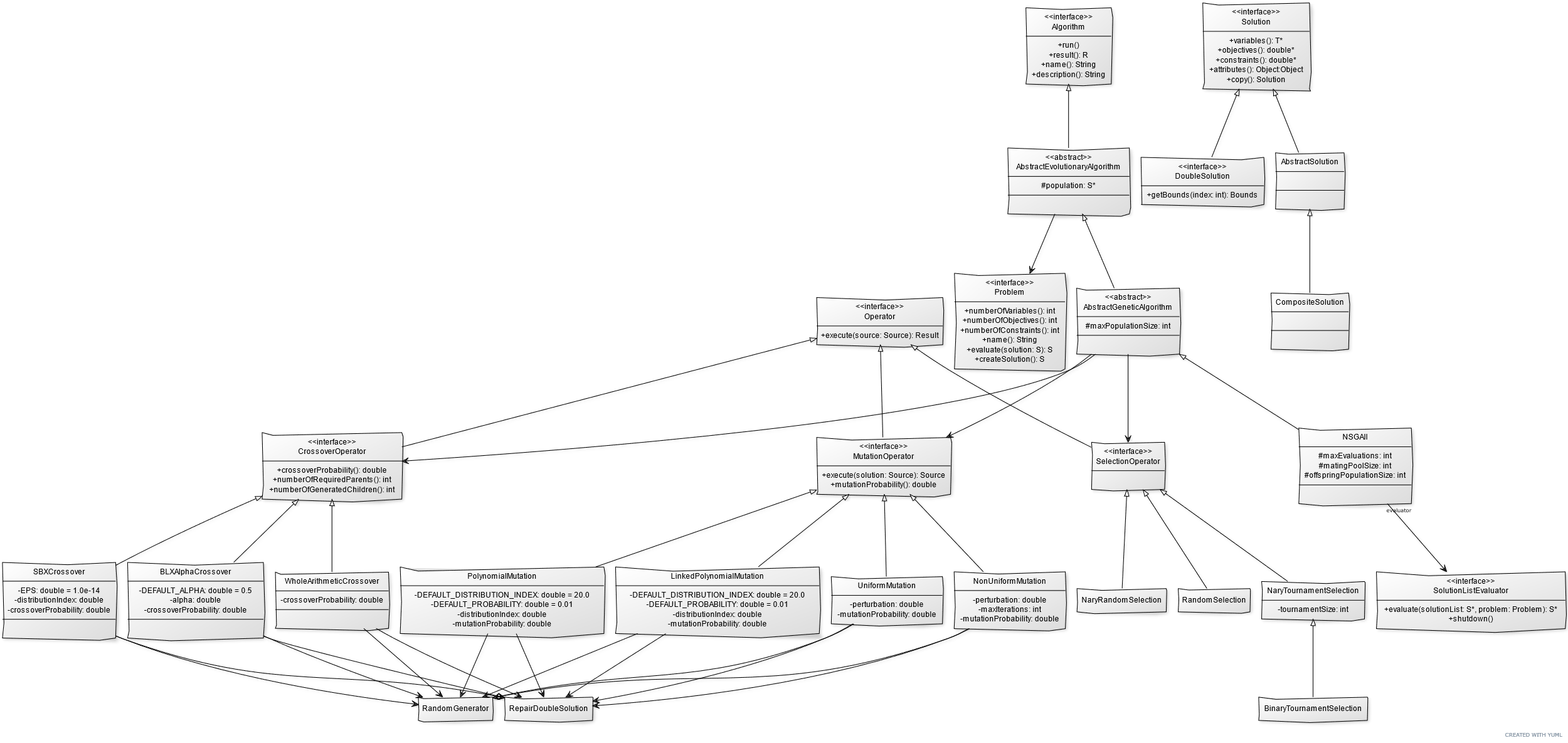
Gestionando la variabilidad de jMetal

* **Repositorio GitHub:** <https://github.com/jmhorcas/jmetal-variability.git>

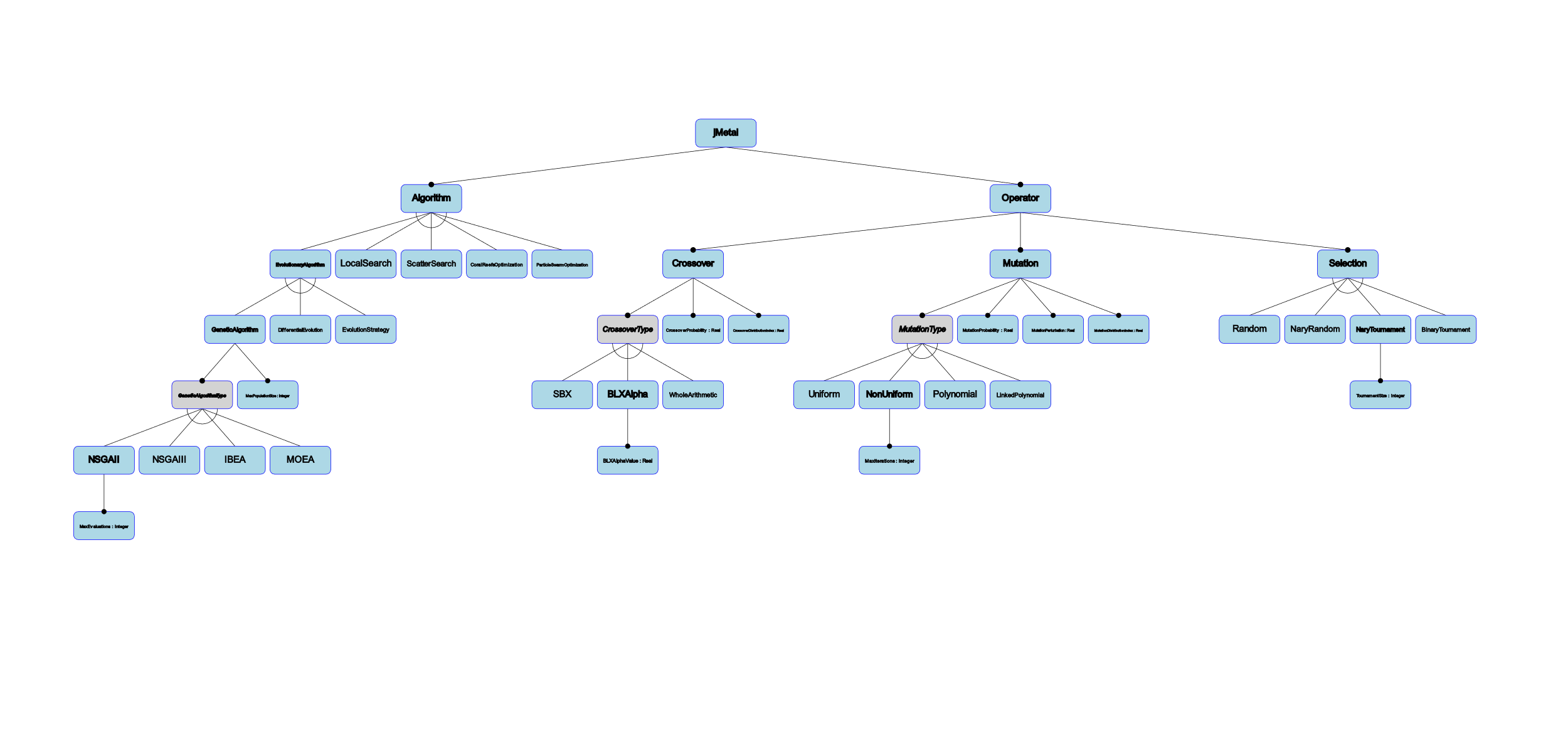
El repositorio contiene todos los recursos para ver los detalles de la siguiente documentación.

# Metodología

1. Nos centramos en el core de la librería jMetal y a partir del código extraemos un diagrama de clases que nos facilitará el análisis de la variabilidad.



1. A partir del diagrama de clases, el código, y la documentación de jMetal creamos un modelo de variabilidad (feature model) en UVL:

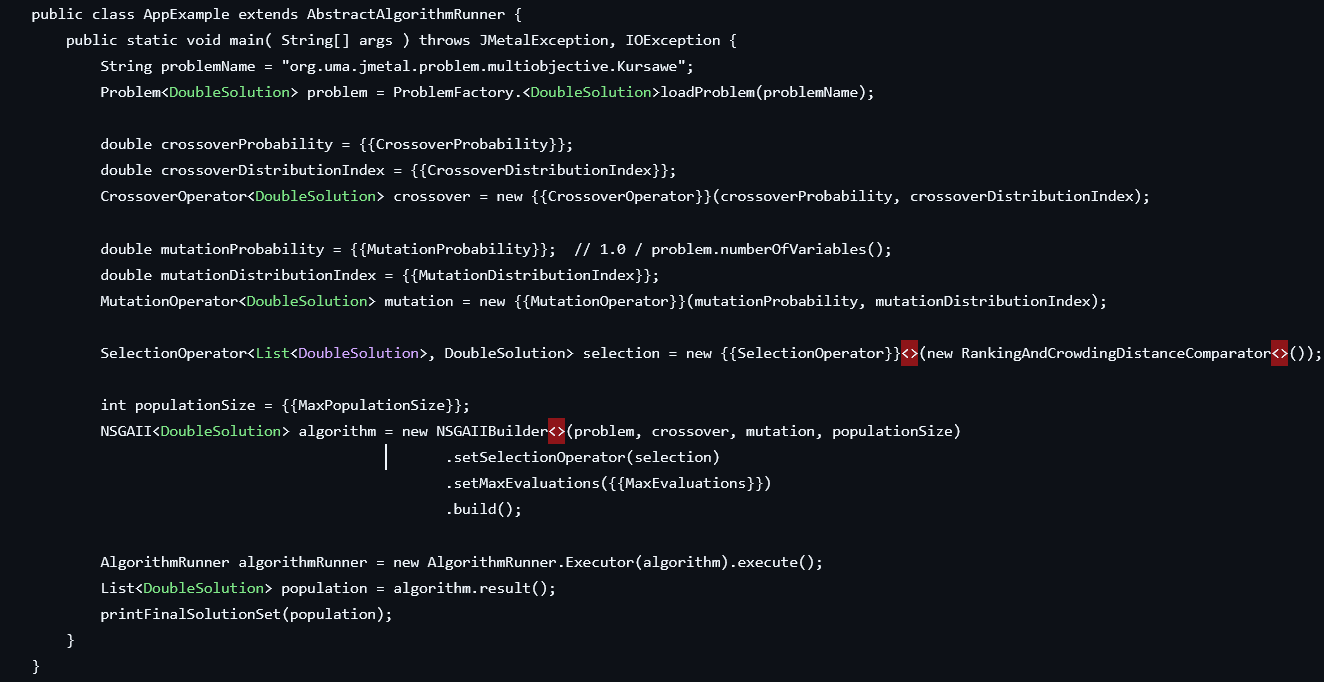


Texto

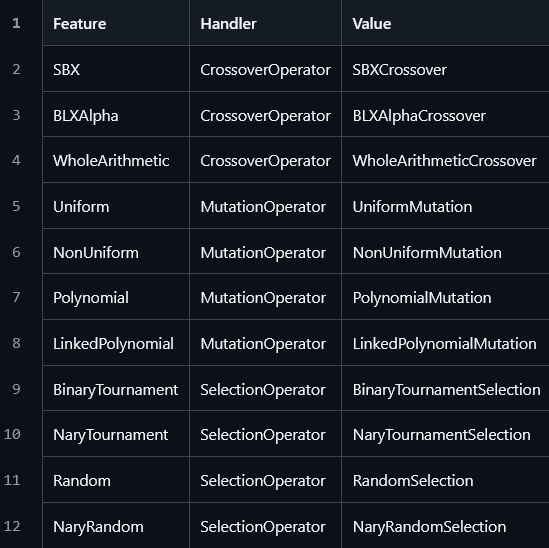
El contenido generado por IA puede ser incorrecto.

Básicamente tenemos algoritmos (nos centramos en el algoritmo genético NSGAII). Cada algoritmo puede configurarse con un operador de Crossover, un operador de Mutation, y un operador de Selection. Cada uno de los operadores tiene sus propios parámetros configurables. Aunque hay parámetros que son genéricos. El algoritmo en sí también tiene parámetros configurables.

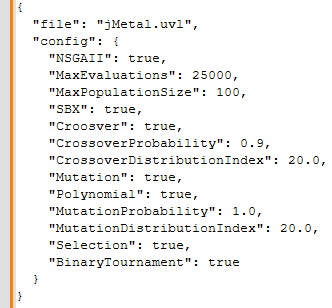
1. Cogemos un caso de uso de jMetal que resuelve un problema concreto usando el algoritmo NSGAII. Para una primera versión vamos a usar el core limpio de jMetal instanciando los componentes (es decir, nos olvidamos de todas clases que tiene Antonio para ayudar en la configuración). A partir de este código, creamos una plantilla Jinja marcando las partes variables. Las partes variables están entre dobles corchetes {{}} y a cada una de ellas se le da un identificador (llamado Handler).



1. Definimos un fichero de mapping que asocia las features del feature model con los Handlers de la plantilla Jinja. Este fichero podría ser opcional, pero en caso de omitirlo hay que complicar la definición de la plantilla con declaraciones {% if XXX %}, además de usar los mismos nombres para las features y para los Handlers. Con el mapping, la plantilla queda más limpia y fácil de gestionar.



1. Creamos una configuración del feature model en .json:



1. Usando mi UVEngine resolvemos la variabilidad.

Entradas necesarias:

1. Plantilla .jinja con el código variable.
2. Configuración .json del feature model.
3. Fichero de mapping.

Salida:

Código listo para ejecutarse.

