Diseño Detallado de Software

Entrega II

Módulo de Inteligencia Artificial

Integrantes:

- Pablo Lluch
- Daniel Merrill

● Descripción, haciendo énfasis en la responsabilidad del componente dentro del programa. El alcance del componente debe justificarse desde los cuatro principios fundamentales del diseño detallado.

Descripción general:

El módulo de Inteligencia artificial toma como *input* las palabras extraídas de las fuentes por el módulo de parsing, y entrega dos *output*, uno es un listado con las palabras más relevantes, y el otro es un listado de las fuentes más relevantes. Se puede concluir entonces, que la responsabilidad del módulo en el programa es la de **tomar decisiones** y **organizar la información** de forma que esté lista para ser usada por los visualizadores del programa.

Input y su procesamiento:

El input hacia el módulo será un texto en formato JSON. Este JSON contendrá toda la información extraíble de las fuentes originales. El esquema del JSON es el siguiente:

```
[
        {
                author: "autor", // Opcional
                date: "fecha", // Opcional
                content: "noticia, twitt, cualquier contenido",
                source:
                        {
                                url: "ruta de la fuente",
                                type: "emol, twitter, gobierno_de_chile, etc",
                                extras: "cualquier otro parámetro que se quiera entregar, por ejemplo, el número de followers si es un twitt" // Opcional
                        }
        },
        ...
```

]

Este JSON será entregado al método *get_words_by_relevance* y *get_source_elements_by_relevance*. El primero llamará a los métodos necesarios para obtener un arreglo con las palabras y sus valores de *relevancia*, mientras que el segundo lo hará para obtener un arreglo con la información original, pero con los puntajes de *relevancia* agregados.

Análisis del componente:

Este análisis se basó en la estructura del componente, que se puede observar en el diagrama de clases de más adelante. Al diseñar este componente, quisimos enfocarnos en crear un programa fácil de usar, eficiente, y fácil de modificar. Para esto, tomamos las siguientes decisiones de diseño:

1- La puerta de entrada al componente será una sola interfaz, que usará los componentes creados (componentes con nivel medio - alto de abstracción), y expondrá solo dos métodos específicos que guardan relación con el problema que se nos plantea, ordenar fuentes y palabras según relevancia (nivel bajo de abstracción). El principio de diseño de software que se ve afectado es claramente el de **abstracción**, y en segundo lugar el de **bajo acoplamiento**, pues la dependencia con otros módulos será solo a través de estos dos métodos y del formato del JSON de entrada.

2- Al momento de elegir qué algoritmo usar para calcular las relevancias de tanto palabras como fuentes, se podrá elegir cualquier implementación, mientras esta extienda la clase que resuelve el problema de forma genérica. Esto nos permite extender el programa fácilmente para soportar nuevos algoritmos, y nos permite modificar el comportamiento de un algoritmo en particular sin afectar todo el resto del programa. Los principios que se relacionan son nuevamente el de **abstracción**, y el de **ocultamiento**, pues para la clase utilizadora del algoritmo es irrelevante la forma en que se lleve a cabo el algoritmo, actúa como caja negra.

3- El módulo, internamente, actúa de una forma secuencial. A partir del método inicial de la interfaz de entrada, cada clase va llamando a métodos de otras clases que ven el problema de una forma cada vez más específica. Esto se traduce en una **alta cohesión**, pues cada clase opera sobre un mismo objeto y lo entrega, formando parte de una especie de cadena de ensamblaje. Cada clase del sistema está relacionada con la siguiente clase en la línea de ensamblaje.

Output del componente:

Una vez calculados los puntajes de relevancia de las palabras o de la fuente, se entregará un JSON de la siguiente forma:

JSON de palabras:

```
[
        {
                word: "palabra1",
                relevance: 100
        },
        {
                word: "palabra2",
                relevance: 95
        },
        …
]
```

JSON de fuentes:

```
[
        {
                author: "autor", // Opcional
                date: "fecha", // Opcional
                content: "noticia, twitt, cualquier contenido",
                relevance: 100,
                source:
                        {
                                url: "ruta de la fuente",
                                type: "emol, twitter, gobierno_de_chile, etc",
                                extras: "cualquier otro parámetro que se quiera entregar, por ejemplo, el número de followers si es un twitt" // Opcional
                        }
        },
        ...
]
```

● Diagrama de clases del componente y dos diagramas de secuencia de procesos clave del componente. Indicar claramente al menos 3 patrones de diseño que surjan de este análisis.

Diagrama de Clases:

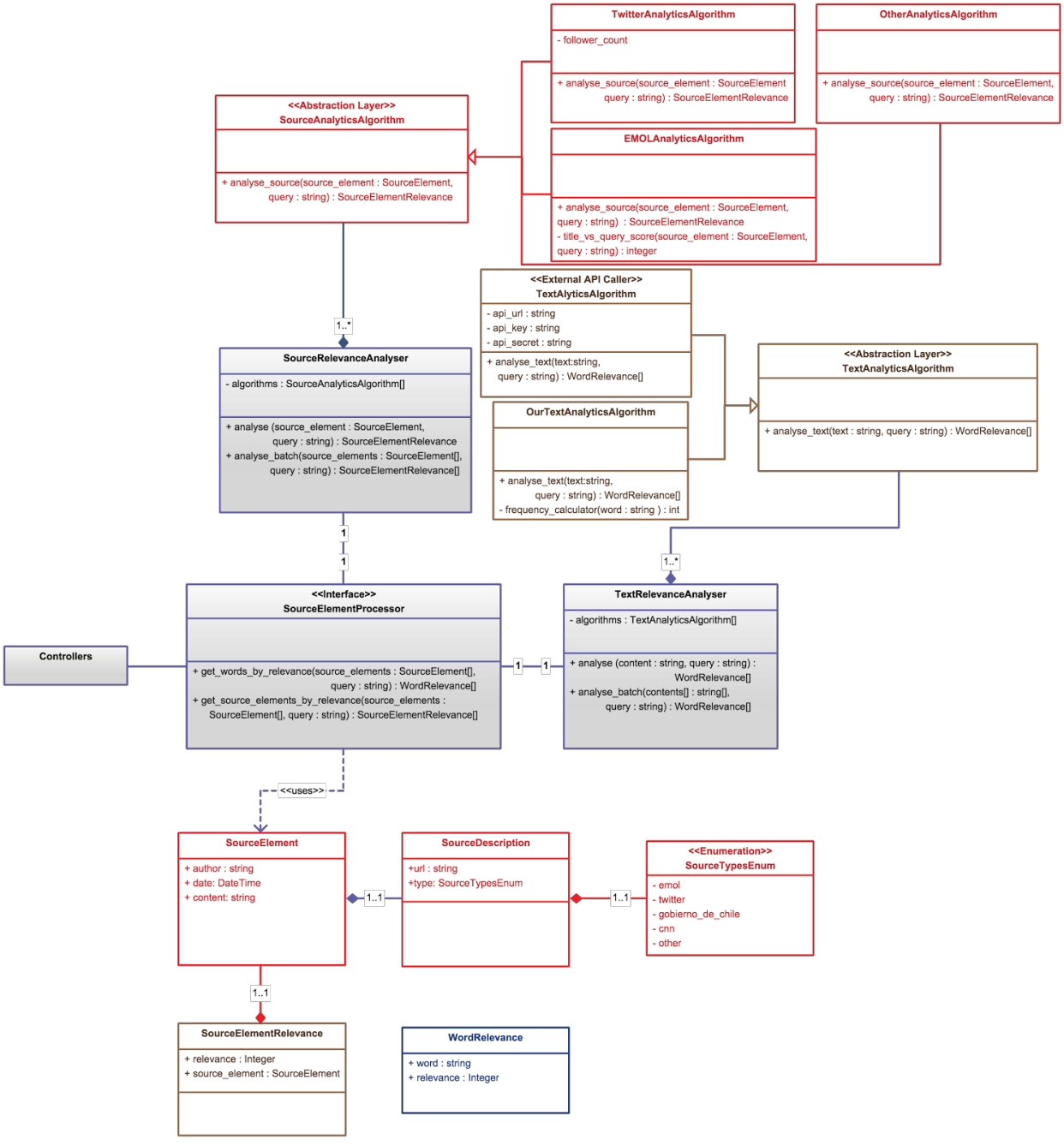


Diagrama de Secuencia 1:

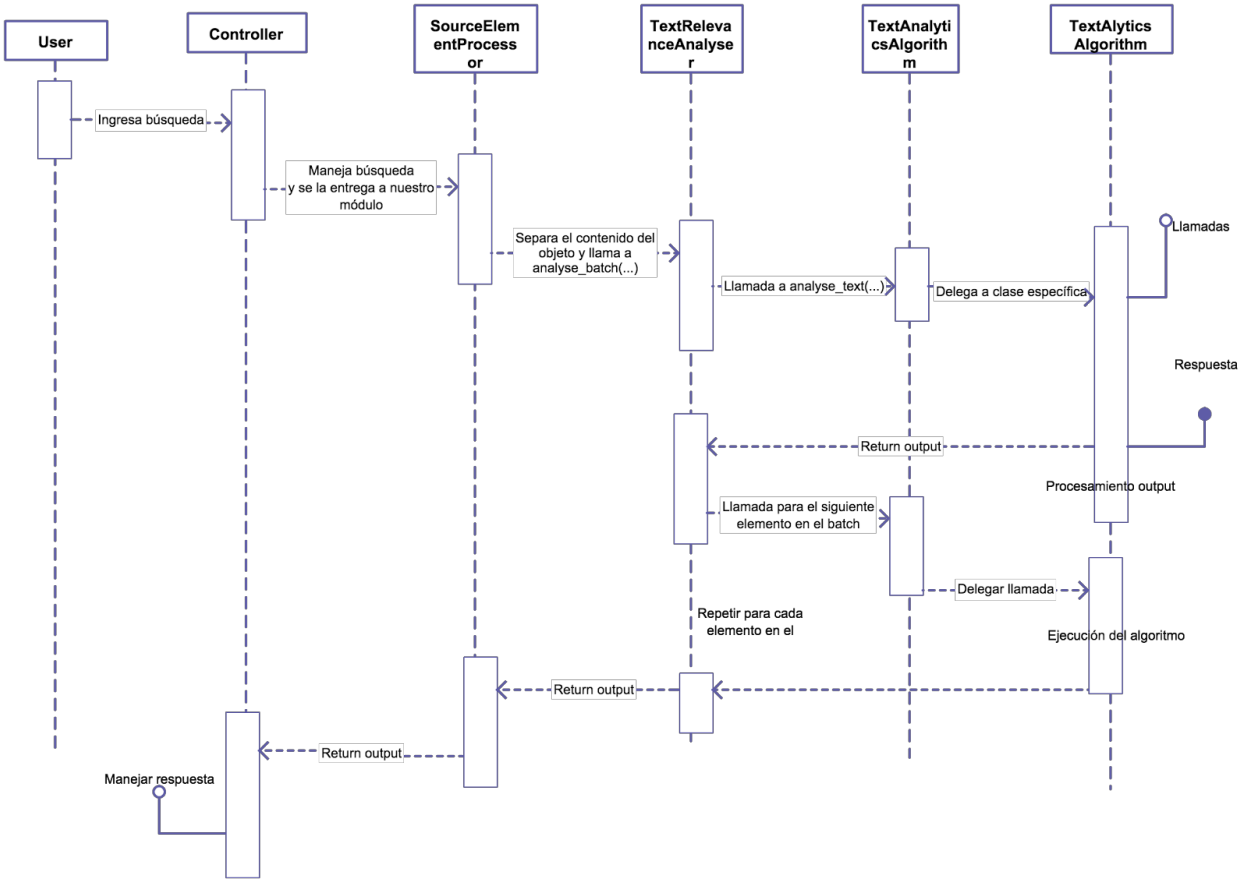
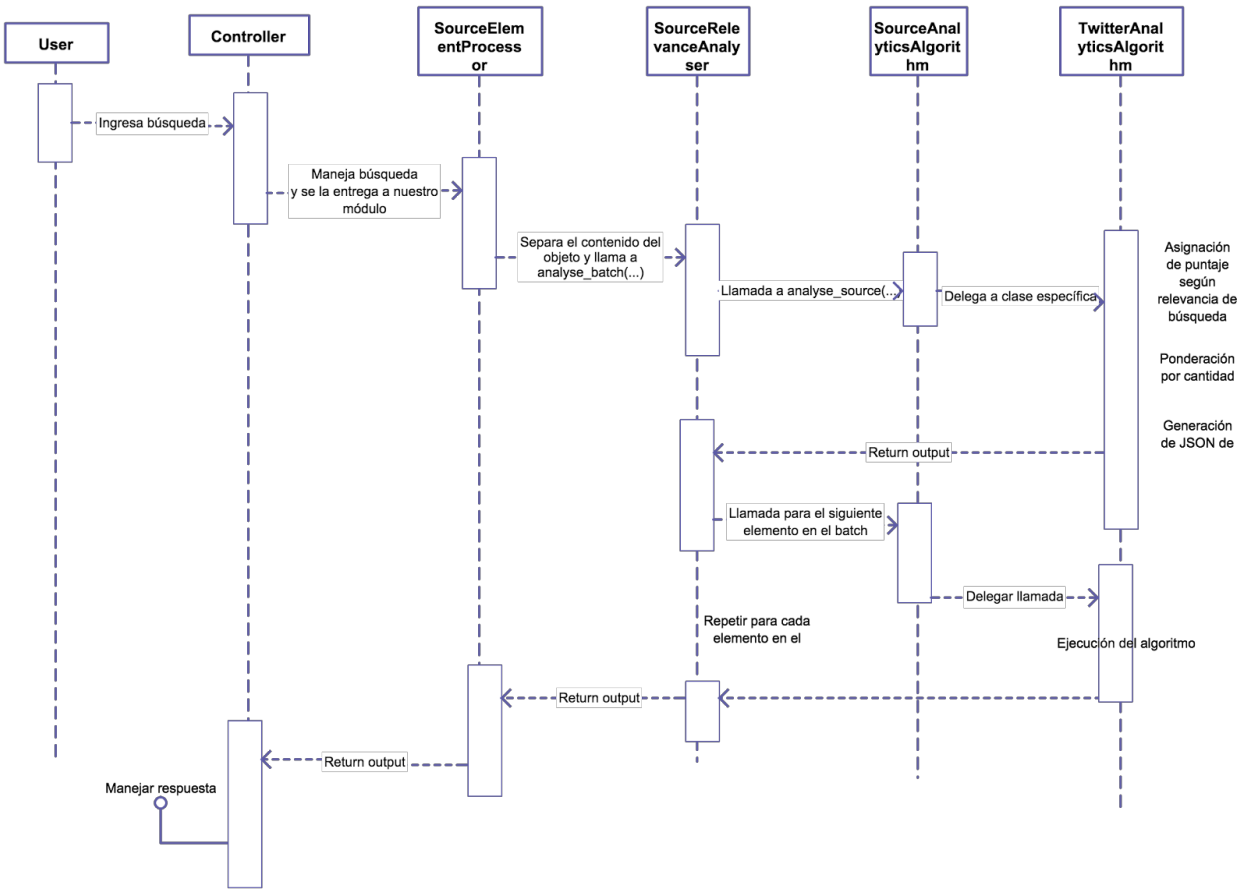


Diagrama de secuencia 2:



Patrones de Diseño

En el módulo de Inteligencia Artificial se utilizan los siguientes patrones GoF:

- Strategy - Este patrón permite definir una familia de algoritmos, encapsulando a cada uno haciéndolos intercambiables con una interfaz común. Específicamente, nosotros queremos poder tener diversos algoritmos de análisis de fuentes y textos. Analizar el contenido de un tweet es diferente a analizar una noticia en EMOL, pero las firmas de los métodos son las mismas. Para implementarlo, se definió una clase **source_analyzers/base.rb**, de la cual heredan las clases definidas en **emol.rb** y **twitter.rb**. Adicionalmente, se definió una clase SourceRelevanceAnalyser, la cual tiene como uno de sus atributos a una de las estrategias definidas.

- Adapter - Patrón utilizado para encapsular una clase existente y acomodar sus métodos a una interfaz requerida. En este caso en particular, hemos decidido hacer un wrapper

para las llamadas HTTP que sea simple y extensible, el cual se encuentra en **lib/http/client.rb**. A través de esta clase, queremos encapsular los métodos de alguna librería HTTP ruby (como Faraday), teniendo la posibilidad de elegir otra en caso de necesidad.

- Además de los patrones GoF mencionados, se hace un uso claro de los siguientes principios SOLID:
    - Single responsibility principle - Clases como SourceElementProcessor tienen una funcionalidad clara y definida, en este caso, obtener palabras o elemento de una fuente segun su relevancia.
    - Dependency inversion principle - Dependemos de abstracciones, no implementaciones. Se usan interfaces abstractas como clases bases para los analizadores en vez de depender de una clase concreta.

- Vale la pena mencionar que la naturaleza misma de ruby permite evitar muchos problemas de dependencia en un cierto tipo de objeto (pues las variables no tienen tipo), y extender de forma natural cualquier clase o modulo existente.