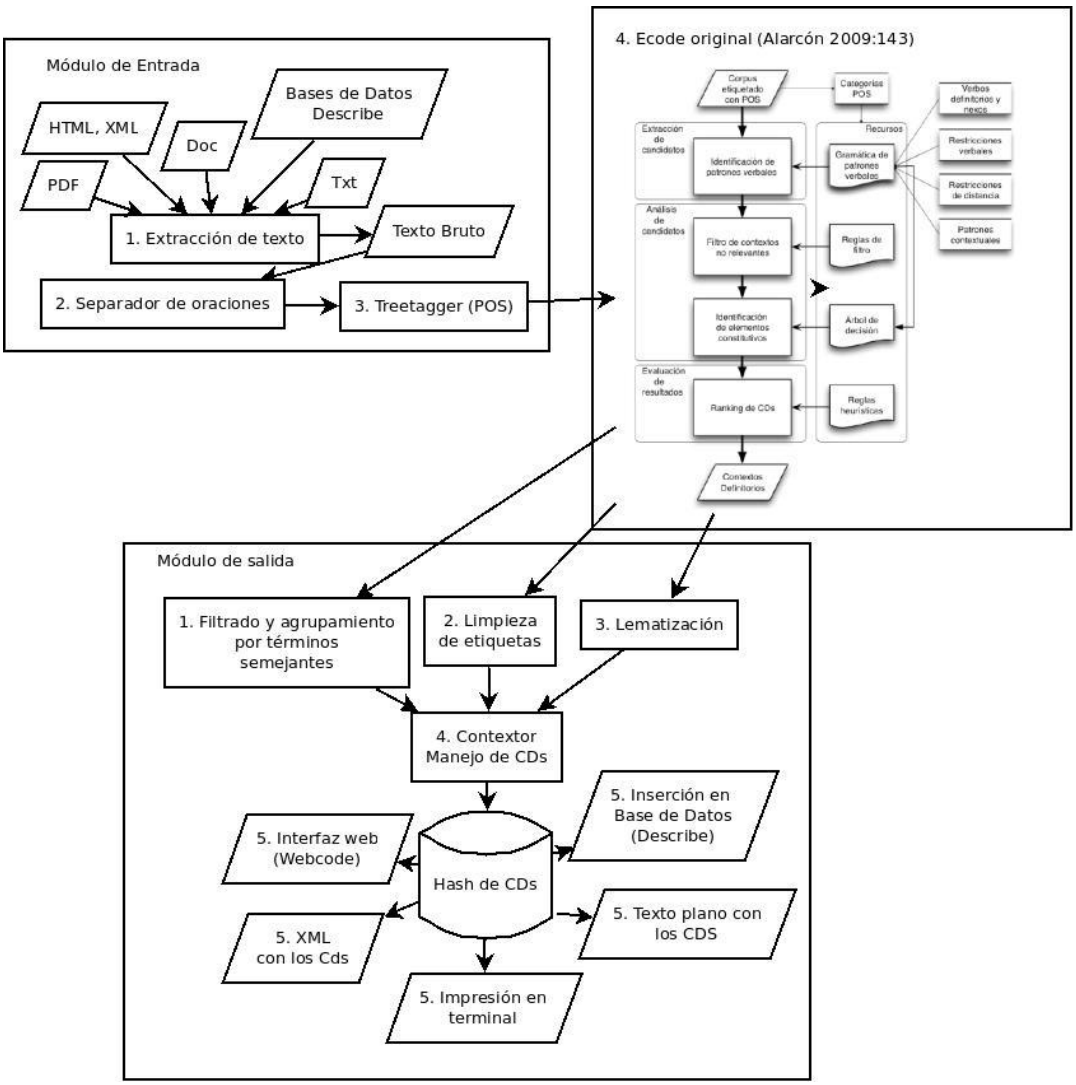

Figura 4.1 Panorama final del Ecode

Como se puede ver en la figura anterior, la adaptación del sistema se basó, fundamentalmente, en flexibilizar las entradas de datos además del acoplamiento con el sistema Describe. Por una parte, la optimización se realizó a nivel de código y de expresiones regulares del Ecode original; mientras que la expansión se llevó a cabo proponiendo una estructura de datos que entregue fácilmente los CDs previamente extraídos y permita otras funcionalidades como lematizar los contextos y limpiar sus etiquetas.

Además de los archivos con los que cuenta originalmente el sistema, se agregaron archivos en donde se contienen los módulos de entrada y salida. Veamos con mucho más detalle, en la siguiente tabla, los nuevos archivos que se integraron al sistema original:

Archivo	**Descripción**
00_Inicia.pm	Inicializa variables globales, carga y preprocesa gramáticas.
ES_entrada.pm	Extracción de texto, etiquetador POS, separador de oraciones.
ES_salida.pm	Eliminación de etiquetas, lematización, creación de Hash de CDs y filtrado por términos.
ES_DB_describe.pm	Funcionalidad para trabajar con la base de datos de Describe.
ES_File_Type.pm	Módulo de Perl para identificar el tipo de archivo de entrada.
Porter_Stem_Es.pm	Lematizador Porter para el español.
util.pm	Funciones utilizadas a lo largo del sistema como apoyo.
libecode.pm	Permite la ejecución secuencial del módulo de entrada, Ecode y salida.

Tabla 4.1 Archivos agregados a Ecode

4.1 Optimización del Ecode original

La optimización del módulo central, es decir, el Ecode original, se realizó en tres etapas. Después de un análisis del flujo del programa se encontró que lo que debía optimizarse eran las operaciones de entrada y salida entre los módulos de procesamiento del Ecode, el uso indebido y extenso de las estructuras de control y de variables globales, y la utilización de expresiones regulares de forma inapropiada o sin aprovechar el potencial de éstas.

4.1.1 Operaciones de Entrada/Salida

El Ecode original presentaba lectura y escritura a disco duro entre cada archivo, lo cual, si bien sirve para obtener información intermedia del procesamiento, lo ideal sería almacenar dicha información y realizar una sola operación de escritura al final del procesamiento disco.

Por ejemplo (Archivo: 01_vd.pm 156-164):

```
# Imprime un documento con todas las líneas que tienen un verbo definitorio en la carpeta de OUTPUT
open FILE, (">outputAux/s01_preCandidatos.txt") or die "No se puede escribir el documento con preCandidatos!";
print FILE our @preCandidatos;
close FILE;

# Imprime un documento con todas las líneas que NO tienen un Verbo Definitorio en la carpeta de OUTPUT
open FILE, (">outputAux/n01_preNRsSinVDs.txt") or die "No se puede escribir el documento con n01_preNRsSinVDs!";
print FILE our @NRs1;
close FILE;
```

Este fenómeno se repite a lo largo de cada archivo y para pasar la información se utilizan arreglos globales que pueden suplir directamente en la memoria la tarea de los archivos sin tener que recurrir a la escritura en el disco duro. La siguiente tabla muestra cuántas lecturas y escrituras de archivos se producen en el sistema:

Archivo (Módulo)	Operaciones de Entrada –(lectura de archivos)	Operaciones de Salida (Escritura de archivos)
01_vds.pm	1	2
02_pvds.pm	2	2
03_td.pm	2	1
04_filtro.pm	3	2
05_arbol.pm	2	2
06_retagging.pm	1	1
07_rankingTyD.pm	2	1
08_rankingGlobal.pm	2	1
09_final.pm	7	6
01_gramaticaPOS.pm	0	0
ecode.pl	1	3

Tabla 4.2 Ocurrencias de operaciones de entrada/salida en los módulos de Ecode

De la tabla anterior salta a la vista que la escritura y lectura se llevan a cabo en cada archivo para pasar al arreglo de información a través del sistema. Las lecturas restantes pertenecen a lectura de las gramáticas, realizándose repetidas veces en los módulos, lo que resulta redundante y probablemente implique que sean cargadas solamente una vez. Además de lo anterior, las escrituras restantes representan archivos de procesamiento intermedio para motivos de corrección de errores, esta funcionalidad se mantuvo durante el desarrollo de la optimización para la realización de pruebas; sin embargo, el usuario final será capaz de decidir si requiere o no esta información.

4.1.2 Simplificación de código

El código del Ecode original presentaba un uso ineficiente de las estructuras de control y realizaba varios procesamientos previamente realizados en módulos anteriores, por lo que se optó por simplificar el código con las siguientes consideraciones:

4.1.2.1 Inicialización de variables y gramáticas

En primer lugar, se creó el módulo (00_inicia.pm) que inicializa las variables globales que van a ser usadas (arreglos de gramáticas, arreglos de paso de información entre módulos, etc.). En este paso, se preprocesan y cargan las gramáticas en arreglos globales para ser llamadas desde la memoria en los diferentes módulos. El preprocesamiento consiste en separar la información de las gramáticas y colocarla en arreglos.

4.1.2.2 Simplificación de variables y su alcance

En todos los módulos se intentaba saber cómo fueron declaradas las variables y su uso interno; con esto, se descubrió que en todos los casos las variables fueron declaradas como globales en Perl, lo que podría propiciar un conflicto entre nombres de variables, además de que resulta innecesario tener en la memoria toda la información de procesos anteriores, pues ya no es relevante su existencia. Adicionalmente se encontró que el operador *our* era usado para cada aparición de las variables; si bien esto no genera algún error por parte del intérprete de Perl, el proceso equivale a declarar muchas veces las variables, razón por la cual fue corregido. Veamos el siguiente ejemplo que aclara más el asunto:

(Archivo Ecode original: 02_pvds.pm: 89-105):

```
foreach our $patron (our @arrayPVD1)
{
if ($patron =~ /<lm>(.*?)<.lm>/)
{
                    our $lema = $1;
}
our $lema;
our $vd = "<vd lema=\"$lema\">[^]*?<.vd>";
foreach our $preCandidato (@preCandidatos)
{
# PVD - Rule 05
while ($preCandidato =~ / <id[^]*?>$vd /gi)
{
```

```
$preCandidato =~ s/ <id([^ ]*?)>($vd) / pvd_r05_$1><vd_$1>$2<\/vd_$1><\/pvd_r05_$1> /i;
}
}
}
```

En el ejemplo se aprecia el uso extensivo del operador *our* que declara variables, pero se utiliza en cada aparición de la variable lo cual resulta ser confuso e innecesario. El problema se reparó de esta manera (Archivo Ecode final: 02_pvds.pm: 18-26 76-86):

```
our (@gramaPVDs,@patrones,@preCandidatos,@candidatos,@NRs);
our ($sele, $se, $tag,$aux,$verboCon);
my (@arrayPVD1, @arrayPVD2,@arrayPVD3);
my ($patron,$lema,$vd,$nexoAux,$nexo,$preCandidato,$distancia,$distanciaAux);
…
for $patron (@arrayPVD1)
{
$lema = $1 if ($patron =~ /<lm>(.*?)<.lm>/) ;

        $vd = "<vd lema=\"$lema\">[^ ]*?<.vd>";
        for $preCandidato (@preCandidatos)
        {
                # PVD - Rule 05
$preCandidato =~ s/ <id([^ ]*?)>($vd) / <pvd_r05_$1><vd_$1>$2<\/vd_$1><\/pvd_r05_$1> /I
while ($preCandidato =~ / <id[^ ]*?>$vd /gi) ;
}
}
```

En el ejemplo anterior se puede apreciar a las variables que quedan declaradas al inicio del módulo, separadas por globales y locales, y que posteriormente son usadas sin el operador *our* en el módulo, además se observa de qué manera se comprimieron las estructuras de control.

Otro punto interesante, es que no se utilizó el paso de información entre las diferentes funciones por medio de sus argumentos, simplemente se instanciaba la variable requerida con el operador *our* y se utilizaba, por lo que se implementó el paso de argumentos entre las diferentes funciones.

4.1.2.3 Reescritura de expresiones regulares

Para el mejoramiento del sistema Ecode original, se hicieron cambios en algunas expresiones regulares, por ejemplo: se eliminó la variable *$iniciolinea* que marcaba el inicio de una línea, pues las expresiones regulares proveen esta funcionalidad, además de utilizar los operadores de las expresiones regulares para simplificar condicionales y bucles en el sistema. Un ejemplo de la reescritura de expresiones regulares es ilustrado en el siguiente ejemplo (Archivo Ecode Original: 05_arbol.pm: 465-482):

```
if (/<izq>.*? \(\/[^]*? ($termino1.*?)<.izq>/i)
{
        if ($1 =~ /\)\/[^]*?/i)
        {
        }
        elsif ($1 =~ /$verboCon/i)
        {
        }
        else
        {
                s/<izq>(.*? \(\/[^]*?) ($termino1.*?)<.izq>/$1 <t>$2<\/t>/gi;
                s/<der>(.*?) (\)\/[^]*? .*?)<.der>/<d>$1<\/d> $2/gi;
                s/<der>(.*?) (\)\/[^]*?)<.der>/<d>$1<\/d> $2/gi;
                s/<der>(\)\/[^]*? .*?)<.der>/<d>$1<\/d>/gi;
                s/<der>(\)\/[^]*?)<.der>/<d>$1<\/d>/gi;
                s/(<cand.>)/$1<cd_RI16\/>/gi;
        }
}
```

En esta secuencia se observa claramente el uso erróneo de la estructura de control condicional, por lo que es posible simplificarla uniendo los 2 primeros condicionales en uno solo reescribiendo la estructura de control y la expresión regular, de esta manera (Archivo Ecode Final: 05_arbol.pm: 430-441):

```
if (/<izq>.*? \(\/[^]*? ($termino1.*?)<.izq>/i)
{
        if ($1 !~ /\)\/[^]*?|$verboCon/i)
        {
                s/<izq>(.*? \(\/[^]*?) ($termino1.*?)<.izq>/$1 <t>$2<\/t>/gi;
                s/<der>(.*?) (\)\/[^]*? .*?)<.der>/<d>$1<\/d> $2/gi;
                s/<der>(.*?) (\)\/[^]*?)<.der>/<d>$1<\/d> $2/gi;
                s/<der>(\)\/[^]*? .*?)<.der>/<d>$1<\/d>/gi;
                s/<der>(\)\/[^]*?)<.der>/<d>$1<\/d>/gi;
                s/(<cand.>)/$1<cd_RI16\/>/gi;
        }
}
```

El procedimiento seguido para el mejoramiento del sistema en este punto en particular fue el siguiente: se reescribió la expresión regular del condicional englobando las 2 condicionales que no producían ningún procesamiento y negándolas, con esto se simplificaría enormemente el código y se haría más rápido, ya que genera menos comparaciones.

4.1.2.4 Compresión de estructuras de control

Por otra parte, se decidió comprimir las estructuras de control, en especial los condicionales, ya que en su uso extensivo no se tomaron en cuenta los operadores que permite Perl para comparar entre las expresiones regulares. Durante el proceso se consideraron diferentes formas de comprimir las estructuras de control: primero, se comprimieron todas aquellas estructuras de control (*if, for, while*) que sólo produjeran una línea; esto puede considerarse meramente por estilo, sin embargo, se realizó por simplificación.

Por ejemplo, el Archivo Ecode Original: 03_td.pm: 62-65 a continuación ilustrado:

```
if ($patron=~ /<nx>(.*?)<.nx>/i)
{
        our $nexoDef = $1;
        }
```

Se comprimió en (Archivo Ecode Final: 03_td.pm: 39):

```
$nexoDef = $1 if ($patron=~ /<nx>(.*?)<.nx>/i);
```

Además, en el caso de los condicionales *if* se comprimieron los que presentaban su producción principal y un *elsif* que producía sentencias, por ejemplo, el Archivo Ecode Original: 03_td.pm: 85-91:

```
if (/<nx_$lemaID>/i)
{
}
else
{
        s/($inicioLinea)/<tipoD=\"$tipoDef\"\/>$1/gi;
}
```

Se comprimió en el Archivo Ecode Final: 03_td.pm: 59:

```
        s/^/<tipoD=\"$tipoDef\"\/>/gi  if (! /<nx_$lemaID>/i) ;
```

Asimismo, existen estructuras condicionales que presentan varias condiciones anidadas que no producen nada y su producción es en el *else*. Retomando un ejemplo anterior (Archivo Ecode Original: 05_arbol.pm: 465-482), podemos ver la situación de manera concreta:

```
if (/<izq>.*? \(\/[^ ]*? ($termino1.*?)<.izq>/i)
{
        if ($1 =~ /\)\/[^ ]*?/i)
        {
```

```
        }
        elsif ($1 =~ /$verboCon/i)
        {
        }
        else
        {
                s/<izq>(.*? \(\/[^]*?) ($termino1.*?)<.izq>/$1 <t>$2<\/t>/gi;
                s/<der>(.*?) (\)\/[^]*? .*?)<.der>/<d>$1<\/d> $2/gi;
                s/<der>(.*?) (\)\/[^]*?)<.der>/<d>$1<\/d> $2/gi;
                s/<der>(\)\/[^]*? .*?)<.der>/<d>$1<\/d>/gi;
                s/<der>(\)\/[^]*?)<.der>/<d>$1<\/d>/gi;
                s/(<cand.>)/$1<cd_RI16\/>/gi;
        }
}
```

Se sustituyó reescribiendo la expresión regular y comprimiendo el condicional (Archivo Ecode Final: 05_arbol.pm: 430-441), de la manera siguiente:

```
if (/<izq>.*? \(\/[^]*? ($termino1.*?)<.izq>/i)
{
        if ($1 !~ /\)\/[^]*?|$verboCon/i)
        {
                s/<izq>(.*? \(\/[^]*?) ($termino1.*?)<.izq>/$1 <t>$2<\/t>/gi;
                s/<der>(.*?) (\)\/[^]*? .*?)<.der>/<d>$1<\/d> $2/gi;
                s/<der>(.*?) (\)\/[^]*?)<.der>/<d>$1<\/d> $2/gi;
                s/<der>(\)\/[^]*? .*?)<.der>/<d>$1<\/d>/gi;
                s/<der>(\)\/[^]*?)<.der>/<d>$1<\/d>/gi;
                s/(<cand.>)/$1<cd_RI16\/>/gi;
        }
}
```

Al final, por estilo se sustituyeron las estructuras de control *foreach* por *for*, ya que son equivalentes y tienen la misma sintaxis[13].

4.1.2.5 Conjunción de ciclos similares

Durante el proceso experimental, resultó muy frecuente que se presentaran situaciones en las que un ciclo *for* era usado dos o más veces consecutivas para iterar un arreglo y realizar operaciones sobre él, situación que podría optimizarse simplificando y conjuntado varios ciclos *for* en uno solo. Con este proceso se ahorran muchos ciclos de procesamiento y se simplifica el código. Veamos un ejemplo (Archivo Ecode Original: 02_pvds.pm: 169-237):

```
# Sustituye <vd_#></vd_#> por <vd_# lema.*?></vd_#>
foreach (@preCandidatos)
{
        # Sustituye <vd_ID><vd lema=".*"> por <vd_ID lema=".*">
        s/<vd(_[^]*?)>(<vd)( lema)/$2$1$3/gi;
        s/<\/vd>(<\/vd_[^]*?>)/$1/gi;
        # Limpia dobles espacios entre <\/vd> y <nx>
        s/  (<nx)/ $1/gi;
}
# Sustituye etiqueta AUXILIAR: <prAux_#></prAux_#>
foreach (@preCandidatos)
{
        our ($sele, $aux);
s/<prAux_([^ ]*?)>($sele) ($aux)<.prAux_[^ ]*?>/<pr_$1>$2<\/pr_$1>
<aux_$1>$3<\/aux_$1>/gi;
        s/<prAux_([^]*?)>($sele)<.prAux_[^]*?>/<pr_$1>$2<\/pr_$1>/gi;
        s/<prAux_([^]*?)>($aux)<.prAux_[^]*?>/<aux_$1>$2<\/aux_$1>/gi;
        # Añade etiqueta SE: <\/vd> se <NX>
        s/(<.vd_)([^]*?)(>) ($sele) (<nx_[^]*?>)/ $1$2$3 <pr_$2>$4<\/pr_$2> $5/gi;
}
```

[13] (http://perldoc.perl.org/perlsyn.html#Foreach-Loops)

```
# Expande etiqueta de PVD
foreach (@preCandidatos)
{
	our ($sele, $se, $aux);
	# Expande <pvd_r..> a la IZQUIERDA (excepto para PVDs con lema SER)
	while (/ ((?:$sele $aux|$sele|$aux) <pvd_r.._[^]*?><vd_[^]*? [^]*?>)/gi)
	{
		if ($1 =~ /<vd_[^]*? lema=\"ser\">/)
		{
		}
		else
		{
			# SELE + AUX
s/ ($sele) ($aux) (<pvd_r.._)([^ ]*?)(>)(<vd_[^ ]*? [^ ]*?>)/ $3$4$5<pr_$4>$1<\/pr_$4>
<aux_$4>$2<\/aux_$4> $6/gi;
			# SELE
s/ ($sele) (<pvd_r.._)([^]*?)(>)(<vd_[^]*? [^]*?>)/ $2$3$4<pr_$3>$1<\/pr_$3> $5/gi;
			# AUX
s/ ($aux) (<pvd_r.._)([^]*?)(>)(<vd_[^]*? [^]*?>)/ $2$3$4<aux_$3>$1<\/aux_$3> $5/gi;
		}
	}
	# Expande <pvd_r..> a la DERECHA
	# SELE
	s/(<.vd_[^]*?>)(<.pvd_r.._)([^]*?)(>) ($sele) /$1 <pr_$3>$5<\/pr_$3>$2$3$4 /gi;
}

# ------------------------------------------------->>>>>>>>
# PUSH + WARNING
# ------------------------------------------------->>>>>>>>
foreach (@preCandidatos)
{
	our $KpreCandidatos++;
	# Push candidatos
	if (/(<pvd_[^]*?>)/)
```

```
        {
                push our @candidatos, ($_);
        }
        else
        {
                # Push No Relevantes 2 - Sin Patrones Verbales
                push our @NRs2, ("<NRs tipo=\"sinPVDs\"\/>".$_);
                # Comprueba que no queden verbos definitorios anotados sin procesar
                if (/<.vd_[^]*?>/)
                {
                        push our @warnings, ("Faltan vds por procesar - 02_pvds.pm linea 163\n");
                }
        }
}
```

Si se mira con detenimiento en el ejemplo anterior se aprecia como un ciclo *foreach* se repite cuatro veces consecutivas que itera sobre *@preCandidatos*, esto se puede comprimir de la siguiente manera, para obtener mayor eficiencia en el proceso y un mayor ahorro de espacio (Archivo Ecode Final 02_pvds.pm: 135-176):

```
# Sustituye <vd_#></vd_#> por <vd_# lema.*?></vd_#>
for (@preCandidatos)
{
        # Sustituye <vd_ID><vd lema=".*"> por <vd_ID lema=".*">
        s/<vd(_[^]*?)>(<vd)( lema)/$2$1$3/gi;
        s/<\/vd>(<\/vd_[^]*?>)/$1/gi;
        # Limpia dobles espacios entre <\/vd> y <nx>
        s/  (<nx)/ $1/gi;
        # Sustituye etiqueta AUXILIAR: <prAux_#></prAux_#>
s/<prAux_([^ ]*?)>($sele) ($aux)<.prAux_[^ ]*?>/<pr_$1>$2<\/pr_$1><aux_$1>$3<\/aux_$1>/gi;
        s/<prAux_([^]*?)>($sele)<.prAux_[^]*?>/<pr_$1>$2<\/pr_$1>/gi;
        s/<prAux_([^]*?)>($aux)<.prAux_[^]*?>/<aux_$1>$2<\/aux_$1>/gi;
```

```
        # Añade etiqueta SE: <\/vd> se <NX>
        s/(<.vd_)([^]*?)(>) ($sele) (<nx_[^]*?>)/ $1$2$3 <pr_$2>$4<\/pr_$2> $5/gi;

        # Expande etiqueta de PVD<pvd_r..> a la IZQUIERDA (excepto para PVDs con lema SER)
        while (/ ((?:$sele $aux|$sele|$aux) <pvd_r.._[^]*?><vd_[^]*? [^]*?>)/gi)
        {
                if ($1 !~ /<vd_[^]*? lema=\"ser\">/)
                {
                        # SELE + AUX
s/ ($sele) ($aux) (<pvd_r.._)([^ ]*?)(>)(<vd_[^ ]*? [^ ]*?>)/ $3$4$5<pr_$4>$1<\/pr_$4>
<aux_$4>$2<\/aux_$4> $6/gi;
                        # SELE
s/ ($sele) (<pvd_r.._)([^]*?)(>)(<vd_[^]*? [^]*?>)/ $2$3$4<pr_$3>$1<\/pr_$3> $5/gi;
                        # AUX
s/ ($aux) (<pvd_r.._)([^]*?)(>)(<vd_[^]*? [^]*?>)/ $2$3$4<aux_$3>$1<\/aux_$3> $5/gi;
                }
        }
        # Expande <pvd_r..> a la DERECHA  SELE
        s/(<.vd_[^]*?>)(<.pvd_r.._)([^]*?)(>) ($sele) /$1 <pr_$3>$5<\/pr_$3>$2$3$4 /gi;

        # Push candidatos
        if (/(<pvd_[^]*?>)/)
        {
                push @candidatos, ($_);
        }
        else
        {
                # Push No Relevantes 2 - Sin Patrones Verbales
                push @NRs, ("<NRs tipo=\"sinPVDs\"\/>".$_);
        }
}
```

Al final, solamente en este ejemplo se pudieron comprimir cuatros ciclos en uno solo.

4.1.3 Impacto de la optimización del Ecode original y trabajo futuro

La optimización realizada al sistema es el primer paso para convertir al Ecode, no sólo en un sistema eficiente, sino también en una fuente para la investigación de contextos definitorios que fue concebida para solucionar problemas computacionales no resueltos y como un apoyo para los lingüistas en sus estudios del léxico.

Como ya hemos explicado, se reescribió gran parte del código del sistema, se revisaron las expresiones regulares y se planteó un esquema de trabajo para simplificar el código y arreglar problemas detectados. Asimismo, se redujeron, en gran medida, el número de líneas del código y la cantidad de ciclos de procesamiento que se requieren para ejecutarlo, lo que se tradujo en un sensible mejoramiento del sistema.

El siguiente paso en la optimización de Ecode consistía en permitir a los usuarios modificar las gramáticas y agregar nuevas reglas, esto mediante una interfaz que convierte las gramáticas a XML; ya que existen reglas importantes que deberían estar en gramáticas y, además, deberían ser configurables (por ejemplo, las reglas de evaluación de los CDs (Ranking)).

Las secuencias de expresiones regulares deben ser analizadas una por una para tratar de simplificar el conjunto basado en las entradas y salidas que requiere cada módulo, es decir, que en vez de revisar cada expresión regular una por una, se analizarían como un todo; como si fueran una gramática y, al mismo tiempo, tratar de simplificar la cantidad de expresiones regulares totales.

4.2 El Módulo de entrada de datos

En un principio el Ecode recibía de entrada un texto previamente separado por oraciones y etiquetado con POS, por lo que se tuvo que generar un módulo completo para producir esa entrada a partir de los archivos de aplicación e internet, para leer o producir texto, más frecuentes entre los usuarios de computadoras de hoy en día (Doc, HTML, PDF, XML).

Fue necesario encontrar un nuevo etiquetador POS para adecuarlo al sistema, ya que el etiquetador disponible era muy lento. Entonces, se adaptó el TreeTagger (véase 2.1.3.2) al sistema y se desarrolló una interfaz que permitiera dicho acoplamiento.

El módulo de entrada se puede apreciar en la siguiente figura:

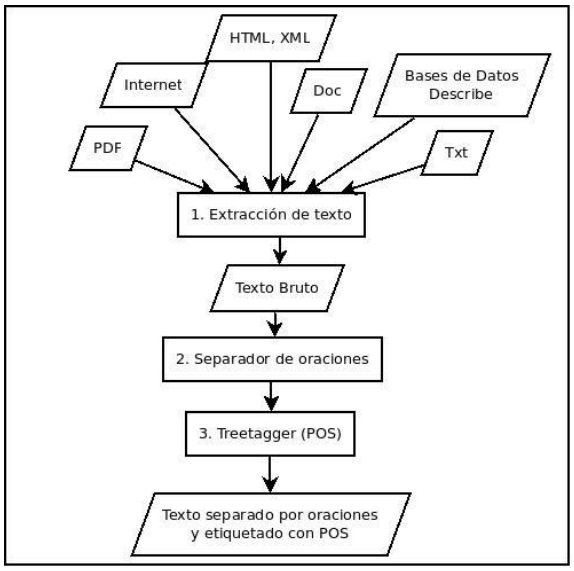

Figura 4.2 El módulo de entrada de Ecode

4.2.1 La fuente de los datos

El uso del sistema estaba restringido a llevar una serie de procesos previos al Ecode, anteriormente mencionados; sin embargo, uno de los objetivos de esta tesis es desarrollar un sistema que acoplara Ecode al sistema Describe como motor principal, además de presentar una herramienta independiente a usuarios que requieran extraer información de sus propios textos. Por lo tanto, fue necesario desarrollar una interfaz independiente para el usuario interesado únicamente en el Ecode y los usuarios que generaran una ejecución de Ecode desde el sistema Describe.

4.2.1.1 Texto plano y extracción de éste a partir de archivos de aplicación de usuario e internet

La conexión a la base de datos de Describe era un requerimiento importante del Ecode; sin embargo, también fue requerida una herramienta que permitiera extraer los CDs de textos y orígenes particulares a usuarios que los requirieran.

Los usuarios concentran su lectura y escritura de texto en formatos de aplicación (Doc, pdf, html, xml), por lo que lo más natural para un usuario final es mantener su corpus textual en alguno de éstos formatos. Algunos usuarios podrían utilizar texto plano (txt), lo cual también está contemplado, y adicionalmente internet; por lo tanto, se pensó en desarrollar

un módulo de entrada flexible que se ajustara a las necesidades de entrada de los usuarios.

Para la identificación del tipo de archivo se utilizó el módulo estándar File::type[14] de Perl, que recibe el archivo, lee su cabecera e identifica el tipo de archivo que es; este módulo se utiliza para poder discernir entre el método de extracción del texto de acuerdo con el tipo de archivo.

En la siguiente tabla se describen los tipos de archivo de aplicación que soporta Ecode y la herramienta que se usa para extraer el texto de ellos:

Tipo de archivo de aplicación	Herramienta de extracción de texto
Documento de Word (Doc) hasta la versión 2003	Antiword, herramienta de extracción de texto de archivos de Word
Documento portátil (PDF)	Cam::PDF, Módulo de manejo de PDFs
Lenguajes de etiquetado (XML, HTML, XHTML)	Elinks, explorador web de modo texto con capacidad de volcar el texto
Texto plano (txt)	Perl, Ecode
Todos los archivos de un directorio	Perl
Internet	Elinks, explorador web de modo texto con capacidad de volcar el texto

Tabla 4.3 Herramientas para la extracción de texto

Todas estas herramientas convergen en el flujo del programa entregando el texto bruto que extrajeron del tipo de archivo correspondiente.

4.2.2. El problema de las codificaciones

Uno de los problemas que llevó más tiempo resolver, amén de que no se pudo llegar a una solución total, fue el de las codificaciones de entrada al sistema. Si bien el texto en español suele escribirse sólo en dos o tres codificaciones de caracteres, en la práctica no es así, ya que frecuentemente se encuentran caracteres mal formados o no reconocidos, errores de codificación, mezcla entre codificaciones y errores en el reconocimiento de éstas.

Así que se realizó un análisis y se encontró que la gran mayoría de texto en español se

[14] (http://search.cpan.org/dist/File-Type/lib/File/Type.pm)

encuentra o en UTF-8 o en ISO-8859-1, lo cual permitió buscar herramientas que detectaran y cambiaran las codificaciones de los documentos de entrada.

Además, se presentó la problemática de que un texto podría tener varias codificaciones, en especial viniendo de la base de datos de Describe que maneja todo tipo de sitios web de donde extrae el texto en codificaciones variadas.

La solución que se implementó consiste en:

a) Tomar el texto en bruto y separarlo por líneas.
b) Analizar si cada línea pertenecía a la codificación UTF-8.
c) En caso afirmativo, se convierte a ISO-8859-1.
d) En caso negativo, se presume que es ISO-8859-1, sin garantía, debido a la falta de herramientas confiables de detección de codificación ISO-8859-1 y que la gran mayoría del texto que no sea UTF-8 es ISO-8859-1.
e) Se une el texto bajo la codificación ISO-8859-1.
f) Se ejecuta POS y la salida se convierte a la codificación a UTF-8 que será la única a lo largo del resto del sistema.

De esta manera se pudieron corregir una gran cantidad de errores de codificación que ocasionaban resultados equívocos. Sin embargo, es apropiado mencionar que el problema de las codificaciones continúa arrojando errores y que la gran cantidad de variables entre ellas, las herramientas disponibles y los errores en los documentos son duros obstáculos que se tienen que superar en el Ecode y en general en el tratamiento de texto en general.

4.2.3 Separador de oraciones

Otro de los problemas de difícil acercamiento en el procesamiento del texto es la distinción entre oraciones; la propia dificultad técnica de encontrar reglas que permitan separar el texto en oraciones se ve opacada por la inconsistencia en la teoría lingüística sobre qué es o no una oración. Para fines prácticos del Ecode, se definió a una oración como una porción de texto entre 2 separadores de oración (. ¿? ¡), aunque existen ciertas reglas antes y después de cada separador de oración que especifica si es o no una oración.

En la siguiente tabla se muestran reglas de separación de oraciones usadas en el Ecode:

Excepción
Abreviaciones: etc.
Títulos: Dr. Ing. Lic. Sr. Sra. Srita.
Marcas de puntuación: …
Reglas de separación
Separador de oración (!?.) espacio
Separador de oración (¡?.) digito

Tabla 4.4 Excepciones y reglas de separación de oraciones

Si bien las reglas y excepciones son muy simples, separan bien el texto para el procesamiento de Ecode; sin embargo, es necesario plantear un análisis más detallado de la conformación de los CDs como oraciones y sobre cómo pueden ser delimitadas para una mayor precisión en este proceso.

En un futuro se debe permitir a los usuarios modificar las reglas de separación de oraciones de acuerdo con sus necesidades; sin embargo, esta posibilidad aún no está implementada.

4.2.4 El etiquetador POS

En un principio Ecode no contaba con un etiquetador POS integrado al sistema. El usuario debía preprocesar su texto para poder ingresarlo al Ecode; de tal manera que fue necesario integrar un etiquetador POS al sistema y permitir libertad al usuario en el procesamiento de sus textos.

El etiquetador POS para el cual el Ecode estaba acoplado era un etiquetador de Brill modificado que funcionaba con un grupo de etiquetas del Corpus del Español Mexicano Contemporáneo (Tabla 2.1, conjunto de etiquetas del CEMC).

Inicialmente, se desarrolló un acoplamiento del etiquetador Brill al Ecode; sin embargo, y aunque funcionaba con buenos resultados, tenía marcadas deficiencias en su desempeño, en velocidad, en tamaño del programa y también en la utilización, por lo que se decidió buscar otro etiquetador POS que permitiera evitar estos problemas y se encontró que una solución mucho más eficiente es el TreeTagger.

El etiquetador TreeTagger provee varias ventajas sobre el etiquetador Brill:

- Es más pequeño.
- Salida fácil de acoplar y por Pipes sin archivos.
- Contiene un conjunto de etiquetas más amplio.
- Permite ser entrenado, aunque ya está entrenado para textos del español.

Considerando estas ventajas se implementó una interfaz entre Ecode y TreeTagger; el principal reto fue que TreeTagger tiene más etiquetas POS que el conjunto del CEMC, por lo que resultó necesario elaborar una tabla de equivalencia y convertir las etiquetas. La tabla de equivalencia es la siguiente:

Etiqueta del CEMC, usadas por Ecode	**Etiquetas de TreeTagger**
/2 (Adjetivo)	ADJ
/1 (Adverbio)	ADV
/6 (Artículo)	ART
/B (Nombre Propio)	NP, ACRNM (Acrónimo), NMON (nombre del mes)
/8 nombre común	NC, NMEA (medida de medición *pesos*, *litros*, etc.)
/4 (preposición)	PREP, PREP/DEL (*del* como preposición)
/3 (Conjunción)	CC, CQUE (*que* como conjunción)
/s (Símbolo)	CM (coma ,), COLON (dos puntos *:*), DOTS (tres puntos ...), LP (paréntesis izquierdo), SEMICOLON (punto y coma), SLASH (diagonal), BACKSLASH (contra diagonal) DASH (guion -), PERCT (Porciento %) QT , comilla (‘), RP (paréntesis derecho) SYM (otros símbolos)
/9-CON (Verbo Conjugado)	VHfin (verbo haber conjugado), VEfin (verbo estar conjugado), VLfin (verbo conjugado), VMfin (verbo modal conjugado), VCLIfin (clítico conjugado), VSfin (verbo ser conjugado)
/9-INF (Verbo en infinitivo)	VEinf (verbo estar en infinitivo), VLinf (verbo en infinitivo), VMinf (verbo modal en infinitivo), VHinf (verbo haber en infinitivo), VCLIinf (clítico en infinitivo), VSinf (verbo ser en infinitivo)
/9-GER (verbo en gerundio)	VEger (verbo estar en gerundio), VLger| (verbo en gerundio), VMger (verbo modal en gerundio), VHger (verbo haber en gerundio), VCLIger (clítico gerundio), VSger (verbo ser en gerundio)
/9-PAR (Verbo en participio)	VEadj (verbo estar en participio), VLadj (verbo en participio), VMadj (verbo modal en participio), VHadj (verbo haber en

	participio), VCLIadj (clítico en participio), VSadj (verbo ser en participio)
/1 (Ambigua, no es usada por Ecode o no se pudo incluir en alguna otra categoría)	ACRNM (acrónimo) ALFP(letras en plural del alfabeto *as, bes*), ALFS (letra del alfabeto), CARD (cardinales), CCAD (conjunción coordinada adversativa *pero*), CCNEG (conjunción coordinada negativa *ni*), CODE(código alfanumérico), CSUBF (conjunción subordinada que introduce clausulas finitas *apenas*), CSUBI (conjunción subordinada que introduce clausulas infinitas *al*), CSUBX (conjunción subordinada de tipo no especificado *aunque*), -DM (pronombres demostrativos, son usados pero interceptados en la gramática POS), FO (formula), FS (marcas de puntuación finales), INT (pronombre interrogativo *quiénes cuántas, cuánto*), ITJN (interjección *oh, ja*), NEG (negación), ORD (ordinal), PAL (*al*), PDEL (*del*), PE (palabra extranjera), PNC (palabra no clasificada), PPC (pronombre personal clítico), PPO(pronombre posesivos *mi, sus, su*), PPX(clíticos y pronombres personales, *nos, me, nosotras, te, sí*), QU (cuantificadores *sendas, cada*), REL (pronombres reflectivos *cuyas, cuyo*), UMMX (unidad de medida *MHz, km, mA*)

Tabla 4.5 Tabla de relación entre etiquetas de etiquetadores

Como se aprecia, muchas etiquetas POS se marcan como ambiguas y esto puede ser por dos razones: que el Ecode no utiliza la etiqueta y por lo tanto la etiqueta fue traducida como ambigua, o bien, que el Ecode sí utiliza ese tipo de palabra, pero que la gramática POS (Archivo 01_gramaticaPOS.pm) detecta la palabra sin importar qué etiqueta se le asignó.

Además, cabe mencionar que dicha relación se realizó sin un análisis profundo de los tipos de palabras que entrega TreeTagger; esto debido a que escapa de la finalidad de este trabajo por lo extenso y laborioso que requiere dicho análisis. Tampoco participó un experto y se realizó sólo por comparación simple entre textos.

Claramente, la relación anterior es susceptible de ser optimizada o corregida por un experto lingüista que permita seleccionar mejores relaciones entre las etiquetas para garantizar que la traducción de palabras sea la adecuada y quizás este pendiente pueda ser parte de un proyecto futuro.

El etiquetador ya presentaba un entrenamiento para textos en español, por lo que no fue necesario reentrenar el TreeTagger; sin embargo, el tipo de lenguaje con el que se definen los CDs es culto, por lo que un reentrenamiento de TreeTagger para textos técnicos y cultos en español permitirá ciertamente una mejora en el etiquetamiento y por lo tanto en el rendimiento general de Ecode.

El sistema se ejecuta enviando el texto a etiquetar por medio de un pipe y se recibe con otro haciendo más eficiente el proceso; posteriormente se procesa la salida del TreeTagger y se vuelven a acoplar las oraciones originales pero ya etiquetadas. En esto consiste el

etiquetamiento de las partes de la oración.

4.3 El módulo de salida

Al ejecutarse el Ecode se obtiene una lista de CDs en un arreglo, por lo que es conveniente procesar más la información para preparar la salida del sistema de acuerdo con las necesidades de los usuarios.

En la siguiente figura se muestra la estructura del módulo de salida:

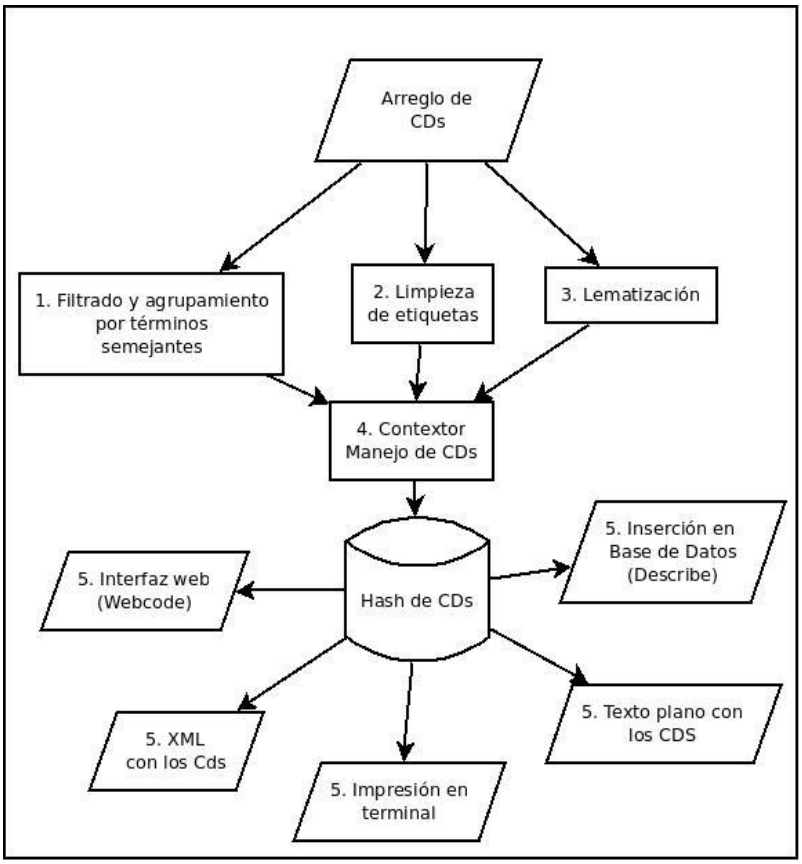

Figura 4.3 El módulo de salida de Ecode

Como se aprecia en la figura, uno de los procesos fundamentales es el número cuatro, "contextor", que tiene la capacidad de extraer los diferentes términos de los CDs y agruparlos por términos semejantes en una estructura de datos que se describirá en breve: el hash de CDs tiene la finalidad de contener a los CDs y presentar una interfaz para las diferentes salidas que requieren los usuarios.

4.3.1. Filtrado y agrupamiento de términos semejantes y asociación de CDs por sus términos

Los CDs, al entrar al módulo de salida en un arreglo, son revisados para extraer el conjunto de términos que corresponde a cada contexto, y posteriormente se aplican reglas que permiten agrupar los términos que son semejantes para luego unificar los CDs que contienen términos parecidos.

La agrupación de términos semejantes, por si sola, requiere de un gran desarrollo lingüístico, por lo que las reglas que se consideraron para agrupar los términos son burdas y únicamente basadas en el contenido de las palabras de los términos y algunas consideraciones de POS. En la siguiente tabla se describen las reglas utilizadas para agrupar y filtrar términos semejantes:

Regla	**Descripción**
Reglas de filtrado	
Tamaño > 10 palabras	Los términos con más de 10 palabras se filtran.
Contiene al término buscado	Si los términos encontrados no contienen al que se ingresó de búsqueda.
Reglas de agrupamiento	
Artículo + término	Se agrupan términos que sean iguales, pero que uno contenga un artículo y el otro no.
Número o símbolo + término	Si el término presenta números o símbolos al inicio, los demás se agrupan.
Palabras funcionales + término	Se agrupan términos que contengan palabras funcionales al inicio o final y que contengan al término semejante.

Tabla 4.6 Reglas de filtrado y agrupamiento de términos semejantes

Un ejemplo de agrupamiento de términos se muestra en la siguiente figura:

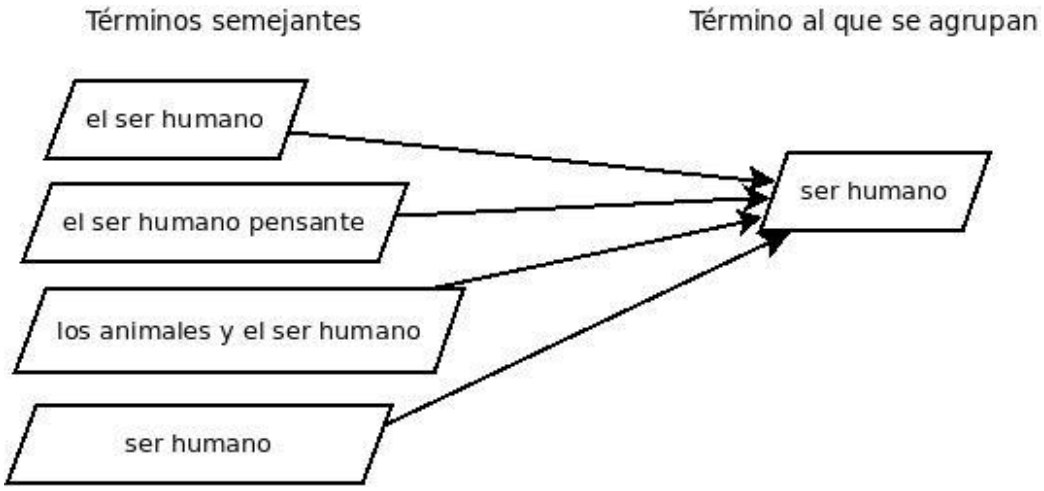

Figura 4.4 Ejemplo de agrupamiento de términos semejantes

Así quedan agrupados los términos semejantes y posteriormente se crea una estructura de datos que contendrá dichos términos agrupados y a los CDs que pertenecen a dicho término.

4.3.2 El hash de CDs

Una vez que se encontraron términos semejantes y se agruparon los contextos definitorios en torno al término que definen, se crea una estructura de datos que permite manipular los CDs y agregarles características, a la vez que se provee una interfaz para la salida de los datos.

La estructura de datos que se forma es un hash asociativo que permite ligar los arreglos de CDs con su término semejante, así como permite el acceso a los CDs que fueron limpiados y/o lematizados.

La estructura del hash de CDs y ejemplos se representan en la siguiente tabla:

Hash de CDs	
Llave: Término semejante	**Valor: Referencia a Hash con arreglo de CDs**
Humano	Puntero a Hash 1
Virus	Puntero a Hash 2
Hash 1	

Llave: CD con etiquetas completas	**Valor: Referencia a Arreglo de limpios-lemas**
<t>el ser humano</t> <pvd lema=ser><vd lema = ser> es </vd> </pvd> <d> un animal</d>	Puntero a arreglo 1
<d> Un ser pensante con un gran cerebro <pvd> es el <vd lema= denominar> denominado</vd> </pvd><t>ser humano</t>	Puntero a arreglo 2
Hash 2	
Llave: CD con etiquetas completas	**Valor: Referencia a Arreglo de limpios-lemas**
<t>un virus <pvd>está <vd lema = denominar> conformado</vd> </pvd><d> por una serie de proteínas.</d>	Puntero a arreglo 3
<d>al ser vivo más pequeño</d><pvd> se le <vd lema = denominar> denomina</vd></pvd> <t> virus.</t>	Puntero a arreglo 4
Arreglo 1	
[0] CD limpio de etiquetas	**[1] CD lematizado**
El ser humano es un animal	El ser hum es un anim
Arreglo 2	
[0] CD limpio de etiquetas	**[1] CD lematizado**
Un ser pensante con un gran cerebro <pvd> es el <vd lema= denominar> denominado</vd> </pvd><t>ser humano</t>	Un ser pens con un gr cerebr es el denom ser hum
Arreglo 3	
[0] CD limpio de etiquetas	**[1] CD lematizado**
Un virus está conformado por una serie de proteínas.	Un vir est conf por una ser de prot
Arreglo 4	
[0] CD limpio de etiquetas	**[1] CD lematizado**
Al ser vivo más pequeño se le denomina virus.	Al ser viv mas peq se le denom vir

Tabla 4.7 Estructura del Hash de CDs

De esta forma se conforma el hash permitiendo al usuario obtener únicamente los CDs cuyo término es semejante al término buscado, o bien una agrupación para su fácil lectura.

4.3.2.1 Limpieza y lematización de CDs

La estructura Hash de CDs al ser creada requiere procesamiento de los CDs para limpiarlos de sus etiquetas y prepararlos para ser usados, ya sea por los usuarios o por otros sistemas que no requieren de las etiquetas de los CDs.

Primeramente, el proceso limpia las etiquetas POS y las etiquetas de partes de los CDs dejándolos como texto limpio, lo cual es más legible para el usuario que sólo se interesa en los CDs y no en sus partes. Posteriormente a los CDs limpios se les aplica un proceso de lematizado que reduce el tamaño de las palabras colocando los lemas de ellas, esto genera que los CDs semejantes sean "medidos" o agrupados con mayor facilidad; además, se permite una salida apta para un mayor análisis lingüístico.

Para lematizar los CDs se utilizó el módulo de Perl "Lingua::Stem::Es" que permite lematizar palabras basado en el algoritmo de Porter.

La lematización se desarrolló en un principio en el Ecode por la necesidad de clasificación de CDs en el módulo consecutivo de Describe, que agrupa los contextos debido a su similitud y hace más adecuado procesarlos estando lematizados.

4.4. Implementación de Ecode

Una vez que se terminaron de construir los módulos de Ecode, fue necesario implementarlos para que los usuarios pudieran trabajar con el sistema; si bien la interacción con las entradas y salidas ya fue completada, era necesario proporcionar al usuario una interfaz con el sistema, además de conectarlo con el sistema Describe.

Las interfaces de usuario son dos: la interfaz web (WebCode) y una interfaz en consola estilo Unix.

4.4.1 Interfaz de usuario web de Ecode (WebCode)

El internet es la vía de transferencia de información más importante. En él se puede enviar todo tipo de datos y accesar aplicaciones Web que permiten una interacción con los usuarios, que pocas veces se obtiene con algún otro tipo de aplicación; esta fue la razón por la que se decidió realizar una interfaz web para Ecode y poder permitir el acceso inmediato de los usuarios.

WebCode se desarrolló en un principio para demostración del funcionamiento de Ecode; sin embargo, ese mostró mucho interés por tener una aplicación web abierta al público.

La aplicación cuenta con todas las características que permite Ecode: tiene una página de entrada en donde el usuario puede subir sus archivos a analizar al servidor o proporcionar una URL a procesar; y una página de salida que permite presentar claramente los CDs encontrados y proporciona herramientas de trabajo, como son el guardar los contextos de interés, obtener algún tipo de CD en específico o presentar los CDs limpios para su lectura o envío de resultados elegidos por correo.

La aplicación web, debido a los requerimientos de Ecode, se montó sobre el servidor que contiene a Describe como una aplicación independiente, aunque por el momento no es accesible para los usuarios pues se encuentra todavía en desarrollo.

4.4.2 Interfaz de consola

El Ecode se desarrolló en un principio como una aplicación de consola que no permitía más argumentos que el archivo de entrada, y no tenía ninguna opción de procesamiento. Se desarrolló, entonces, una interfaz de consola estilo Unix con opciones y argumentos.

La interfaz permite elegir el tipo de entrada y las salidas requeridas, además de contar con ayuda y detectar errores en la sintaxis del programa.

La sintaxis es:

Perl ecode2.pl –opciones Archivo_url_entrada [termino de interés]

En la siguiente tabla se muestran las opciones de la aplicación y su uso:

Opción	Descripción
Entradas:	
-a	La entrada es un archivo (txt, pdf, html, xml, doc)
-u -	La entrada es una URL (debe contener "http://" al inicio)
-i	Busca un término en específico que tiene ser el segundo argumento después de la entrada
Salidas:	
-e	Quita etiquetas de partes de CDs (incluye etiquetas de paso de información)
-p	Quita las etiquetas POS
-l	Archivo de CDs limpios sin POS y sin etiquetas de partes de CDs (CDs_limpios.txt)
-m	CDs con su texto lematizado (CDs_lematizados.txt)
-x	Todas las salidas: principal con etiquetas (depende de -e y -p); CDs limpios; CDs lematizados.
-t	Archivo de texto plano de entrada extraído de la fuente y separado por oraciones (texto_extraido.txt)
-g	Archivo con el texto de entrada y etiquetado con POS
Otros:	
-d	Debug (genera archivo de debuggeo)
-h	Uso del sistema
-s	Modo silencioso
Del formato de Salida: La salida es uno o varios archivos en el directorio /sal : + (CDs_final.txt) - Archivo conteniendo los CDs según fueron requeridos en la entrada + (CDs_lematizados.txt) - Archivo que contiene a los CDs lematizados [opcional] + (CDs_limpios.txt) - Archivo que contiene a los CDs limpios [opcional]	

+ (Texto_Extraido.txt) - Texto extraído de la fuente original [opcional] + (Texto_Extraido_POS.txt) - Texto extraído y etiquetado con POS [opcional]
Ejemplos: + Busca CDs en archivo.pdf (-a), limpio de etiquetas de partes de CDs (-e), sin etiquetas POS (-p), con archivo de texto extraído intermedio: **perl ecode2.pl -e -p -t -a archivo.pdf** + Busca CDs en URL http://es.wikipedia.org/wiki/Ling%C3%BC%C3%ADstica_computacional (-u) y busca el término "lingüística computacional" (-i) perl ecode2.pl -u –i “http://es.wikipedia.org/wiki/Ling%C3%BC%C3%ADstica_computacional" "lingüística computacional"

Tabla 4.8 Argumentos y uso de la interfaz de consola de Ecode

4.4.3 Integración a Describe

Si bien se pensaba que el sistema Ecode sería el motor de extracción de información del proyecto Describe, el sistema requería desarrollo para los requerimientos de Describe, por lo que ambos sistemas estaban desconectados y se tuvo que diseñar una interfaz que permitiera conectar a Ecode a las necesidades del Describe.

Para acoplar el Ecode al sistema Describe fue necesario hacer un análisis de las posibles formas de conectar los demás módulos de Describe escritos en Java con el Ecode escrito en Perl. Entonces, se consideró pasar la información en archivos y hacerlo por tuberías (*Pipes*). Para poder llevar a cabo un control más estricto con estadísticas en Describe, se decidió que fueran módulos independientes y se comunicaran a través de una base de datos.

Esta base de datos es un conjunto de tablas que permite mantener el control sobre los términos buscados, los documentos de dichos términos, el estado del procesamiento del término, los CDs de Ecode, además de los campos y tablas para los diferentes módulos, tanto funcionales como experimentales que lo componen.

La base de datos de Describe está montada con Mysql, por lo que se tuvo que utilizar el módulo estándar de Perl, DBI, que permite conectar a Perl con cualquier base de datos incluyendo a Mysql. El módulo que maneja la entrada y salida de Ecode con la base de datos de Describe se escribió en un archivo aparte (Archivo: ES_DB_describe.pm).

En la siguiente figura se muestra como interactúa Ecode con la base de datos de Describe:

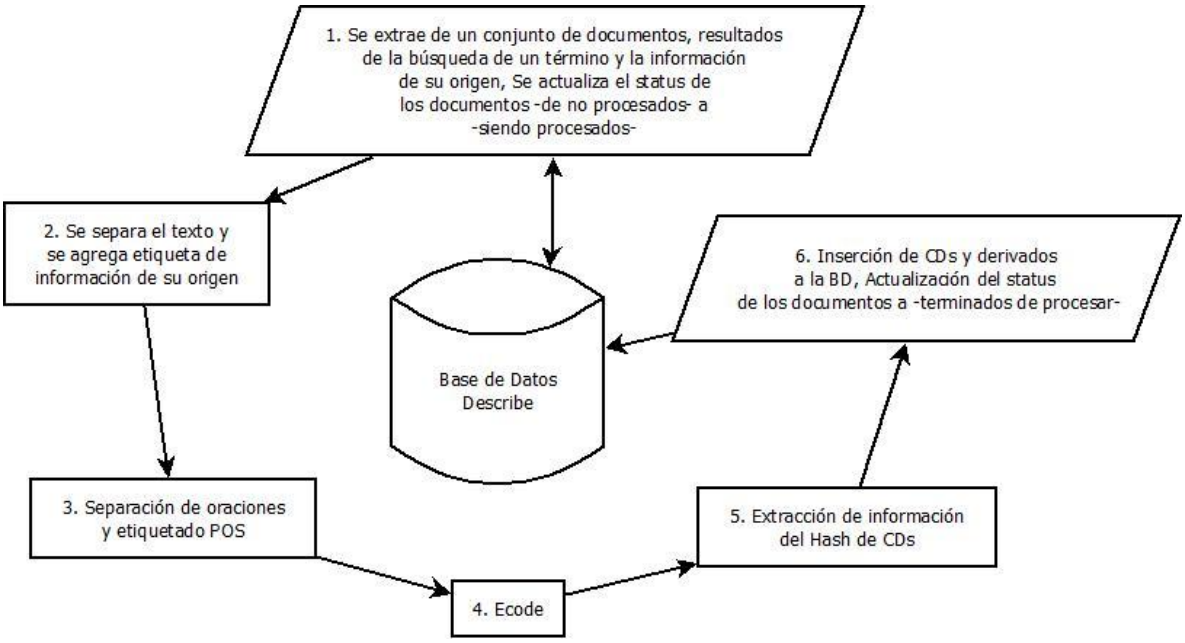

Figura 4.5 Interacción de Ecode con la base de datos de Describe

En la figura, el número uno representa la alimentación de Ecode con texto, que es una consulta para buscar qué documentos de un término o términos ya están listos para ser procesados. En esta consulta, además, se trae información del origen de los documentos, URL, término buscado, id de documento, etc. Además se inserta información en una tabla, indicando que los documentos serán procesados por Ecode con el tiempo, para llevar estadísticas y cálculo de resultados.

El número dos es sólo un preprocesamiento que permite añadir a cada oración una etiqueta que no será tocada por Ecode durante su procesamiento y permite pasar información sobre el origen de la línea, lo que finalmente permitirá saber de dónde se obtuvo un CDs.

Los números tres, cuatro y cinco corresponden en sí al Ecode: el tres es el módulo de entrada, el cuatro es el procesamiento principal y el cinco es la preparación de los CDs para sus diferentes salidas, es decir, es el módulo de salida.

La inserción final a la base de datos está dada por el número seis; en este paso se insertan los CDs, los CDs lematizados y los CDs limpios a la base de datos, decodificando la etiqueta de información e insertando los datos en la tabla correspondiente. Además, se marca el estado de los documentos procesados como terminados para que puedan ser atendidos por el módulo siguiente.