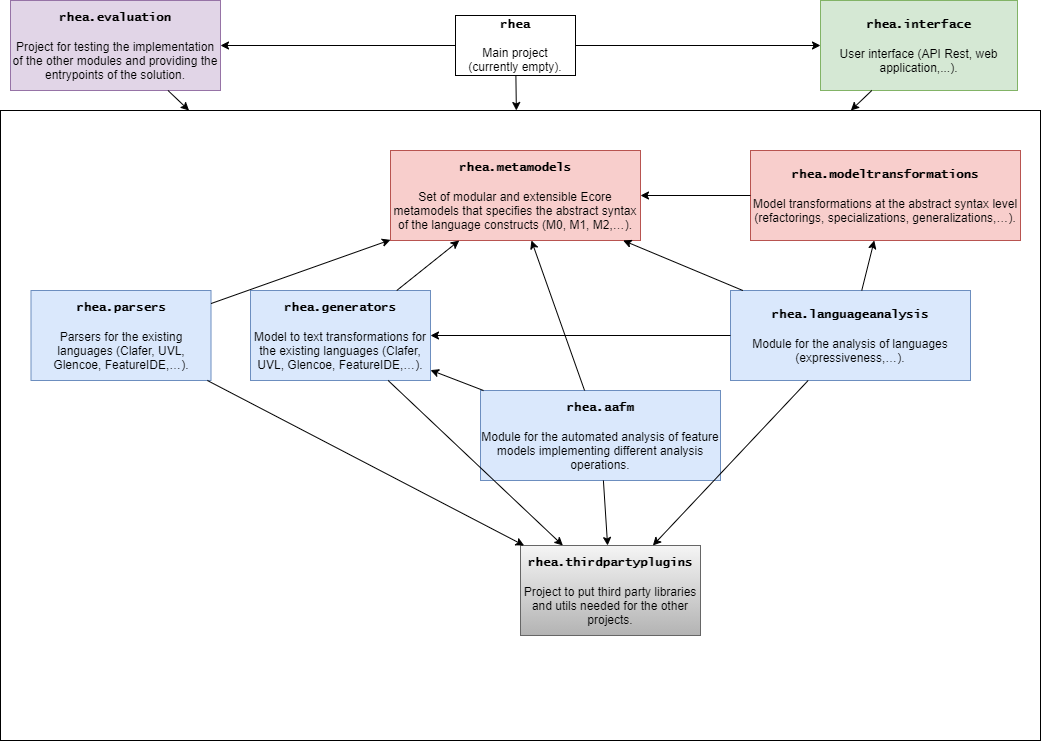
Proyecto RHEA

# Estructura del proyecto



# ToDos

✔️: hecho. **✇**: work in progress (medio a hacer).❌: por hacer.

## rhea.metamodels

Definición de los metamodelos modulares (ver paper Horcas2020\_SPLC).

* ✔️Modelos de características básicos (**BasicFMs.ecore**) **Constructores:** FeatureModel, Feature, AbstractFeature, Root, Optional, Mandatory, Parent-Child, FeatureGroup (abstract), AlternativeGroup, SelectionGroup, CrossTreeConstraints (abstract)
* ✔️Restricciones básicas (**BasicCTCs.ecore**)  
  **Constructores:** BasicContraint, Requires, Excludes
* ✔️Restricciones en lógica proposicional (**PropLogicCTCs.ecore**)  
  **Constructores:** AdvanceConstraint, Term, TermType (feature, not, and, or, xor, implies, excludes, equiv)
* **✇** Modelos de características con cardinalidad (**CardinalityBasedFMs.ecore**)  
  **Constructores:** GroupCardinality, Multiplicity  
  **Tarea:** Añadir constructor para características Clonables.
* ❌Características con tipos de datos (**DataTypes.ecore**) [Hay que rehacerlo]  
  **Constructores:** DataType, PrimitiveType, TypedFeature, Value, ObjectType, NonBooleanFM
* ❌Características numéricas (**NumericalFMs.ecore**) [Hay que rehacerlo]  
  Constructores: NumericalFM, NumericalFeature
* ❌Características con atributos (**AttributedFMs.ecore**) [Hay que rehacerlo]  
  **Constructores:** AttributedFM, Attribute

## rhea.parsers

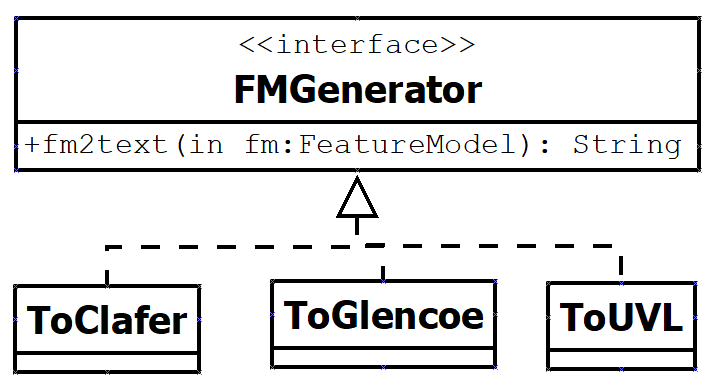
Implementación de los parsers para los diferentes lenguajes de variabilidad (transformaciones de la sintaxis concreta a la abstracta).



* **✇** Parser para Clafer [complicado]  
  **Constructores soportados actualmente:** FeatureModel, Feature, Root, Optional, Mandatory, Parent-Child, AlternativeGroup, SelectionGroup, GroupCardinality  
  **Tarea:** Añadir el resto de constructores (restricciones, numéricas,…)
* **✇** Parser para Glencoe  
  **Constructores soportados actualmente:** FeatureModel, Feature, Root, Optional, Mandatory, Parent-Child, AlternativeGroup, SelectionGroup, GroupCardinality  
  **Tarea:** Añadir el resto de constructores (restricciones, numéricas,…)
* **✇** Parser para UVL [no necesario por ahora]
* ❌Parser para FeatureIDE [no necesario por ahora]
* ❌Parser para SPLOT [fácil pero no necesario por ahora]

## rhea.generators

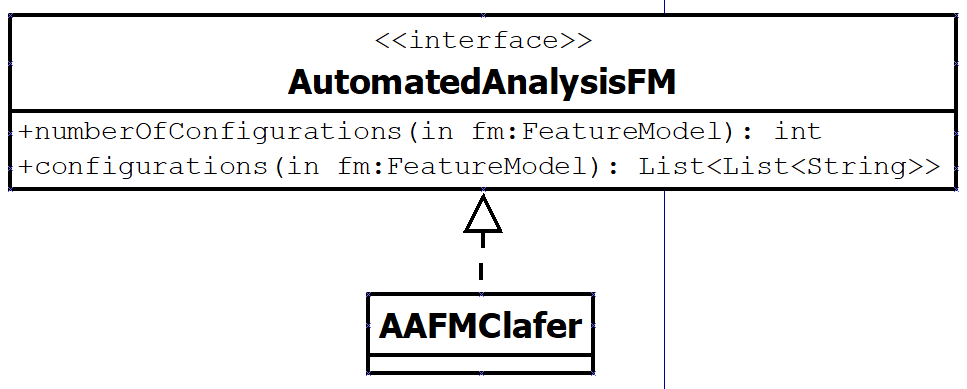
Implementación de los generadores para los diferentes lenguajes de variabilidad (transformaciones de la sintaxis abstracta a la concreta).



* **✇** Generator para Clafer  
  **Constructores soportados actualmente:** FeatureModel, Feature, Root, Optional, Mandatory, Parent-Child, AlternativeGroup, SelectionGroup, GroupCardinality  
  **Tarea:** Añadir el resto de constructores (restricciones complejas, numéricas,…)
* ❌Generator para Glencoe
* ❌Generator para UVL

## rhea.aafm

Implementación de las operaciones de análisis de los feature models en base a la sintaxis abstracta.



* **✇** Implementación basada en los binarios de Clafer y Chocosolver   
  **Operaciones soportadas:** configuraciones, número de configuraciones.  
  **Constructores soportados actualmente:** todos los de Clafer para ficheros Clafer y los mismos que el generador de Clafer para nuestros feature models.
* ❌Implementación basada en SAT4J y/o Chocosolver y otros solvers.

## rhea.languageanalysis

Este proyecto ahora mismo tiene dos clases:

* Una clase **LanguageExpresiveness** ejecutable para analizar la expresividad a nivel de constructores de lenguajes. Esta clase también tiene código para generar todos los posibles productos, configuraciones y productos distintos de una SPL dado un conjunto de features concretas y abstracts (independientemente de del feature model).
* Una clase **FeatureModelGenerator** para generar todos los posibles feature models dados un conjunto de constructores del lenguaje y un conjunto de features.

❌ **Tarea:** organizar este código, crear interfaces, refactorizarlo…. Para poder usarlo como parte de la evaluación.

## rhea.transformaciones

Implementación de los refactorings, language generators,… (cualquier transformación de modelo) en Henshin:

* Refactorings:
* Language generators:

## rhea.evaluation

Este proyecto sirve para testear la funcionalidad del resto de proyectos y para la evaluación de los diferentes aspectos del proyecto.

Ahora mismo contiene una clase que dado un FM en Clafer, lo parsea y convierte a la sintaxis abstract. Para comprobar que la transformación es correcta se calculan las configuraciones de ambos modelos.

A continuación se aplicarán los refactorings y se volverá a comprobar las configuraciones.

Por último se pasará el modelo a la sintaxis concreta de Clafer y se volverá a comprobar las configuraciones.