TALLER 3

Aplicación de información geográfica

DANIELA VILLEGAS OSORIO
VALENTINA LENIS CAICEDO
SEBASTIAN LUNA REINOSA

Profesor

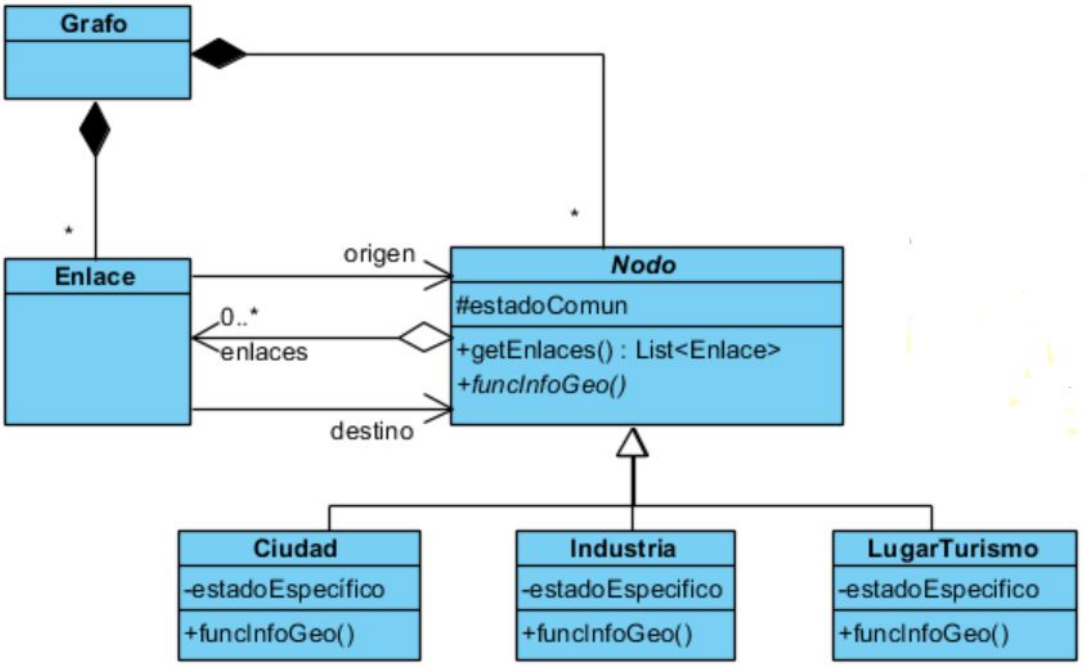
Jesús Andrés Hincapié Londoño

UNIVERSIDAD DE MEDELLÍN
FACULTAD DE INGENIERÍA
INGENIERÍA DE SISTEMAS

Medellín, 2020

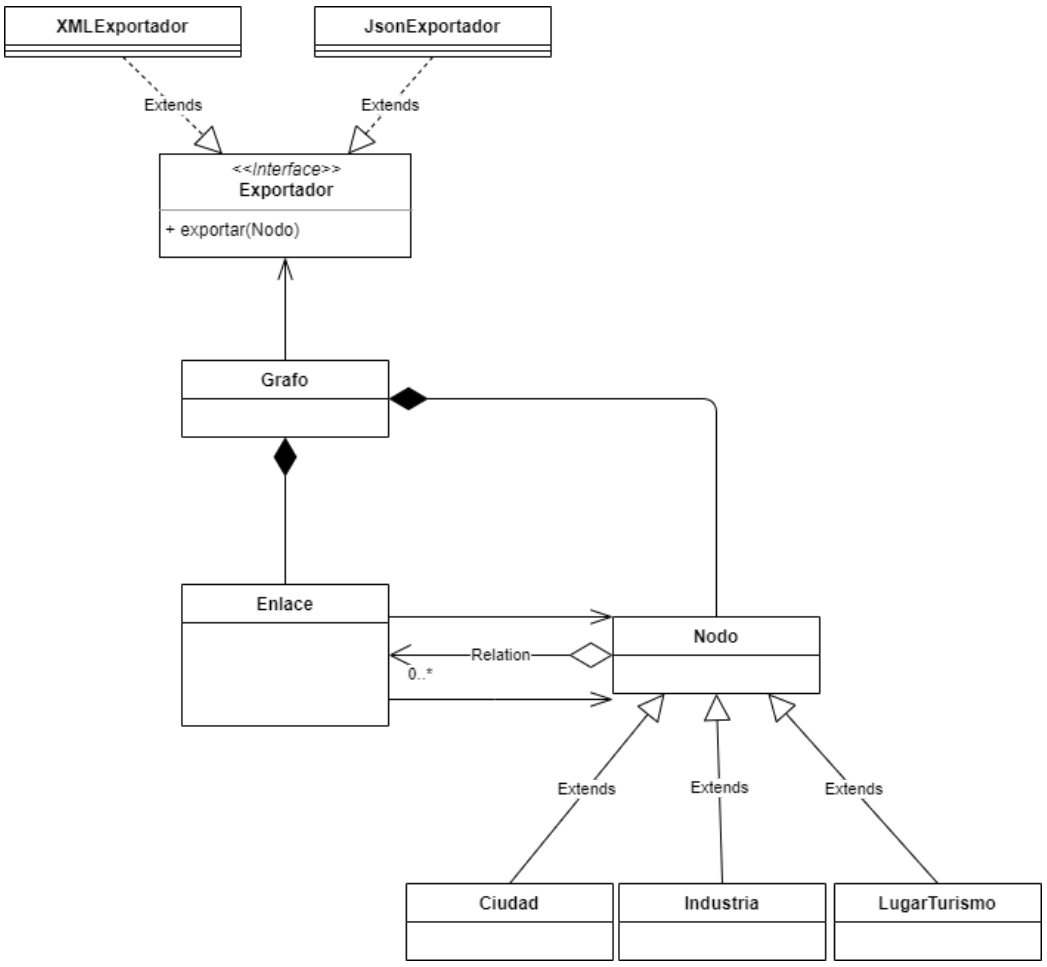
UNIVERSIDAD DE MEDELLIN

Análisis del diseño



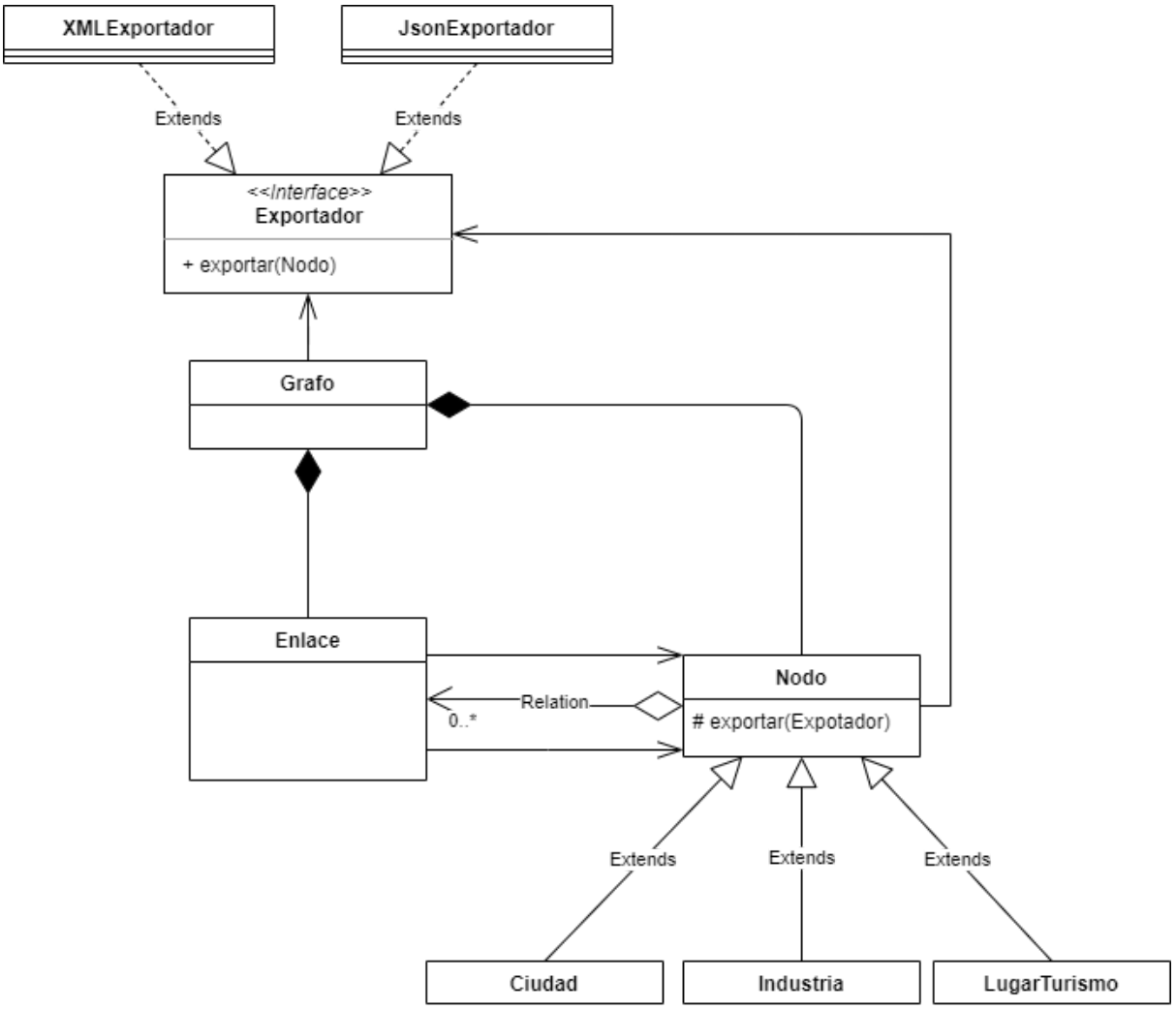
Nos piden una forma de exportar inicialmente en XML pero con la posibilidad de extender varios formatos el grafo tratando de no afectar la implementación de las clases.

El principal problema que evidenciamos es que cada tipo de nodo implementa una manera diferente de exportar ya que su estructura puede cambiar de nodo a nodo, para una versión inicial contemplamos esta solución:

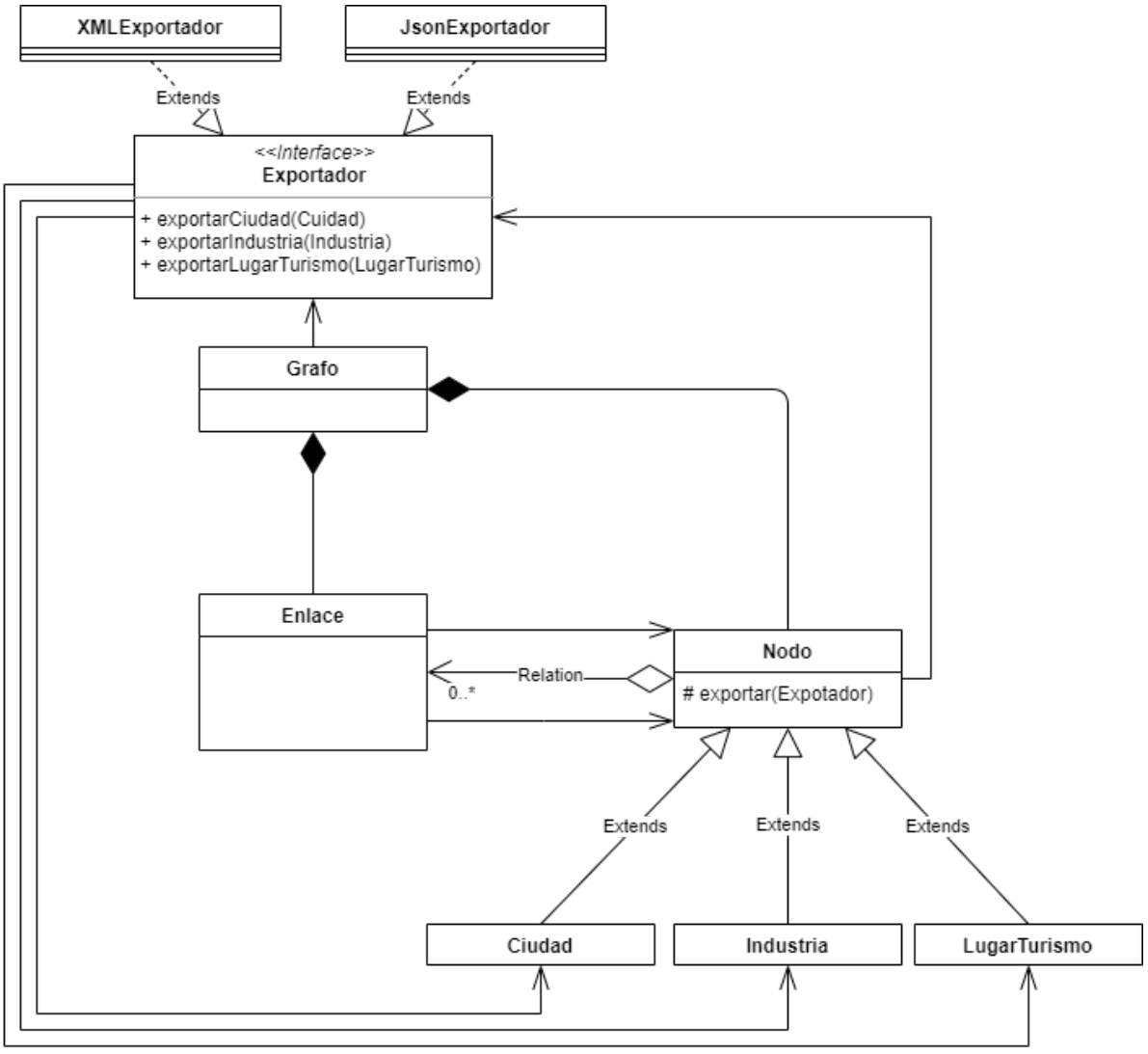


Ahora, con este modelo la clase Grafo usa la *interface* exportar con el fin de que se pueda exportar de diferentes maneras y según sea el tipo de nodo a exportar. El problema de esta solución radica en que tendríamos que preguntar el tipo de cada nodo para poder exportar de diferente manera, además no se podría tener acceso a los atributos privados de cada nodo.

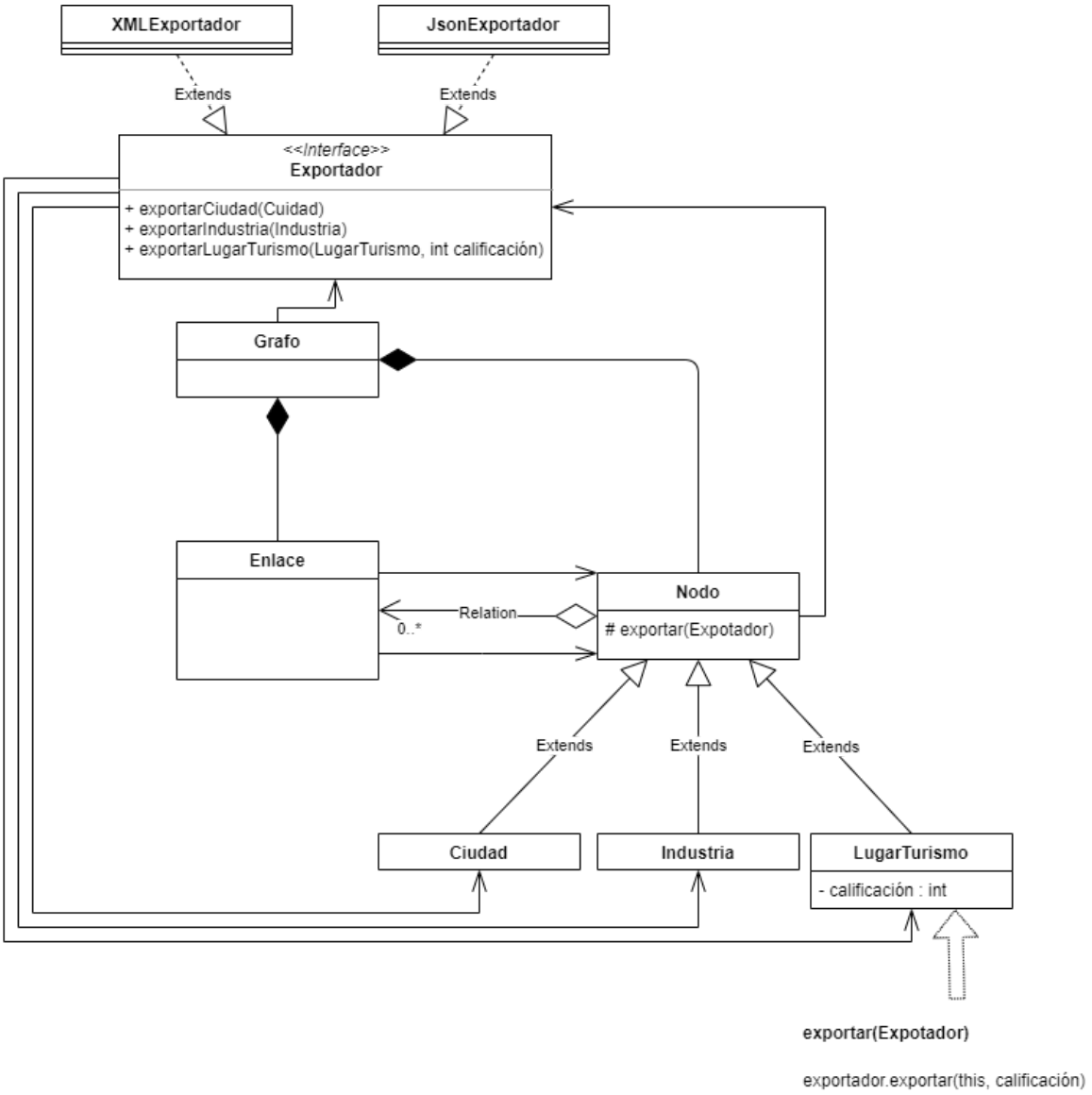
Para acceder a los atributos privados de cada tipo de nodo lo que hicimos fue delegar la función de exportar a cada tipo de nodo para que podamos utilizar todos sus atributos, tanto públicos como privados. El modelo quedaría así:



Pero, aún el Exportador recibe un Nodo, por lo tanto no estaríamos aprovechando esta nueva funcionalidad (acceder a los atributos privados de la clase). Así que tendremos que individualizar la forma de exportar cada nodo. Para esto decidimos crear un método para exportar cada uno y asociar el exportador a cada tipo de nodo para que se implemente su método. Con esto el modelo cambia así:



Con esta solución recibimos únicamente el nodo en cuestión, pero podemos parametrizar cada método para que reciba atributos específicos de cada tipo de la siguiente manera:



Con la solución implementada tenemos acceso a todos los atributos de cada tipo de nodo y lo podríamos exportar dependiendo de este; en la anotación del final se ejemplifica cómo sería implementado el código en este ejemplo específico.

En conclusión con este diseño estamos implementando, el patrón ***Visitor,*** el cual hace parte de los patrones de comportamiento que permite añadir funcionalidades a una clase sin tener que modificarla, siendo usado para manejar algoritmos, relaciones y responsabilidades entre objetos, en este caso manejando los algoritmos de exportar cada tipo de nodo.

Otro rol relevante en este caso los **visitantes concretos** serían cada tipo de exportación (XML, JSON), el cual implementa varias versiones del mismo adaptado a los diferentes **elementos concretos** que serían los tipos de nodos.

Además implementamos ***Double Dispatch***, el cual usamos con el fin de realizar llamados dinámicos de un método a otro, permitiendo así agregar operaciones en ciertas clases sin cambiar la estructura del código, ahorrandonos así el uso de múltiples condicionales con el fin de validar el tipo de nodo.

La principal ventaja de este diseño es que se puede extender la funcionalidad de exportar sin tener que tocar los tipos de nodos, es decir, teniendo en cuenta el principio de abierto/cerrado podemos introducir un nuevo comportamiento que trabaje con objetos de diferentes clases sin modificar estas.

Enlace al repositorio: https://github.com/Sebassllr/InformacionGeografica

Referencias

informaticapc. (2020). *Patrones de Diseño Software*. Obtenido de Informaticapc: https://informaticapc.com/patrones-de-diseno/visitor.php