## [Editeur Xtext ET Transformation Divine](http://www-master.ufr-info-p6.jussieu.fr/2013/Editeur-Xtext-et-Transformation,744)

**RAPPORT DU PROJET**

**eNCADRe pAR :**

Yann THIERRY-MIEG

**

**Réalisé Par :**

**Mokrane KADRI**

**Tahar OUAZIB**

**Sarah DAHAB**

**SOMMAIRE :**

1. Préambule………………………………………………………………………………………………..
2. Objectifs Du Projet…………………………………………………………………………………..
3. Outils de développements ………………………………………………………………………..
4. Description des Langages……………………………………………………………….

4.1. **Divine** ……………………………………………………………………………………

1. Le Système Devine………………………………………………………………..…
2. Les Types de données………………………………………………………………..
3. Les Processus………………………………………………………………..…………
4. Les Channels………………………………………………………………..………………
5. Les Transitions ………………………………………………………
6. Les Expressions………………………………………………………………..…………

4.2. Le système Gal …………………………………………………………………………..

4.3. **Xtext & Xtend**…………………………………………………………………………….

1. Description Du travail ………………………………………………………………………..…………..
2. Étapes et Réalisation
   * 1. Modélisation ……………………………………………………………………………..
     2. L’éditeur de Texte ……………………………………………………………………….
     3. Divine to Gal
3. Conclusion ………………………………………………………………..………………………………
4. Annexe………………………………………………………………..………………………………………

1. Préambule:

Ce projet s’inscrit dans le cadre de l’UE (PSTL/PSAR) du deuxième semestre proposé par L’équipe MoVe du Laboratoire Informatique de l’Université Pierre et Marie Curie et supervisé par Yann Thierry- Mieg.

C’est également un projet basé sur les travaux réalisés par cette équipe dans les années précédentes (Plugin Coloane et GAL).

Le but du projet et de réaliser un plugin pour l’IDE Éclipse. Ce plugin se présente sous la forme d’un éditeur de texte pour le langage Divine (voir les sections suivantes) .Il offre également la possibilité de réaliser une transformation (Conversion) de fichiers écrits en Divine vers Gal (**G**uard **A**ction **L**anguage).

2. Objectifs Du Projet:

Ce projet vise à réaliser un plugin pour le Eclipse .Ce plugin Fournira entre autre :

* Un Éditeur de texte pour le langage Divine .Cet éditeur tirera sa puissance du Framework Xtext (voir Section Xtext & Xtend) et fournira ainsi un ensemble de fonctionnalités comme celles proposés par les éditeurs modernes à savoir : La coloration syntaxique, les outlines, les quickfix etc…
* Une Fonctionnalité permettant de réaliser une transformation d’un fichier (.dve) écrit en divine vers le langage Gal (voir le système Gal) depuis notre éditeur de texte.

3. Outils de développement:

//todo manque dinspiration !

L’outils utilisé pour la réalisation du projet est Eclipse.

* **Eclipse :** c’est L’ide sur lequel se base notre projet .On a enrichi notre IDE avec Plusieurs plugins comme Xtext, Xtend EcoreDocumentation afin de pouvoir manipuler les technologies(Langages) en question.
* **SVN :** L’utilitaire de gestion de version

4. Description des Langages:

Dans cette partie nous allons présenter les diffèrent langages manipulé lors du le projet, nous commenceront par Divine, puis Gal et nous terminerons par Xtext et Xtend.

**4.1 Divine :**

A. Le système Divine :

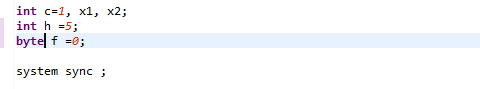
Un système Divine est constitué de variables globales et d’ensemble de processus contenant des variables locales, des états, des channels et des transitions qui seront exposés plus précisément dans cette partie.

Un système Divine commence directement par la déclaration de variable globale si il y en a ou par la déclaration de processus et se termine par une accolade fermante.

Chaque ligne de code se termine par un point-virgule.

B. Les types de données :

* Variable : une variable peut être de type byte ou int. elle est déclarée par son type et son nom et se termine par un point-virgule, elle peut être initialisée lors de la déclaration ou plus loin dans le code. On distingue 2 types de variable :
  + Variable globale : une variable globale est toute variable déclaré dans un system Divine.

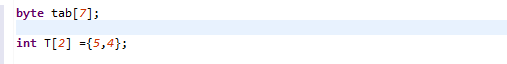


* + Variable locale : une variable locale est une variable déclaré au début dans un system de processus. Elle est visible uniquement dans le system où elle a été déclarée.



* Tableau : Tout comme les variables, un tableau a un nom et il un type (byte ou int).

Il est déclaré de la même manière qu’une variable mais avec des crochets contenant la taille du tableau. Il peut être initialisé lors de la déclaration par des accolades contenant la suite de valeur séparé par une virgule.

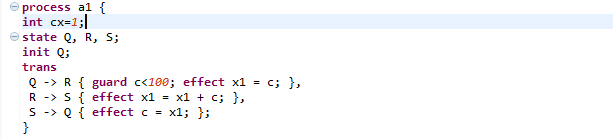


* Les constantes : Les constantes sont déclarées avec le mot clés const, tout comme les variables elles sont typé int ou byte.

C. Les processus :

En Divine, les Processus (mot clef Process) peuvent êtres considérer comme des sous système. Ils comportent un ensemble transitions (Voir Transitions), des déclarations de variables suivies des déclarations d’états (ordre important). L’État initial peut être initialisé avec le mot clef init suivie d’un état donné.

Voici un exemple de Processus en divine :



D. les Channels :

Les Canaux sont utilisés en divine pour l’envoie et réception de données entre des processus. Ainsi les canaux pourrait être considérés comme un mécanisme de synchronisation entre processus (2 processus différents exécute chacun une transition en même temps).

* Channels non tamponnés :

Ce genre de canaux peut ne pas être typé et ne transmet pas forcement de données. L’envoie de données en utilisant ce type de canaux est dite synchrone, i.e. Les deux processus qui émettent/reçoivent les données exécute leur Transitions dans une même transition système.

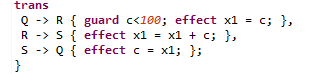
* Channels tamponnés :

C’est des canaux typé (int, byte), réalisant une transmission de données asynchrone, i.e. deux processus différents peuvent emmètre/recevoir dès que leur tampon le permettent.

D. Les transitions :

Les transitions sont toujours déclarées dans un processus (Process) avec le mot clef Trans .Une transition indique le(s) changement(s) d’état(s) de processus (syntaxe : flèches allant d'un état ​​à un autre). Elle définit également une condition (garde) pour qu’elle puisse être exécutée (en utilisant le mot-clé Guard). Elle pourrait avoir parfois un canal de synchronisation (Voir Channel) ainsi que des effets (Effect) qui sont de simples affectations aux variables.

Voici un exemple de transition :



E. Les Expressions :

Comme dans la majorité des langages de programmation, divine peut manipuler les expressions arithmétiques et expressions booléennes.

* les expressions arithmétiques :

Les expressions arithmétiques en divine peuvent être utilisé dans des assignements, l’initialisation de variables voir même en garde de transition quand elles ont une interprétation booléenne (valeur 0 ou 1).On se divise en deux catégories :

* + expressions arithmétiques  binaires :

|  |  |
| --- | --- |
| Opération | Operateur |
| Addition | Soustraction | **+ | -** |
| Multiplication | Division | **\* | /** |
| Modulo | **%** |
| Shift Gauche | Droit | **<< | >>** |

* + expressions arithmétiques  Unaires :

|  |  |
| --- | --- |
| Opération | Operateur |
| Moins unaire | **-** |
| Complément bit à bit (tilde) | **~** |

* les expressions booléennes :

Elles sont généralement utilisées dans les gardes des transitions ou les assertions.

On en retrouve les types basic (true et false) mais aussi les opérations usuelles comme le ou Logique (||) le Et Logique (&&) et les différentes comparaisons entre deux expressions arithmétiques (>, < , >= ,<= , != ,etc…) .

**4.2 Le Système Gal :**

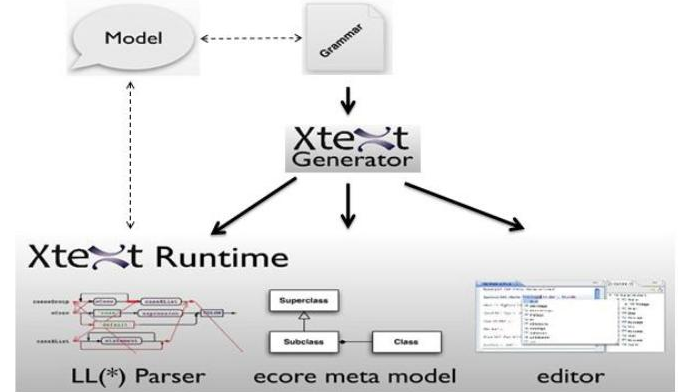
**4.3 Xtext et Xtend :**

**1. Xtext :**

Xtext permet le développement de la grammaire des langages spécifiques aux domaines (DSL: **D**omain **S**pecific **L**anguages) et d’autres langages textuels. Il est étroitement lié à l'**E**clipse **M**odeling **F**ramework (EMF).

Il permet de générer un méta-Modèle Ecore, un analyseur syntaxique (parser, en anglais) basé sur le générateur ANTLR ou JavaCC et un éditeur de texte sous La plate-forme Eclipse afin de fournir un environnement de développement intégré IDE spécifique au langage. La figure fournit une vue d’ensemble de l’outil Xtext. La forme textuelle du DSL constitue le point de départ. La grammaire du DSL est le point d’entrée de l’outil. Xtext produit, en sortie, un analyseur syntaxique, un éditeur et un méta-modèle pour le DSL.

Xtext permet de considérer la forme textuelle du DSL comme un modèle conforme à la grammaire d’entrée, donc au méta-modèle généré.

****

**1. Xtend:**

Xtend est un langage orienté objet de haut niveau .Syntaxiquement et sémantiquement Xtend tire ces racines du langage Java. Il se distingue par sa syntaxe plus concise ainsi que quelques fonctionnalités supplémentaires telles que l'inférence de type, les méthodes d'extension, et la surcharge des opérateurs. Il intègre également des fonctionnalités connues de la programmation fonctionnelle, comme les expressions lambda. Xtend est typé statiquement et utilise le système de type de Java sans modifications. Il est compilé en code Java et s’intègre ainsi parfaitement avec toutes les bibliothèques Java.

Concernant notre projet, Xtend était indispensable pour éditer certaines classes générées par Xtext permettant ainsi d’implémenter un ensemble de fonctionnalités (Règle de validation, quickFix, etc.) . Ces fonctionnalités seront détaillées dans les sections qui suivent.

5. Description du travail :

La Réalisation du projet s’est déroulée en plusieurs étapes. Dans cette section, on soulignera les parties majeures du travail effectué, qui seront présentés plus en détail dans le chapitre suivant.

* **La Définition du Langage :**

La première étape du projet consistait à écrire une grammaire Xtext pour le langage « Divine » tout en respectant la spécification et la Syntaxe de ses éléments .Voir le site officiel  (<https://divine.fi.muni.cz/manual.html> ).

* **L’éditeur de texte :**

Une fois notre grammaire opérationnelle, on a implémenté un ensemble de fonctionnalités (Régle de validations, quickfix, outline etc…) afin d’améliorer notre éditeurs de texte et le rendre plus riche.

* **La Transformation :**

Cette partie est le cœur du projet. Le concept est simple : transformer à un fichier décrivant un system donné, écrit en « Divine » (.dve), en un autre fichier « Gal » (.gal) décrivant le même système.

6. Étapes et Réalisation :

Dans cette section, nous aborderons plus en détails les différentes étapes de la réalisation du projet décrites dans la partie précédente, tout en les illustrons par des extrait de code et/ou des captures d’écrans.

1. Modélisation:

Nous allons maintenant voir comment est construite la grammaire pour chaqu’un des éléments du langage « divine ».Nous utiliseront plutôt une approche basé sur la Meta modélisation pour appuyer les explications fournis.

Les différents aspects des langages « divine » et « Xtext » ont été présentés dans les sections précédentes. La documentation officielle est également disponible sur leurs sites respectifs :

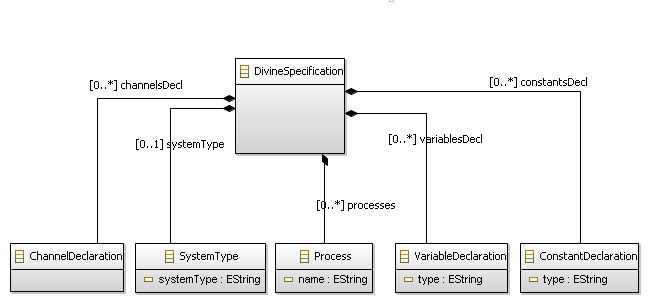
Xtext:  “<https://www.eclipse.org/Xtext/documentation.html> “

Divine: “ <https://divine.fi.muni.cz/manual.html> “

**Remarque :** Pour plus de détaille sur la façon dont les modèles ci-après ont été généré, ainsi que le code Xtext correspondant faut se référer au règles correspondantes de la grammaire qui se situe dans le fichier :fr.lip6.move.divine/src/fr/lip6/move/Divine.xtext

* **Le système Divine :**

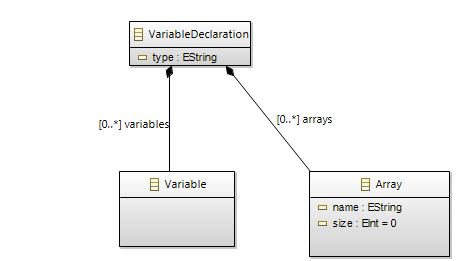
Cette entité représente la structure globale d’un programme divine, tous les programme écrit en Divine devraient s’appuyer sur le Meta-modèle ci-après :

****

Ce Diagramme est construit à partie de la règle DivineSpecification de notre grammaire Xtext. Il montre bien que la méta-classe DivineSpecification englobe  les définitions des différentes type d’entité du système, tel que Les Processus (Meta-classe Process), ainsi que les différents type de données tel que les variables (Meta-classe VariableDeclaration), les Constantes (Meta-classe ConstantDeclaration), etc…

* **Les Structure de données**

Divine permet de manipuler plusieurs types de données comme les variables, les constantes et les tableaux.

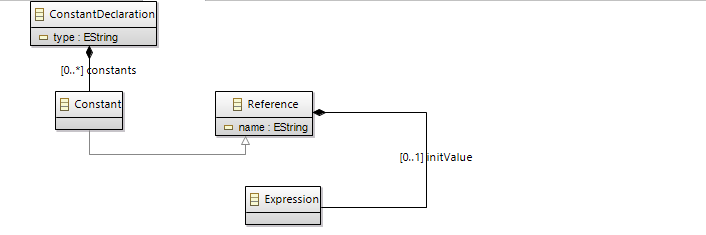
* + Les variables :

Comme l’indique le Modèle VariableDeclaration de la figure ci-dessus, la déclaration des variables se décline en un ensemble de :

* tableau (Méta-classe Array) , ce tableau sera identifié par son nom (unique) ainsi que sa taille.
* Variable (Méta-classe Variable), des variables classiques portant un nom.

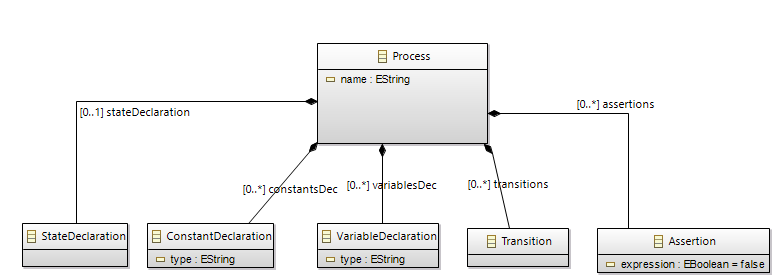
Ces deux déclarations sont typées soit par int ou byte.

* + Les Constantes :

****

* **Les Process :**

Le diagramme qui suit illustre la structure d’un Process (méta-classe Process) .Un process comporte donc un ensemble de déclarations de variables, de constantes, ayant une portée locale, mais aussi une déclaration d’un ensemble d’états (méta-classe StateDeclaration). Il est caractérisé par son nom unique et un ensemble de transitions. (Voir Transition).



**Remarque :**

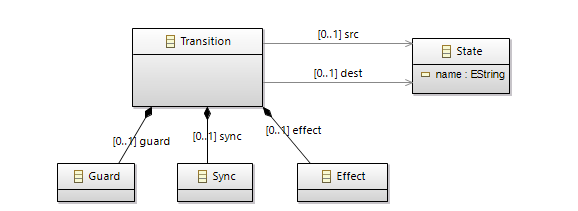
À noter que les variables déclarées dans le système divine, à l’exception de celles déclarées dans les process sont toutes considérés comme étant variables globales.

* **Les transitions :**

Une transition est responsable des changements opérer sur les états. Concrètement elle fait passer un processus d’un certain état (méta-classe State) « X » vers (->) un nouvel état « Y ». Tout en réalisant un ensemble d’effets de bord sous forme d’assignements

(Méta-classe Effect). Cependant ceci n’est possible que si la condition de la garde (classe Guard) est satisfaite.

Le modèle UML suivant représente une transition en Divine :



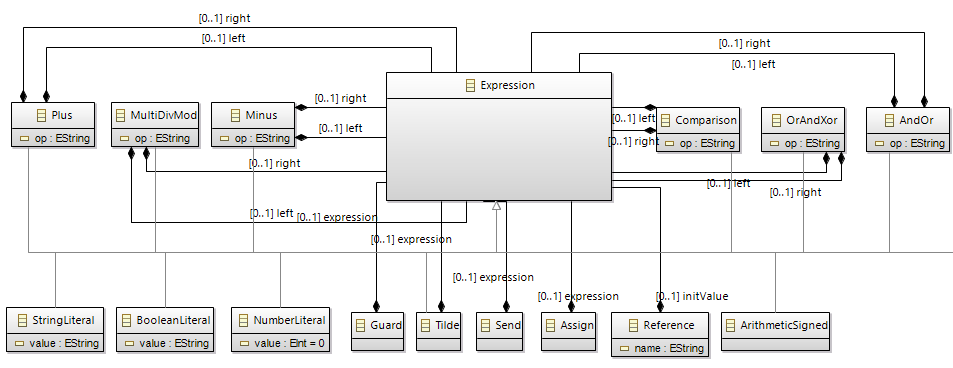
Remarque :

Les transitions ont un nom unique dans le système et sont uniquement déclarées dans un process.

* **Les Expressions :**

Les règles représentants les expressions étaient sans doute les règles plus laborieuses a qu’on a eu à définir. Il fallait gérer les priorités des opérations ainsi que leur distributivité à gauche.

Le diagramme ci-après, montre la représentation qu’on a choisie pour définir les expressions Divine dans notre grammaire Xtext.



Comme le montre ce diagramme, il s’agit d’un pattern Composite. Cette modélisation nous permet ainsi de manipuler deux Grande familles d’expressions :

Les Expressions arithmétiques:

Ce sont des expressions formées avec les opérateurs arithmétiques classiques.

Ce genre d’expression se divise en plusieurs sous types :

* + Les expressions arithmétiques Binaires, comporte une opération (type String) ainsi que deux opérandes qui sont des expressions à leurs tour. C’est le cas de l’addition, soustraction etc... (Voir. Les méta-classe Plus, Minus, MultiDivMod etc…)
  + Les expressions arithmétiques Unaires pour la représentation des entiers négatifs par exemple, Classe ArithmeticSigned.
  + Les constantes, qui ont une valeur. (NumberLiteral).

Les Expressions Booléennes:

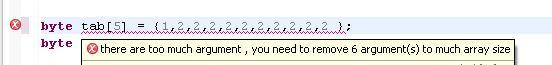
Les expressions booléennes suivent le même pattern que les expressions entières, on y trouve :

* des expressions binaires comme la comparaison entre deux expressions entières (méta-classe Comparaison), mais aussi les opérateurs logique usuels tel que le «ET », le « OU »,  modélisées par AndOr.
* des expressions booleen binaires
* Un type constant, représenté par la classe BooleanLiteral, qui contient un champ indiquant sa valeur (true /false).

1. L’éditeur Divine :
2. **Les Règles De validation**

L’ensemble des règles de validation définies vise à assurer :

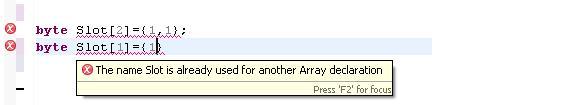
* Une cohérence entre la taille des tableaux et le cardinal de l’ensemble leurs éléments lors de la déclaration et l’initialisation.





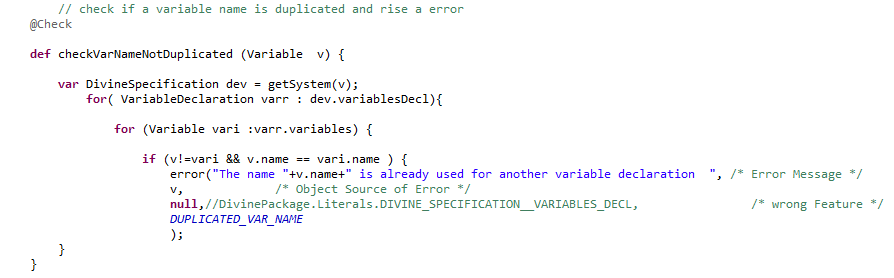
* L’unicité des noms de Variables, Tableaux et Constantes.





L’ensemble de ces règles est définie d’un fichier annexe ( DivineValidator.xtend ) fournie par Xtext. Voici un extrait de la méthode chargé de vérifier l’unicité de nom de variables.

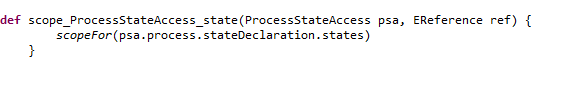
**Remarque** : l’annotation @Check indique au Framework que la méthode qui suit est une règles de validation à vérifier.

****

1. **Le Scoping**

Xtext fournie un mécanisme de référencement, le problème est que les références au niveau syntaxique sont vues comme des identifiants (au même titre que les noms de variables) et sont résolues après que l’arbre syntaxique de la grammaire ait été généré.

Dans notre Projet, on a eu recours au mécanisme de scoping pour résoudre les problèmes d’accès aux états des processus dans le corps de ces derniers .Voici un extrait du code :



Les règles de scoping sont définies quant à elles dans un autre fichier annexe ( DivineScopeProvider.xtend ), généré automatiquement par Xtext aussi.

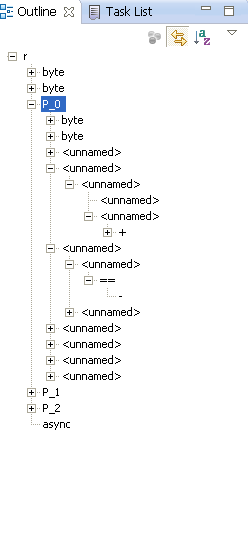
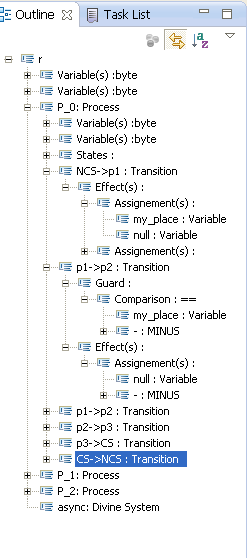
1. **Outlines :**

Les Outlines permettent de visualisé la structure des programmes, c’est comme un sommaire .il a pour vocation de faciliter navigation dans le code. Dans le code. Xtext génère par défaut des outlines qui portent le nom des labels (ID) des nœuds simples de la grammaire définie. Cependant pour les nœuds composés Xtext produit des balises  <unamed>. Dans cette optique, on définit un ensemble de méthodes qui ont pour but de guider Xtext à traiter tous les nœuds possibles de notre grammaire et générer ainsi Des Outlines Complet. Le fichier à éditer est fourni par Xtext ( DivineOutlineProvider.xtend )

Ce fichier regroupe un ensemble de méthodes qui se charge de renvoyer une chaine de caractère adéquate qui sera affiché dans les outlines.

Pour illustrer, voici des captures d’écran correspondant à l’aperçu des outlines avant et après la définition des règles de leur génération.

AVANT APRES

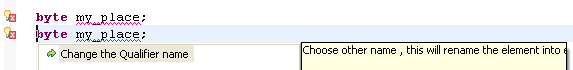
 

1. **QuickFix:**

Les deux figures suivantes illustres deux exemples de quickFixs implémentés.

Le premier permet de corriger les erreurs liés à l’absence de cohérence entre la taille d’un tableau est le cardinal des éléments qui lui sont affecter, le second permet de renommer les éléments ayant le même nom.

****

****

Le code correspondant aux quickFixs se trouve dans le fichier « DivineQuickFixProvider.xtend » du package « fr.lip6.move.divine.ui.quickfix »

1. **Divine to Gal :**

La tâche principale consistait donc réaliser une conversion du langage « divine » vers le langage « Gal » en s’appuyant à la fois sur le « Meta-Model » qu’on a obtenu grâce à notre grammaire mais aussi sur un ensemble de fonctions offertes par « Gal » via le plugin « Coloanne ».

L’ensemble des règles des transformations sont définies dans le paquage « togal », qui est composé de deux fichiers :

* « Converter.java » : ce fichier regroupe un ensemble de fonction utilitaire, capable notamment de faire des conversions de certain type d’éléments « divine » en leur équivalent en « gal », stocker les Variable et vérifier le typage des éléments.
* « DveToGalTransformer.java » : ce fichier est composé d’un certain nombre de fonction réalisant la transformation « Dve » « Gal ».

**TODO DETAILS**

ANNEXE :

* **Les documentations des langages de programmation**
* **La documentation Xtext:**

[**http://www.eclipse.org/Xtext/documentation.html**](http://www.eclipse.org/Xtext/documentation.html)

* **La documentation Xtend:**

[**http://www.eclipse.org/xtend/documentation/index.html**](http://www.eclipse.org/xtend/documentation/index.html)

* **Java Doc:**

[**http://docs.oracle.com/javase/7/docs/api/**](http://docs.oracle.com/javase/7/docs/api/)

* **GAL :**

[**http://move.lip6.fr/software/DDD/gal.php**](http://move.lip6.fr/software/DDD/gal.php)

* **Divine et sa spécification :**

[**https://divine.fi.muni.cz/manual.html#the-dve-specification-language**](https://divine.fi.muni.cz/manual.html#the-dve-specification-language)

* **Liens Utiles :**
* **Les expressions en Xtext:**

[**http://xsemantics.sourceforge.net/xsemantics-documentation/Expressions-example.html**](http://xsemantics.sourceforge.net/xsemantics-documentation/Expressions-example.html)

* **Code source pour comprendre la grammaire:**

[**http://divine.fi.muni.cz/trac/browser/divine/dve/**](http://divine.fi.muni.cz/trac/browser/divine/dve/)

* **Autre Liens Utiles**
* **Le site du Laboratoire :**

[**http://move.lip6.fr/software/DDD/**](http://move.lip6.fr/software/DDD/)

* **TeamCity :**

[**http://teamcity-systeme.lip6.fr/login.html**](http://teamcity-systeme.lip6.fr/login.html)

* **Modèles Utiliser pour les tests :**

[**https://projets-systeme.lip6.fr/svn/research/libddd/libits/trunk/tests/test\_models**](https://projets-systeme.lip6.fr/svn/research/libddd/libits/trunk/tests/test_models)

[**https://projets-systeme.lip6.fr/svn/research/libddd/libits/trunk/perfs/test\_models**](https://projets-systeme.lip6.fr/svn/research/libddd/libits/trunk/perfs/test_models)