INSA Rennes

Equipe DIVERSE

Département informatique - Stage été 2015

Amélioration du support de Melange dans l'environnement de développement Eclipse

Du 06 Juillet au 28 Août 2015 (2 mois)

Auteur :
François Boschet

Maître de stage :
M. Arnaud Blouin

Correspondant INSA :
Mme. Marie Babel

Remerciements

Je remercie toutes les personnes de l'équipe Diverse et plus particulièrement Mr Arnaud Blouin, pour m'avoir permis de décrocher ce stage, et Mr Thomas Degueule pour avoir passé beaucoup de temps a répondre à toutes mes questions.

Table des matières

1

Présentation

INRIA

L'INRIA, ou Institut National de Recherche en Informatique et en Automatique, est un centre de recherche français composé de plus de 2700 chercheurs répartis dans des centres partout en France. Le site de Rennes est très lié avec l'IRISA (Insitut de Recherche en Informatique et Systèmes Aléatoires) et abrite près de 400 scientifiques répartis dans une douzaine d'"équipe-projet". Chaque équipe travaille sur un domaine précis, comme par exemple l'équipe Hybrid qui travaille sur le réalité virtuelle et les interactions 3D.

Diverse

Durant mon stage, j'ai fait partie de l'équipe Diverse, composé d'une quarantaine de personnes, dont 8 chercheurs permanents et dirigé par Mr Benoit Baudry. Diverse est un acronyme pour "Diversity-centric Software Engineering". Les axes de recherche de l'équipe sont l'ingénierie des langages et la variabilité, l'adaptation et la diversification des logiciels.

Ces recherches ont amenés au dévelopement de plusieurs logiciels notament Kevoree, une platforme de dévelopement logiciel distribuée, Kermeta, un langage de manipulation de métamodèles, et Melange un outil de manipulation de DSL[1] sur lequel j'ai travaillé.

[1] Langage dont les specifications sont conçues pour un domaine d'application précis

2

L'environnement de travail

Melange

J'ai travaillé sur le langage Melange (http://melange-lang.org/), le projet de thèse de Thomas Degueule. Melange est un language de programmation de type : "Langage-based Model-oriented programming language"; c'est à dire qu'il manipule des langages sous forme de métamodèles dans le but de créer un nouveau langage spécifique composé d'un ou plusieurs fragments de langages.

Melange est écris en grande partie en Xtend, et est basé sur la platforme "client riche" d'Eclipse connue sous le nom de Eclipse RCP. L'utilisation de la plateforme Eclipse permet d'avoir une base de plugins importante et don un travail de développement plus aisé.

Eclipse

RCP - Rich Client Platform

Il s'agit de la plateforme de développement de plugins Eclipse. Néanmoins ce n'est pas un EDI a part entiere. RCP ne fournit pas de client riche de developpement, il fournit uniquement une suite de framework qui permet de developper des "clients riches". Pour faire simple, Eclipse RCP est un sdk[1] pour le développement de plugins Eclipse.

[1]Software Development Kit: Kit de développement de logiciel

Le kit de développement propose tout un panel de "points d'extensions" sur lesquels le développeur vient contribuer. Ce design en point d'extension permet notamment de minimiser la quantité de code à produire dans le cadre du développement de plugins Eclipse, étant donné que bon nombre de ces extensions ont déjà un comportement défini par défaut. Une bonne manièred'apprendre le foncitonnement de la plateforme client riche est de suivre les tutoriels de Lars Vogel disponibles sur le site internet http://www.vogella.com/tutorials/

EMF / ECORE

EMF, ou Eclipse Modeling Framework, est un framework de modélisation et de génération de modèles. Il permet notamment de générer, à partir du modèle, un éditeur associé au modèle, une série de tests ainsi que beaucoup de petits plugins Eclipse comme un système d'hyperlien vers les définitions des objets manipulés, une gestion d'erreur et un déboggeur.

Langages

XTEXT

Xtext est un framework de développement de langages de programmation. A partir d'une grammaire sous la forme EBNF, xtext génère un compilateur pour ce langage ainsi que les fonctionnalités de bases de l'éditeur eclipse associé à ce langage, c'est à dire la coloration syntaxique, l'auto-complétion et une gestion des erreurs de syntaxe.

XTEND

Xtend est un langage de programmation qui, une fois compilé en java, tourne sur la *java virtual machine*. Le principal interêt est l'éclaircissement de la syntaxe de java. Par exemple les expression lambdas, dont la syntaxe est assez repoussantes en Java, deviennent très lisible et utilisable avec xtend. L'expression lambda `anIterable.filter(p -> p.getAttribute() == value);` en java devient `anIterable.filter[it.getAttribute() == value]`. On remarque qu'en java, on doit redéfinir un attribut au début de l'expression alors que cet attribut est implicite avec Xtend. Appliqué à de enchainements de fonctions et des comportements plus complexes, les lambdas java deviennent beaucoup moins lisibles comparés à Xtend.

Pour faire simple, xtend est simplement du sucre syntaxique pour java. Ce qui rends le développement d'applications java plus aisé et moins verbeux.

3

Réalisation

Outline

Une des premiere fonctionnalité d'Eclipse sur laquelle j'ai travaillé été l'outline. Il s'agit d'un panneau qui permet, dans notre cas, l'affichage nos langages sous forme d'arbres.

Eclipse RCP propose, avec les outils de generation d'EMF de creer une outline minimale qui affiche simplement les classes du modèle du langage. Le problème c'est que l'on aimerait bien afficher des informations supplémentaires sur cet arbre, notamment les types des langages, et donc leur hiérarchie, ainsi que les "aspects" de ces langages.

Le principe est assez simple, il suffit d'étendre la classe qui créé l'arbre pour créer nos propres branches et feuilles dans l'abre qui représentent les principales feature du langage. Il est intéressant de noter que la classe de base qui s'occupe de la création de l'arbre, `DefaultOutlineTreeProvider`, utilise principalement les deux méthodes: `createNode(IOutlineNode parent, org.eclipse.emf.ecore.EObject modelElement)` et `_isLeaf(org.eclipse.emf.ecore.EObject modelElement)` qui sont des dispatchers dynamiques capables de "choisir" la bonne implémentation de la méthode en fonction du type du `modelElement`. Mon travail a don été dans un premier temps de sous-typer ces fonctions dans le but de créer mes propres branches de l'arbre.

Par la suite, le travail a été de créer des provisionneurs de labels pour afficher correctement les éléments du langage dans l'abre avec leurs types associés.

Hyperlink

Une feature extrémement pratique de la plateforme Eclipse est l'hyperlinking qui permet, au ctrl+click (ou F3) sur un nom qualifié, d'afficher la définition du-dit nom qualifié. Par exemple lorsque l'on "ctrl+click" sur un appel à une méthode, Eclipse nous affiche la définition de la méthode.

Le problème est que le mécanisme d'hyperliens ne permet de sauter qu'à la définition de noms qualifiés, il ne permet de suivre les URI[1] Eclipse qui sont situés dans une chaine de caractère notamment. La grammaire de melange possède la règle de grammaire suivante `'syntax' EcoreURI = STRING` qui permet de référencer l'URI du metamodel du langage que l'on essaye d'importer. Il peut être pratique de pouvoir naviguer facilement vers ce métamodèle via l'URI fournie et pour cela il faut modifier le comportement du système d'hyperlien de Eclipse.

Etant donné que la feature hyperlien de melange a été généré par xtext, nous avons simplement redéfini le comportement de la méthode `createHyperlinkByOffset(XtextRessource resource, int offset, IHyperLInkAccpetor acceptor)` en contribuant au point d'extension de xtext concerné. Notre nouvelle méthode est un petit dispatcher qui s'occupe de reperé dans quelle partie de la syntaxe se trouve le curseur. Si la resource visé est notre élément "EcoreURI", la méthode créé un hyperlien vers le métamodèle grace à l'URI fournit par l'utilisateur; et dans le cas contraire, la méthode suit le comportement par défaut de création d'hyperlien.

Grammaire et metamodel

Pour continuer plus loin dans l'ajout de fonctionnalités à l'éditeur melange, il a fallut revoir la grammaire et le métamodèle de melange.

Le principal problème rencontré lors du développement de fonctionnalités comme la complétion automatique a été un problème de grammaire. Un exemple, la fonction de merge de melange dont la syntaxe suit la règle de grammaire "`Merge returns merge: 'merge' mergedLanguage = STRING ;`" ne permet pas d'effectuer de complétion automatique puisque xtext ne propose pas de mécanisme de complétion pour des objets de type STRING. Or les seuls objets qui peuvent être merge sont des objets de type `Language`. On peut donc réécrire la précédente règle de grammaire en "`Merge returns merge: 'merge' mergeLanguage = [Language | QualifiedName]`" qui permet, en plus d'accéder aux mécanismes de complétion integrés à Xtext, d'avoir une vérification de type directement

[1]URI: Unified Resource Identifier. Identifiant unique qui permet de repérer un fichier en fonction de son type, de sa plateforme et de son chemin. Dans Eclipse, les URI sont de la forme "plateforme":/"type"/"chemin"/

das l'éditeur et non plus uniquement à la compilation. Cette modification de la grammaire va evidemment de pair avec la mise à jour du métamodèle du langage, en ajoutant un lien de composition entre la classe Merge et le type Language.

Le passage des STRING aux noms qualifiés dans la grammaire de melange a permis d'ajouter une véritable vérification des types qui facilite l'édition des fichiers melange, ainsi que de faciliter améliorer l'auto-complétion par défaut d'Xtext qui sera d'ailleurs la prochaine étape de développement.

Scope

Comme expliqué précédement, la modification de la grammaire melange a permis d'améliorer les possibilités d'auto-complétion et c'est donc la suite logique dans l'ajout de fonctionnalités à melange.

La complétion avec Xtext ressemble assez à l'Outline, il faut faire du polymorphisme sur la méthode `getScope(EObject object, EReference ref)` en sous-typant la classe `AbstractDeclarativeScopeProvider`. Le mécanisme de proposition de complétion est asez simple. L'appel au `CompletionProvider`, dans le cadre de mon implémentation, est un appel à un dispatcher qui appelera la bonne fonction `getScope(EObject object, EReference ref)` avec comme EObject, la règle de grammaire dans laquelle se trouve l'utilisateur, et en EReference sur quel élément terminal précisement.

Par exemple, pour la règle merge expliquée précédement, on va proposer de merge tous les objets de type `Language` présent dans le scope du fichier melange. Pour d'autres fonctionnalités, nous ne voulons pas proposer n'importe quels éléments du scope. Par exemple, dans le cadre de l'héritage, nous ne voulons pas proposer de sous-typer notre langage par lui-même, il faut donc restreindre le nombre de choix de complétion possibles.

Pour ce faire, on récupère tous les objets du scope qui ont le type demandé et on applique un filtre. On recontruit ensuite le scope de scomplétion avec les objets précédements triés.

Menu (W9)

La dernière fonctionnalité sur laquelle j'ai travaillé durant ce stage à été une contribution au menu popup. Plus précisement le travail a consisté à faire apparaitre un sous-menu dans le menu déroulant lorsque l'utilisateur effectue un clique droit sur un fichier d'un langage défini dans un fichier melange.

Lorsque l'on fait un clique droit sur un de nos fichiers de langage, un sous-menu doit apparaitre et proposer d'ouvrir ce fichier avec l'editeur de tous les langages dont il hérite.

Cette partie a été la plus compliquée de ce stage puisque Eclipse ne permet pas nativement d'ouvrir un fichier avec nimporte quel éditeur.

Il faut tout d'abord ajouter une contribution au `popupMenu` pour créer un menu qui contiendra un nombre de commandes variables, ce nombre dépendera du nombre de langages dont hérite notre langage courant. On accède au nombre de langages avec les editeurs desquels on veut ouvrir notre langage courant grace à un registre d'extension melange qui s'occupe de répertorier les sous-types d'un langage présent dans le fichier melange. Ensuite on créé une commande pour chaque langage. On associe à chaque commande une Map de trois paramètres avec comme clés (EXACTTYPE, SUBTYPE, EDITORID) avec EXACTTYPE le type du langage courant, SUBTYPE le type d'un langage sous-type du courant et EDITORID l'id de l'editeur du langage sous-type.

Ensuite j'ai developpé le Handler qui s'occupe de gérer les actions au clique sur une commande. Dans un premier temps, mon idée a été d'utiliser l'id de l'editeur souhaité pour ouvrir un fichier d'un langage A avec l'éditeur d'un langage B. Malheuresement, ce n'est pas aussi simple, Eclipse utilise un système d'"Adapters" pour ouvrir un fichier avec un éditeur. Cette classe permet a Eclipse de faire le lien entre les concepts du langage et la façon de les représenter. Des adapters sont créés par melange à la compilation et font les liens entre les concepts d'un langage et des langages sous-types. Mais ces classes sont complétement hors de portée du plugin, elle sont créées durant la compilation du fichier melange et mon Handler fait partie d'un plugin Eclipse.

Pour remédier à tous ces problèmes, il a été décidé d'utiliser les URI melange qui sont de la forme `melange:/resource/"chemin_du_fichier à ouvrir"?mt="SOUS_TYPE"` qui sert à ouvrir un fichier d'un langage selon un modèle donné en paramètre. Cette URI est traitée par une classe spéciale qui est censée ouvrir une resource selon un modèle compatible passé en paramètre. Malheuresement cette feature n'était pas entièrement terminée durant mon stage et je n'ai eu ni le temps ni les connaissances suffisantes à propos du fonctionnement de melange pour pouvoir corriger tous les bugs. Néanmoins, une fois ces bugs corrigés, melange proposera un système d'ouverture de fichier d'un langage avec l'éditeur d'un sous-type du langage, ce qui est assez inédit.

4

Conclusion

En définitive ce stage m'a permis de découvrir le monde de la recherche en travaillant sur un projet de thèse. Ce projet, Melange, m'a permis d'apprendre une facette de l'informatique que je ne connaissais pas, à savoir l'ingénièrie des modèles. Je remercie particulièrement Thomas Degueule pour avoir passé du temps à m'expliquer ce qu'est un langage et comment fonctionne melange.

De plus ce stage a été un défi technique pour moi; j'ai du apprendre un nouveau langage, xtend, et un nouvel environnement de développement, Eclipse RCP. Ce que j'en retiens, c'est que xtend est une manière sexy de faire du java et que ce langage permet de se détacher de certaines parties "lourdes" de la syntaxe java (notamment les expressions lambdas).

La plateforme riche d'Eclipse quant à elle est assez lourde et difficile d'accès. Son principal problème est un cruel manque de documentation sur son fonctionnement, notamment en ce qui concerne les classes de xtext/emf. Cela a énormement ralenti le développement, a chaque fois que je commençais à travailler sur une nouvelle fonctionalité, je devais faire de l'archéologie pour essayer de comprendre le fonctionnement des classes.

En définitive ce stage a été très enrichissant intellectuellement et m'a permis d'apprendre les bases de l'ingénièrie des modèles.

5

Summary

In the end, this internship made me discover the world of research by working on a thesis project. This project, melange, taught me the basics of model engineering, an IT field I barely knew. Shoutout to Thomas Degueule for having spend a lot of time explaining me what a language is and how melange is working.

Moreover, it has been a technical challenge; I had to learn a new lanaguage syntax, xtend, and a new environnement, Eclipse RCP. What stays in my mind is that xtend is a sexy way of doing Java and it helps focusing on modeling and design because of its powerful yet simple syntax. (Lambdas expressions are a great example)

On the contrary, the Eclipse Rich Client Platform is really heavy and not accessible. The main problem is the lack of documentation about its behaviour, especially when you want to customize thing generated by xtext/emf. This have slowed down the developpement because each time I wanted to work on a new feature, I had to do some archeology and try to understand how things work.

To sum up in a few words, the internship has been very enriching intellectually and taught me the basics of model-driven engineering.