## [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 project…………………………………………………………………………………..
3. Outils de développements ………………………………………………………………………..
4. Description des Langages……………………………………………………………….

1. **Divine** ……………………………………………………………………………………

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

2. **Xtext & Xtend**…………………………………………………………………………….

1. Description Du travail ………………………………………………………………………..…………..
2. Etapes et Réalisation
   * 1. Grammaire « Xtext » pour « Divine »
     2. L’éditeur Divine
        1. Règles De validation & Scooping
        2. Coloration Syntaxique et Outline
     3. Divine to Gal
        1. Le Système Gal……………………………………………………………
3. Conclusion ………………………………………………………………..………………………………
4. Annexe………………………………………………………………..………………………………………
5. **ANNEXE :**

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

<http://www.eclipse.org/Xtext/documentation.html>

* **La documentation Xtend:**

<http://www.eclipse.org/xtend/documentation/index.html>

* **Java Doc:**

<http://docs.oracle.com/javase/7/docs/api/>

* **GAL :**

<http://move.lip6.fr/software/DDD/gal.php>

* **Divine et sa spécification :**

<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/>

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

<http://move.lip6.fr/software/DDD/>

* **TeamCity :**

<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/perfs/test_models>

**5. Description du travail :**

**La Définition du Langage :**

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

**Règles de Validation & scoping**

Une fois notre grammaire opérationnelle, On a défini certain nombre de règles de Validation pour éviter les problèmes d’incohérence qui peuvent survenir lors de l’écriture de programme avec notre éditeur.

A cela s’ajoute les règles dites de « scoping » pour gérer la visibilité et la portée des variables définies.

**La Transformation**

L’idée est simple, à partir d’un fichier décrivant un system donné, écrit en « Divine » (.dve), on devait pouvoir générer en sortie un fichier « Gal » (.gal) décrivant le même système.

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 ».

**6. Etapes et Réalisation :**

Dans cette section, nous aborderons plus en détails les différentes étapes de la réalisation du projet tout en les illustrons par des extrait de code ainsi que des capture d’écrans.

* 1. Grammaire « Xtext » pour « Divine » :

On s’appuyant sur cette grammaire, Xtext produit alors un « Meta-Model ». Ce « Meta-Model » est décrit dans un langage spécialisé appelé « ECORE » qui un langage de Meta-modélisation basé sur des notions de L’orienté –Objet.

Les détails sur la grammaire définie ainsi que certaines des difficultés rencontrées lors de sa création sont exposés dans la partie XXX, page XXX.

Au bout de cette étape on a obtenu donc une grammaire (fichier « Divine.xtext »), qui décrit Les différents éléments de ce langage tout en respectant leurs spécifications.

L’exécution de ce fichier génère le fichier : « GenerateDivine.mwe2 » (mwe : Modeling Workflow Engine), qui permet à son tour de lancer une instance de notre plugin (Editeur de texte) susceptible de reconnaitre la syntaxe « Divine ».

import fr.lip6.move.divine.divine.ProcessStateAccess

import org.eclipse.emf.ecore.EReference

import org.eclipse.xtext.scoping.impl.AbstractDeclarativeScopeProvider

import static org.eclipse.xtext.scoping.Scopes.\*

import fr.lip6.move.divine.divine.DivineSpecification

import org.eclipse.emf.ecore.EObject

import fr.lip6.move.divine.divine.ProcessStateAccess

import org.eclipse.emf.ecore.EReference

import org.eclipse.xtext.scoping.impl.AbstractDeclarativeScopeProvider

import static org.eclipse.xtext.scoping.Scopes.\*

import fr.lip6.move.divine.divine.DivineSpecification

import org.eclipse.emf.ecore.EObject

import fr.lip6.move.divine.divine.ProcessStateAccess

import org.eclipse.emf.ecore.EReference

import org.eclipse.xtext.scoping.impl.AbstractDeclarativeScopeProvider

import static org.eclipse.xtext.scoping.Scopes.\*

import fr.lip6.move.divine.divine.DivineSpecification

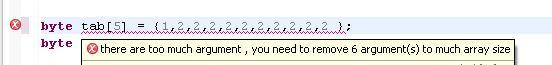
import org.eclipse.emf.ecore.EObject

* 1. L’éditeur Divine :

1. **Les Règles De validation**

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

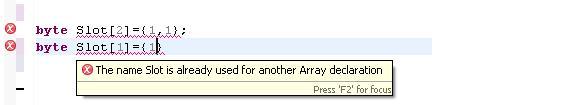
* Une cohérence entre la taille des tableaux et le cardinal de l’ensemble leurs éléments lors de la déclaration et initialisation.





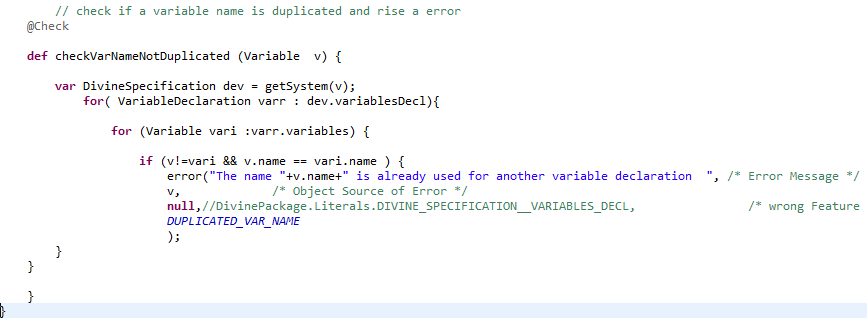
* L’unicité des noms de Variables, Tableaux et Constantes.





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

Voici un exemple de code :



1. **Le Scooping**

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 cas, on a eu recours à ce mécanisme pour résoudre les problèmes d’accès aux états des processus dans le corps de ces dernier .Voici un extrait du code :

Les règles de scooping sont définies quant à elle dans un autre fichier annexe « DivineScopeProvider.xtend », généré automatiquement par Xtext aussi.

****

1. **Coloration Syntaxique et Outline**

**// TODO**

* 1. **Divine to Gal :**