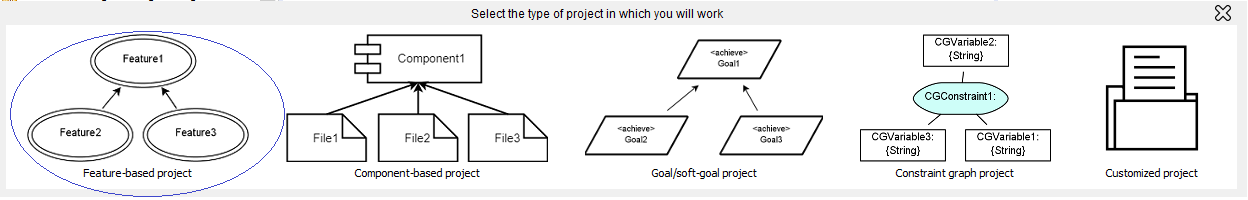
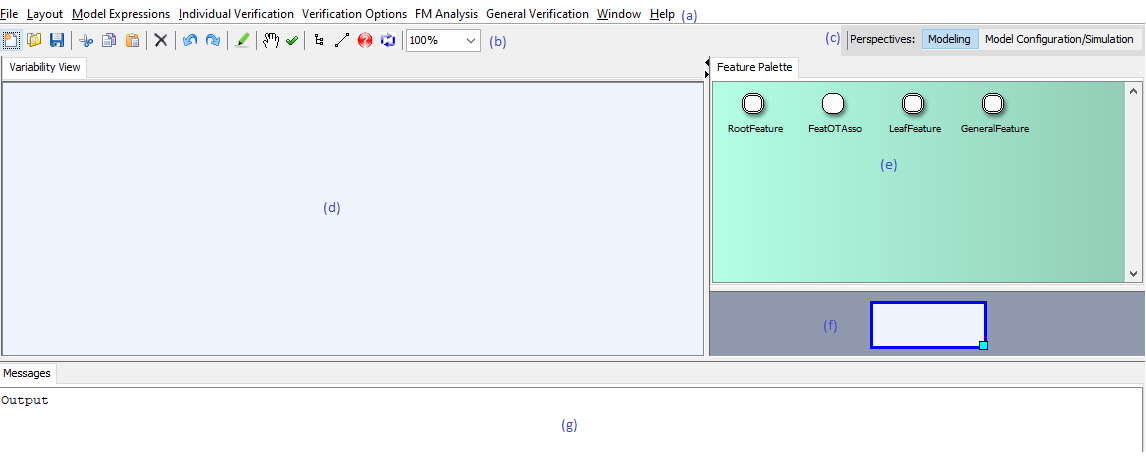
**VariaMos**

VariaMos es una herramienta de gestión de la variabilidad que soporta diferentes tipos de modelos y también ofrece una amplia flexibilidad para que el usuario adapte la herramienta a sus necesidades. Es una aplicación gráfica desarrollada en el lenguaje Java que se puede ejecutar en diferentes sistemas operativos (Windows, Mac OS y Linux) y además puede interactuar con otras herramientas mediante archivos de intercambio de modelos.

Una vez abrimos la herramienta de VariaMos encontramos una interfaz que presenta todos los modelos soportados en VariaMos, se exponen cada uno de los modelos empezando por ***Feature-based***, que permite representar todos los productos involucrados en la línea de producto de software en términos de “características”.

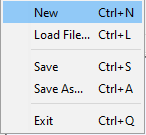


Una vez seleccionamos este modelo nos encontramos con la siguiente interfaz que nos ofrece todas las facilidad para empezar a construir

**Menú principal (a):**

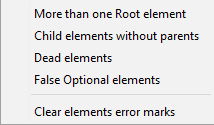
Permite acceder a los controles de la aplicación para que el usuario pueda interactuar con ella.

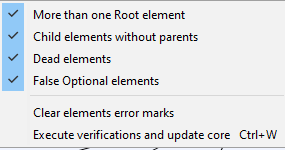
* **File:** Controla las operaciones que involucran la creación y el guardado de archivos.



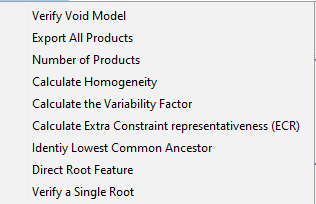
* + **New:** Podemos hacer uso de esta opción en caso de que deseemos empezar de cero con un nuevo modelo, pero debemos tener sumo cuidado pues podríamos perder el modelo actual en caso de que no lo hayamos guardado. Se presentarán las opciones iniciales para elegir entre los tipos de modelo.
  + **Load File:** Si tenemos un modelo listo (la extensión del modelo debería ser \*.**vnum**) , podremos cargarlo, incluso si el modelo ha sido creado en versiones anteriores de VariaMos, esta funcionalidad abrirá el explorador de archivos hasta que seleccionemos el modelo que deseemos cargar y continuar con el proceso de modelado

.

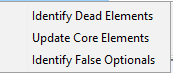
* + **Save:** Éste guardado corresponde a los cambios que se han hecho sobre un modelo existente o previamente creado, en caso de que el modelo no exista esta opción proporciona las mismas cualidades que **Save as**.
  + **Save as:** Se despliega el explorador de archivos para navegar y elegir la ruta donde queremos que nuestro modelo sea almacenado por primera vez.
  + **Exit:** Cierra Variamos
* **Layout:** el menú Layout son un conjunto de opciones de organización automática de los elementos diagramados en la aplicación.
* **Model Expressions:** En model expression encontramos los marcos de trabajos que definen los modelos que posteriormente se van a construir. Hasta el momento, el único marco soportado en Variamos es REFAS (Requirements engineering for self-adaptive systems). REFAS permite a los desarrolladores evaluar y mejorar los modelos de requisitos a través de simulaciones, además REFAS puede verificar que todos los componentes del modelo que se está construyendo sean coherentes.
* **Individual Verification:** Nos permite atomizar los parámetros de verificación, de esta forma es más fácil depurar los errores que puedan existir en nuestro modelo
  + **More than one root element:** Cuando construimos un modelo basado en características, el elemento raíz debe ser único, en caso de que sospecharemos que hay dos raíces podremos verificarlo haciendo uso de esta opción.
  + **Child elements without parents:**Esta opción nos permite verificar casos en donde existe alguna característica aislada, es decir, una característica que no puede ser alcanzada al hacer una derivación por la jerarquía de características desde la raíz.
  + **Dead elements:** Podemos verificar características innecesarias, cuando hacemos uso de las relaciones entre características (exclusión, requerido, obligatorio, opcional, etc...) accidentalmente podemos crear un modelo en donde alguna característica no esté disponible en ningún producto entonces esta característica no vale la pena incluirla, o deberíamos reconsiderar la estructura del modelo.
  + **False optional elements:** En ocasiones se tiene que una característica es opcional, sin embargo, es requerida por alguna característica que está presente en todos los productos. En este caso deberíamos replantear las relaciones.
  + **Clear elements error marks:** Cuando un modelo es verificado, cada característica que tenga un error, tendrá una marca que lo indique. Esta opción limpia todas las marcas antes de hacer una nueva verificación.
* **Verification Options:** Además de poder verificar de manera individual cada caso de error, también podemos hacer la verificación combinando las opciones que deseemos.



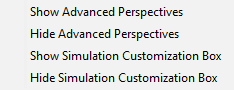
* **FM Analysis:** 
  + **Verify Single Root:** En los modelos de características la raíz debe ser única, por tanto con esta funcionalidad podemos verificar la unicidad de la raíz.



* **General Verification**
  + **Update core elements:** Al hacer uso de esta funcionalidad, podemos ver en verde, cuáles son los elementos o características que estarán de forma invariable en todos los productos generados.
  + **Identify Dead Elements:** Los elementos muertos, son características que nunca pueden existir en un producto, por ende, son innecesarias en el modelo al usar esta función podríamos verificarlas para eliminarlas o replantear una solución.
  + **Identify False Optionals:** Algunas características podrían verse comprometidas cuando son opcionales y una característica obligatoria o de núcleo las excluye.



* **Window:** Para acceder a estas funciones debe estar en la perspectiva de simulación
  + **Show advanced perspectives:** Nos muestra la perspectiva avanzado, que es donde podemos definir los meta-modelos y después construir nuestros modelos a partir de la definición.
  + **Hide advanced perspectives:** Esconde la perspectiva avanzada.
  + **Show simulation customization Box:** Muestran las perspectivas avanzadas, para crear modelos propios o verificar sintaxis del meta-modelo en curso.
  + **Hide simulation customization Box:** Esconda las paletas de donde se editan los modelos



**Barra de Herramientas**

En la barra de herramientas, se encuentran las funcionalidades de uso frecuente. Algunas funciones soportadas en VariaMos se pueden usar de manera fácil y rápida desde este menú.

* Empezar un nuevo modelo desde cero.
* Cargar un modelo existente
* Guarda el modelo actual en una ruta específica, o general un punto de guardado si el modelo ya existe en el computador.
* Corta un elemento y lo mantiene en el portapapeles para luego ser pegado. [Esta función también puede ser alcanzada usando Ctrl+X]
* Copia un elemento y lo mantiene en el portapapeles para luego ser pegado. [Esta función también puede ser alcanzada usando Ctrl+C]
* Pega el elemento que se guardó en el portapapeles. [Esta función también puede ser alcanzada usando Ctrl+V]
* Elimina el elemento que está siendo seleccionado.
* Estas opciones permiten deshacer y rehacer las últimas acciones ejecutadas. [Puede ser alcanzado también por medio de Ctrl + Z]
* Se actualiza el modelo, mostrando cuáles son las características invariables en cualquier modelo.
* Elimina las marcas que se generan después de hacer algún tipo de verificación.
* Realiza la verificación a nivel de núcleo (Este tipo de verificación es explicado en las funcionalidades del menú principal)
* Verifica si la raíz del modelo es única.
* Verifica si existe alguna característica que no tenga padre.
* Es un atajo para ejecutar las opciones habilitadas dentro *Verification options.*
* Usando este botón fácilmente podremos comprobar si existen características falsas. [*False optional elements*]
*  Permite acercar y alejar el modelo, graduando el enfoque con un porcentaje.

**Perspectivas**

Separan las acciones que se pueden hacer en cada vista, estas perspectivas son a nivel de modelado y a nivel de configuración.

* **Modelling:** En esta perspectiva se crean, editan y verifican los modelos de variabilidad y de características. (Creados en *Variability View*)
* **Model configuration / Simulation:** Dentro de esta perspectiva se configura y se realiza la simulación de los modelos basados en características. Una vez terminado nuestro trabajo en la perspectiva **modelling**, aquí podremos ver cuales son los productos derivados de nuestro modelo.

**Variability view**

Dentro del modelo de características tenemos la vista de variabilidad que es donde arrastraremos cada uno de los componentes de la paleta, y es el lienzo que contendrá la jerarquía de nuestro modelo.

**Feature Palette (e):**

Aquí encontramos el conjunto de elementos necesarios para dibujar los modelos dentro de la vista de variabilidad.

* **Leaf feature:** Estas son las características de las cuales no se pueden desprender más características, es decir que dentro de la jerarquía del modelo estas se encuentran en el fondo.
* **FeatOTAsso:** Hay un conjunto de relaciones grupales que se pueden establecer entre las características de un modelo, este elemento permite hacer uso de esas relaciones y definir diferentes tipos de cardinalidades.
* **GeneralFeature:**Son características padres, es decir características de las cuales otras características pueden ser desprendidas en la jerarquía del modelo.
* **RootFeature:** Es la característica inicial o raíz, de esta se desprenden todas las demás características debemos recordar que la característica raíz es única.

**Visor del área de modelado (f):**

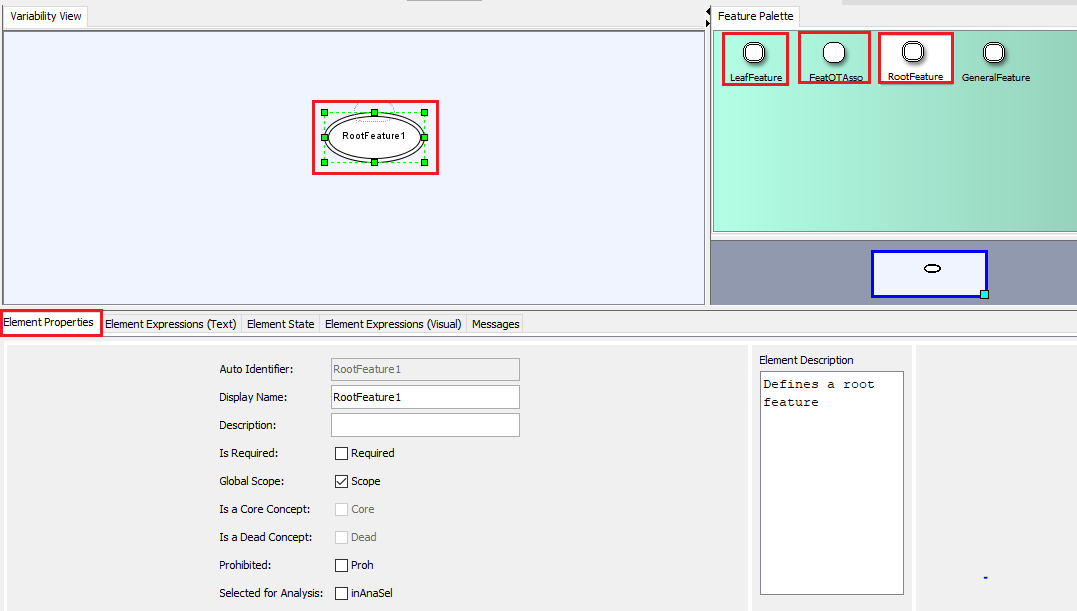
A través de un panel permite visualizar la vista actual de manera panorámica además podemos desplazarla desde aquí.

**Sección de propiedades de los elementos que hacen parte de los modelos (g):**

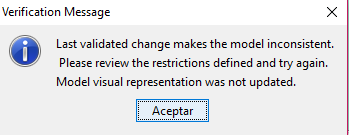
Permite editar a través de formularios la información y propiedades de los elementos (nodos del modelo).

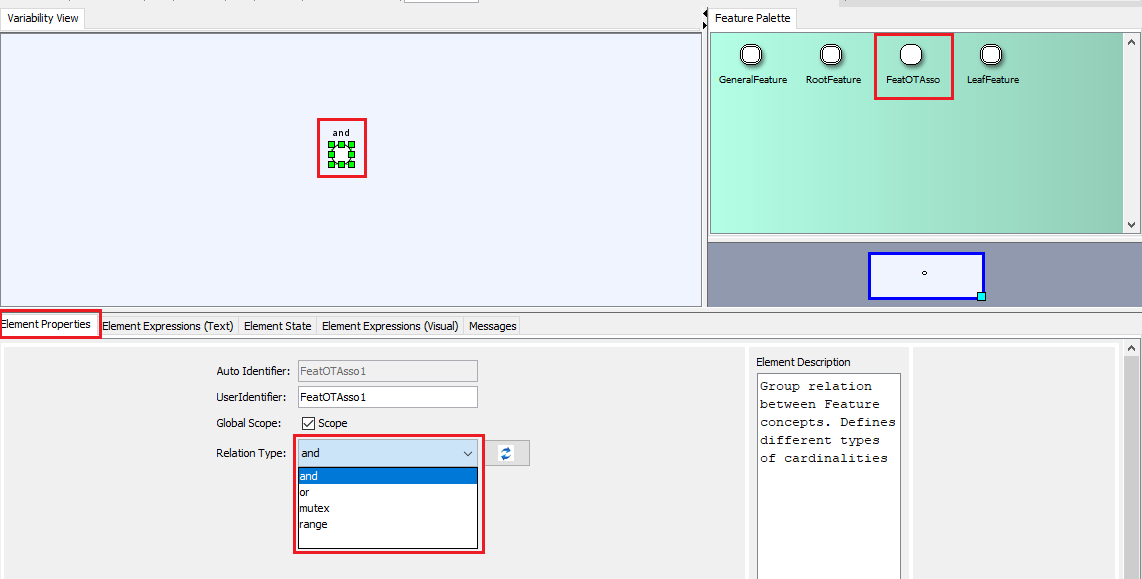
**Definición de los tipos de características y relaciones entre ellas:**

Para un modelo de características debemos empezar haciendo uso de una característica raíz de la cual se derivarán las demás. Por lo general, una característica raíz es aquella que define el producto sin importar las variaciones que este pueda tener.

Las propiedades que tiene tanto una característica raíz, como una hoja o una general, son las siguientes:

* **Auto Identifier:** Este es el nombre o identificador que enlaza lógicamente la característica con la herramienta VariaMos, y es por esto que es inmutable.
* **Display Name:** Este es el nombre de la característica raíz y puede ser editado para reemplazarlo por un nombre que pertenezca al dominio de nuestro modelo.
* **Description:** Aquí podemos añadir algún comentario, pues agregar una pequeña descripción a las características, es siempre una buena idea con el objetivo de tener en claro el rol que cumplen cada una de ellas en la concepción del modelo.
* **Is required:** Establece que una característica va a ser ahora de núcleo, es decir que estará disponible en cualquier producto. [Not Sure, if so, no necessary]
* **Is a core concept:** Esta opción informa si la característica, es una de núcleo dentro del modelo.
* **Is a Dead concept:** Esta opción informa si la característica, nunca existirá en algún modelo.
* **Prohibited:** Cuando seleccionamos esta opción para alguna característica, automáticamente la excluimos de existir en algún producto.

Debe tenerse especial cuidado cuando se seleccionan algunas opciones de las propiedades, podríamos incurrir en el error de seleccionar requerido [*Is required*] a la vez de prohibido [*Prohibited*], y obtendremos un problema de consistencia como el siguiente. 

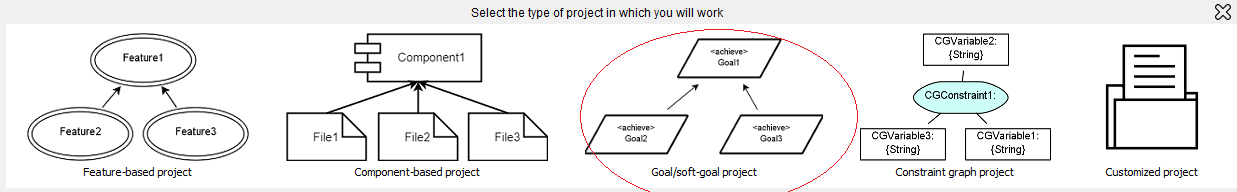
Las relaciones entre características dentro de VariaMos, puede ser de varios tipos y son las siguientes

* + Para VariaMos se debe especificar una cardinalidad grupal, esta permite establecer de manera fácil y rápida cualquier combinación que se desee tener entre características, por ejemplo, el **XoR** tendría una cardinalidad grupal [1..1] , es decir solo una característica podría ser elegidas a la vez dentro de algún conjunto, un **“or”** sería una cardinalidad grupal [ 1..n] y un **“and”** [n..n]. Las cardinalidades grupales permiten ahorrar esfuerzo al especificar el número de características necesarias para representar en un modelo, por ejemplo, *“se necesitan 4 de 5 características”* y serán fácilmente elegibles en lugar de construir este requisito por medio de un **AND** y un **OR**.
  + **Range:** Permite especificar el mínimo y máximo número de características hijas que se pueden incluir en una configuración cuando se selecciona la característica padre en dicha configuración.***And:*** Si la característica padre se incluye en una configuración, también se incluyen todas las características hijas en dicha configuración. ***Or:*** Una característica hija tiene una relación opcional con su padre cuando la característica hija puede o no, estar incluida en todos los productos en los cuales su característica padre aparece

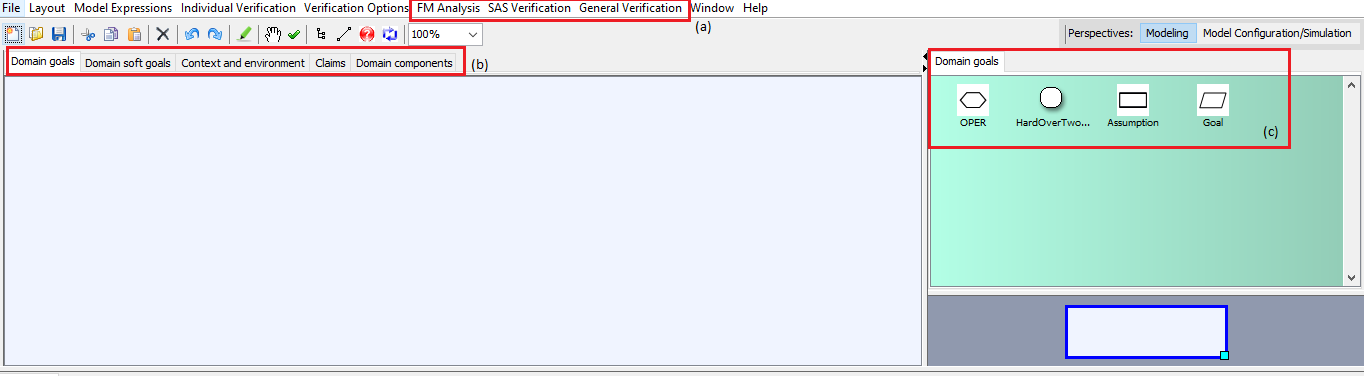
Una característica puede tener una relación directa con sin usar FeatOTAsso, para ello basta con establecer una línea entre una característica y la otra, hay dos tipos de relaciones:

* **TraversalF:** Seleccionar el icono refrescar para poder disponer de estos tipos de relaciones.
  + **Require:** Si una característica A requiere una característica B, la inclusión de A en un producto incluye la característica B en ese producto.
  + **Excludes:** Si una característica A excluye una característica B, la inclusión de A en un producto excluye la característica B en ese producto.
* **Structural**
  + **Mandatory:** Una característica hija tiene una relación de obligatoriedad con su padre cuando la característica hija está incluida en todos los productos en los que la característica padre aparece
  + **Optional:** Una característica hija tiene una relación opcional con su padre cuando la característica hija puede o no, estar incluida en todos los productos en los cuales su característica padre aparece.

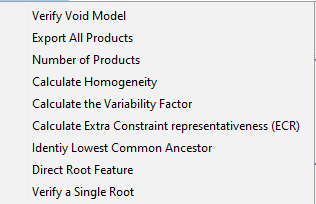
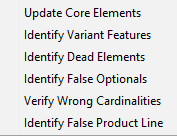
Los modelos de características son muy útiles dentro del marco de trabajo(REFAS), sin embargo, estos modelos podrían complementarse definiendo otro tipo de modelos en la concepción de nuestro proyecto.



El modelo de metas (***Goal/Soft-goal***) permite describir los propósitos funcionales y no funcionales de la línea de productos, una vez seleccionado nos encontramos con la siguiente.

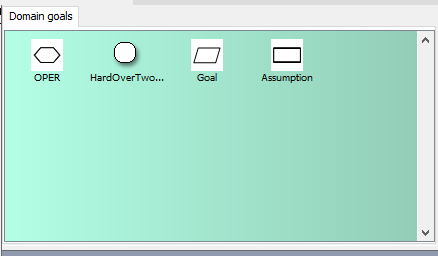


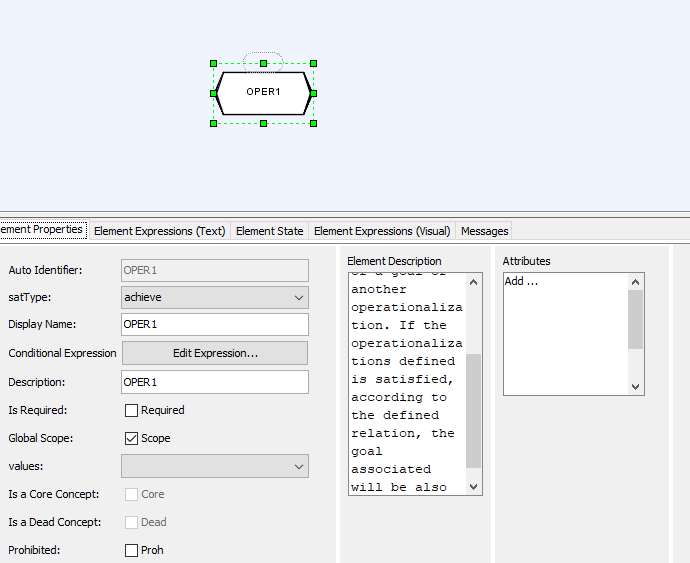
Existen diferencias respecto al menú principal disponible en el modelo de características.

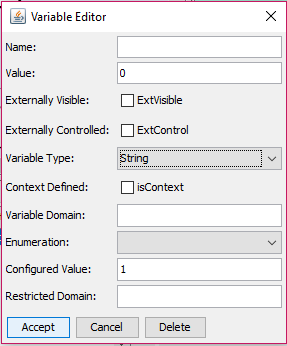
* **FM Analysis:** Analizamos los resultados ahora, no en términos de las características que componen el modelo, sino, en términos de los posibles productos.
  + **Verify void model:** Nos permite identificar si del modelo que acabamos de crear, no hay ninguna solución posible, es decir, verificar cuando no haya ningún producto derivable de las características y las relaciones entre ellas.
  + **Export all products:** Podemos llevar a una hoja de Excel todos los productos que pueden obtenerse de un modelo basado en características.
  + **Number of products:** Retorna el número de productos que pueden obtenerse de un modelo. [Ojo, esto no funciona dice cero. Sin embargo cuando se calcula la homogeneidad si aparece que hay un número de productos determinado, quizá hay problema en el retorno de la variable]
  + **Calculate Homogeneity:** Un modelo más homogéneo es uno con pocas variables únicas en un producto, una variable única es aquella que solo aparece en un producto. Computamos la homogeneidad así: [Debo preguntar que es mejor, tener homogeneidad cercana a uno o no, diversificar los productos o mantenerlos relacionados]
  + **Calculate the variability factor:** El factor de variabilidad indica cuan diversificados son los productos, esta proporción se calcula así:
  + **Calculate extra constraint representativeness (ECR)**
  + **Identify Lowest Common Ancestor**
  + **Direct Root Feature**
* **SAS Verification:** Soft-dependencies expresan el nivel requerido de satisfacción en un modelo en lugar del nivel esperado, es decir, con una condición específica podemos establecer el grado requerido que se espera cumplir en los objetivos implícitos.
  + **Identify SoftDeps Always Active:** Esta opción permite verificar cuáles son esas dependencias que estarán presentes en cualquier producto, son importantes porque son los atributos que sin lugar a duda deben ser garantizados.
  + **Identify SoftDeps Never Allowed:** Esta opción permite verificar cuáles son esas dependencias que no estarán presentes en algún producto gracias al nivel que se les ha establecido.
  + **Identify Claims Always Active:** Los ***claims*** indican las asunciones que se hacen acerca del trabajo de las ***operationalizations****,* esta opción nos permite verificar cuales son las claims que siempre están activas.
  + **Identify Claims Never Allowed:** Verifica cuales son las asunciones que nunca están activas para las ***operationalizations.***
  + **Identify SG Contribs with Conflict:** Verifica si tenemos problemas o errores a nivel de especificación de *Soft-Goals.* Problemas generados por los tipos de relaciones que se establecen en los *Soft-Goals*[**Contribution**, **Require**]
  + **Identify Claims with Conflicts:** Verifica conflictos generados en *Claims,* los conflictos en este caso también son de carácter relacional establecido junto a las ***operationalizations.***
  + **Identify SoftDeps with Conflicts:** Verifica si existen problemas con las dependencias, tanto de constante activación como de nula aparición en los productos.
* **General Verification**
  + **Update core elements:** Al hacer uso de esta funcionalidad, podemos ver en verde, cuáles son los elementos o características que estarán de forma invariable en todos los productos generados.
  + **Identify Variant Features:**Al hacer uso de esta funcionalidad, podemos ver en color gris, cuales son los elementos o características que podrían estar presentes o no en algún producto.
  + **Identify Dead Elements:** Los elementos muertos, son características que nunca pueden existir en un producto, por ende, son innecesarias en el modelo al usar esta función podríamos verificarlas para eliminarlas o replantear una solución.
  + **Identify False Optionals:** Algunas características podrían verse comprometidas cuando son opcionales y una característica obligatoria o de núcleo las excluye.
  + **Verify Wrong Cardinalities:** Verifica cuando en un modelo tenemos cardinalidades que no tienen sentido, es decir, de una característica se pueden desprender un número ***n*** de características cuando la característica padre está disponible en un producto, sin embargo en la LP este número ***n*** nunca es alcanzado.
  + **Identify False Product Line:** Es una línea de producto llena de errores, o incluso que no genera ningún producto, y no está cumpliendo su objetivo.
  + ****

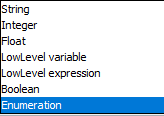
Tenemos un rango de vistas más amplio en donde podremos definir nuestro modelo basado en objetivos, dependiendo la vista en la estemos trabajando la paleta cambiará .Las responsabilidades para cada vista son las siguientes:

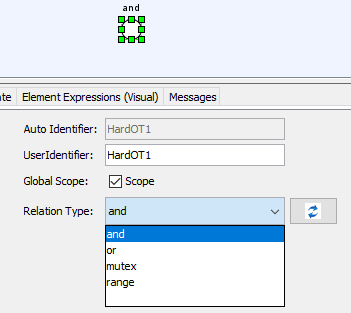
* **Domain goals:** En esta vista definimos los objetivos que especifican el problema inicial, y con un recorrido *“Top-down”* se establecen las relaciones en alto nivel para modelar la solución. La paleta de esta vista se describe a continuación:



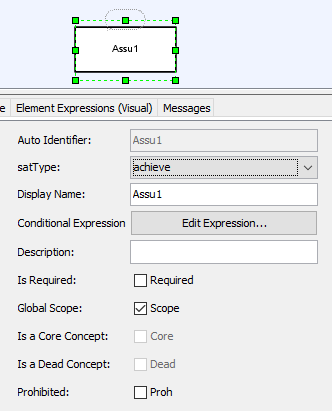
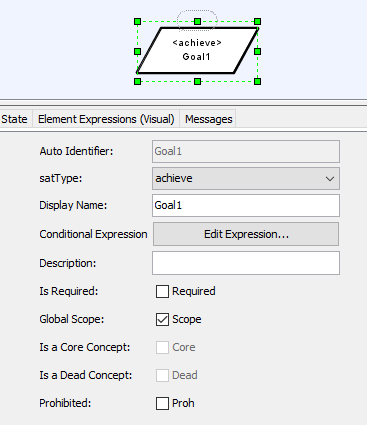
* + **OPER:** Son las formas más atómicas que especifican la solución, y varían desde la estructura planteada en hardware hasta el algoritmo con el que se abordará el problema, para facilitar el cumplimiento de los objetivos. Cuando hacemos uso de una *operationalization,* podremos definir lo siguiente.
    - ***satType****:* El tipo de satisfacción para la **operationalization** *,* es decir dentro de su rol de implementación, como deberá cumplirse.
      * **Achieve (++):** Garantiza que esta OPER debe ser alcanzada en los modelos.
      * **Maintain( = ):** Esta OPER deberá mantenerse constante dentro del modelo, incluso cuando variaciones externas ocurren.
      * **Optimize (+):**
      * **Avoid (--):**
      * **Cease ( - ):**
    - ***Display Name:***Es el nombre visible dentro del modelo
    - ***Description:***Por defecto esta descripción es igual a el identificador que VariaMos establece (OPERX), pero puede ser editada por el usuario para describir algo más propio de su dominio.
    - ***Is required:***Convierte la **operationalization** en una de núcleo, es decir que debe estar presente en todos los modelos.
    - ***Prohibited****:* Cuando seleccionamos esta opción para alguna característica, automáticamente la excluimos de existir en algún producto.
    - ***Attributes****:* Podemos establecer atributos que permitan el manejo generalizado de una **operationalization**.

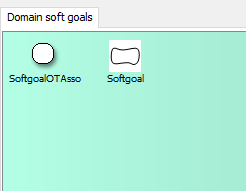
**

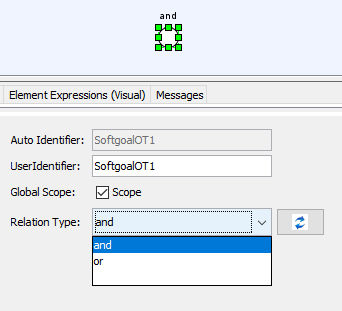
* + - * **Name:** El nombre visible de la OPER , también puede alcanzarse desde aquí
      * **Value:** Las OPER pueden tener valores enteros, que indican el grado o importancia que tiene esta dentro de la definición del modelo.
      * **Externally Visible:** Al seleccionar este atributo, la OPER desaparece del modelo ( el identificador del usuario se vuelve invisible ).
      * **Variable Type:** Los tipos de variable, son similares a los de cualquier lenguaje de programación
      * ****
        + **String:** Es una cadena de caracteres. (Ejm : “Base de Datos”)
        + **Integer:** Es un número perteneciente al conjunto de los enteros
        + **Float:** Números de punto flotante (Ejm : 3,14)
        + **LowLevel Variable:**
        + **LowLevel Expression:**
        + **Boolean:** Es un valor de verdad (True) o falsedad (False).
        + **Enumeration:** Es un dato definido por el usuario, su definición se especifica dentro de la vista de contexto y ambiente.
    - **Enumeration:** Un atributo para una OPER, puede ser una enumeración, sin embargo la definición de la enumeración debe existir previamente. Su definición se puede lograr desde la vista de ***Context and environment***
  + **HardOverTwo:** Es el elemento intermedio que permite definir las siguientes operaciones entre dos OPERs



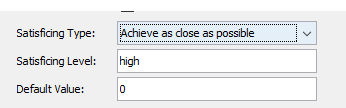
* + - ***And:*** Si la OPER padre se incluye en una configuración, también se incluyen todas las OPER hijas en dicha configuración.
    - ***Or:*** Una OPER hija tiene una relación opcional con su padre cuando la OPER hija puede o no, estar incluida en todos los productos en los cuales su OPER padre aparece.
    - ***Mutex:*** Un conjunto de OPER hijas tienen una relación ***Mutex*** con su padre cuando solo una de las OPER hijas puede ser seleccionada cuando la OPER padre es parte de un producto.
    - **Range:** Permite especificar el mínimo y máximo número de OPER hijas que se pueden incluir en una configuración cuando se selecciona la OPER padre en dicha configuración.
  + **Goal:** Especifica los objetivos de nuestro modelo, los objetivos pueden jerarquizarse haciendo uso de las relaciones especificadas en ***HardOverTwo...***Los objetivos en la parte superior de la jerarquía, serán aquellos en más alto nivel, y los que estén más abajo son los objetivos que ayudarán a que los principales se alcancen.
  + **Assumption:** Es una condición que debe ser verdad respecto a un objetivo, o a un OPER para que se pueda cumplir.
    - **Display Name:** El nombre que estableceremos para nuestro objetivo o asunción.
    - **Description:** En este campo describimos el objetivo o la asunción en términos de nuestro dominio específico.
    - **Is Required:** Convierte el objetivo/asunción en uno de núcleo, es decir, que debe satisfacerse en todos los productos.
    - **Is a core concept:** Esta opción informa si el objetivo o asunción, es uno de núcleo dentro del modelo.
    - **Is a Dead concept:** Esta opción informa que el objetivo o asunción, nunca existirá en algún modelo, bien sea por incoherencia en las relaciones o por señalarse prohibida.
    - **Prohibited:** Cuando seleccionamos esta opción para algún objetivo o asunción, automáticamente la excluimos de existir en algún producto.

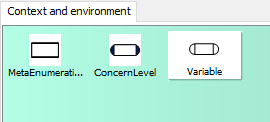


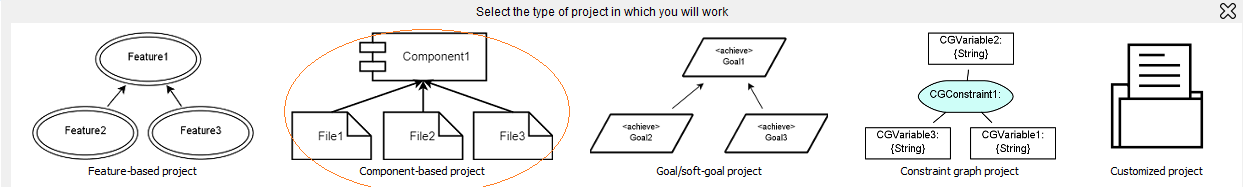
* **Domain soft goals:** Especifica los requisitos no funcionales del sistema, aunque no son explícitamente programados o construidos dentro del campo hardware o software, estos constituyen los objetivos implícitos para garantizar que los objetivos se alcancen apropiadamente. La paleta de esta vista se define a continuación:
  + 
  + **SoftgoalOTAssoa:** Permite especificar el tipo de relación que tendrán dos “objetivos implícitos o suaves”, por ejemplo, dos objetivos que podrían considerarse son “tolerancia a fallos” y “seguridad”.



* + **Softgoal:** Define el objetivo implícito perse, desde una perspectiva de los usuarios finales, incluye un nivel que permite medir la satisfacción. Es importante resaltar los siguientes atributos:

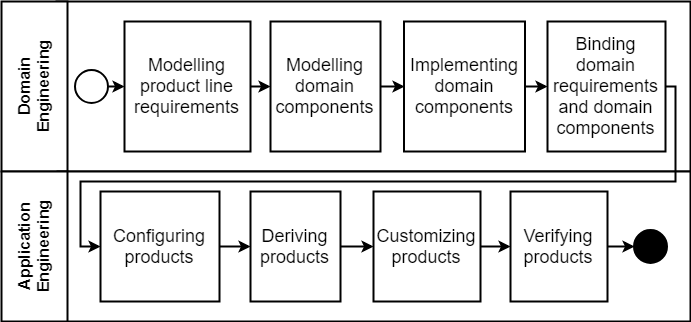


* + - **Satisficing Type:** Especifica el tipo de satisfacción, indicando cuáles objetivos deben cumplirse o mantenerse, el alcance de estos objetivos está categorizado o adjetivado por.***as low / high / close as possible.***
    - **Satisficing Level:** Nivel de satisfacción es subjetivo, sin embargo, para manejar un lenguaje común se recomienda clasificar en: ***as low, high, close.***
    - **Default Value:** Define la importancia de un requisito no funcional u objetivo suave, el valor es establecido por el creador del modelo, por defecto es cero.
* **Context and environment:** Esta vista contiene la definición de variables contexto que el usuario puede añadir a su modelo, dentro de la paleta de los elementos del contexto y ambiente se tiene:
  + **Meta-Enumeration:** Permite la creación de una enumeración definida por el usuario, la enumeración contiene un atributo que establece un identificador y un nombre que posteriormente estarán en la vista de contexto.
  + **ConcernLevel:** Un ***ConcernLevel***se define para asociar variables con características comunes. El tipo define si las variables son detectadas o perfiladas; en el primer caso, son modificadas por el sistema o el entorno; en el segundo caso, los define el administrador o el usuario. El alcance puede ser local o global, el primero significa que el valor es independiente para cada usuario, mientras que las variables globales se comparten entre todos los usuarios.
    - **Display name:** Es el nombre que identifica desde la perspectiva del usuario un ***ConcernLevel***
    - **Description:** Permite escribir de forma detallada el propósito de este ***ConcernLevel***
    - **Variable:** Una variable es una abstracción de un componente del sistema o entorno que son relevantes para el sistema. Los valores variables deberían simplificarse tanto como sea posible teniendo en cuenta los requisitos de modelado
    - **Variable Type:**
      * **String:** Es una cadena de caracteres. (Ejm : “Base de Datos”)
      * **Integer:** Es un número perteneciente al conjunto de los enteros
      * **Float:** Números de punto flotante (Ejm : 3,14)
      * **Boolean:** Es un valor de verdad (True) o falsedad (False).
      * **Enumeración:** Es especificado por el usuario.
* **Claims:** Dentro de la vista de claims se indican las asunciones que se hacen acerca del trabajo de las ***operationalization*** para satisfacer y cumplir los objetivos implícitos (Soft Goals). Dentro de la paleta de Claims tenemos:
  + **Claim:** Un *claim* incluye un grupo de operacionalizaciones y una condición lógica para evaluar la satisfacción del *claim*. El *claim* se activa solo cuando se seleccionan todas las operacionalizaciones y la expresión condicional es verdadera. El *claim* incluye una relación con un softgoal (SG). Dentro de *VariaMos* es importante resaltar de las *claims,* lo siguiente. (Las demás características han sido discutidas en varias ocasiones previamente [Display name, Description, Is required…etc])
    - **Cond Expression Text:** Permite dentro de la vista, debajo del ***Display name,*** agregar la condición que hemos establecido, pero en texto, es decir, información adicional.
    - **OPER:** Una ***operationalization*** permite la satisfacción parcial o completa de un objetivo u otra operacionalización. Si se cumplen las operacionalizaciones, de acuerdo con la relación definida, el objetivo asociado también se cumplirá.
  + **OperClaimOver:** Define las relaciones que se pueden establecerse entre ***operationalizations*** y ***claims.*** Las relaciones soportadas son **and, or, mutex, y range** que han sido discutidas previamente, solo que ahora están en función de OPERs y CLAIMS.
  + **SoftGoal:** Su definición puede encontrarse en la vista ***Domain soft goals.***
  + **Soft Dependencies:** Expresan el nivel requerido de satisfacción en lugar del nivel esperado, es decir, con una condición específica podemos establecer el grado requerido que se espera cumplir en los objetivos implícitos. ***Soft Dependency*** se activa solo cuando la expresión condicional es verdadera. El SD incluye una relación con un softgoal (SG)
    - **Cond Expression Text:** Permite dentro de la vista, debajo del ***Display name,*** agregar la condición que hemos establecido, pero en texto, es decir, como información adicional
* **Domain components**



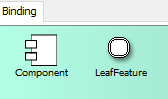
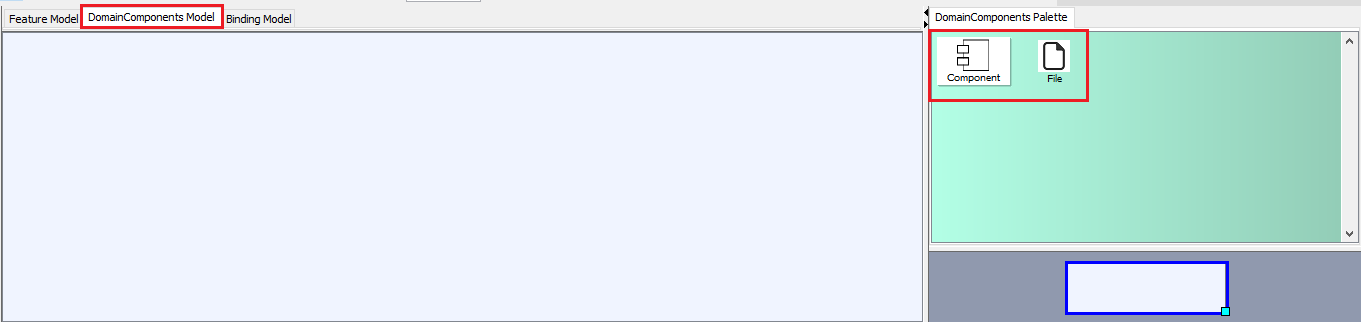
**Component-based project**

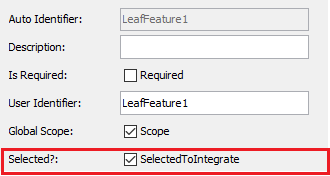
El modelado basado en componente, sigue el siguiente flujo:



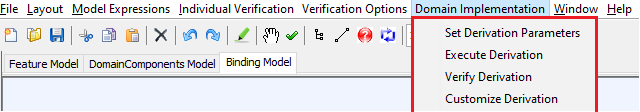
Dentro de este flujo, es importante modelar los componentes de un producto, por ello una de sus vistas es ***Feature Model,*** la cual ha sido discutida previamente. En este caso en lugar de características tendremos componentes, las demás premisas continúan invariables.

Las diferencias empiezan en el modelado de los componentes, la vista y su respectiva paleta luce así:

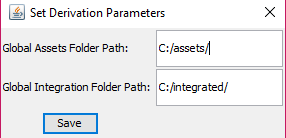
* **Domain Components Model:** Un componente puede ser asociado con uno o varios archivos.
  + **Component:** Representa un directorio donde se almacenan los archivos del componente.
    - **Id:** Es el nombre que en la vista tendrá este componente. Este nombre debe ser igual al nombre del directorio donde está almacenado los archivos del componente.
  + **File**
    - **Id:** Es el nombre que en la vista tendrá este componente. Se sugiere colocar el identificador como una combinación del nombre del componente y el nombre del archivo. Por ejemplo: si el componente se llama Login, y el archivo vista\_login.jsp, entonces el Id del archivo debería ser Login-VistaLogin.
    - **FileName:** El nombre del archivo que se almacena en el directorio. Ejm “ClaseX.java”, “Doc.json”, “vista\_login.jsp”.
    - **Destination:** Define la ruta en la cual se derivará el archivo. Por ejemplo: si se trabaja con un arquitectura MVC y el archivo actual es una vista, posiblemente ese archivo se vaya a derivar en una ruta como esta “WebContent/views/vista\_login.jsp”.
* **Binding Model:** En esta vista se enlazan los componentes con las características hoja.
  + **Component :** Los componentes definidos en esta vista o en la anterior serán los mismos pero ahora podremos relacionarlos con las ***Leaf features*** definadas en la vista ***Feature Model***. La relación entre una característica hoja y un componente es 1 a 1 por el momento. Y la dirección del enlace va desde un componente a una característica hoja.
  + **Leaf Feature:** A las leafFeatures se le agregaron una opción nueva llamada “SelectedToIntegrate”. Esta opción permite configurar un producto. De tal manera de que si queremos que un producto contenga unas características particulares, entonces debemos seleccionarla para ser integrada, y verificar que la flecha esté en verde. Esto permitirá la integración de esta característica.



Dentro del menú principal notamos una nueva opción para este tipo de modelado.

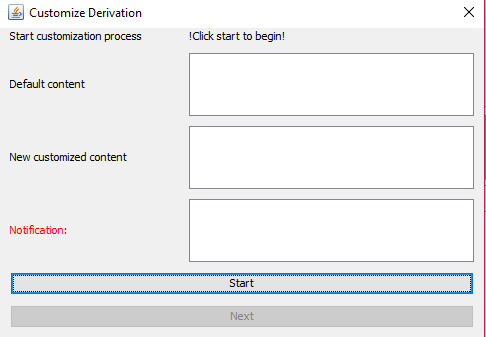


* **Set Derivation Parameters**
  + **Global Assets Folder Path:** La ruta en donde se encuentran almacenados los componentes y sus archivos.
  + **Global Integration Folder Path:** Especifica la ruta en donde, una vez ensamblado todo, la derivación quedará guardada.



* **Execute Derivation (solo funciona con componentes desarrollados bajo FragOP):** Al terminar los componentes y después de relacionarlos con las características, entonces habremos terminado y es posible ejecutar el ensamble de nuestro modelo. Una vez hecho esto, podremos ir a la ruta que especificamos de salida y verificar que todo ha sido ensamblado exitosamente.
* **Verify Derivation:** Verifica la sintaxis de los archivos derivados. VariaMos utiliza ANTLR y unos lexers y parsers para verificar la sintaxis de cada archivo basado en su extensión.
* **Customize derivation (solo funciona con componentes desarrollados bajo FragOP):** Al desarrollar componentes con FragOP se pueden definir unos puntos de personalización. Estos puntos indican lugares en el código que se deberían personalizar después de la derivación. Tales como: variables que deben ser parametrizadas, textos de ejemplo, entre otros. Después de dar clic en Start y si existen puntos de personalización, entonces se muestra lo siguiente:
  + **Default content:** Muestra el código por defecto del archivo sin personalizar y en un campo de solo lectura.
  + **New customized content:** Muestra el código por defecto del archivo sin personalizar y en un campo que se puede editar. Aquí el usuario puede editar el código para personalizarlo a su gusto.
  + **Notification:** Muestra avisos, de si se terminaron todos los puntos de personalización o si se encuentran errores.

Una vez modificados el contenido dentro de los puntos de personalización, se modifican los archivos derivados.



**Simulación**

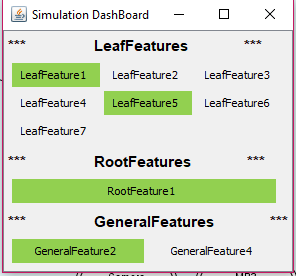
Para simular el modelo que hemos terminado, tenemos que movernos de perspectiva, la simulación permitirá observar todos las configuraciones posibles que se obtienen del modelo.

Y encontraremos la siguiente **barra de herramientas** en la perspectiva de simulación:

1. **:** Nos regresa a la primera configuración del modelo
2. **:** Nos enseña una configuración aleatoria del modelo
3. **:** Enseña la primera configuración del modelo, los elementos pertenecientes a esta configuración se marcarán con el siguiente símbolo sobre ellos.
4. **:** El zoom permite disminuir y aumentar el tamaño del modelo sobre la vista.

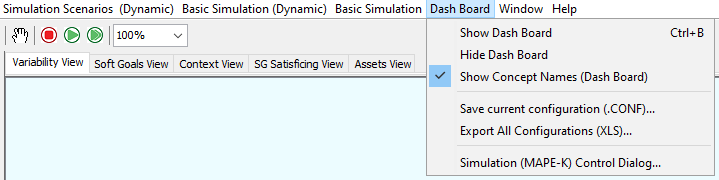
De esta manera, el desarrollador de SPL puede simular diferentes configuraciones de productos.

Dentro de la simulación podemos encontrar un “*Dashboard”,* que puede lucir así:



Este ***dashboard*** es generado por un modelo de características, sin embargo los elementos o componentes que estén seleccionados en una configuración determinada aparecerán resaltados en verde.

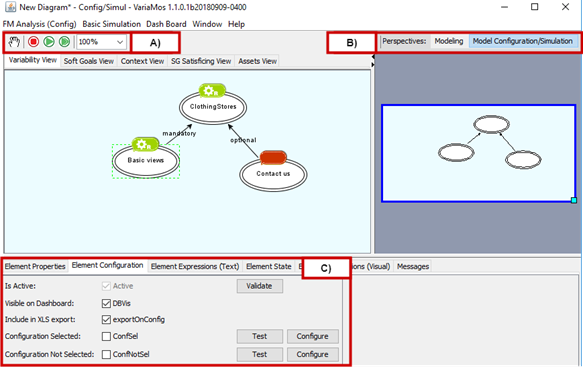
Dentro del menú de simulaciones, la pestaña **Dashboard,** ofrece algunas opciones que no pueden ser alcanzadas desde la barra de herramientas:



* **Show dashboard:** Muestra el tablero con las configuraciones disponibles de un producto en específico.
* **Hide dashboard:** El tablero de simulación puede esconderse, si se quieren ver los elementos seleccionados desde la vista de variabilidad.
* **Save current configuration:** La configuración del modelo actual, permite exportarla y guardarla.
* **Export All configurations:** Exporta todas las posibles configuración del modelo en formato XLS, soportado en excel.

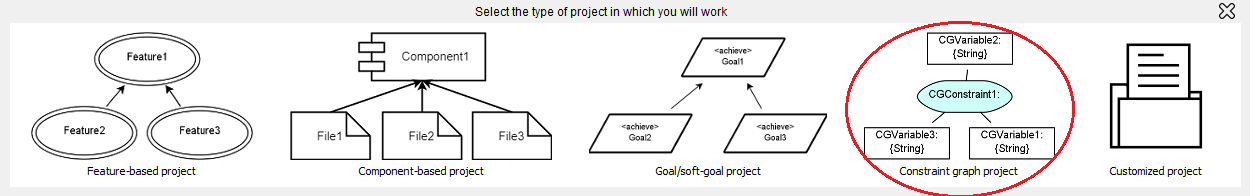
Cuando estamos haciendo configuraciones en la simulación, encontraremos que nuestras características están marcadas con diferentes símbolos, los cuales corresponden a estos significados:

* Las características que tienen un símbolo rojo sobre ellas indican que no formarán parte de la especificación de requisitos del producto que se está configurando.
* Las características con un símbolo verde y una letra "C" sobre ellas son aquellas que seleccionamos para ser parte de la especificación de requisitos del producto que se está configurando.
* Las características con un símbolo verde y una letra "R" sobre ellas son aquellas que fueron seleccionadas automáticamente por la herramienta VariaMos porque eran obligatorias en el modelo de características SPL.
* Las características con un símbolo verde y una flecha sobre ellas son aquellas que la herramienta VariaMos seleccionó automáticamente gracias a los mecanismos de propagación de restricciones del *SOLVER* que respalda la actividad de configuración.



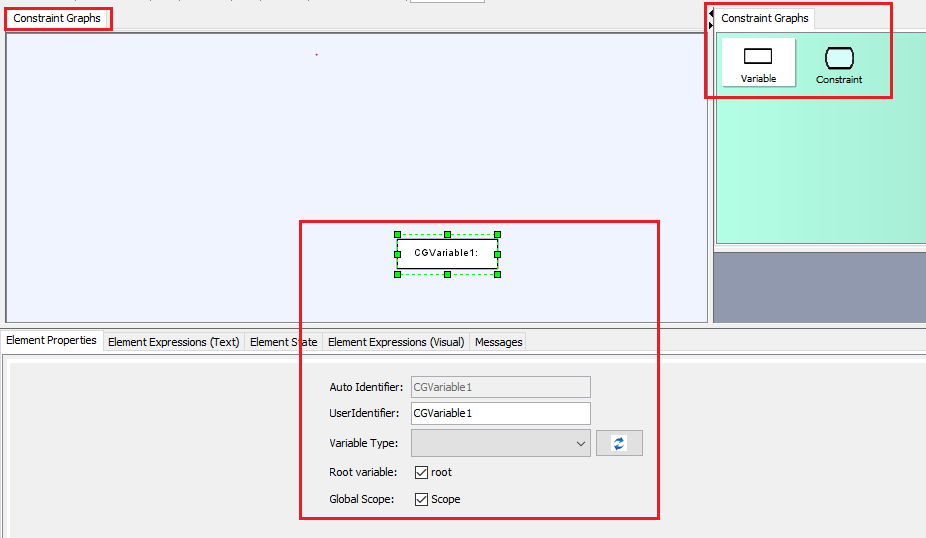
**Constraint graph project**

Un *grafo de restricciones* puede ser usado para representar relaciones entre variables que están sujetas a conjunto de restricciones. En *VariaMos,* esta funcionalidad puede seleccionarse en el proyecto que se indica a continuación.



Una vez el proyecto fue seleccionado la interfaz que se presenta ofrece la posibilidad de añadir variables y restricciones a estas variables:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ No funciona correctamente \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_OJO\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_:

Se necesita entender el tipo de variables en conjunto con las restricciones que serán añadidas, es decir, no todas las relaciones se aplican a todos los tipos de variables, por eso se propone lo siguiente para acotar la cardinalidad de estas relaciones y lograr generar un grafo lógico que permita obtener configuraciones que cumplan con las condiciones especificadas por el usuario diseñador de SPL.

Los tipos de datos soportados en la herramienta son:

***String, Integer, Float, Boolean, Enumeración* [**Fueron explicadas anteriormente**]**

Los tipos de restricciones establecidas para el grafo son las siguientes, y podrían ser clasificadas por tipo de datos así:

* **Or:** Alguna de las dos variables, o ambas estén presentes en la configuración final. ***Tipo de datos:*** *Boolean o Enumeración definido por el usuario*
* **Greater:** Debe satisfacerse que una variable es mayor que alguna otra elegida. ***Tipo de datos:*** *Integer y Enumeración [Si es definida por el usuario ]*
* **GreaterOrEq:** Debe satisfacerse que una variable es mayor o igual que alguna otra elegida.***Tipo de datos:*** *Integer y Enumeración [ Si es definida por el usuario ]*
* **NotEquals:** Dos variables no pueden ser iguales.***Tipo de datos:*** *Integer , Boolean y Enumeración [ Si es definida por el usuario ]*
* **Equals:** Dos variables deben ser iguales. ***Tipo de datos:*** *Integer , Boolean y Enumeración [ Si es definida por el usuario ]*
* **And:** Que ambas estén presentes en la configuración final. ***Tipo de datos:*** *Boolean o Enumeración definido por el usuario*
* **Less:** Debe satisfacerse que una variable es menor que alguna otra elegida. ***Tipo de datos:*** *Integer y Enumeración [ Si es definida por el usuario ]*
* **LessOrEqual:** Debe satisfacerse que una variable es menor o igual que alguna otra elegida ***Tipo de datos:*** *Integer y Enumeración [ Si es definida por el usuario].*
* **Implies:** Una variable implica la existencia de otra en la misma configuración esta relación podría ser de cualquier tipo e implicar cualquier tipo dentro de la lógica de la SPL.
* **DoubleImplies**: Esta relación define pares de variables, es decir, variables que siempre que una es elegida para alguna configuración, la otra también debe existir.