

Ejercicio: Domain Specific Languages

Víctor Julio Ramírez Durán

Master en Sistemas Informáticos Avanzados

Universidad del País Vasco - EHU/UPV

Donostia - Gipuzkoa

Objetivo:

Crear un DSL y su transformación en código PlantUML usando el software JetBrains MPS. Se usará como ejemplo la siguiente notación de definición de entidades:

Ejemplo DSL

```
entity Account {  
    Int amount  
    String number  
    ref Customer [+] holders  
    val Transaction [*] transactions  
}  
  
entity Customer {  
    String name  
    String address  
    ref Account [+] accounts  
}  
  
entity Transaction {  
    Date date  
    Int amount  
}
```

Código PlantUML a generar

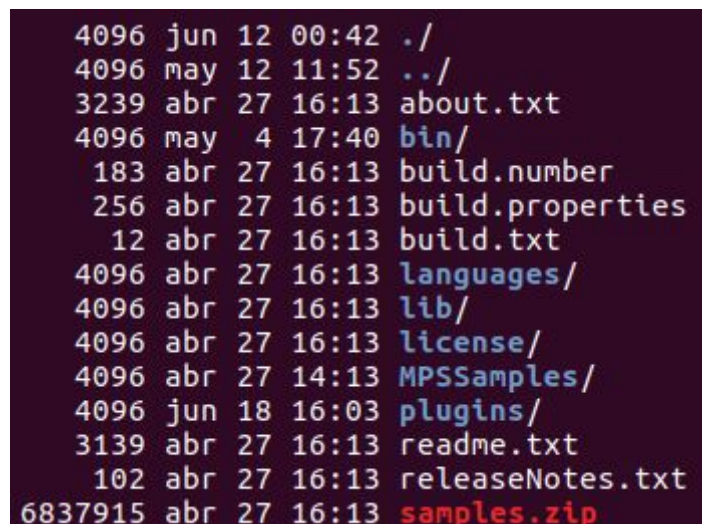
```
@startuml  
class Account {  
    Int amount  
    String number  
}  
Account -- "1" Customer : holders  
Account *-- "0..*" Transaction : transactions  
  
class Customer {  
    String name  
    String address  
}  
Customer -- "1" Account : accounts
```

```
class Transaction {  
    Date date  
    Int amount  
}  
@enduml
```

Prefacio: Creación del proyecto en JetBrains MPS

Se utilizará la versión de JetBrains MPS para Linux (ya que se trabajará sobre Ubuntu 17.04) la cual se descarga desde <https://www.jetbrains.com/mps/download/#section=linux>.

Una vez descargado el fichero MPS-2017.1.2.tar.gz (versión estable en el momento), procedemos a extraerlo en un directorio, donde encontraremos la estructura de ficheros de la imagen 1:



```
4096 jun 12 00:42 ./  
4096 may 12 11:52 ../  
3239 abr 27 16:13 about.txt  
4096 may  4 17:40 bin/  
 183 abr 27 16:13 build.number  
 256 abr 27 16:13 build.properties  
  12 abr 27 16:13 build.txt  
4096 abr 27 16:13 languages/  
4096 abr 27 16:13 lib/  
4096 abr 27 16:13 license/  
4096 abr 27 14:13 MPSSamples/  
4096 jun 18 16:03 plugins/  
3139 abr 27 16:13 readme.txt  
 102 abr 27 16:13 releaseNotes.txt  
6837915 abr 27 16:13 samples.zip
```

Imagen 1

Nos dirigimos al directorio *bin* donde ejecutaremos *./mps.sh*, una vez cargado MPS, seleccionamos *Create New Project*. Allí seleccionamos *Language Project*, indicamos un nombre para el proyecto, un nombre para el lenguaje, seleccionamos la casilla *Create Sandbox Solution* y le damos *OK* (Imagen 2).

Esto nos crea un proyecto con la estructura de la figura 3, en el módulo *structure* definiremos los conceptos del DSL, en el módulo *generator* definiremos la transformación y en el módulo *sandbox* probaremos el funcionamiento del DSL y la transformación generada.

Luego de los pasos anteriores se procede a crear el DSL.

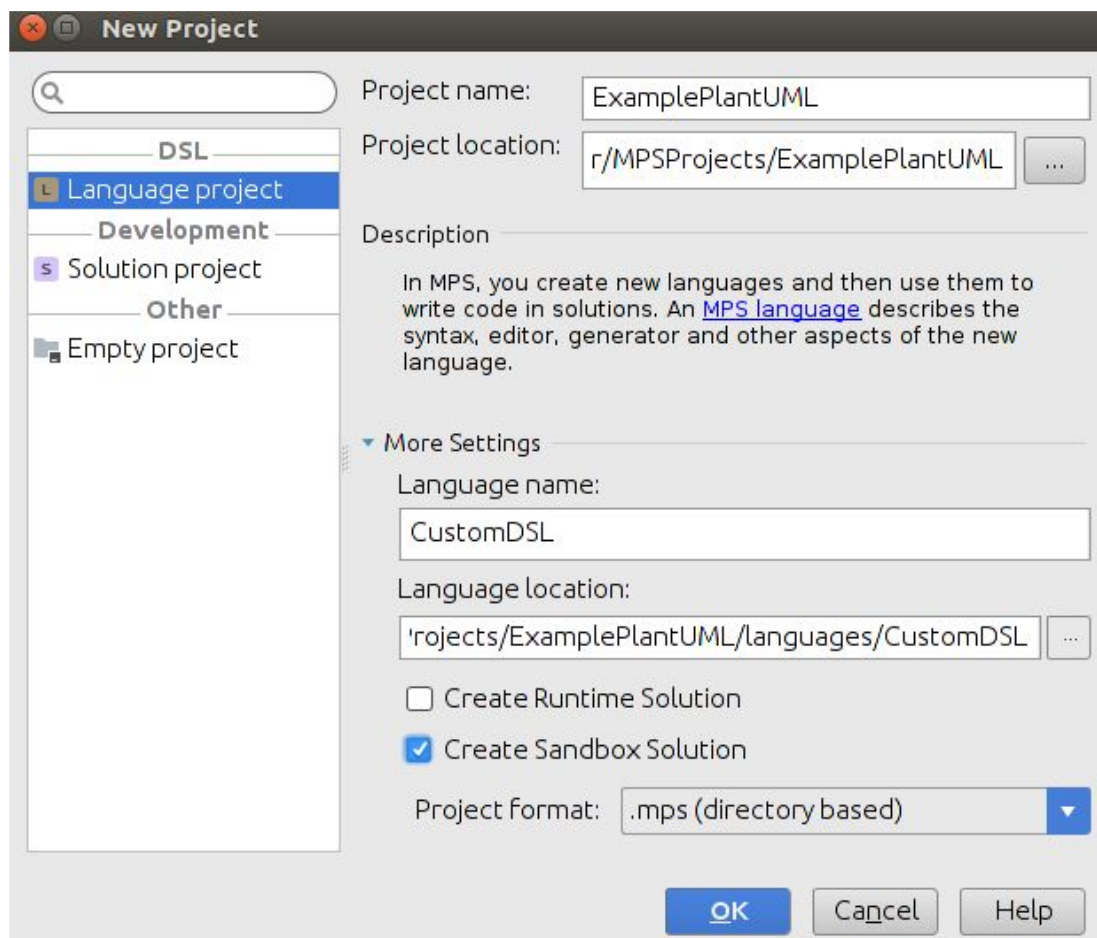


Imagen 2

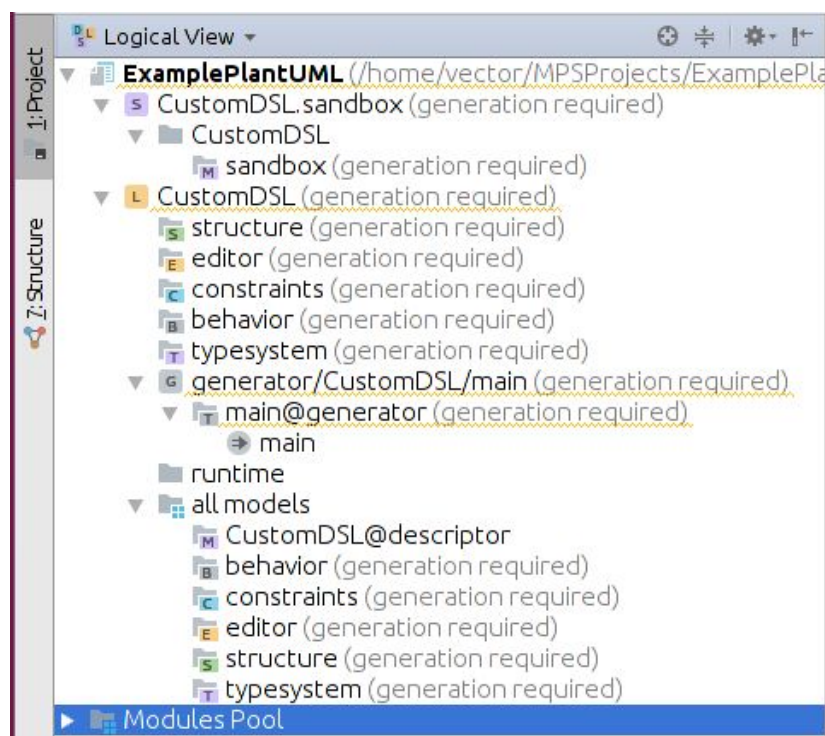


Imagen 3

Creación del DSL

Luego de los pasos anteriores se procede a crear el DSL. Como primera medida se deben generar todos los conceptos y sus editores, empezaremos con los conceptos básicos y luego los que envuelven a otros conceptos.

Sobre el módulo *structure* damos *click derecho* -> *New* -> *concept*. Esto nos genera un nuevo concepto vacío (Imagen 4).

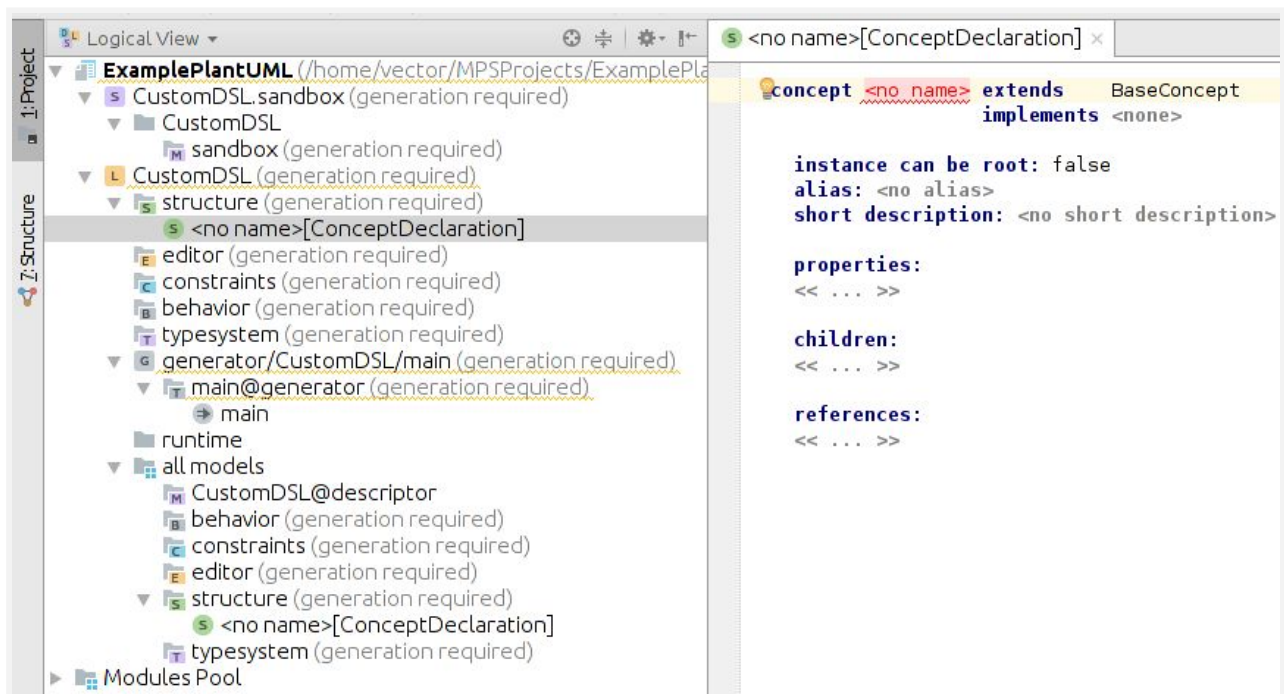


Imagen 4

Concepto: property

Creamos el concepto para las **propiedades** de las entidades de nuestro DSL, las propiedades se definen de la siguiente manera:

```
String number
Int amount
Date date
```

Podemos observar que constan de dos partes, el tipo de dato y el nombre. Entonces para este concepto necesitamos dos propiedades: tipo y nombre. Definimos entonces el concepto como se muestra en la figura 5.

Las propiedades de un concepto en MPS (no confundir con las propiedades de la entidad que estamos definiendo) constan igualmente de un nombre y un tipo. Al extender el concepto de *BaseConcept* tenemos disponibles los tipos de datos primitivos. Con la combinación de teclas *Ctrl* + *space* podemos acceder a la ayuda cuando tenemos el cursor sobre el tipo de datos (Imagen 6).

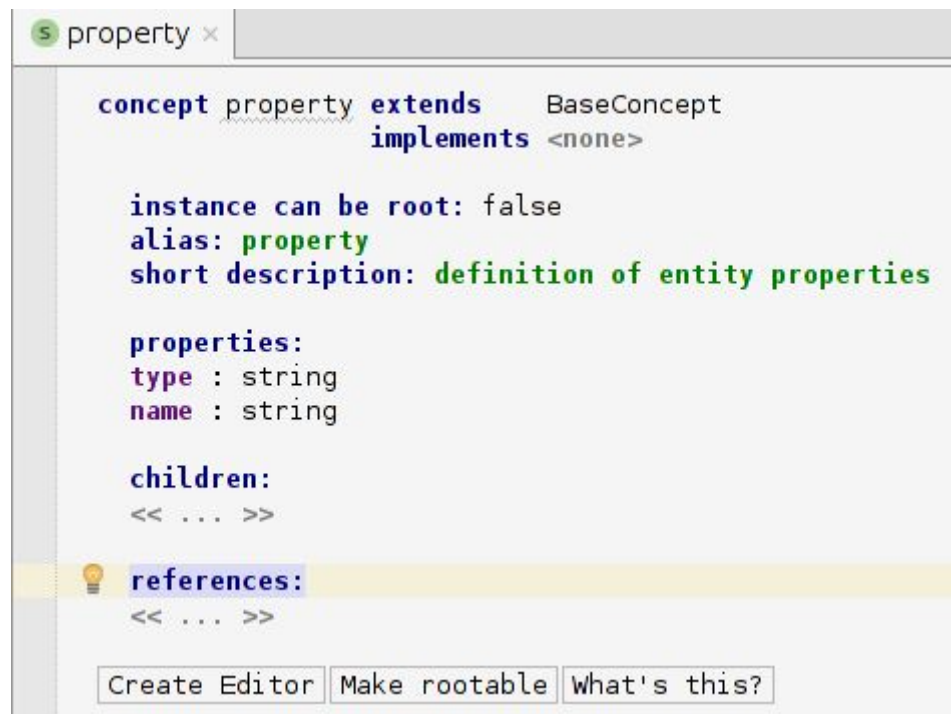


Imagen 5

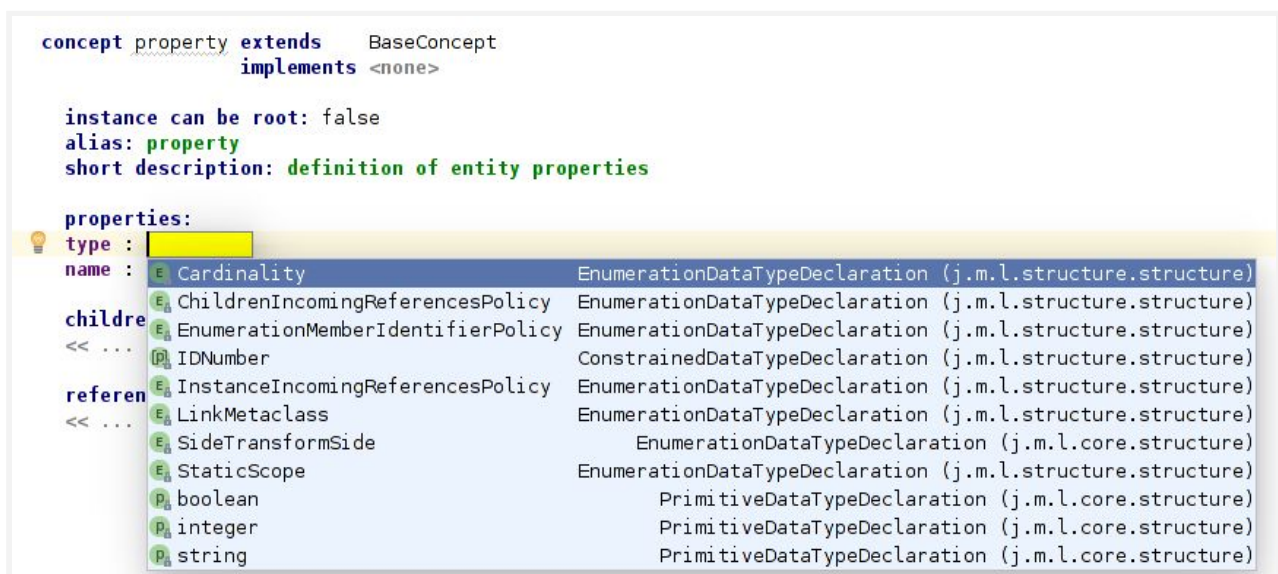



Imagen 6

Una vez creado el concepto procedemos a definir cómo debe interpretarse este concepto en el editor cuando estemos escribiendo ejemplos de nuestro DSL. En este paso se define la semántica que debe tener nuestro DSL, para el caso de las propiedades de entidades, definiremos que primero se debe escribir el tipo y luego el nombre.

Damos click en el símbolo + de color verde que se encuentra en la parte inferior  -> editor -> *Concept Editor*. Esto nos genera un nuevo editor para el concepto property (Imagen 7).

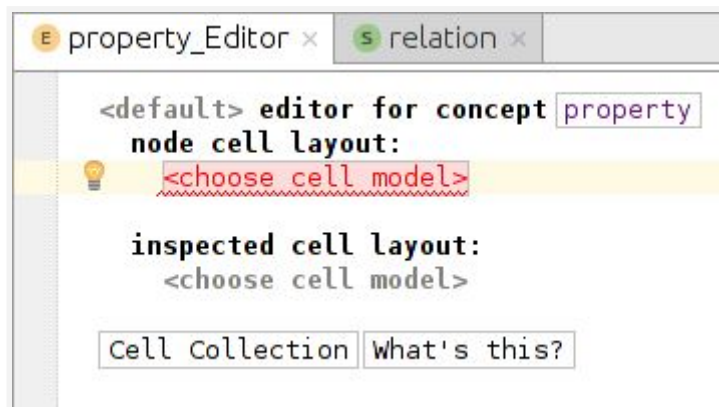



Imagen 7

Nos situamos en node cell model, y le damos click a la ayuda en forma de foco amarillo  y seleccionamos *Surround with Indent Collection*, con esto generamos una colección de términos que comienzan con una indentación [- -]. Luego utilizamos nuevamente la combinación de teclado para la ayuda (*Ctrl + space*) y vemos que tenemos disponibles las propiedades del concepto *property* que definimos anteriormente (Imagen 8), seleccionamos primero el tipo, luego damos Enter para generar un nuevo campo en el editor y luego el nombre usando la ayuda de nuevo.

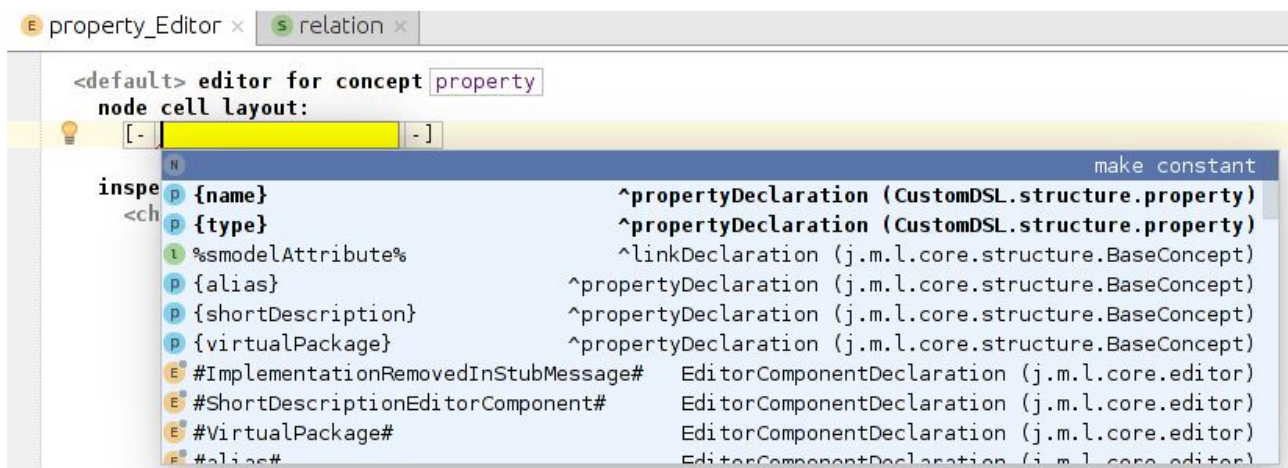



Imagen 8

También podemos indicar el estilo que queremos que tenga cada una de las propiedades del concepto que definimos en el editor, esto se hace haciendo click en el inspector situado en la esquina inferior derecha  2: Inspector . Nos situamos en el tipo [- { type }] y en la sección *Style* que nos muestra el inspector cambiamos el *base style* a *keyword* (Imagen 9). Con esto logramos que en el editor cada vez que escribamos un tipo para nuestra propiedad, esta se resalte como una palabra clave (bold y color púrpura).

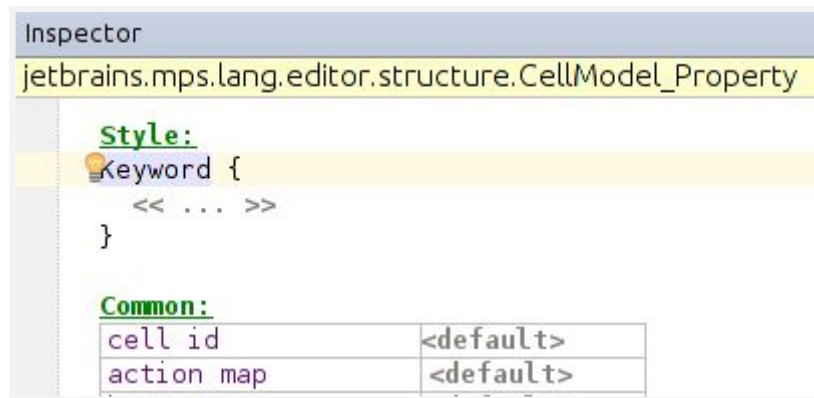


Imagen 9

De esta forma hemos definido la semántica para las propiedades de nuestras entidades (Imagen 10).

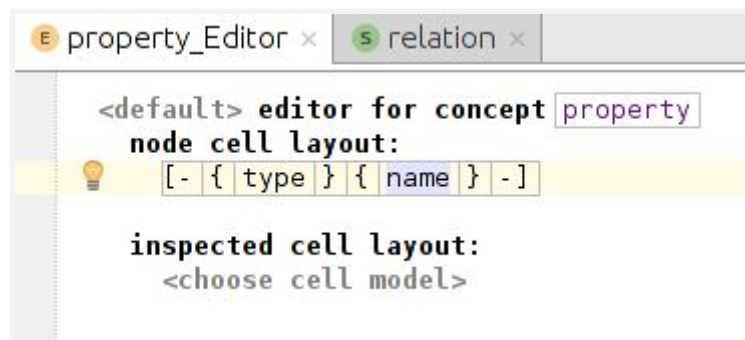


Imagen 10

Concepto: relation

Procedemos a definir el concepto **relaciones** de las entidades. Observamos que las relaciones se definen de la siguiente manera:

```
ref Customer [+] holders
val Transaction [*] transactions
ref Account [+] accounts
```

Constan de 4 partes: tipo, nombre de entidad a relacionar, cardinalidad, y nombre para la relación. Con esta definición procedemos a crear el concepto para las relaciones (Imagen 11).

Una vez creada la definición del concepto relation, procedemos en el editor a crear su visualización con el orden que se ve en el ejemplo (Imagen 12), definiremos un estilo base *keyword* de la misma manera que lo hicimos con el tipo de las propiedades, para el nombre de la entidad a relacionar no pondremos estilo base sino dejaremos el por defecto pero le añadiremos *font-style: italic* y la cardinalidad la dejaremos en Bold (Imagen 13).

Una vez terminado el editor y con el inspector añadidos todos los estilos pasamos al concepto entidad.

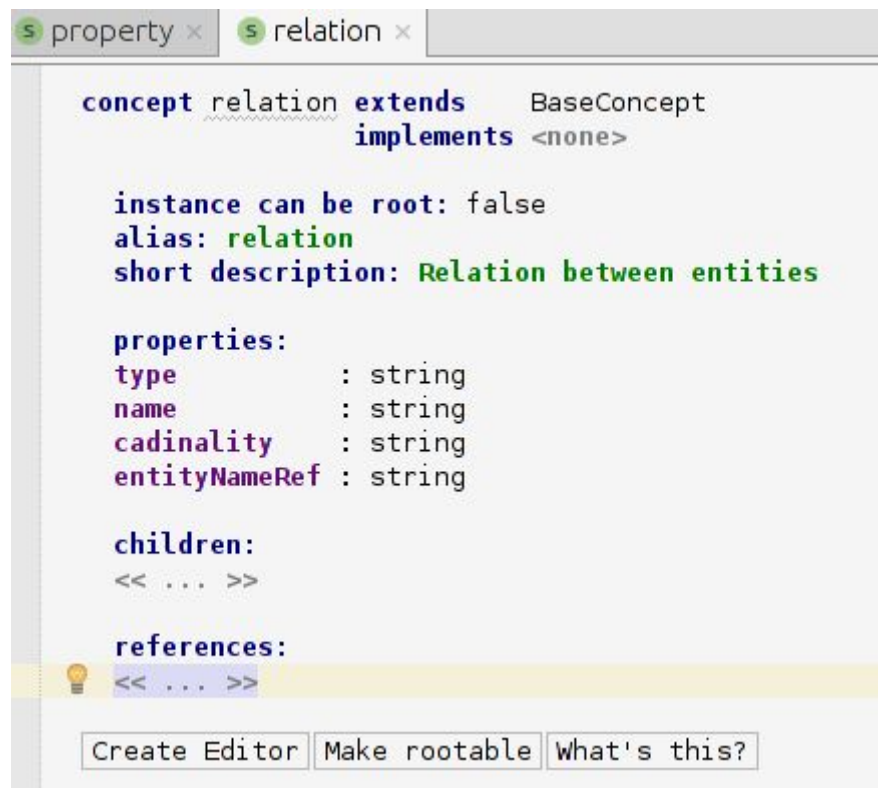


Imagen 11

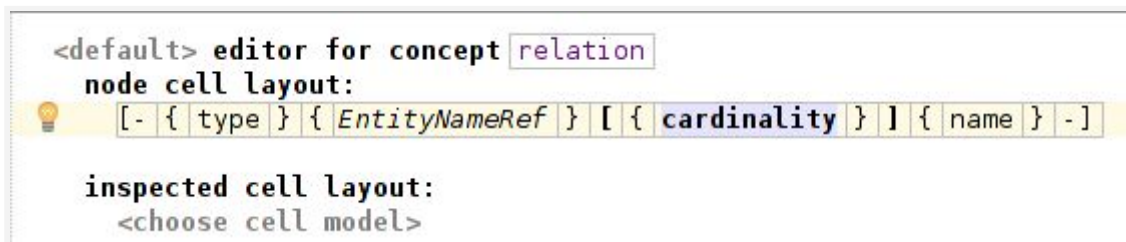


Imagen 12

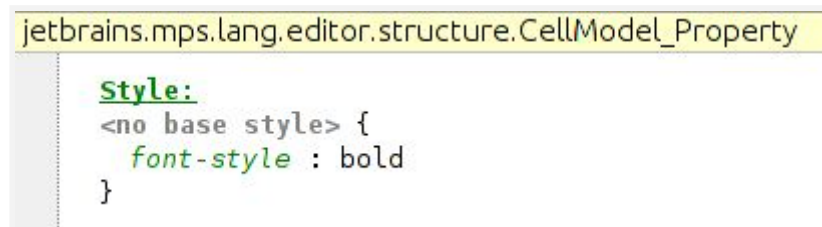


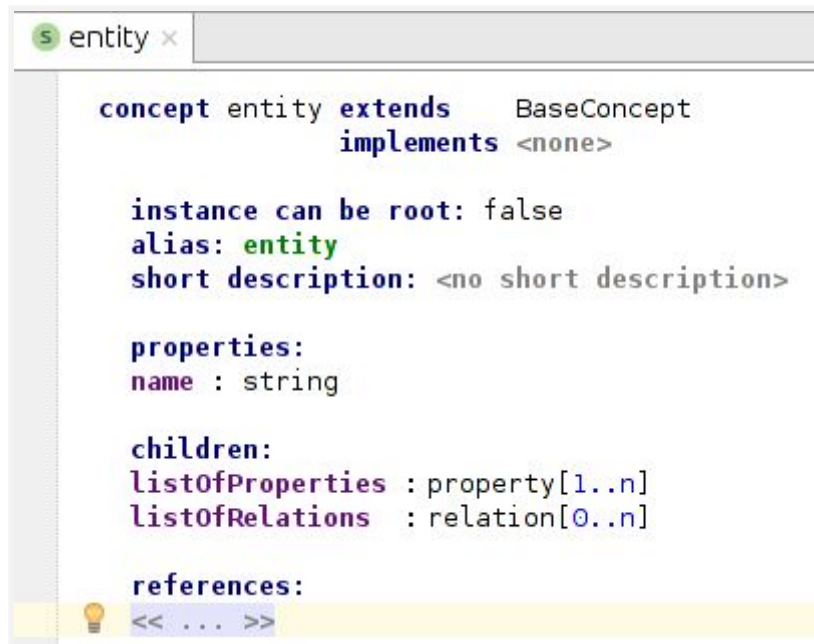
Imagen 13

Concepto: entity

Ahora procederemos a definir el concepto **entidad**. Las entidades están compuestas de un nombre, un listado de propiedades y un listado de relaciones. Al menos debe existir una propiedad en la entidad pero las relaciones pueden o no existir, esto lo tendremos en cuenta al definir la cardinalidad de los conceptos.


```
entity Customer {
    String name
    String address
    ref Account [+] accounts
}
```

Para el concepto entidad definiremos una propiedad nombre y dos hijos: el listado de propiedades y el listado de relaciones a los cuales les añadiremos la cardinalidad 1..n y 0..n respectivamente (Imagen 14).



```
concept entity extends BaseConcept
    implements <none>

    instance can be root: false
    alias: entity
    short description: <no short description>

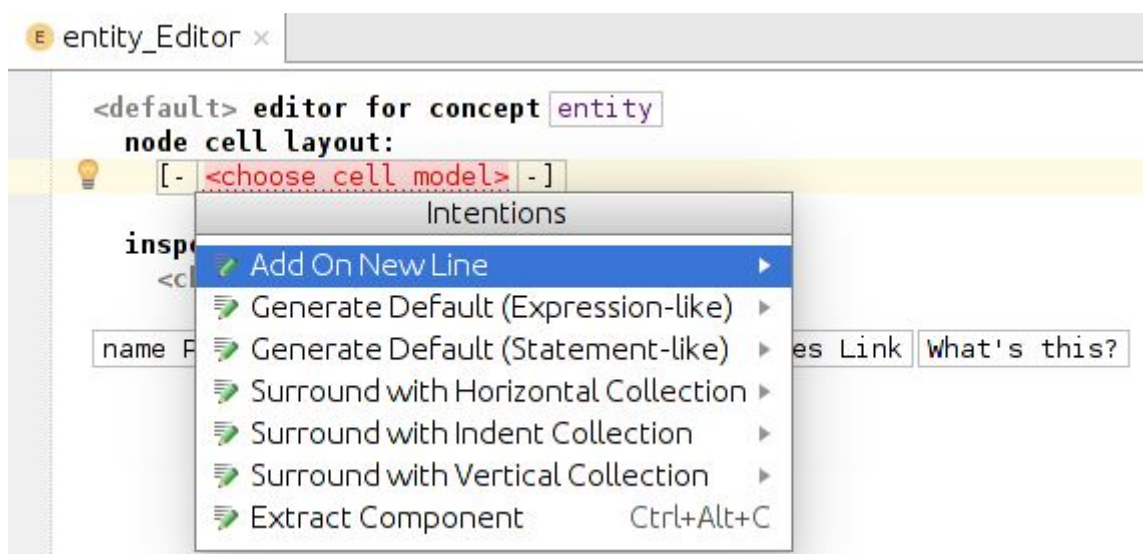
    properties:
        name : string

    children:
        listOfProperties : property[1..n]
        listOfRelations : relation[0..n]

    references:
        << ... >>
```

Imagen 14

Ahora en el editor procedemos a definir la semántica de las entidades, sabemos que debe empezar con un literal “entity” luego el nombre de la entidad seguido de llaves, dentro de las llaves van las propiedades y luego las cardinalidades.



```
<default> editor for concept entity
node cell layout:
[- <choose cell model> -]
```

Intentions

- Add On New Line
- Generate Default (Expression-like)
- Generate Default (Statement-like)
- Surround with Horizontal Collection
- Surround with Indent Collection
- Surround with Vertical Collection
- Extract Component Ctrl+Alt+C

name P es Link What's this?

Imagen 15

Para crear una nueva línea usamos el menú desplegable *Intentions* al que podemos acceder con la combinación *Alt + Enter* (Imagen 15) y seleccionamos *Add On New Line*.

Para insertar el listado de propiedades y de relaciones usamos el menú de ayuda e insertamos los *linkDeclarations* a los hijos creados en la declaración del concepto *entity* (Imagen 16).

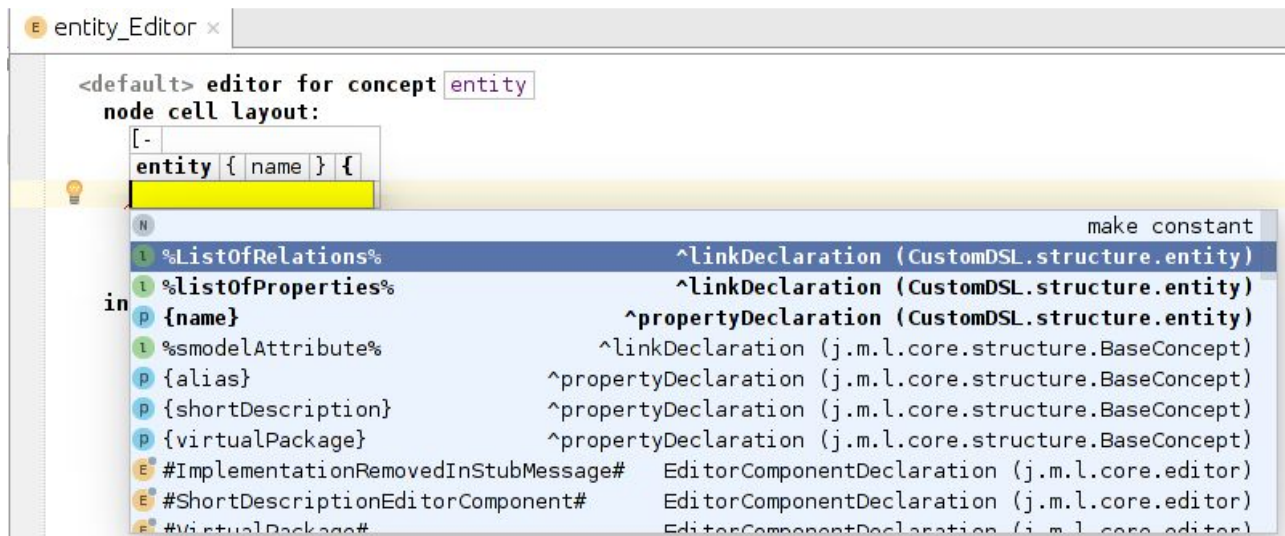


Imagen 16

Para finalizar, la semántica del concepto entidad debe quedar como se describe en la imagen 17.

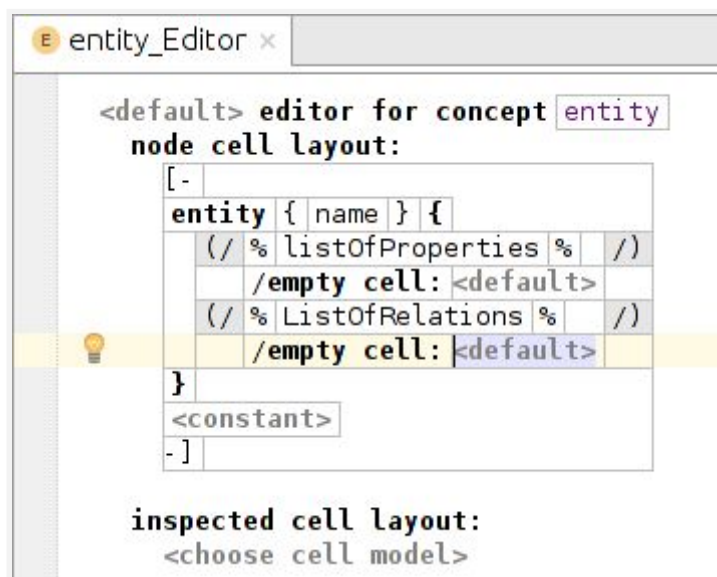


Imagen 17

Concepto: entities

Por último, definiremos un concepto *entities* que será el elemento raíz de nuestro DSL y encapsulará todas las entidades. En el editor no tendrá semántica alguna diferente al listado de entidades (Imagen 19) pero se usará en la transformación para definir las etiquetas *@startuml* y *@enduml* de inicio y cierre del código PlantUML.

En la definición del concepto pondremos que es el elemento raíz: *instance can be root: true*. Además en la sección de hijos añadiremos el listado de entidades (Imagen 18).

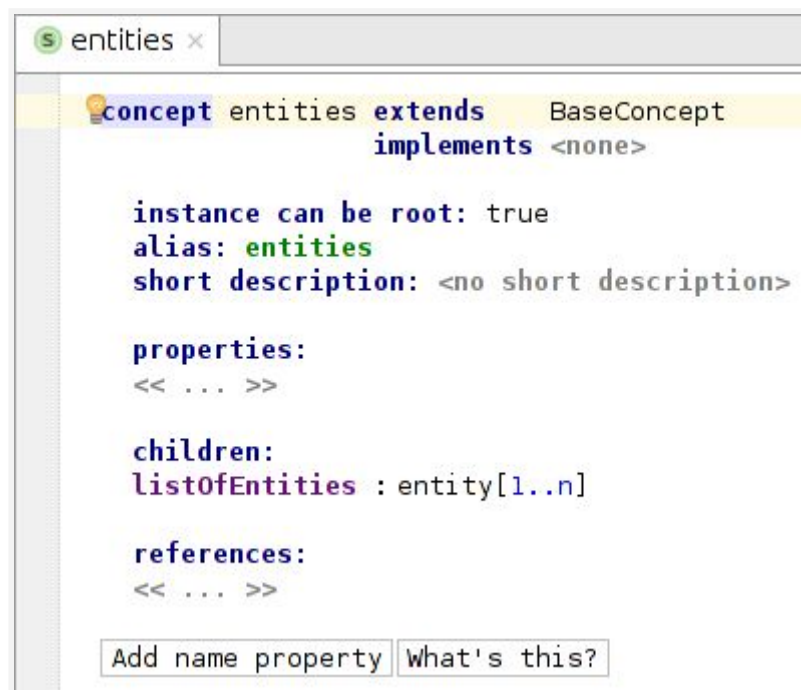


Imagen 18

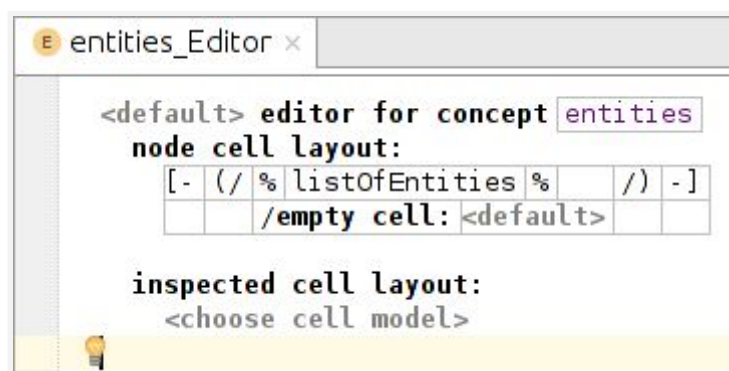


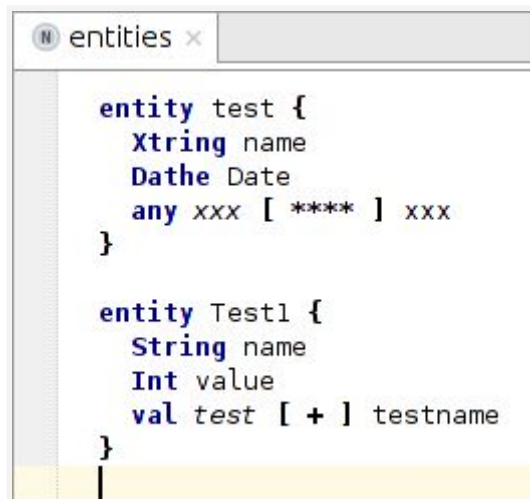
Imagen 19

Prueba DSL

Para probar nuestra definición de los conceptos primero nos situamos en el módulo principal del lenguaje y presionamos Ctrl + F9 para compilar la solución, vamos al módulo sandbox, damos *click derecho* -> *New* -> *entities*. Entities aparece en el menú New del Sandbox porque definimos que el concepto entities podía ser un elemento raíz.

Se nos despliega el editor con un ejemplo de una entidad, aquí podremos crear entidades, relaciones y propiedades según la definición de conceptos descritos anteriormente. Como podemos observar, aunque nuestro DSL mantiene la estructura semántica correcta, podemos definir tipos de datos erróneos como: *Xtring*, *any*, *Dathe*, así mismo como cardinalidades y

referencias incorrectas (Imagen 19). Para corregir esto, crearemos tipos de datos restringidos para las propiedades y relaciones.



```
entity test {  
  Xstring name  
  Dathe Date  
  any xxx [ **** ] xxx  
}  
  
entity Test1 {  
  String name  
  Int value  
  val test [ + ] testname  
}
```

Imagen 19

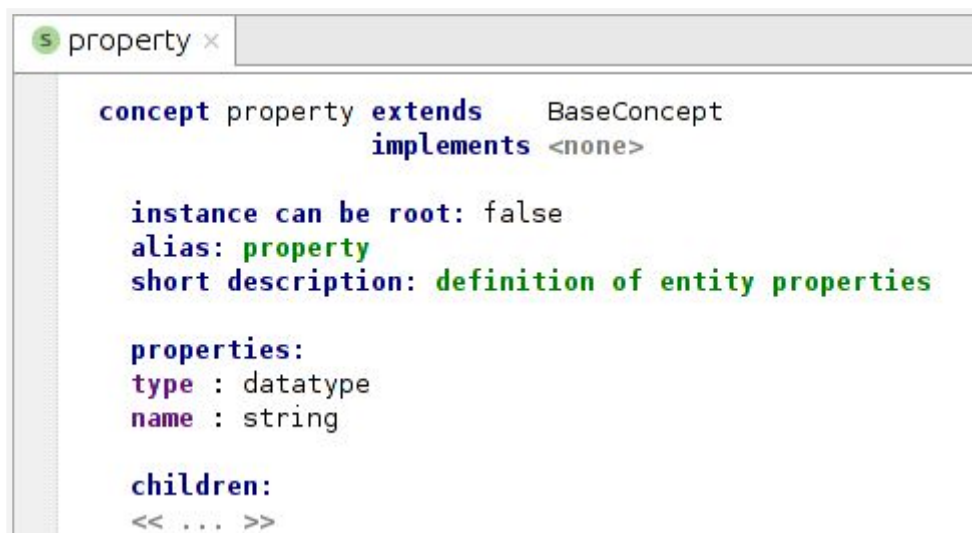
Para las propiedades crearemos un tipo de datos restringido dando click derecho sobre el módulo structure -> New -> Constrained Data Type. Indicaremos un nombre y una expresión regular para definir los tipos de datos que son permitidos (Imagen 20).



```
constrained string datatype: datatype  
  
matching regexp: String|Int|Date|Boolean
```

Imagen 20

Y ahora asociaremos este tipo de datos al tipo de datos del concepto property (Imagen 21).



```
concept property extends BaseConcept  
  implements <none>  
  
instance can be root: false  
alias: property  
short description: definition of entity properties  
  
properties:  
  type : datatype  
  name : string  
  
children:  
  << ... >>
```

Imagen 21

Haremos lo mismo para los tipos de datos de referencias y cardinalidades (Imagen 22).

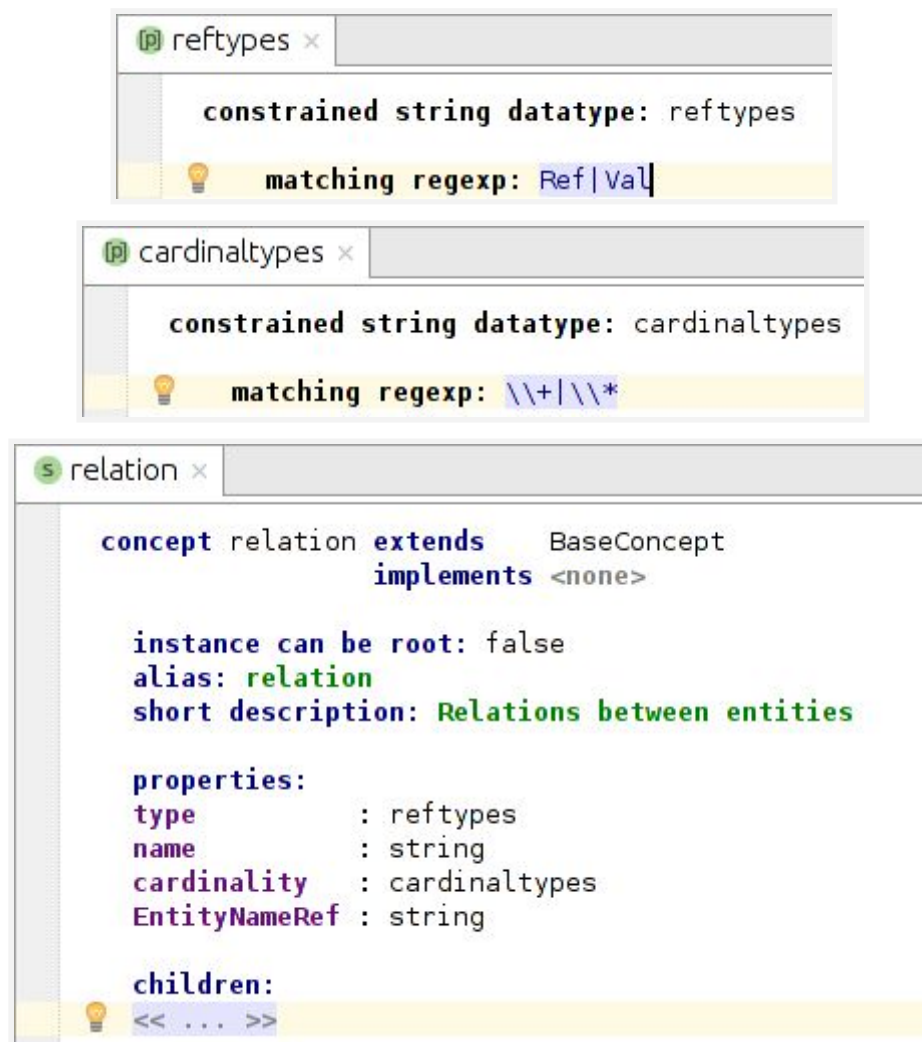


Imagen 22

Una vez modificados los conceptos y la solución sea compilada, veremos que en el editor del módulo sandbox se muestran los errores para los tipos de datos erróneos (Imagen 21).

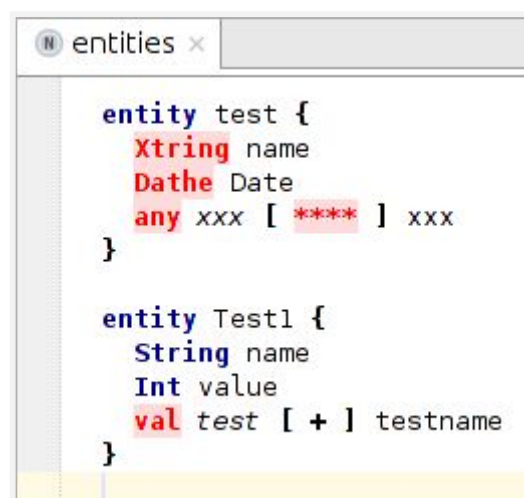
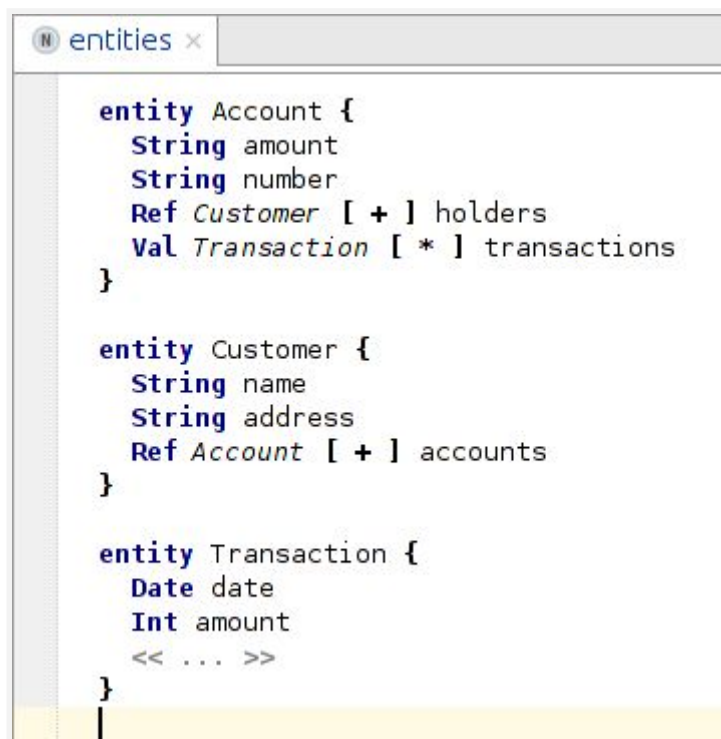


Imagen 21

Al corregir los errores, nuestro DSL estará completo (Imagen 21).




```
entity Account {  
    String amount  
    String number  
    Ref Customer [ + ] holders  
    Val Transaction [ * ] transactions  
}  
  
entity Customer {  
    String name  
    String address  
    Ref Account [ + ] accounts  
}  
  
entity Transaction {  
    Date date  
    Int amount  
    << ... >>  
}
```

Imagen 21

Transformación a código PlantUML

Para crear la transformación haremos uso del módulo *generator*, además instalaremos un plugin para generar texto plano: *com.dslfoundry.plaintextgen*. Este plugin lo descargamos de: <https://plugins.jetbrains.com/plugin/8444-com-dslfoundry-plaintextgen> y lo extraemos al directorio plugins dentro de la instalación de JetBrains MPS.

En el módulo *generator* vemos que por defecto tenemos un módulo llamado *main@generator* y un mapping configuration llamado *main*. Daremos click derecho sobre *main@generator* -> *Model Properties*, en la pestaña *Used Languages* hacemos click sobre el simbolo + de color verde  y añadimos el lenguaje *com.dslfoundry.plaintextgen* (Imagen 22).

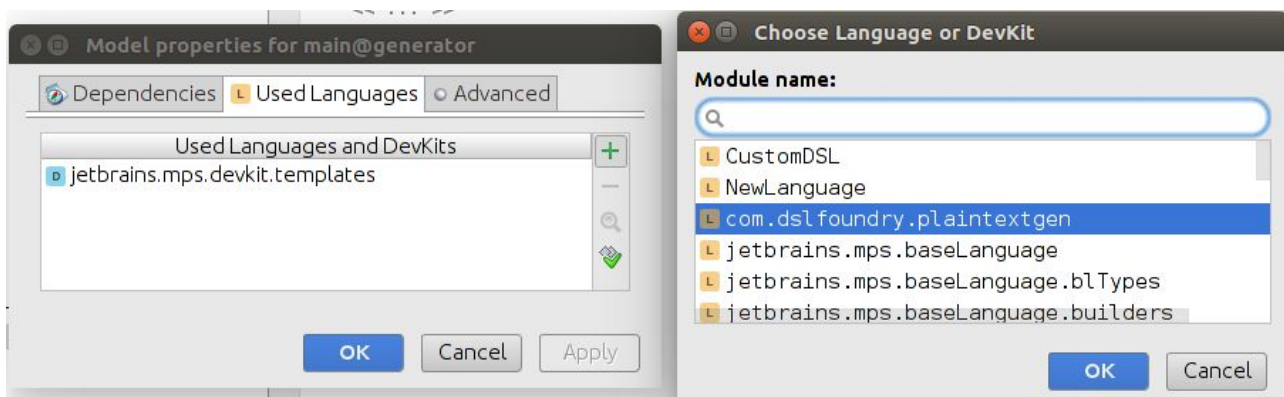


Imagen 22

Si no se muestra el lenguaje luego de haber añadido el plugin, se deben guardar los cambios en el proyecto, reiniciar la aplicación e intentar nuevamente.



Ahora procederemos a crear las plantillas para cada uno de los conceptos declarados en el lenguaje. Debemos tener en cuenta las transformaciones que se deben generar según la siguiente tabla:

	DSL	PlantUML
Propiedades	<code>String number</code> <code>Int amount</code>	<code>String number</code> <code>Int amount</code>
Relaciones	<code>val Transaction [*] transactions</code> <code>ref Customer [+] holders</code>	<code>Account *-- "0..*" Transaction : transactions</code> <code>Account -- "1" Customer : holders</code>
Entidades	<code>entity Account {</code> <code>Int amount</code> <code>String number</code> <code>ref Customer [+] holders</code> <code>val Transaction [*] transactions</code> <code>}</code>	<code>class Account {</code> <code>Int amount</code> <code>String number</code> <code>}</code> <code>Account -- "1" Customer : holders</code> <code>Account *-- "0..*" Transaction : transactions</code>

Plantilla: property

Damos click derecho sobre *main@generator* -> *New* -> *j.m.lang.generator* -> *template declaration*. Esto nos crea un nuevo *template* al que llamaremos: *reduce_property* cuya entrada será el concepto *property*.

En la sección *content node* definiremos la transformación que se debe realizar para cada instancia del concepto *property*. Lo primero que observamos es que nos arroja un error y nos indica que el concepto que estamos usando es el *BaseConcept*. Cambiaremos esto por una instancia del concepto *Vertical Lines* del plugin *com.dslfoundry.plaintextgen*, para esto abriremos el menú *Intentions* (Alt + Enter) y escribimos en el buscador “vertical” y seleccionaremos: *Replace with instance of VerticalLines concept*. Luego en el mismo menú *Intentions* seleccionamos *Create Template Fragment*. Con esto ya definimos el concepto con el que queremos trabajar y un fragmento de plantilla donde colocaremos la interpretación de los conceptos.

Dentro de la etiqueta  presionamos Enter para crear una nueva palabra . Dentro de estas palabras (*com.dslfoundry.plaintextgen.structure.word*) escribiremos los matches entre los conceptos y nuestro lenguaje de salida, así como constantes.

Para las propiedades no existen transformaciones complicadas, simplemente siguen la misma estructura que nuestro DSL tal como lo vimos en la tabla anterior. Entonces añadiremos 3 palabras como plantilla: `String name`. En este momento esas tres palabras son constantes, si generamos la transformación, por cada propiedad se nos creará una copia de las tres constantes, debemos ahora generar los matches entre los datos de cada instancia de los conceptos y la plantilla. Para añadir el match debemos crear macros, en este caso añadiremos el macro para enlazar la propiedad del concepto *property* a nuestro *template*. Nos situamos en la palabra “String”

y abrimos el menú Intentions y seleccionamos *Add Property Macro*: `$ [String name]`. Se nos muestra ahora en el inspector la declaración del *PropertyMacro*, aquí seleccionaremos el nodo y la propiedad a la cual enlazar escribiendo: "node.type" (Imagen 23). Con esto le indicamos a la plantilla que cada vez que transforme una instancia del concepto *property* debe colocar donde está la palabra "String" el valor correspondiente a la propiedad *type* de la instancia del concepto.

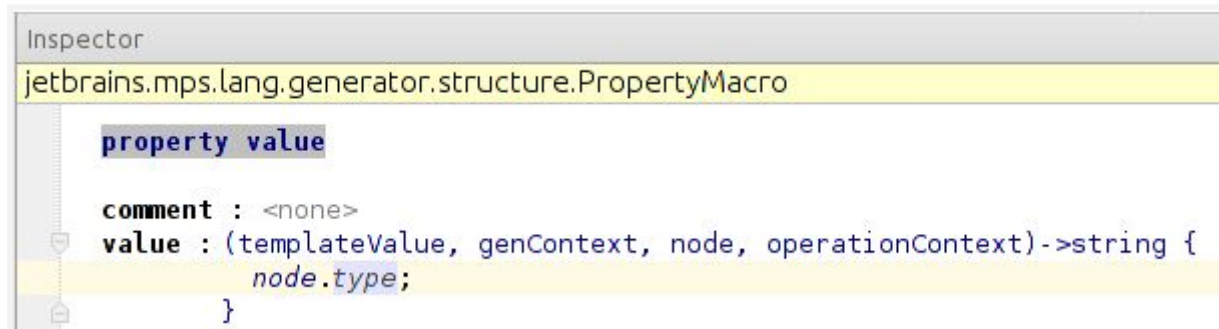


Imagen 23

Hacemos el mismo procedimiento con la propiedad *name* del concepto *property* y con esto terminamos la definición de la plantilla (Imagen 24).

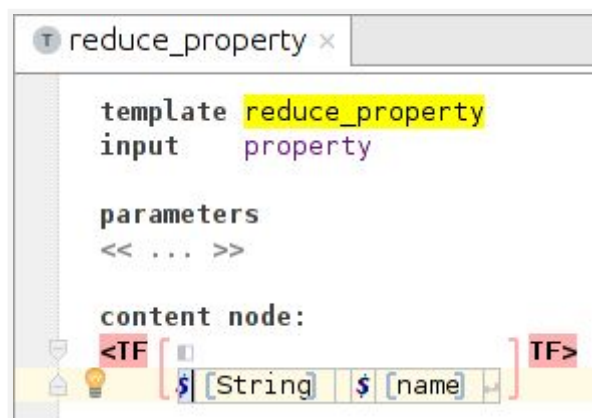


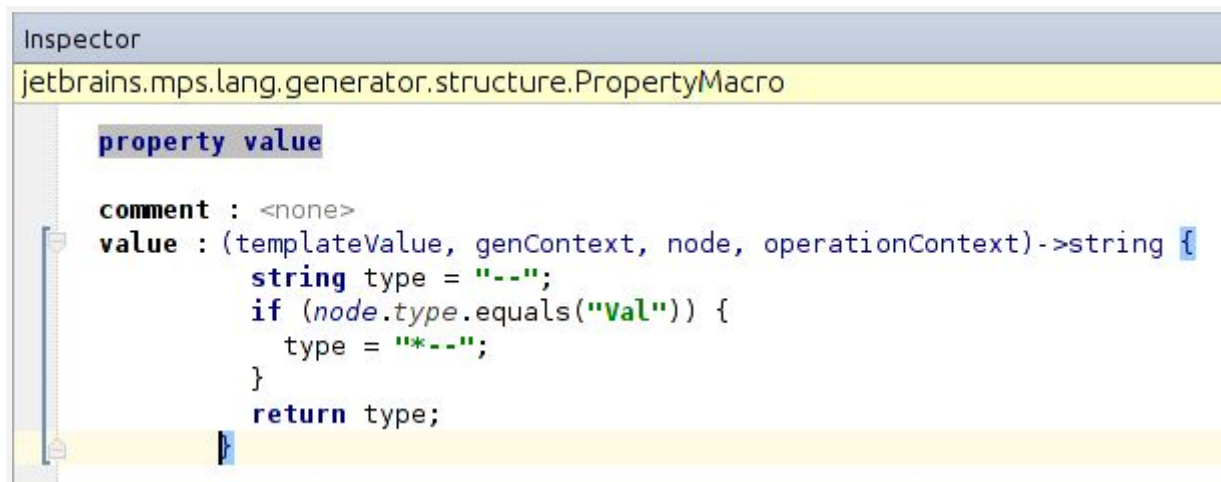
Imagen 24

Plantilla: relation

Seguimos el mismo procedimiento que para la plantilla *property*, creamos la plantilla, le ponemos un nombre, una entrada y definimos el nodo contenido asociando los matches necesarios a las propiedades de las instancias de los conceptos.

Observamos que no tenemos problema para asociar los matches para el nombre de la relación y la entidad a relacionar ya que los nombres están declarados explícitamente en el DSL. Pero para el tipo de relación, la cardinalidad y el nombre de la entidad a la que pertenece la relación no es posible asignar los matches directamente.

Empezaremos con el tipo de relación el cual tiene dos valores en el DSL: "Val" y "Ref", los cuales se deben transformar en "*" y "-" respectivamente. Para esto podemos usar código java y hacer la modificación correspondiente en la definición del *Property Macro* como se muestra en la imagen 25.



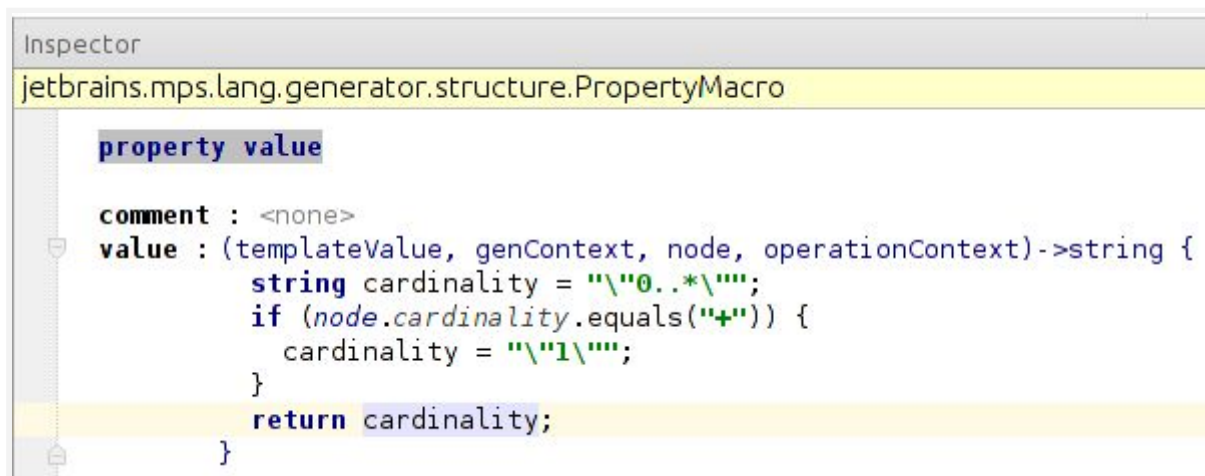
```
Inspector
jetbrains.mps.lang.generator.structure.PropertyMacro

property value

comment : <none>
value : (templateValue, genContext, node, operationContext)->string {
    string type = "--";
    if (node.type.equals("Val")) {
        type = "*--";
    }
    return type;
}
```

Imagen 25

De esta misma manera hacemos la modificación para la cardinalidad la cual tiene los valores: “+” y “*”, y se deben transformar en “1” y “0..*” respectivamente (Imagen 26).



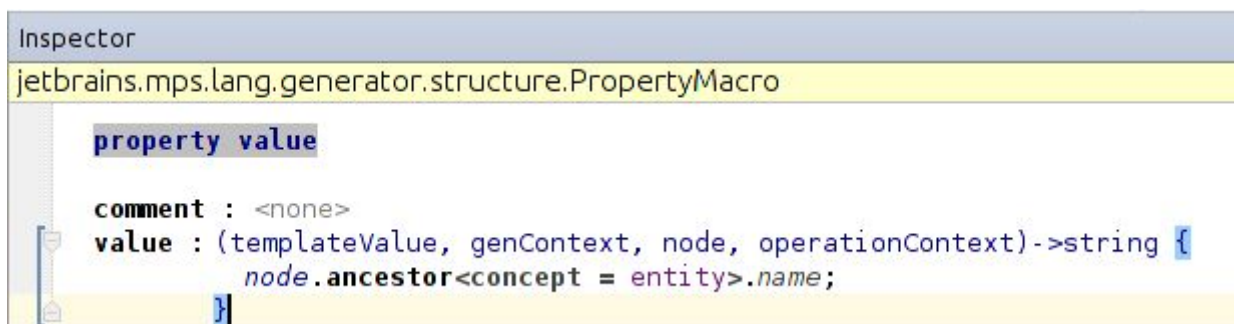
```
Inspector
jetbrains.mps.lang.generator.structure.PropertyMacro

property value

comment : <none>
value : (templateValue, genContext, node, operationContext)->string {
    string cardinality = "\"0..*\"";
    if (node.cardinality.equals("+")) {
        cardinality = "\"1\"";
    }
    return cardinality;
}
```

Imagen 26

Para el nombre de la entidad a la cual pertenece la relación, debemos hacer el enlace al nombre de la instancia del concepto padre, para esto usamos la operación del nodo: “ancestor”. Con la cual podemos hacer llamados al concepto padre del concepto actual. Entonces declaramos el *Property Macro* como se muestra en la imagen 27.



```
Inspector
jetbrains.mps.lang.generator.structure.PropertyMacro

property value

comment : <none>
value : (templateValue, genContext, node, operationContext)->string {
    node.ancestor<concept = entity>.name;
}
```

Imagen 27

Con esto finalizamos la plantilla para el concepto Relación, en la imagen 28 se muestra terminada.

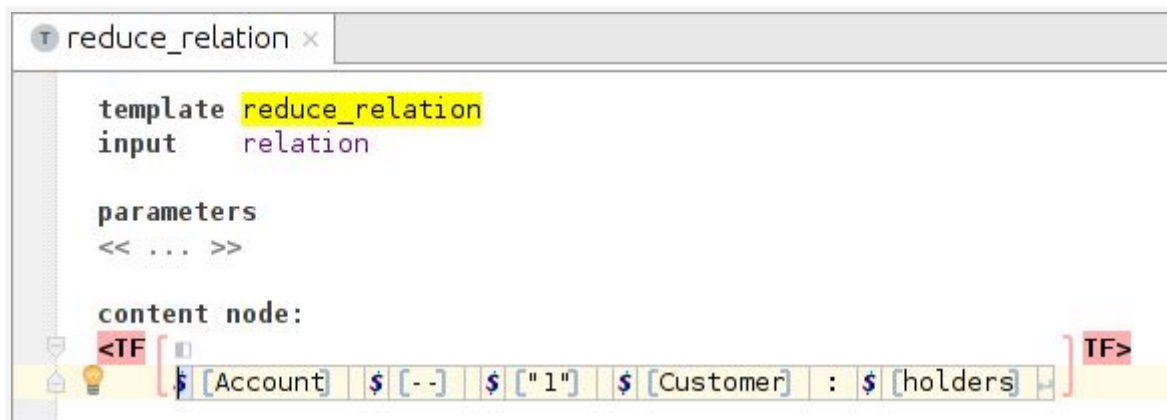



Imagen 27

Plantilla: entity

En la plantilla de entidad debemos definir su estructura y también cómo interpretar sus hijos: propiedades y relaciones. Procedemos a crear la plantilla, colocarle un nombre y una entrada de tipo entity. En el nodo de contenido cambiamos el *baseConcept* por una instancia de *VerticalLines* y creamos el *Template Fragment*.

Creamos la primera parte del template donde colocaremos los literales que lleva cada entidad, recordar que para crear nuevas palabras se usa la tecla Enter, también se usa para crear nuevas líneas arriba o abajo dependiendo de donde esté el cursor, si antes o después del símbolo . Las palabras `<no name>` se han puesto porque en estas posiciones irán tanto las propiedades como las relaciones según la definición del código PlantUML (Imagen 28).

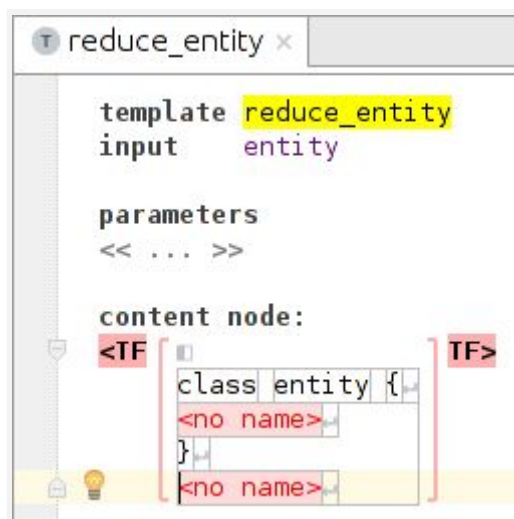


Imagen 27

Como primera medida enlazaremos el nombre de la entidad, creamos un Property Macro apuntando al nombre de la entidad `class $ [entity] {` (Imagen 28).

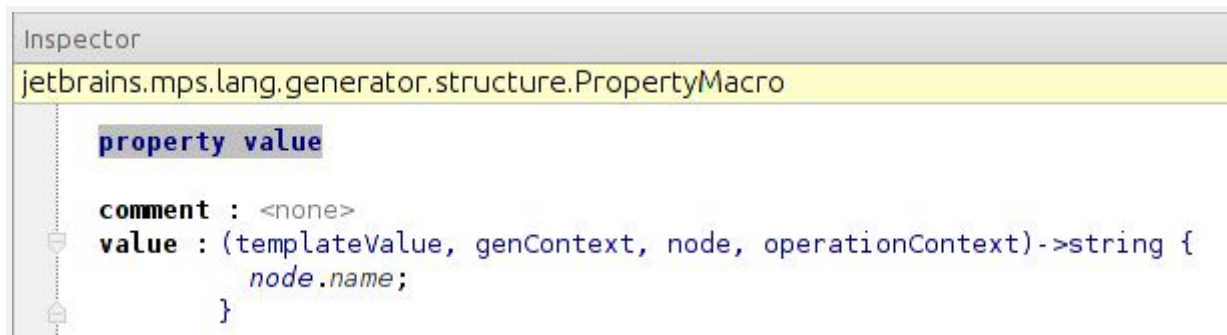


Imagen 28

Ahora, en la sección donde deben ir las propiedades de la entidad, colocaremos un sangrado (indent) para separarlas del margen izquierdo, esto lo hacemos situándonos en la palabra, presionando la combinación de teclas *Ctrl + Space* y seleccionando *Indent Collection*. Una vez hecho el sangrado nos situamos en la etiqueta `<< ... >>` y presionamos *Ctrl + Enter* para generar una nueva palabra (Imagen 29).

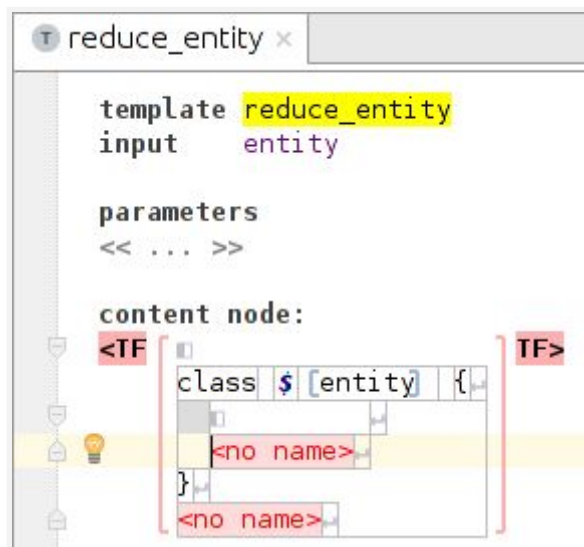


Imagen 29

Situados en esta palabra, con la tecla *Shift* presionada y acto seguido presionamos la tecla de flecha a la derecha señalamos tanto la palabra como el símbolo de nueva línea `<no name>`, este procedimiento es muy importante para que las propiedades no salgan en una sola línea sino separadas por un salto de línea, luego desplegamos el menú *Intentions* y seleccionamos *Add LOOP macro over node.listOfProperties*. Con esta instrucción creamos el bucle que se repetirá por cada una de las propiedades que tenga la entidad. Nuevamente dentro de la palabra, en el menú de *Intentions* seleccionamos *Add Node Macro* y dentro de los dos símbolos de dólar escribiremos "CALL" y luego presionamos *Enter*. Esta instrucción nos crea un llamado a una plantilla definida anteriormente, en el Inspector tendremos que escribir el nombre de la plantilla, en nuestro caso: `reduce_property`. Como seguimos viendo la palabra `<no name>`, nos situamos al lado izquierdo de la palabra y presionamos la tecla *backspace* para borrarla. Repetimos el mismo procedimiento para el bucle de las relaciones el cual debe ir en la palabra después de la llave de cierre de la etiqueta `class`.

Al terminar, la plantilla para las entidades debe quedar como en la imagen 30, muy importante que los símbolos de salto de línea se encuentren dentro de los bucles.

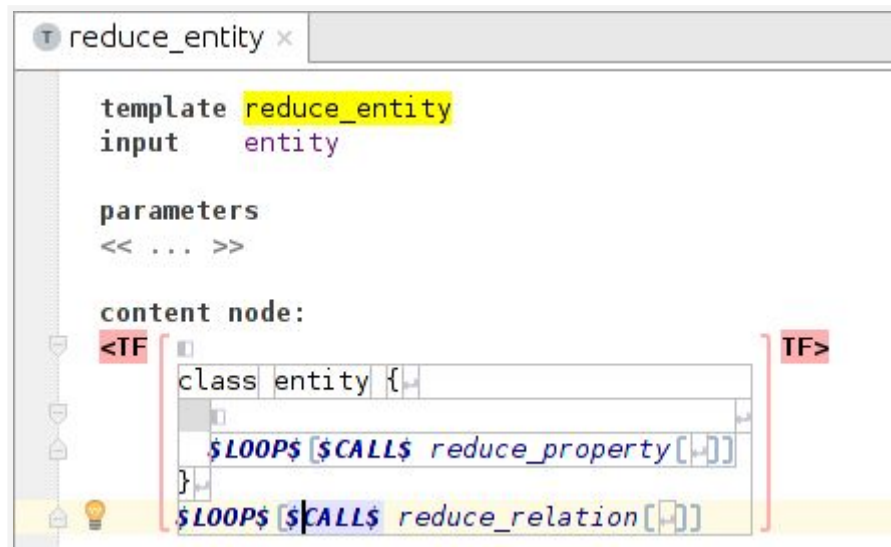


Imagen 30

Plantilla Raíz: map_entities

Damos click derecho nuevamente sobre main@generator -> New -> c.dslfoundry.plantextgen -> TextgenText. Esto nos genera un nuevo root template que será el elemento raíz cuya entrada será el concepto entities, definiremos también el nombre del archivo a generar y su extensión.

Dentro de la etiqueta '<< ... >>' generamos una nueva palabra (Ctrl + Enter), Allí colocaremos la etiqueta de apertura del código PlantUML: "@startuml", luego generamos dos líneas nuevas y en la última colocamos la etiqueta de cierre "@enduml" (Imagen 31).

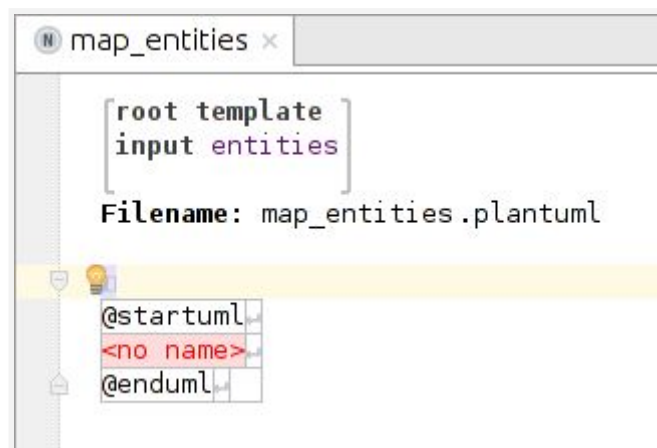


Imagen 31

Ahora crearemos en la palabra de la línea media un bucle sobre la plantilla de entidades de la misma forma que hicimos para las propiedades y relaciones. La plantilla raíz finalizada se muestra en la imagen 32.

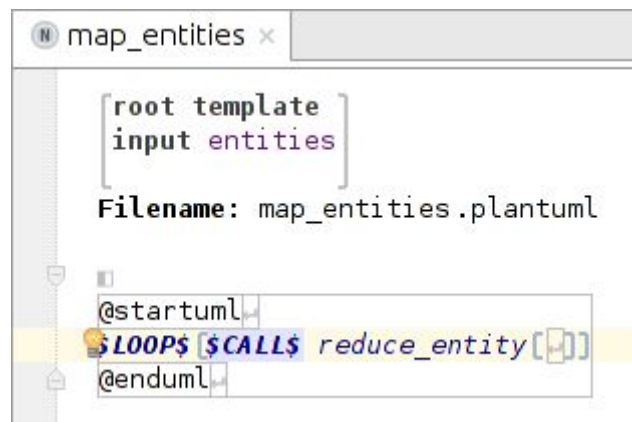


Imagen 32

MappingConfiguration: main

Ahora debemos indicar en la transformación cuál es el elemento raíz, esto lo hacemos en el mapping configuration: main. Damos doble click y vamos a la sección root mapping rules, situados allí presionamos Enter. Registramos el concepto raíz el cual para nuestro caso es entities y su plantilla map_entities (Imagen 33).

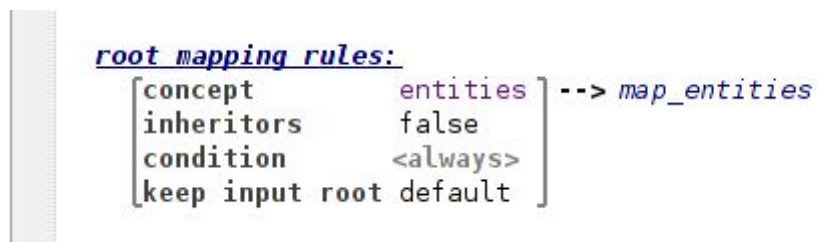


Imagen 32

Prueba de la transformación

Ahora compilamos nuevamente el lenguaje (Ctrl + F9), nos situamos en el sandbox entities y dentro del editor damos *click derecho* -> *PreviewGeneratedText*, con esto podemos visualizar el fichero que se ha creado luego de la transformación (Imagen 33).

Con este último paso hemos terminado el ejercicio, establecimos un lenguaje de dominio adaptado a las necesidades de los usuarios y hemos generado código en un lenguaje de máquina que puede ser fácilmente interpretado por un compilador.

Así como se ha hecho con PlantUML, es posible generar código en cualquier lenguaje una vez establecidas las reglas y transformaciones. JetBrains MPS genera de forma nativa código java pero es posible adaptarlo a cualquier otro.

```
map_entities.plantuml x
1 @startuml
2 class entity {
3     Int amount
4     String Number
5 }
6 Account -- "1" Customer : holders
7 Account *-- "0..*" Transaction : transactions
8
9 class entity {
10     String name
11     String address
12 }
13 Customer -- "1" Account : accounts
14
15 class entity {
16     Date date
17     Int amount
18 }
19
20 @enduml
```

Imagen 33

Agradecimientos

Jesús J. García Molina de la Universidad de Murcia (jmolina@um.es).

Vaclav Pech del MPS Support Community forum. (<https://github.com/vaclav/MultiLangEntities>)