Implementación de los refactorings con Henshin

El primer paso es identificar los constructores básicos que vamos a usar, de forma que los **refactorings** que definamos traduzcan de otros constructores más complejos a estos. Vamos a usar como constructores básicos aquellos que sean soportados por todas las herramientas **(Clafer, Glencoe, FeatureIDE, UVL)**, es decir, aquellos constructores que tienen en común. Vamos a llamar a este languaje **LBasicTools**:

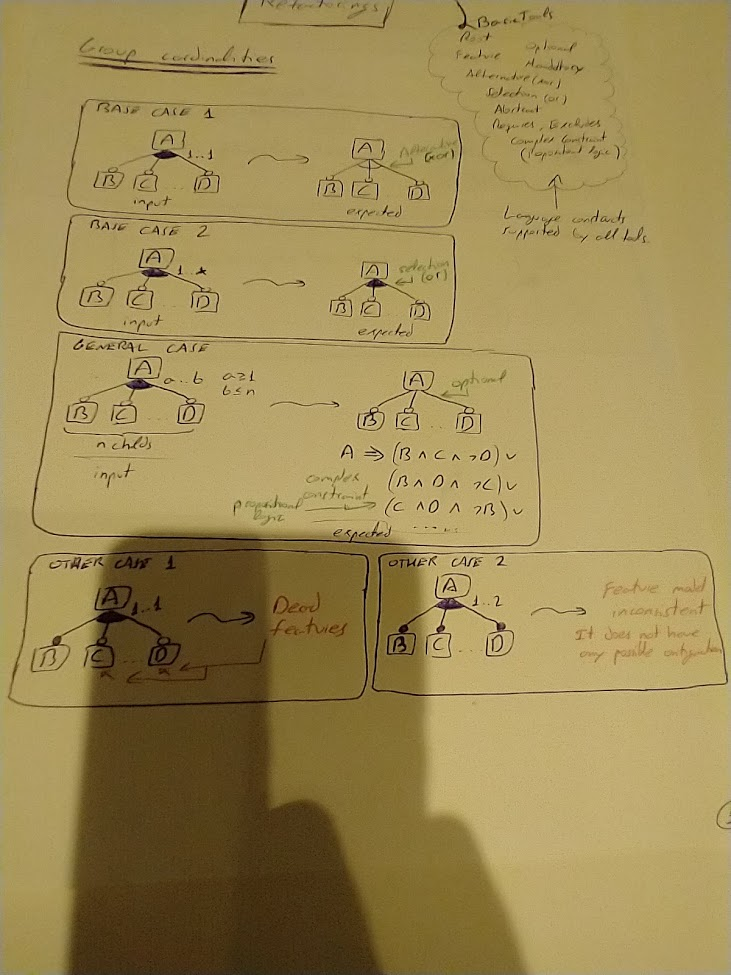
**LBasicTools = {Root, Feature, Abstract, Optional, Mandatory, Alternative (xor), Selection (or), Requires, Excludes, Complex Constraint (propositional logic)}**

# **Group Cardinalities**

Vamos a empezar definiendo el refactoring para los grupos con cardinalidad (**Group Cardinalities**). Un refactoring puede consistir en una o más transformaciones de modelo que llamaremos a partir de ahora **Henshin rules**.

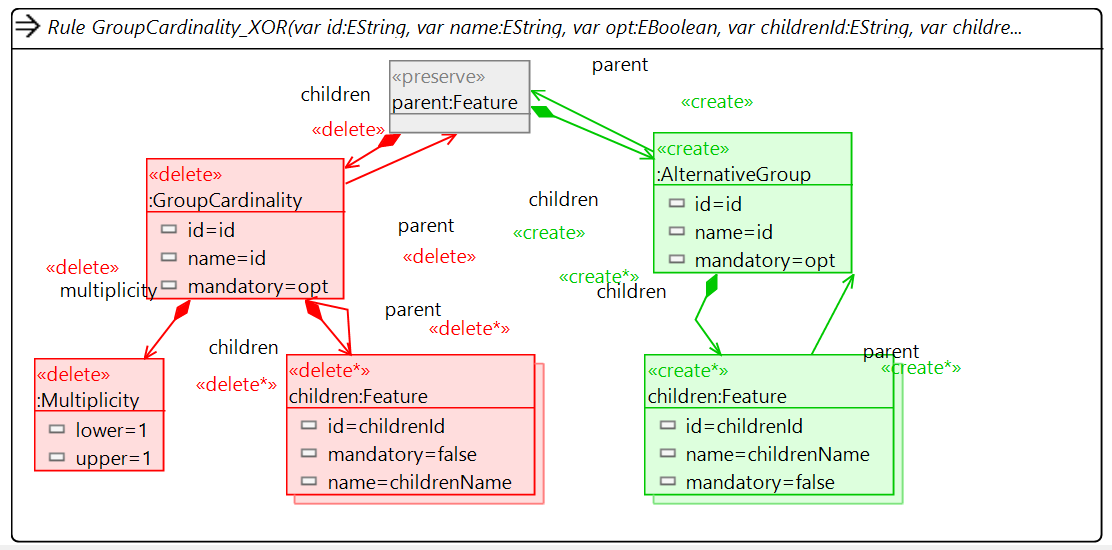
He identificado dos casos base y un caso general (ver página siguiente) que resultarán en una o más transformaciones de modelo (**Henshin rules**). También he identificado algunos casos que se permite en algunas herramientas (e.g., Glencoe) pero que dan como resultado **feature models inconsistentes** o feature models con **dead features**. Éstos casos no los vamos a considerar en las Henshin rules pero podríamos definir Henshin rules específicas que identificaran éstos casos y dieran como salida información del error (análisis) del feature model.

**¿Están cubiertos todos los posibles casos?**



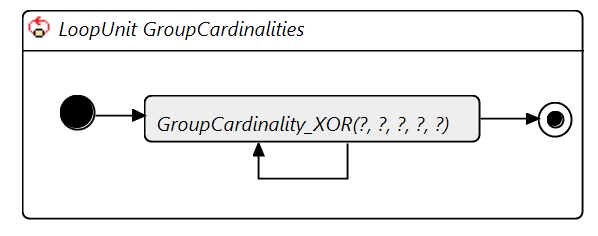
## Definiendo las transformaciones

Por ahora he definido la Henshin rule para el caso base 1:



**Comentario 1.** Implementar directamente el caso general resulta muy complicado, de ahí que empiece definiendo los casos base. Puede que luego cuando tenga definido el caso general incluya también los casos base, pero eso no ocurrirá para los Group Cardinalities porque definiendo los casos base de forma independiente al caso general resulta en feature models mucho más simples.

**Comentario 2**. Esta transformación se aplica por defecto una única vez al feature model haciendo un “match” de forma aleatoria. Es decir, transforma únicamente un grupo (una instancia de Group Cardinalities). Para que traduzca todas los Group Cardinalities del modelo, añado la siguiente regla (**unit**), que aplica la regla anterior tantas veces como haga falta hasta que no encuentre más “matchs”.



## Correctnesss de las transformaciones

**Comentario 3.** Para comprobar que la regla es correcta debo definir varios tests con entrada (**input model**) y salida (**expected output)** conocida. Esto implica que los feature models aleatorios para la evaluación debo usarlos más adelante PARA LA EVALUACIÓN del artículo pero no en este punto donde necesito entradas y salidas bien conocidas. En la siguiente página pongo un par de ejemplos de tests. El primero (FM\_Test1) nos ayuda a ver si el caso base 1 se aplica correctamente, teniendo en cuenta que hay más de una ocurrencia en el feature model, así como si se mantienen las propiedades (optional/mandatory) de las features. El segundo test es muy interesante porque nos obliga a definir otra Henshin rule adicional, ya que la root no se considera en la regla implementada arriba (debido a que la root no tiene padre).

**¿Qué otros tests deberíamos definir para cubrir todos los posibles casos?**

**Comentario 4.** Los tests se pueden directamente sobre los metamodelos creando una instancia dinámica (feature model) directamente en la sintaxis abstracta (y así lo he hecho para este primer test). Sin embargo, esto no es recomendable ahora mismo porque cualquier cambio en el metamodelo invalida completamente todas las instancias (todos los feature models de tests) y en este punto es muy fácil que tengamos que modificar bastante los metamodelos mientras vamos puliendo los detalles.

Una alternativa mucho más viable es definir los feature modelos en un lenguaje (e.g., Clafer) y usar un parser para pasarlos automáticamente a la syntaxis abstracta. De esta forma cualquier cambio en los metamodelos solo afectaría al parsers que sería trivial de modificar.

