# Capítulo 4

# Diseño

El diseño del sistema sigue una estructura modular donde cada módulo corresponde con una de las fases de la metodología. Entonces los módulos de los que hablaremos a continuación serán: entrenamiento, evaluación, refinamiento de fórmulas y pruebas. Con excepción del refinamiento de fórmulas, cada módulo será dividido en tres módulos lógicos, como es típico de estructurar los proyectos en minería de textos; estos módulos lógicos son: **preprocesamiento**, **procesamiento** y **postprocesamiento**; los objetivos de cada uno de estos módulos se describen a continuación:

1. **Preprocesamiento**. Para procesar los textos es necesario que tengan una estructura más o menos uniforme, y es en esta etapa donde se aplican todos los algoritmos para limpiar el texto y estructurarlo.

2. **Procesamiento**. En esta etapa es donde se lleva a cabo el procesamiento de los textos para obtener conocimiento.

3. **Postprocesamiento**. Se mejorará la representación o el formato del conocimiento obtenido como salida o resultado.

## 4.1. Arquitectura del sistema

Basados en la definición anterior se tiene una vista global del sistema a implementar (Fig. 4.1). A continuación se describe la arquitectura del sistema, dejando de lado el módulo de refinamiento de fórmulas que no se va a implementar en esta ocasión.

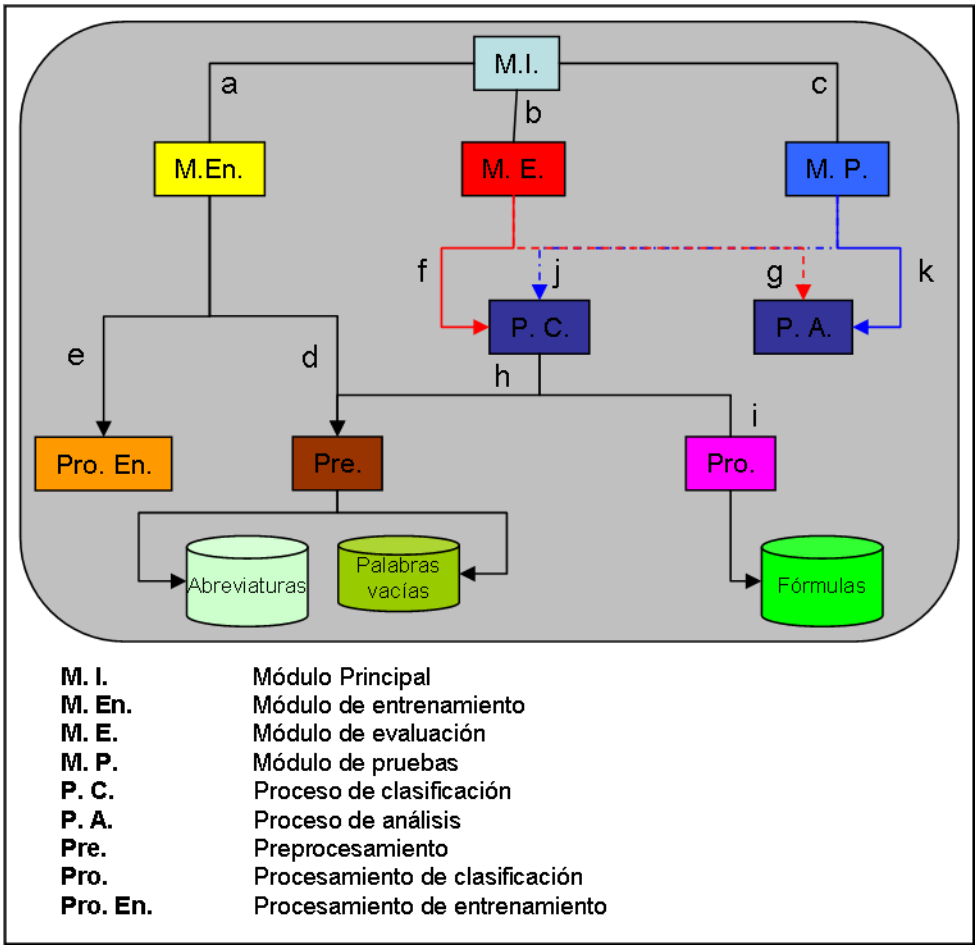


Figura 4.1: Vista modular del sistema

A continuación se describen cada uno de los módulos con sus casos de uso, y sus diagramas de flujo de datos.

| Entradas | Salidas |
|---|---|
| a. Localización del corpora de entrenamiento | Listas de fórmulas asociadas a cada uno de los dominios de entrenamiento |
| b. Localización del corpus de clasificación<br>Corpus de análisis | Corpus clasificado<br>Porcentaje de éxito |
| c. Localización del corpus de clasificación<br>Corpus de análisis | Corpus clasificado<br>Porcentaje de éxito |
| d. Texto | Lista de oraciones |
| e. Lista de oraciones | Lista de fórmulas asociadas a un dominio |
| f. Corpus de clasificación | Corpus clasificado |
| g. Corpus de análisis<br>Corpus clasificado | Porcentaje de éxito |
| h. Texto | Lista de oraciones |
| i. Lista de oraciones | Texto asociado a un dominio |
| j. Corpus de clasificación | Corpus clasificado |
| k. Corpus de análisis<br>Corpus clasificado | Porcentaje de éxito |

Figura 4.2: Parámetros de la arquitectura

## 4.2. Preprocesamiento

Como se puede ver en la figura 4.1, el preprocesamiento será el mismo para todos los módulos y a continuación se describirán las partes que lo conforman, para cada uno de estos se describen los casos de uso. Cómo está estructurado el módulo de preprocesamiento se puede ver en la figura 4.3.

### 4.2.1. Eliminar palabras vacías

Se tendrá una base de conocimiento con palabras vacías para cada idioma; basados en ésta y mientras se lleve a cabo el parser, se eliminarán las palabras que estén contenidas en el diccionario de palabras vacías (Véase Fig. 4.4 y 4.5).

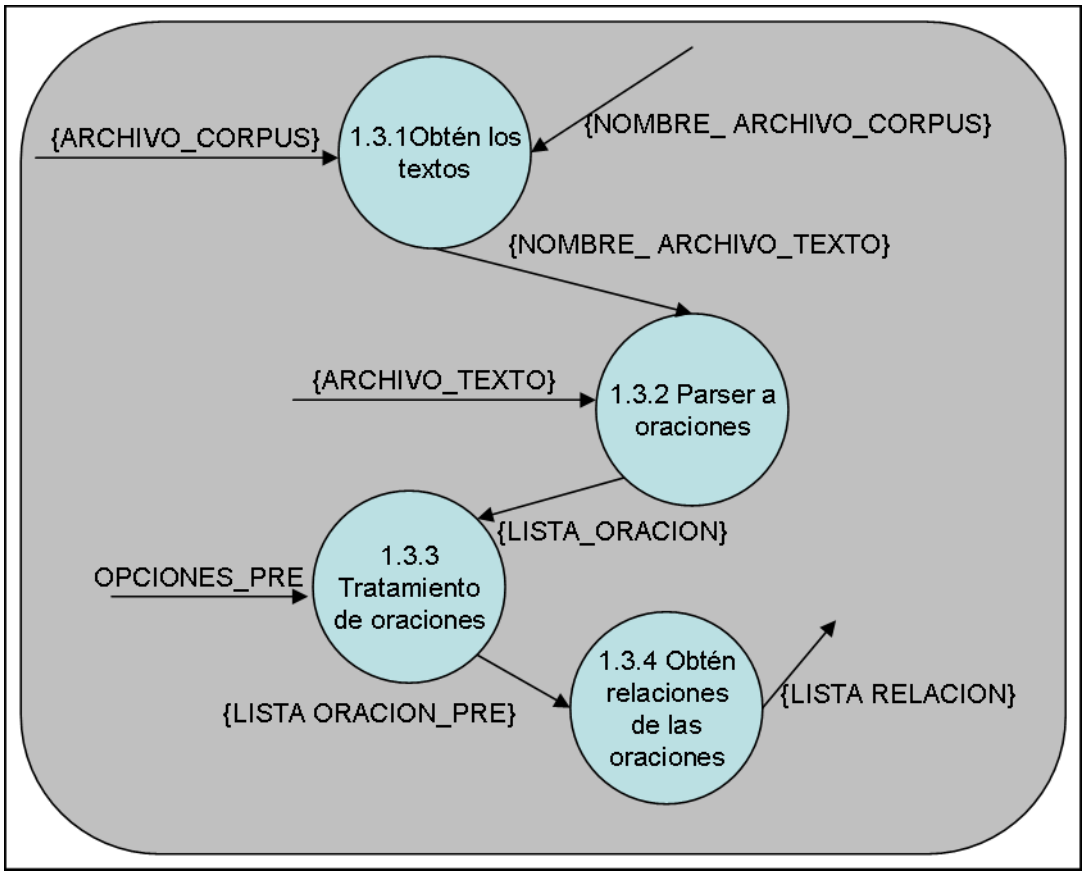


Figura 4.3: Preprocesamiento

| | |
|---|---|
| Caso de uso: | Eliminar palabras vacías |
| Autor: | Alejandra López Fernández |
| Última actuación: | 10/02/05 |
| Resumen: | El usuario desea realizar algún proceso sobre los textos y éste decide que necesita hacer pruebas con los textos eliminando las palabras vacías, el sistema busca las palabras vacías, las elimina. Al terminar este proceso, el usuario obtiene textos sin palabras vacías. |
| Actores: | Usuario |
| Pre-condiciones: | El usuario brindó los textos. |
| Post-condiciones: | El usuario obtiene los textos sin palabras vacías. |
| Disparador: | Eliminar de los textos palabras vacías que por lo general son muy frecuentes en los textos. |
| Proceso: | 1. El usuario selecciona la opción de eliminación de palabras vacías.<br>2. Los textos previamente procesados están representados como oraciones; cada oración se desglosa en palabras.<br>3. Cada palabra se busca en la base de conocimiento, previamente formada, donde se encuentran listadas las palabras vacías de un idioma.<br>4. Al ser identificada una palabra vacía, ésta es eliminada de la oración, reduciéndose el tamaño de ésta.<br>5. Al terminar el usuario obtiene oraciones que no contienen palabras vacías. |
| Excepciones: | |

Figura 4.4: Eliminar palabras vacías

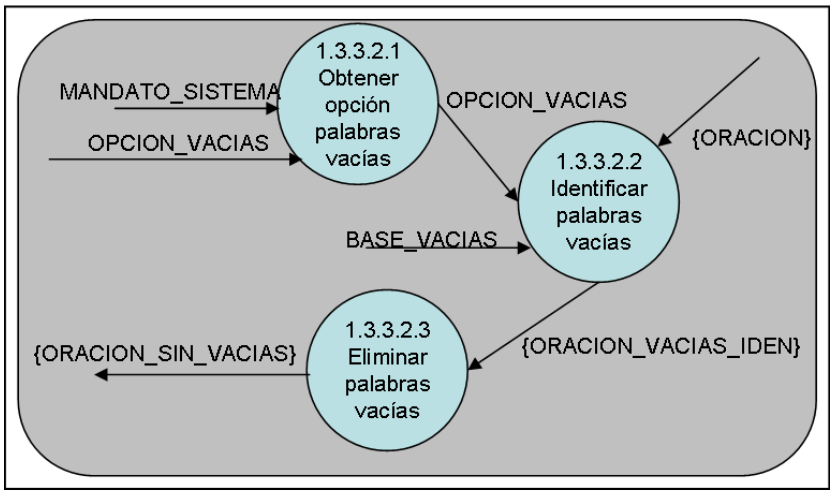


Figura 4.5: Eliminar palabras vacías

### 4.2.2. Aplicar reducción

La reducción a raíz léxica (*stemming*) se realizará mediante el famoso algoritmo de Porter. Se utilizará una versión de este algoritmo en español, que se obtuvo de una paquetería del ICT de la UDLA [12]. Tiene algunos errores, pero será de utilidad para realizar las pruebas. La versión en inglés también está disponible en Internet, existen varias versiones del algoritmo (Fig. 4.6 y 4.7).

### 4.2.3. Puntuación

#### Eliminar signos de puntuación

Los textos serán divididos en oraciones, así que los signos de puntuación son los que servirán de delimitadores para obtener las oraciones. Los signos de puntuación serán eliminados completamente de los textos, serán ignorados a través del parser.

| | |
|---|---|
| Caso de uso: | Aplicar algoritmo de reducción (*stemming*) |
| Autor: | Alejandra López Fernández |
| Última actualización: | 10/02/05 |
| Resumen: | El usuario desea realizar algún proceso sobre los textos y éste decide que necesita hacer pruebas con los textos reduciendo las palabras a sus raíces. Al terminar este proceso, el usuario obtiene textos cuyas palabras están reducidas a raíces léxicas. |
| Actores: | Usuario |
| Pre-condiciones: | El usuario brindó los textos. |
| Post-condiciones: | El usuario obtiene los textos con palabras reducidas. |
| Disparador | Eliminar de cada palabra sus inflexiones y obtener su raíz léxica. |
| Proceso: | 1. El usuario selecciona la opción de aplicar algoritmo de reducción (stemming).<br>2. Los textos previamente procesados están representados como oraciones; cada oración se desglosa en palabras.<br>3. Para cada palabra se ejecuta el algoritmo de porter, aplicando cinco pasos del algoritmo solamente.<br>4. Se sustituye la palabra original por la palabra reducida.<br>5. Al terminar el usuario obtiene oraciones que contienen palabras reducidas. |
| Excepciones: | |

Figura 4.6: Aplicar reducción

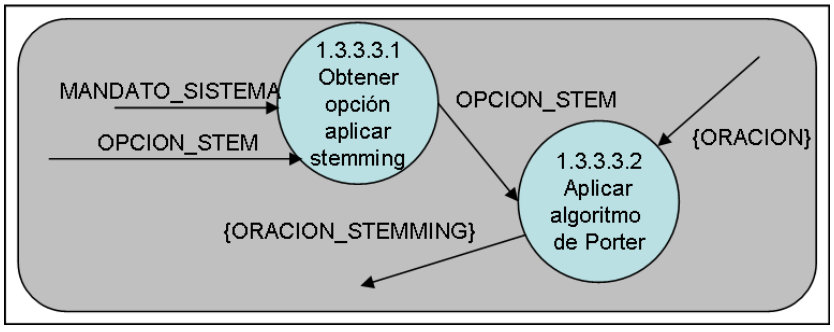


Figura 4.7: Aplicar reducción

| | |
|---|---|
| Caso de uso: | Sustituir abreviaturas |
| Autor: | Alejandra López Fernández |
| Última actualización: | 10/02/05 |
| Resumen: | El usuario desea realizar algún proceso sobre los textos y éste decide que necesita hacer pruebas con los textos eliminando las abreviaturas, el sistema encuentra las abreviaturas y las sustituye por sus valores. Al terminar este proceso, el usuario obtiene textos sin abreviaturas. |
| Actores: | Usuario |
| Pre-condiciones: | El usuario brindó los textos. |
| Post-condiciones: | El usuario obtiene los textos sin abreviaturas. |
| Disparador | Sustituir en los textos las abreviaturas por su significado. |
| Proceso: | 1. El usuario selecciona la opción de sustitución de abreviaturas.<br>2. El sistema al procesar el texto e identifica los puntos que separan las oraciones.<br>3. Al encontrar estos puntos, el sistema hace una consulta a la base de abreviatura<br>4. La abreviatura es sustituida en la oración por el valor que representa.<br>5. Al terminar el usuario obtiene oraciones que no contienen abreviaturas. |
| Excepciones: | |

Figura 4.8: Sustituir abreviaturas

### Identificar abreviaturas

El problema que se genera al obtener las oraciones a través de la puntuación es que pueden existir abreviaturas y podría haber confusión con el punto que está presente al final de cada abreviatura. Este punto podría ser confundido con el fin de una oración, es por ello que se debe discernir entre un punto que marca el final de una oración y un punto que es parte de una abreviatura (Véase Fig. 4.8 y 4.9). Para ello se tiene un diccionario de abreviaturas para cada idioma; basados en él se podrá identificar las abreviaturas y modificarlas por la palabra que ésta representa. Las abreviaturas guardadas en estas tablas son las más comunes y existe la posibilidad de incluir mucho más abreviaturas.

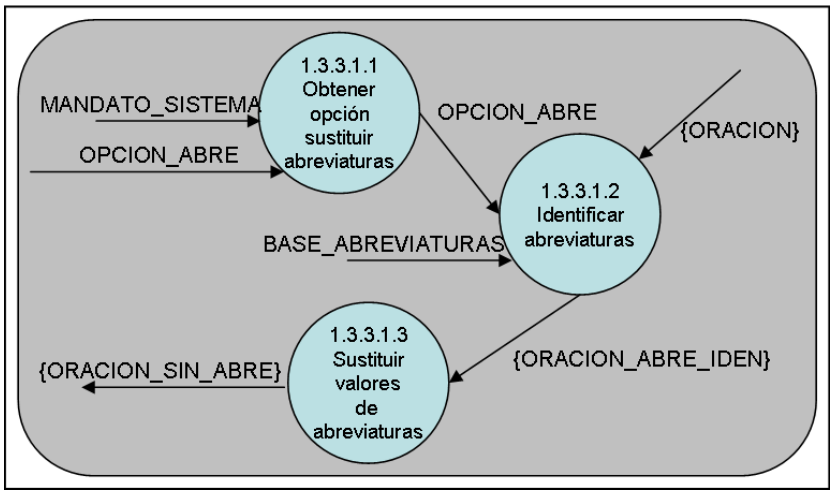


Figura 4.9: Sustituir abreviaturas

### 4.2.4. *Parser*

Esta funcionalidad sirve para dar al texto el formato de entrada para el algoritmo de reconocimiento de patrones, en este caso se realiza un parser de los textos a grafos. Como en este punto no se ha elegido el algoritmo final, tenemos dos *parsers* uno para cada algoritmo de reconocimiento de patrones que se está considerando. En forma general tenemos que se construye un grafo para cada dominio, es decir, un grafo para cada subcorpus que será la muestra de un dominio. Cada texto será dividido en oraciones y dependiendo de los requerimientos del algoritmo de reconocimiento de patrones será la salida final del *parser*.

## 4.3. Módulo de entrenamiento: procesamiento

Esta parte consiste de un algoritmo de reconocimiento de patrones, el algoritmo elegido es el algoritmo ad-hoc. Después de un profundo análisis se llegó a la conclusión que de con objeto de solucionar nuestro problema, nos brindaba mucho más la implementación del algoritmo ad-hoc. Se eligió este sobre el SUBDUE basándonos en las

características de cada uno. Entonces, se tenían dos candidatos que se consideraron para usarse en este módulo. Las opciones que se tenían eran: SUBDUE y un algoritmo ad-hoc, a continuación se describe cada uno y se incluye la discusión para la elección del más adecuado.

Esta parte consiste de un algoritmo de reconocimiento de patrones. Se tienen dos candidatos que se han considerado para usarse en este módulo. Las opciones que se tienen son: SUBDUE y un algoritmo ad-hoc, a continuación se describe cada uno y se incluye la discusión para la elección del más adecuado.

### 4.3.1. SUBDUE

#### Parser (SUBDUE)

Para este algoritmo los textos se tienen que dividir en oraciones, cada oración se representa con un subgrafo cuyos nodos son las palabras y los vértices son las relaciones entre estas palabras, esta relación está dada en función de la distancia con la que se relacionan las palabras que forman cada oración. Por lo tanto en este grafo se representan todas las oraciones que aparecen en un subcorpus, y para cada una se guardan las palabras que la forman y sus relaciones con otras palabras.

Entonces tenemos que para cada subcorpus o muestra de dominio se crea un archivo de texto que tiene la siguiente estructura: cada subgrafo inicia con un vértice, los vértices se describen como: $v < \# >< label >$ donde $< \# >$ es un identificador único para el vértice dentro del grafo y debe empezar en 1 e incrementar en uno sucesivamente, y $< label >$ es una cadena o número que representa el valor del vértice.

Una arista está definida como:

- $e < v1 >< v2 >< label >$,

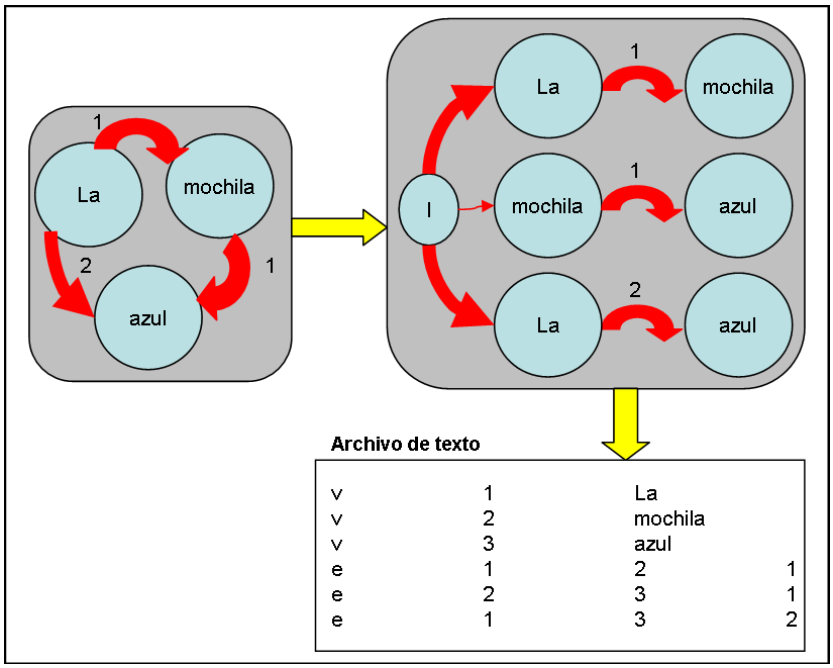


Figura 4.10: Archivo de entrada para SUBDUE

- $d < v1 >< v2 >< label >$,
- $u < v1 >< v2 >< label >$,

donde $< v1 >$ y $< v2 >$ son los identificadores de los vértices origen y destino y $< label >$ es una cadena o número que representa su valor. Las aristas pueden ser de tres tipos: '$e$' si se asume que son aristas dirigidas a menos que en la línea de comando se especifique lo contrario, en ese caso las que estaban especificadas con 'e'se convierten en no-dirigidas. Si empiezan con '$d$' entonces siempre serán dirigidas y si lo hacen con '$u$' siempre serán no dirigidas. Para más detalle de estas especificaciones consultar el manual [9]. Para que la descripción anterior sea mucho más clara supongamos que se tiene la siguiente oración "La mochila azul" en este caso el archivo que se obtiene como salida se puede ver en la figura 4.10.

Si además se tuviera la oración "La mochila verde" el archivo de salida quedaría como se muestra en la figura 4.11.

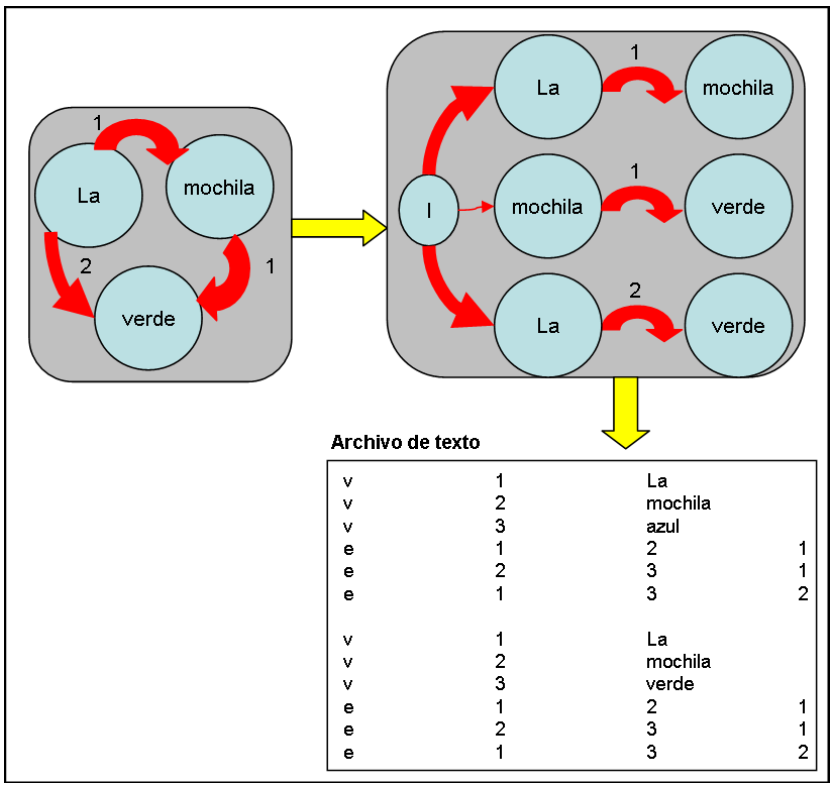


Figura 4.11: Archivo de entrada para SUBDUE

**Lógica de SUBDUE**

SUBDUE es un sistema ya implementado, requiere como entrada un archivo donde se encuentren descritos los grafos. Para ello se toma la salida del *parser* que corresponde a un archivo, éste servirá de entrada. Este software tiene definidos varios parámetros que nos ayudan a configurar la salida del sistema. Se hicieron experimentos exploratorios para ver el formato de salida que queremos o que nos será de mayor utilidad. A continuación se describen los comandos que fueron tomados en cuenta para configurar el procedimiento y la salida del sistema.

- *-iterations* # Es el número de iteraciones hechas sobre el grafo de entrada en donde la mejor subestructura de la iteración anterior es usada para comprimir el grafo, renombrándola en la siguiente iteración. Por defecto el valor de iteraciones es uno, lo que implica que no hay compresión del grafo.

- *-maxsize* # Este argumento especifica el máximo número de vértices que pueden estar en una subestructura. Por defecto se toma el número de vértices en la entrada del grafo.

- *-minsize* # Este argumento especifica el mínimo número de vértices que puede estar en una subestructura. Por defecto se toma uno.

- *-nsubs* # Este argumento especifica el tamaño máximo de la lista de las mejores subestructuras encontradas durante el descubrimiento. Por defecto el valor es tres. Existen otros comandos pero estos son los que fueron manipulados directamente por considerarse relevantes a nuestros experimentos, se pueden revisar con detalle en [9].

| | |
|---|---|
| Caso de uso: | *Parser* a oraciones |
| Autor: | Alejandra López Fernández |
| Última actualización: | 10/02/05 |
| Resumen: | El usuario desea realizar algún proceso sobre los textos y para ello se requiere que estos tengan la estructura adecuada para ser procesados por el algoritmo correspondiente. Al terminar este proceso, el sistema ha transformado textos en un conjunto de oraciones. |
| Actores: | Usuario |
| Pre-condiciones: | El usuario brindó los textos. |
| Post-condiciones: | El sistema ha transformado los textos en un conjunto de oraciones. |
| Disparador | Estructurar los textos de entrada para que sean procesados. |
| Proceso: | 1. El usuario selecciona la opción de procesar textos.<br>2. Utilizando los signos de puntuación se separa cada texto en oraciones.<br>3. Al terminar se obtiene una lista de oraciones que describen a los textos. |
| Excepciones: | |

Figura 4.12: Parser a oraciones

### 4.3.2. Algoritmo AD-HOC

#### Parser (algoritmo AD-HOC)

Para este algoritmo es suficiente tener separados los textos en oraciones (Véase Fig. 4.12) para posteriormente obtener las relaciones entre estas.

#### Lógica del algoritmo AD-HOC

Se diseñó un algoritmo híbrido que combina algunas características del funcionamiento de SUBDUE y del algoritmo de colocación, descritos en el capítulo de marco teórico. El algoritmo tiene dos etapas principales: el análisis de las relaciones y la construcción de fórmulas. Entonces la entrada de este algoritmo son textos representados con oraciones; con cada oración se obtienen las relaciones entre las palabras que la forman; por ejemplo para la oración *"La mochila azul"* obtenemos las relaciones de esta oración (Fig. 4.13).

El proceso de la obtención de estas relaciones se describe en la Fig. 4.14.

Posteriormente con todas las relaciones obtenidas se construye un grafo dividido en

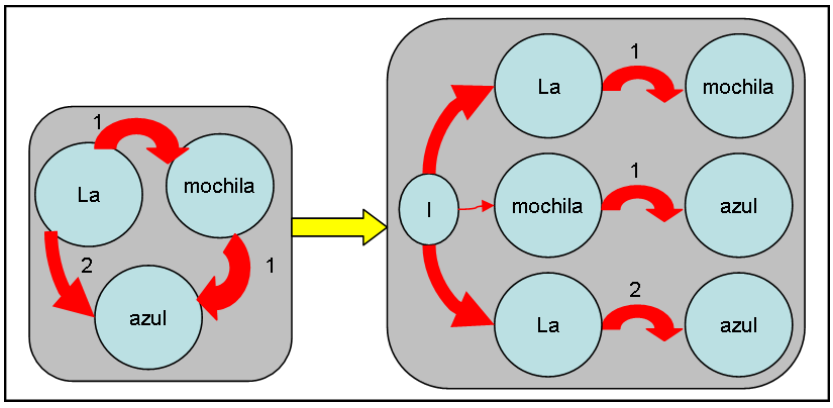


Figura 4.13: Obtención de relaciones

| | |
|---|---|
| Caso de uso: | Obtener relaciones |
| Autor: | Alejandra López Fernández |
| Última actualización: | 10/02/05 |
| Resumen: | El usuario desea obtener las relaciones entre las oraciones obtenidas de los textos brindados por él. Para cada oración se obtiene la adyacencia entre las palabras que la forman. Así se construye un grafo que contiene a todas las relaciones sin repeticiones. |
| Actores: | Usuario |
| Pre-condiciones: | El sistema obtuvo las oraciones de los textos. |
| Post-condiciones: | Se han obtenido las relaciones de las oraciones, construyendo un grafo con ellas. |
| Disparador | Se transforman las oraciones a grafos, basándose en las relaciones que existen entre las palabras que las forman. |
| Proceso: | 1. El sistema obtuvo las oraciones que forman a los textos.<br>2. Utilizando la lista de oraciones, se determina la adyacencia de las palabras que forman la oración y así se obtienen las relaciones de cada palabra con el resto.<br>3. Al terminar se obtiene un conjunto de grafos que representan las relación entre dos palabras, con este conjunto de grafos se construye uno más grande que no contiene repeticiones de la misma relación. |
| Excepciones: | |

Figura 4.14: Obtener de relaciones

subgrafos; cada uno de estos subgrafos tiene dos nodos que representan palabras, unidos por un vértice. Además se describe la relación que existe entre éstas, la cual está dada por la distancia que existe entre ellas en la oración. Dentro del grafo aparecerá sólo una vez cada relación, es decir, si agregáramos la oración *"La mochila verde"* las únicas relaciones que se agregarían serían las relaciones que existen entre "mochila" y "la" con "verde" (Véase Fig. 4.15).

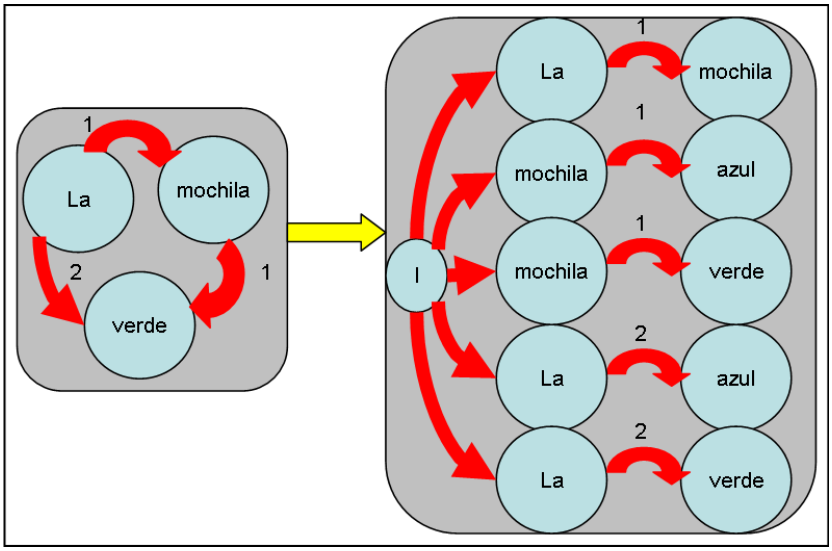


Figura 4.15: Obtención de relaciones

Las repeticiones, la representatividad, la carga semántica y el peso (conceptos definidos en el capítulo anterior) de cada relación también estarán guardados. Estos valores ayudarán y servirán como criterio para filtrar y ordenar las fórmulas de tamaño dos que resultan de esta representación (Fig. 4.16).

### 4.3.3. Elección del algoritmo

Para la elección del algoritmo se llevaron a cabo experimentos exploratorios. SUBDUE tiene como atractivo principal que ya se encuentra implementado y se han hecho varios trabajos que respaldan su eficacia, la lógica de SUBDUE, en teoría, es la forma

| | |
|---|---|
| Caso de uso: | Construir fórmulas |
| Autor: | Alejandra López Fernández |
| Última actualización: | 10/02/05 |
| Resumen: | El usuario desea obtener la lista de fórmulas que describen un dominio. Con las relaciones obtenidas se calculan los valores como el peso, representatividad, carga semántica, etc. Al terminar este proceso, hay una transformación de relaciones a fórmulas. |
| Actores: | Usuario |
| Pre-condiciones: | Se han obtenido las relaciones de los textos. |
| Post-condiciones: | Se ha obtenido la lista de fórmulas que son representativas a un dominio. |
| Disparador | Se transforman la lista de las relaciones en fórmulas; el concepto de fórmulas incluye valores como el peso, representatividad y carga semántica. |
| Proceso: | 1. El usuario selecciona la opción de procesar textos.<br>2. Utilizando la lista de relaciones, obtenida de los textos que el usuario da de entrada, se obtiene para cada una de estás su carga semántica, su representatividad, su longitud y con ello su peso.<br>3. Al terminar se obtiene una lista de fórmulas que incluyen las más representativas del dominio, es decir, las que tienen un peso mucho mayor. |
| Excepciones: | |

Figura 4.16: Construir fórmulas

más natural de solucionar nuestro problema. Por otro lado, nos valemos de variables que tienen la función de restringir el espacio de resultados, pero que pueden influir de manera negativa, porque se pueden desestimar algunos casos que son relevantes. Tal es el caso de las variables *minsize* y *maxsize* que hacen muy difícil manejar fórmulas de longitud variable. Estos parámetros limitan el rango dentro del cual se encuentran el tamaño de las fórmulas buscadas; debido a que no se tiene la certeza de cuál es el rango del tamaño ideal, pueden no tomarse en cuenta algunas fórmulas que se encuentran fuera del rango definido pero son relevantes al dominio. Algo similar sucede con la variable *nsubs* que restringe el tamaño de la lista de subestructuras encontradas en cada iteración. Una posibilidad es que en una iteración una subestructura sea desestimada porque no tiene un valor suficientemente algo para ser incluido en la lista, pero si ésta se siguiera propagando podría tomar importancia en una iteración posterior. Además, incluso si hay un buen manejo de las variables, los resultados obtenidos no tienen una estructura que pueda ser interpretada directamente, se necesita realizar postprocesamiento para realizar el análisis de los resultados.

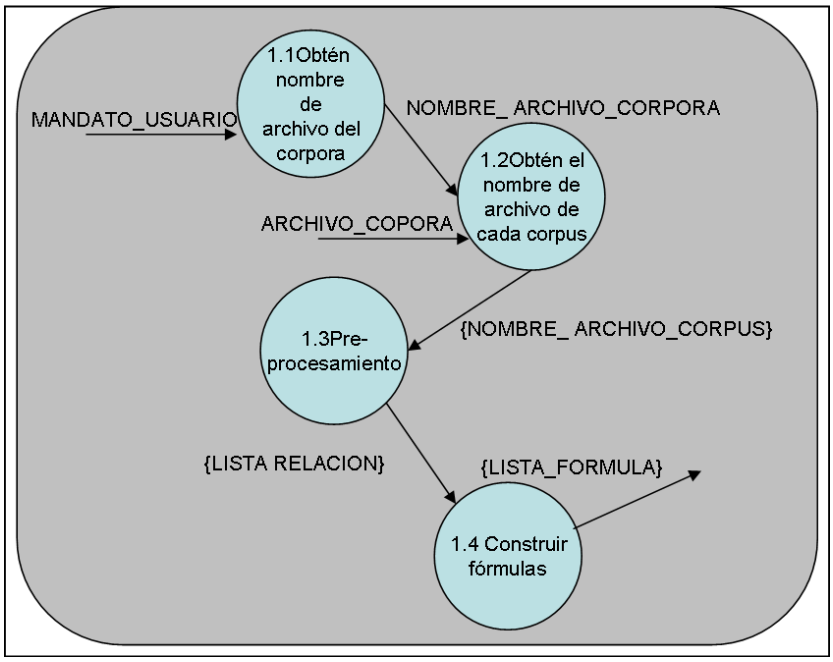


Figura 4.17: Módulo de entrenamiento

El control que se tiene en el algoritmo ad-hoc es total, ésta es la principal ventaja debido a que fue diseñado para resolver este problema. Por otro lado, aunque el algoritmo ad-hoc maneja fórmulas de tamaño dos, se pueden obtener fórmulas más grandes. Lo anterior debido a que se conservan todas las relaciones asociadas a cada palabra, con estas se pueden reconstruir las fórmulas colocando cada palabra en la posición que le corresponde, así se tienen fórmulas de cualquier tamaño siempre y cuando su relación sea fuerte o representativa.

Finalmente, se puede ver en la figura 4.17 el funcionamiento completo de este módulo.

## 4.4. Módulo de evaluación

Para la comprensión de este módulo, lo mejor será dividirlo en sus dos partes principales (Fig. 4.1), éstas son: **clasificación** y **análisis**; a continuación se describe cada

| | |
|---|---|
| Caso de uso: | Determinar índice de similitud |
| Autor: | Alejandra López Fernández |
| Última actualización: | 10/02/05 |
| Resumen: | El usuario desea obtener la similitud entre un texto y un dominio. El texto se transforma a oraciones y a relaciones; en este texto se buscan las fórmulas que describen a un dominio y se realiza el cálculo del índice de similitud de la forma en que fue definido en el capítulo anterior. Al terminar este proceso, se obtiene el índice de similitud entre el texto y el dominio. |
| Actores: | Usuario |
| Pre-condiciones: | El texto se ha dividido en oraciones y se han obtenido sus relaciones. |
| Post-condiciones: | Se ha calcula el índice de similitud entre un texto y un dominio. |
| Disparador | Se compara un texto con la lista de fórmulas de un dominio y se determina que tan parecidos son. |
| Proceso: | 1. El usuario selecciona la opción de clasificar textos.<br>2. Cada texto es transformado a oraciones y se obtienen las relaciones en las oraciones.<br>3. Cada relación obtenida en el texto se busca en la lista de fórmulas que se encuentra asociada a cada uno de los dominios.<br>4. Aplicando la fórmula del 'índice de similitud, se calcula su valor.<br>5. Al terminar se obtiene una lista de 'índices de similitud entre cada texto y cada dominio. |
| Excepciones: | |

Figura 4.18: Calcular índice de similitud

parte.

### 4.4.1. Clasificación

Esta parte se divide también en preprocesamiento, procesamiento y postprocesamiento. El preprocesamiento es el mismo proceso que el empleado en el resto de los módulos.

#### Procesamiento

En esta etapa se utiliza un algoritmo de clasificación; para este algoritmo se necesita que los textos estén convertidos en oraciones. Una vez que cada uno de los textos es parseado a oraciones, se obtienen las relaciones entre las palabras que las forman y se forman fórmulas de tamaño dos; cada una de las fórmulas en las que es convertido el texto se busca en la lista de fórmulas de cada uno de los dominios y se calcula el

| | |
|---|---|
| Caso de uso: | Elegir dominio |
| Autor: | Alejandra López Fernández |
| Última actualización: | 10/02/05 |
| Resumen: | El usuario desea obtener el dominio al que pertenece un texto. Se tiene una lista de índices de similitud asociadas a este texto con todos los dominios; se elige el mayor que es el que mejor lo clasifica y que es lo suficientemente contundente (mayor al 50\%). Al terminar el proceso, se obtiene la lista de textos asociados a un dominio, o en su defecto a ninguno de los dominios conocidos. |
| Actores: | Usuario |
| Pre-condiciones: | Se ha calculado los índices de similitud de cada texto con todos los dominios. |
| Post-condiciones: | Todos los textos se han asociado a un dominio, o en algunos casos a ninguno. |
| Disparador | Para cada texto se elige el dominio cuyo índice de similitud fue mayor y lo suficientemente representativo para ser considerado. |
| Proceso: | 1. El usuario selecciona la opción de clasificar textos.<br>2. Se tienen la lista de índices de similitud que asocian un texto con cada uno de los dominios.<br>3. Los índices de similitud se ordenan de manera ascendente.<br>4. Se elige al que encabeza la lista y se comprueba si es mayor al 50\%, si es así este dominio será elegido como el que lo clasifica mejor.<br>5. Al terminar se obtiene una lista de textos asociado a un dominio. |
| Excepciones: | |

Figura 4.19: Elegir el dominio

índice de similitud (Véase Fig. 4.18) entre el texto y cada dominio (el cálculo se lleva a cabo como se definió en el capítulo anterior). Con cada uno de los índices de similitud entre un texto y cada una de la lista de fórmulas se elige a la que la clasifica mejor, es decir al índice más grande. Así se determina la pertenencia de cada texto a uno de los dominios o a ninguno de ellos (Fig. 4.19); así para clasificar se identificaron dos funciones principales: calcular el índice de similitud y elegir el dominio.

### 4.4.2. Análisis

Esta parte tiene la función de evaluar el funcionamiento del algoritmo de clasificación y además analizar si la lista de fórmulas es lo suficientemente representativa del dominio. Para llevar a cabo este análisis, se cuenta con la clasificación de los textos correcta, es decir una relación de los textos y cada uno de ellos asociado al dominio al que pertenece. Con esta relación y con el resultado del proceso de clasificación, se realiza una comparación y se determina el porcentaje de éxito (definido en el capítulo anterior).

| | |
|---|---|
| Caso de uso: | Calcular el porcentaje de éxito |
| Autor: | Alejandra López Fernández |
| Última actualización: | 10/02/05 |
| Resumen: | El usuario desea obtener y evaluar el comportamiento del proceso de clasificación. Se tiene una lista de textos clasificados por el algoritmo y además se debe tener, antes de procesar el corpus, la clasificación correcta de éste. Se comparan estás dos clasificaciones. Al terminar se tiene el porcentaje de textos clasificados correctamente por el algoritmo. |
| Actores: | Usuario |
| Pre-condiciones: | El sistema tiene una lista de textos clasificados y la lista de esos textos clasificados correctamente. |
| Post-condiciones: | El usuario tendrá el porcentaje de efectividad del algoritmo de clasificación. |
| Disparador | Para la lista de textos clasificados obtenidos en la etapa de clasificación se determina cuántos fueron clasificados correctamente. |
| Proceso: | 1. El usuario selecciona la opción de analizar los resultados.<br>2. Se tienen la lista de textos clasificados por el algoritmo y la lista de los textos clasificados correctamente (antes de que se procesará el corpus).<br>3. Se comprarán las dos clasificaciones y se determina el porcentaje de textos que fueron clasificados correctamente.<br>4. Al terminar se obtiene el porcentaje de éxito. |
| Excepciones: | |

Figura 4.20: Obtener porcentaje de éxito

Con este porcentaje de éxito se realiza la evaluación de éste módulo, el cálculo de esta variable es la función principal de la etapa de análisis (Véase Fig. 4.20).

En la figura 4.21 se muestra el diagrama de flujo de datos de este módulo.

## 4.5. Módulo de pruebas

Este módulo utiliza el mismo procedimiento del módulo de evaluación.

## 4.6. Modelo de datos

Durante la descripción de las funcionalidades y la descripción del flujo de los datos, no se describió formalmente la definición de estos datos. En el apéndice A se agregó un diccionario de datos correspondiente y así se puede entender mejor los diagramas de flujo de datos (figuras 4.3, 4.5, 4.7, 4.9, 4.17 y 4.21).

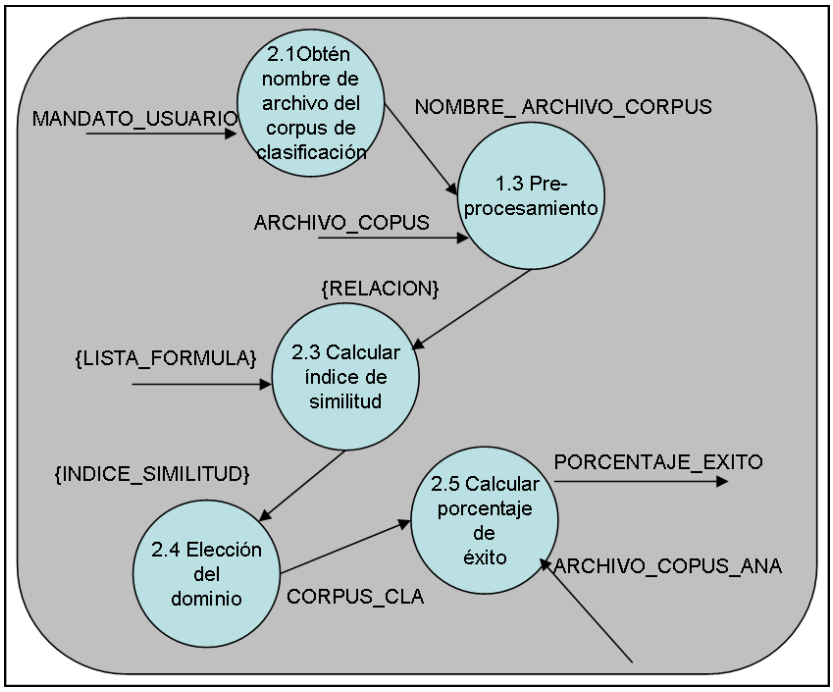


Figura 4.21: Sustituir abreviaturas

Los casos de uso que fueron descritos antes, sirvieron de base para describir las funcionalidades del sistema y describir cómo fluyen los datos, es decir, la interacción del sistema. Esto nos abre brecha para empezar la implementación.