Coulomb Nicolas
Gasiglia Baptiste

Licence Professionnelle
Des Métiers de l'Informatique

OFL-Meta

Réalisation d'un éditeur graphique permettant de définir facilement des composants de langages de programmation conformément au modèle OFL.

Responsables du projet :
M. Pierre Crescenzo
M. Philippe Lahire

Le 21/01/02

Remerciements

Ce projet a été effectué pour le compte de l'Université de Nice-Sophia Antipolis, en collaboration avec le laboratoire I3S (Informatique, Signaux et Systèmes de Sophia-Antipolis), sous la responsabilité de M. Philippe Lahire assisté par M. Pierre Crescenzo.

Qu'il nous soit permis de remercier M. Pierre Crescenzo pour l'aide efficace qu'il nous a apportée, pour sa disponibilité et pour les conseils qu'il a promulgués tout au long de l'évolution du projet.

Nous remercions également M. Philippe Lahire, enseignant responsable de notre projet, pour la confiance dont il a fait preuve à notre égard.

Table des matières

Résumé du rapport

C'est dans le cadre du modèle OFL que s'est déroulé notre projet. Ce modèle fait l'objet d'une thèse réalisée par M. Pierre Crescenzo au sein du laboratoire I3S, dont le sujet exact est « Le modèle OFL pour paramétrer la sémantique opérationnelle des langages à objets : application aux relations inter classes ». C'est dans l'optique de développer des applications mettant en œuvre les caractéristiques de ce modèle, que M. Philippe Lahire a proposé divers projets parmi lesquels, le projet OFL-Meta.

Celui-ci a consisté en la réalisation d'un éditeur graphique permettant de définir facilement des composants de langages de programmation conformément au modèle OFL. Cet éditeur nommé OFL-Meta, développé en Java, servira d'une part à décrire des langages orientés-objets, et sera d'autre part intégré à un ensemble d'applications qui formeront un environnement de développement.

Le rôle d'OFL-Meta est de permettre aux métaprogrammeurs de définir un langage dont le caractéristiques correspondent parfaitement aux besoins du moment. L'objectif fixé n'était pas de terminer l'application, mais d'en avancer le plus possible son développement en respectant les deux axes suivants :

- Produire un code clair et entièrement commenté et
- faire en sorte que celui-ci soit réutilisable.

Notre projet étant désormais terminé, la poursuite du développement (que nous n'avons pas achevé) par d'autres développeurs ne posera aucun problème.

Le présent rapport propose une description de nos travaux et notamment des résultats auxquels nous sommes parvenus.

Introduction

Notre projet d'étude réalisé durant notre année en L.P.M.I. (Licence Professionnelle des Métiers de l'Informatique), l'a été sous la responsabilité de M. Philippe Lahire assisté par M. Pierre Crescenzo.

Le laboratoire I3S souhaite développer une interface graphique permettant de définir des composants de langage objets à classes. Son objectif est, en fait, de proposer un environnement de développement orienté-objets conformément au modèle OFL qui regroupera l'éditeur graphique OFL-Meta mais également un éditeur pour développer des applications depuis une interface UML (nommé OFL-ML) ainsi que d'autre outils. Plus précisément, le modèle OFL est destiné à paramétrer la sémantique opérationnelle des langages objets les plus courant comme C++, Java, Eiffel... L'éditeur OFL-ML fait l'objet d'un autre projet également proposé à des étudiants de la L.P.M.I. de Nice Sophia-Antipolis.

La mission qui nous a été confiée s'intègre dans le projet global d'un environnement de programmation. Elle en constitue d'ailleurs la base même, puisque le logiciel OFL-Meta doit servir à définir les langages dont les caractéristiques seront utilisées plus tard dans le processus de développement d'une application. Développer une application conformément aux attentes des responsables du modèle OFL, avancer le plus loin possible dans le développement du logiciel OFL-Meta, sans pour autant devoir le terminer, faire en sorte que le code fourni soit réutilisable (que l'on puisse aisément comprendre le code réalisé puis y ajouter des fonctionnalités), tels furent les axes essentiels de notre mission.

Dans le cadre de ce projet, le travail sollicité a fait appel à des acquis liés au développement en java, et notamment aux classes graphiques (Swing et awt) ainsi qu'à la compréhension des principes fondamentaux du modèle OFL. En effet, c'est en partant de l'explication du modèle OFL, puis en s'appuyant sur les différentes documentations disponibles sur le langage de développement Java , que le projet a pu progresser.

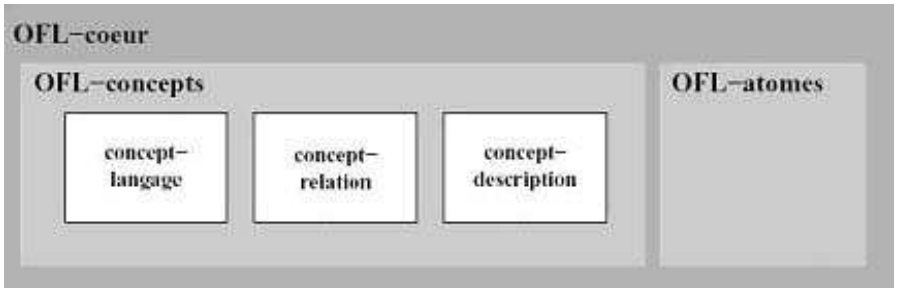
Ce rapport va présenter, expliquer et commenter les différents aspects de notre travail sur le logiciel OFL-Meta. Dans un premier temps, il sera précisé l'intérêt, l'objectif et les opportunités d'un tel logiciel dans le modèle OFL, ce qui constitue une analyse de la situation. Ensuite, les grands axes du projet et de notre mission seront détaillés par rapport aux objectifs qui nous ont été fixés. Les résultats obtenus et apports fournis au cours de notre mission seront recadrés et analysés par rapport au projet initial, nos choix d'implémentations argumentés, ce qui constituera un état des lieu du logiciel tel que nous le rendons. Notons que cette partie est la plus fournie. Nous conclurons par la mise en avant du caractère formateur de ce projet.

1. Intérêt, objectif et opportunité d'OFL-Meta dans le modèle OFL

1.1 Présentation du modèle OFL

L'ensemble des informations relatives au modèle OFL provient de la thèse récemment présentée par M. Pierre Crescenzo, co-responsable de notre projet. OFL s'intéresse aux relations entre classes au sein des langages objets dans lesquels il représente un modèle méta-objet qui peut les décrire, notamment les plus courant comme c++, java ou Eiffel. Il permet donc de définir et d'adapter le comportement et la sémantique opérationnelle des relations entre classes dans les langages objets.

Dans le modèle OFL, nous nous intéresserons plus particulièrement aux OFL-concepts qui font partie de l'OFL-cœur. Le schéma ci-après est une version simplifiée d'un graphique extrait de la thèse de M. Pierre Crescenzo. Il permet de visualiser les trois concepts qui composent les OFL-concepts :



- **Concept-description** :

Les instances de concept-description, c'est à dire les composants-descriptions, représentent la notion de classe (et tout ce qui y ressemble) au sein des langages orientés-objets. Plusieurs paramètres permettent de modifier le comportement d'un composant-description : Name, Context, Genericity, Generator, Overloading…

- **Concept-relation** :

Les instances de concept relation, c'est à dire les composants-relation, représentent les relations qui peuvent être utilisées entre chaque composant d'un langage.
Un concept-relation est une abstraction d'une sorte de relation dans les langages à objets à classes. Les composants-relations, les instances d'un concept relation, peuvent être vues comme des méta-relations. On peut citer plusieurs paramètres utilisés dans le modèle OFL pour modifier le comportement d'un composant-relation : Name, Circularity, Polymorphism-direction, Cardinality…

- **Concept-langage** :

Un concept-langage peut être considéré comme un méta-méta-langage. Chacune de ses instances, nommée composant-langage, décrit un langage modélisé. Son rôle principal est de fédérer des composants-descriptions et des composants-relations.

Ces trois notions sont appelées OFL-composants et définissent un langage. Comme nous venons de le voir, les trois composants citées ci-dessus contiennent chacun un certain nombre de paramètres qui leurs sont propres. Le rôle du logiciel OFL-Meta est justement de pouvoir afficher, modifier puis sauvegarder l'ensemble de ces paramètres pour ainsi générer un langage objet à classes avec les caractéristiques désirées. Ceci est le travail des méta programmeur qui seront les utilisateurs du logiciel OFL-Meta.

1.2 Le logiciel OFL-Meta et ses perspectives d'évolution

1.2.1 Place d'OFL-Meta parmi les différents outils du modèle OFL

Comme nous l'avons déjà expliqué, le logiciel OFL-Meta fait partie d'un ensemble d'outils qui sera utilisé autour du modèle OFL comme environnement de développement. Nous allons donc dans un premier temps, présenter rapidement les quatre outils en cours de développement pour mieux comprendre la finalité du logiciel OFL-Meta :

- OFL-DB

Il consiste en la mise en place d'un formalisme en vue de pouvoir sauvegarder les données traitées par les différents outils. En effet, tous les outils qui gravitent autour du modèle OFL doivent partager les mêmes données. Par rapport à notre projet, c'est le partage des paramètres d'un OFL-composant qui nous intéresse.

- OFL-Meta

Le logiciel OFL-Meta dont le développement constitue notre projet est destiné aux métaprogrammeurs pour leur permettre de facilement créer de nouveaux OFL-composants en définissant les valeurs des paramètres des composants-descriptions, des composants-relations et des composants-langages. D'un point de vue pratique, il s'agira pour les métaprogrammeurs d'attribuer des valeurs aux différents paramètres d'un langage. Il sera ainsi capable de modifier le comportement d'un langage en fonction de ses besoins et d'obtenir le langage « parfait ».

- OFL-ML

Une fois les OFL-composants créés par les métaprogrammeurs et formalisés par OFL-DB, c'est au tour des programmeurs d'utiliser OFL-ML pour créer graphiquement des descriptions, décrire des relations entre elles et donner un corps aux méthodes. C'est donc grâce à ce logiciel qu'il sera possible de développer des applications à partir d'un langage décrit par un méta programmeur. OFL-ML permet de développer des applications en utilisant le formalisme graphique d'UML.

- OFL-Parser

Cet outil doit permettre, à partir de l'application décrite sous OFL-ML de générer une application qui soit exécutable. C'est ce logiciel qui devra mettre en œuvre les actions d'OFL renfermant le code qui tient compte des paramètres des composants.

1.2.2 Rôles et atouts d'OFL-Meta

Cette description simplifiée des différents outils qui seront mis en œuvre pour développer une application met en avant le fait que le logiciel OFL-Meta est amené à évoluer en fonction notamment des spécifications d'OFL-DB.

Ce logiciel tient donc une place primordiale dans la mise en place du modèle OFL étant donné que c'est directement par cette interface que seront apportées les modifications sur la sémantique opérationnelle des langages à objets à classes.

La définition de nouveaux OFL-composants ou la modification de ceux déjà archivés sera accessible à des personne qui n'ont pas forcément une parfaite connaissance du modèle OFL. En effet, tous les paramètres sont modifiables et, pour chacun, un choix des différentes valeurs possibles sera proposé. Le méta-programmeur n'aura donc pas à connaître parfaitement la conception du modèle OFL pour définir un langage adapté à ses besoins étant donné qu'il n'aura qu'a agir sur les valeurs de chaque paramètre.

C'est d'ailleurs là le principal atout du logiciel OFL-Meta. Le fait d'être construit sur une interface graphique intuitive (un arbre décrivant l'OFL-composant courant et les OFL-composants archivés) le rend accessible à tous et simplifie donc la tâche du métaprogrammeur.

2. Les grands axes du projet

2.1 Le cahier des charges du logiciel

2.1.1 Présentation du logiciel OFL-Meta

Dans cette partie, nous allons présenter le logiciel OFL-Meta dans ses grandes lignes, et tel qu'il devrait être à la fin de son développement. Le logiciel OFL-Meta est un éditeur graphique permettant de définir des composants de langages, comme expliqué durant la présentation du modèle OFL. Cet éditeur graphique doit se présenter sous une forme similaire à celle de l'explorateur de fichier de Windows. L'éditeur OFL-Meta doit donc être composé de deux parties :

- Un arbre sur la gauche et
- une partie variable sur la droite.

En ce qui concerne l'arbre, il doit représenter l'espace de travail du méta programmeur en lui permettant d'une part d'accéder à l'OFL-composant sur lequel il travaille, et d'autre part de visualiser l'ensemble des OFL-composant disponibles dans sa bibliothèque.

Comme l'indique le modèle OFL, chaque OFL-composant (aussi bien le langage courant que ceux disponibles en bibliothèque) doit être composé de composants-descriptions , de composants-relations et de composants-langages. Chacun des paramètres de ces composants, ainsi que leurs valeurs associées, sera bien évidemment accessible. Il est évident que ces même paramètres, lorsqu'ils sont archivés, ne doivent pas être modifiables contrairement à ceux du langage en cours de spécification.

Un ensemble de fonctionnalités doit permettre aux métaprogrammeurs d'importer les paramètres de composants-descriptions, de composants-relations et de composants-langages ainsi que leur valeur, depuis les langages archivés vers la même destination dans le langage courant. En effet, des paramètres de composants-descriptions ne doivent pas pouvoir être insérés dans les paramètres composants-relations du langage courant.

Au niveau de la partie variable de l'interface (qui doit se trouver sur la droite) il y a deux applications possibles :

- Afficher / Ouvrir le contenu du répertoire sélectionné
- Afficher / modifier les valeurs du paramètre sélectionné

Cette partie doit se comporter, lorsque l'utilisateur parcourt l'arbre, comme un explorateur « classique ». Nous entendons par explorateur « classique » le fait de visualiser les dossiers ou les nœuds d'un niveau inférieur de l'arbre lorsqu'un répertoire est sélectionné. Bien évidemment, l'affichage doit être interactif et permettre de se déplacer dans l'arborescence à partir des documents affichés.

Dés lors qu'un nœud terminant une branche de l'arbre (dans notre cas il s'agira d'un paramètre) est sélectionné, l'interface de droite doit servir à modifier ou à visualiser la valeur de ce même paramètre. Il faut également que soit présenté à l'utilisateur, la valeur actuelle du paramètre, ainsi qu'une liste des valeurs possibles pour ce paramètre.

On peut enfin préciser que l'ensemble des opérations impliquant l'ouverture ou la fermeture d'un OFL-composant doit également être disponible. Ainsi, l'importation, l'enregistrement ou la mise en archive d'un langage sont des opérations également gérées par le logiciel OFL-Meta. La consigne qui nous a été donnée au niveau du format d'entré et de sortie est d'utiliser des fichier textes (.txt) pour stocker les données. Nous avons vu précédemment que l'étude de ce formalisme des données faisait l'objet d'un projet différent : OFL-DB.

2.1.2 Les consignes de développement

En ce qui concerne le travail requis dans le cadre de notre projet, nous avons dû, durant le développement, suivre deux grands axes :

- Respecter les besoins précédemment décrits et
- fournir un code source clair, commenté et réutilisable.

Il est toutefois nécessaire de préciser que l'objectif du projet n'était en aucun cas de terminer le logiciel OFL-Meta. Nous devions simplement avancer le plus possible notre implémentation qui sera par la suite complétée par d'autres développeurs.
C'est pour cette raison que nous avons attaché la plus grande importance à la clarté de notre code source. Les classes que nous allions rendre devaient être courtes et structurées, l'ensemble de notre code commenté de manière à générer la documentation du logiciel (javadoc).

Enfin, nous pouvons ajouter qu'il nous a été imposé de rédiger l'ensemble des informations écrites (texte visible sur le logiciel mais aussi les commentaires du code source) en anglais. Cette contrainte n'était pas imposée au niveau des noms utilisées dans notre code (attributs, méthodes, classes…), mais conseillée. Nous avons du également tenir compte de la spécification propre à la sémantique des noms précités dans le langage Java.

2.2 Choix du langage et de l'environnement de développement

Le langage Java nous a été imposé pour développer le logiciel OFL Meta. Le choix d'un langage orientés-objets était quasiment inévitable étant donné la structure même du modèle OFL. En effet, le modèle OFL s'appuyant sur des langages objets à classes, il était important d'utiliser un langage répondant aux mêmes critères, ce qui est le cas du langage Java.
L'avantage pour nous d'utiliser ce langage, par rapport à un autre tel que C++, est principalement la simplicité d'implémentation des interfaces. En effet, la majeure partie du développement, dans le cadre de notre projet OFL-Meta, consiste à mettre en place une interface graphique en y greffant les fonctionnalités qui nous sont propres. Sur ce point, les bibliothèques offertes par le langage java (Swing et awt) nous simplifient en partie la tâche.

Enfin, nous avons choisi d'utiliser le logiciel JCreator (version 2.0). Ce logiciel ne fournissant ni débuggeur ni assistant de développement, nous avons codé l'ensemble des instructions manuellement. Pour exécuter le logiciel OFL-Meta, nous avons utilisé le JDK 1.3 et le compilateur fourni avec JCreator.

2.3 Méthode de travail

Aux vues des instructions précédemment exposées, nous avons extrait trois grandes phases de développement pour mener à bien notre programme de travail, et c'est en respectant cette méthodologie que nous avons pu progresser :

2.3.1 Mise en place de l'interface homme-machine

Cette étape qui est la première chronologiquement consiste à mettre en place les classes graphiques et leurs écouteurs en prévision de l'insertion des fonctionnalités.
Par rapport aux consignes données, c'est au cours de cette phase de travail que sera décrite la structure définitive de l'arbre et de l'affichage dans la partie droite.

2.3.2 Intégration des fonctionnalités

Il faut noter que nous considérons comme fonctionnalités l'ensemble des opérations qui doit être mis à la disposition de l'utilisateur pour qu'il puisse faire évoluer le langage dont il est en train de spécifier le comportement, par rapport à ses besoins. C'est donc durant cette étape que seront ajoutées toutes les fonctions de déplacement des paramètres comme expliqué précédemment.

2.3.3 L'utilisation et la sauvegarde d'informations

Cette dernière phase, qui devra constituer la fin du développement du logiciel, nous permettra d'insérer les fonctionnalités de sauvegarde des OFL-composants utilisés. D'un point de vue graphique, c'est au cours de cette étape que l'ensemble des possibilités offertes à l'utilisateur dans le menu « File » sera implémenté.

Une fois cette méthodologie établie, nous allons pouvoir nous attacher à présenter notre travail et l'implémentation que nous avons réalisée du logiciel OFL-Meta.

3. Résultats et description de l'implémentation d'OFL-Meta

3.1 Description du travail

Nous pouvons distinguer différents types de travaux : ceux qui suivent la méthodologie prédéfinie et qui nous ont amené à rédiger nos différentes classes ; ceux qui résultent des consignes générales au niveau du développement et qui s'appliquent à l'ensemble de nos classes.

3.1.1 Description détaillée des classes

Notre travail a consisté, dans son intégralité, à développer en Java le logiciel OFL-Meta tout en respectant les consignes définies auparavant. C'est en suivant la méthodologie présentée dans le paragraphe 2.3, que nous sommes parvenus à rédiger dix classes et une interface.

Les classes :

- Fenetre
- JtdTools
- LeftTree
- MenuFile
- MenuPopup
- NodeComponent
- PopupListener
- RightPanel
- OFL-Meta (nécessaire pour exécuter le programme, donc non traitée dans ce rapport)

L'interface :

- ConstantesString (celle-ci est une interface que nous traiterons dans le paragraphe 3.1.2)

Pour décrire le travail qui fut le notre nous commencerons par détailler les classes qui correspondent à la **mise en place de l'interface homme-machine**.
Il convient de préciser que nous avons utilisé la plupart du temps des classes en provenance du package « javax.swing » car ces classes nous offrent un plus grand nombre de fonctionnalités sur les objets graphiques.

- La classe Fenetre (voir annexe 1):

La classe Fenetre est la classe principale de l'application OFL-Meta. C'est elle qui comporte les différents objets. Elle hérite de la classe « JFrame » et c'est dans celle-ci que sont créés les panneaux nous permettant de définir l'aspect général de l'application :

En voici les principaux :

- Les Barres de navigation (jScrollBar) : pour rendre utilisable en toutes occasions l'affichage
- Le panneau qui contiendra les menus (jLabel)
- Le panneau central (jScreenSeparator), séparé en deux et qui contient l'arbre de gauche(LeftTree) et le panneau de droite (RightPanel)

Nous avons géré l'organisation des différents panneaux à l'aide d'un containeur (Container)

- La classe MenuFile (voir annexe 2):

La classe MenuFile est celle qui permet de créer le menu de l'application OFL-Meta. Elle hérite de la classe « JMenuBar » et comporte un écouteur.

Voici la forme des menus :

File	Option	About
New	Add	A Propos
Open	Remove	Help
Import	Rename	
SaveAs	Copy	
Save	Paste	
Quit	Collaspe	
	Expand	

Pour réaliser ce menu, nous avons utilisé des « jMenu » et pour les remplir, des « jMenuItem ». Il faut noter qu'il a été ajouté pour chaque menu et sous-menu des raccourcis direct depuis le clavier.

- La classe JtdTools (voir annexe 3):

La classe JtdTools est une classe qui n'est actuellement pas utilisée par l'application. Sa création se ferait dans la classe Fenetre (elle y est actuellement en commentaire). Nous l'avions codée dans les première phases du développement puis supprimée car jugée inutile. Toutefois, nous avons souhaité la conserver au cas où elle deviendrait nécessaire. Cette classe hérite de « JToolBar » et permet de créer une barre d'outil sous le menu général. Elle est composé de « JButton » qui sont des raccourcis vers les fonction du menu (l'écouteur sur ces boutons n'est pas implémenté)

Voici la liste des boutons :

New	Open	Save	SaveAs	Add	Remove

- La classe RightPanel (voir annexe 4):

La classe RightPanel est la classe qui contient l'affichage de droite. Elle hérite de la classe « JPanel » et n'est pas encore associée à un écouteur, ce qui permettrait d'envisager l'interactivité de son affichage. En l'état d'avancement du développement, elle se compose de trois fonctions :

- addDir : elle permet d'ajouter à l'interface de droite (le jPanel) des « jLabel » associés avec un « jTextField » (ce qui représente une image de répertoire) lorsque la branche inférieure de la sélection de l'arbre contient des répertoires. Cette fonction est appelée pour chaque répertoire trouvé dans la sélection.
- addFile : cette méthode repose sur le même principe que la fonction « addDir » lorsqu'il s'agit de nœud de terminaison (de paramètres dans le modèle OFL)

- addSubData : elle permet de modifier la partie droite de l'interface lorsqu'un nœud est sélectionné. Nous n'avons pas terminé la mise en place de cette partie de l'interface, mais il est évident que cette méthode entraînera la création d'une nouvelle instance d'une classe qu'il reste à développer. L'affichage alors ouvert permettrait la modification de la valeur du paramètre sélectionné.

Il est utile de noter que pour chacune de ces méthodes, nous devons passer en paramètre la référence de l'arbre pour pouvoir accéder aux informations relatives à l'actuelle sélection dans celui-ci. Actuellement, la gestion de l'affichage dans la partie droite classique est une succession de « jLabel » et de « jTextField ». Il nous semble plus efficace, d'envisager cette gestion à l'aide d'un « jTable ».

- La classe LeftTree (voir annexe 5):

La classe LeftTree contient l'arbre de l'application OFL-Meta. L'arbre est de type « jTree », et la classe LeftTree ne sert en réalité qu'a contenir cet arbre. C'est dans cette classe que sont ajoutés les différents nœuds de l'arbre.
Il faut noter que la création de ces nœuds est réalisée par la classe « NodeComponent » que nous verrons plus tard.

Enfin dans cette classe nous avons également créé une instance de la classe « PopupListener » pour préparer la seconde phase : intégration des fonctionnalités

Une fois ces cinq classes développées, nous avons considéré que la mise en place des interfaces était terminée et, suivant notre méthodologie, nous avons commencé à **intégrer les fonctionnalités** demandées. Il est à préciser que nous avons laissé de coté la partie droite de l'application (RightPanel) pour nous focaliser sur les fonctionnalités de l'arbre :

- La classe NodeComponent (voir annexe 6):

La classe NodeComponent est la plus importante du logiciel OFL-Meta. Dans l'avancement future du programme, il pourrait être préférable d'en extraire la création complète d'un langage vers une classe différente pour soulager la classe NodeComponent qui actuellement est la plus longue de toutes.

La classe NodeComponent hérite de la classe « DefaultMutableTreeNode ».
Premièrement, chaque instance de cette classe contient des attributs (booleens) qui nous permettent de modifier le comportement du nœud :

- isNameLanguage : le nœud est le paramètre de nom (Name) ou non
- isModifiable : la valeur de ce nœud (paramètre d'après OFL) est modifiable ou non
- isAddable : on peut ou non ajouter des sous dossiers à ce noeud
- isRemovable : possibilité ou non de supprimer le nœud (s'il est dans l'archive ou dans le langage courant)
- isCopiable :on peut ou non copier le nœud

Tous les accesseurs et les modificateurs des ces attributs sont également implémentés dans cette classe. C'est notamment grâce à ces attributs qu'il nous est possible de proposer à l'utilisateur un menu contextuel (« MenuPopup »).

Ensuite, nous avons codé dans cette classe **l'ensemble des fonctionnalité** qui sont proposées à l'utilisateur dans le menu « Option ». Ce sont ces même fonctionnalités qui se retrouveront dans la classe « MenuPopup » détaillée ci-après. Nous avons choisi de les développer dans cette classe pour des raisons pratiques. En effet, chaque instance de cette classe étant des nœuds, il nous est plus aisément donné d'agir sur ces même nœuds directement dans la classe NodeComponent.

Enfin, nous avons implémenté dans la classe NodeComponent, les fonctions d'ajout et de suppression des nœuds ainsi que la fonction « addLanguage ». Cette dernière constitue la partie de cette classe dont le traitement devrait être exporté vers une tierce classe. La fonction « addLanguage », appelée depuis la classe « LeftTree », nous permet de formaliser la structure d'un OFL-composant. Le traitement de cette méthode n'est rien d'autre qu'une succession de créations et d'ajouts de nœuds.

- La classe MenuPopup (voir annexe 7):

La classe MenuPopup est celle qui regroupe l'ensemble des fonctionnalités disponibles. Elle hérite de la classe JPopupMenu et construit un menu composé de « JMenuItem ». Le menu généré par cette classe est contextuel et le constructeur de la classe récupère les informations sur la sélection actuelle dans l'arbre : Nous réalisons un contrôle sur chaque attribut du nœud sélectionné pour vérifier si la fonction associée doit être ou non disponible.
Un « ActionListener » attend que l'utilisateur sélectionne un des choix proposés et appelle la fonction de la classe « NodeComponent » correspondante.

- La classe PopupListener (voir annexe 8):

La classe PopupListener est la classe qui permet de rendre accessible dans la classe LeftTree, la sélection actuelle dans l'arbre. En effet, nous créons une instance de cette classe qui hérite de « MouseAdapter » dans LeftTree. Cela nous permet d'obtenir la sélection dans l'arbre en passant en paramètre l'instance courante de la classe LeftTree. Nous utilisons la classe PopupListener pour modifier la comportement de la classe RightPanel par l'intermédiaire de ses méthodes précédemment détaillées.

L'intégration des fonctionnalité est donc une phase qui n'est pas encore terminée, bien que certaines d'entre elles soient déjà implémentées. En ce qui concerne l'utilisation et la sauvegarde d'informations, nous n'avons pas eu le temps d'implémenter les fonctions répondants à ces besoins.

3.1.2 Application des consignes à l'ensemble du code

Par rapport aux consignes qui nous ont été données et que nous avons détaillées précédemment il est important de signaler que nous avons développé une interface non demandée (ConstantesString) qui contient la quasi totalité des textes utilisés dans notre code. Etant donné que notre travail sera susceptible d'évoluer, nous avons jugé qu'il était préférable de mettre en place cette interface qui permet une plus grande souplesse dans la gestion des langages (voir annexe 9).

De façon générale, nous avons commenté de la manière la plus claire possible l'ensemble de nos classes. Les commentaires sont en anglais (pour la plupart) et permettent de générer la documentation (javadoc) du logiciel OFL-Meta. La rédaction de ces commentaires nous a pris un certain temps, mais cela permettra à d'autres programmeurs de rapidement comprendre nos choix d'implémentations pour mieux pouvoir prendre la suite du développement.
Pour les méthodes, nous avons a chaque fois ajouté les commentaires @pre (les pré condition de la méthode) et @post (post condition de la méthode). Ceux-ci viennent s'ajouter aux commentaires déjà spécifiés en Java : @param (les paramètres de la méthode), @since (version du JDK depuis laquelle il est possible de compiler la méthode) et @see (référence vers une méthode ou un attribut d'une autre classe). (voir sources).

Nous avons également du déclarer l'ensemble des attributs de nos méthodes (dans la mesure du possible) de type « private ». Cette étape nous a obligé à implémenter tous les accesseurs vers ces même attributs (voir sources). Cette modification nous a été demandée en cours de réalisation du projet et nous avons donc du reprendre les codes déjà écrits.

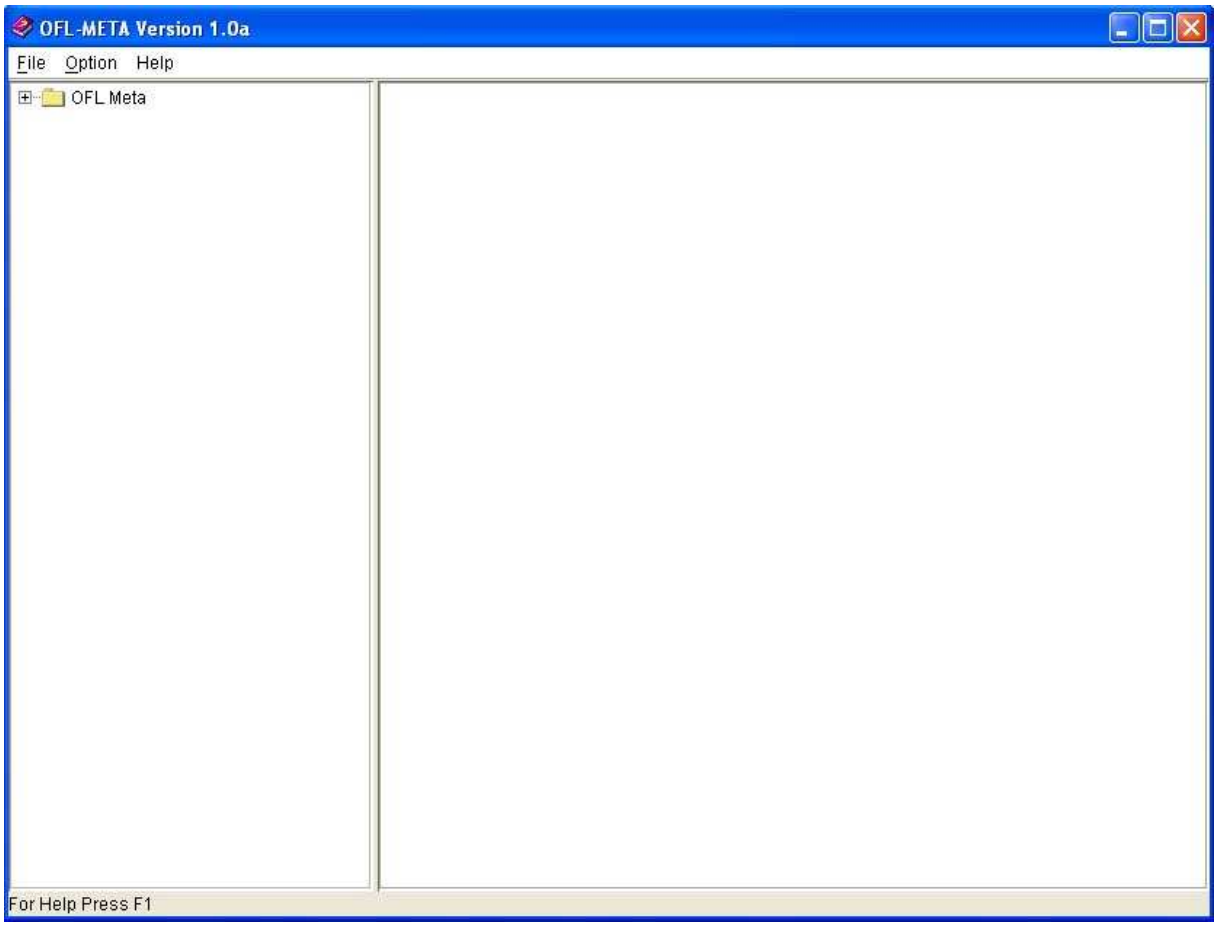
Enfin, nous avons essayé, dans la mesure du possible, de structurer nos classes de manière à ce qu'il soit possible de les faire évoluer aisément. Lorsque nous pensions que nos choix de classes n'étaient pas forcément optimaux, nous l'avons signalé dans la description des classes précédentes.

3.2 Les fonctionnalités actuelles d'OFL-Meta

Comme nous l'avons montré au cours du détail des différentes classes que nous avons développées, le logiciel OFL-Meta n'est pas terminé. Cette partie est destinées à faire un état des lieu du logiciel c'est à dire à mettre en avant les parties ou fonctionnalités qui sont entièrement complétées, celles qui ont été commencées sans être pour autant achevées et enfin présenter quelques idées que nous proposons pour éventuellement améliorer certaines parties du logiciel.

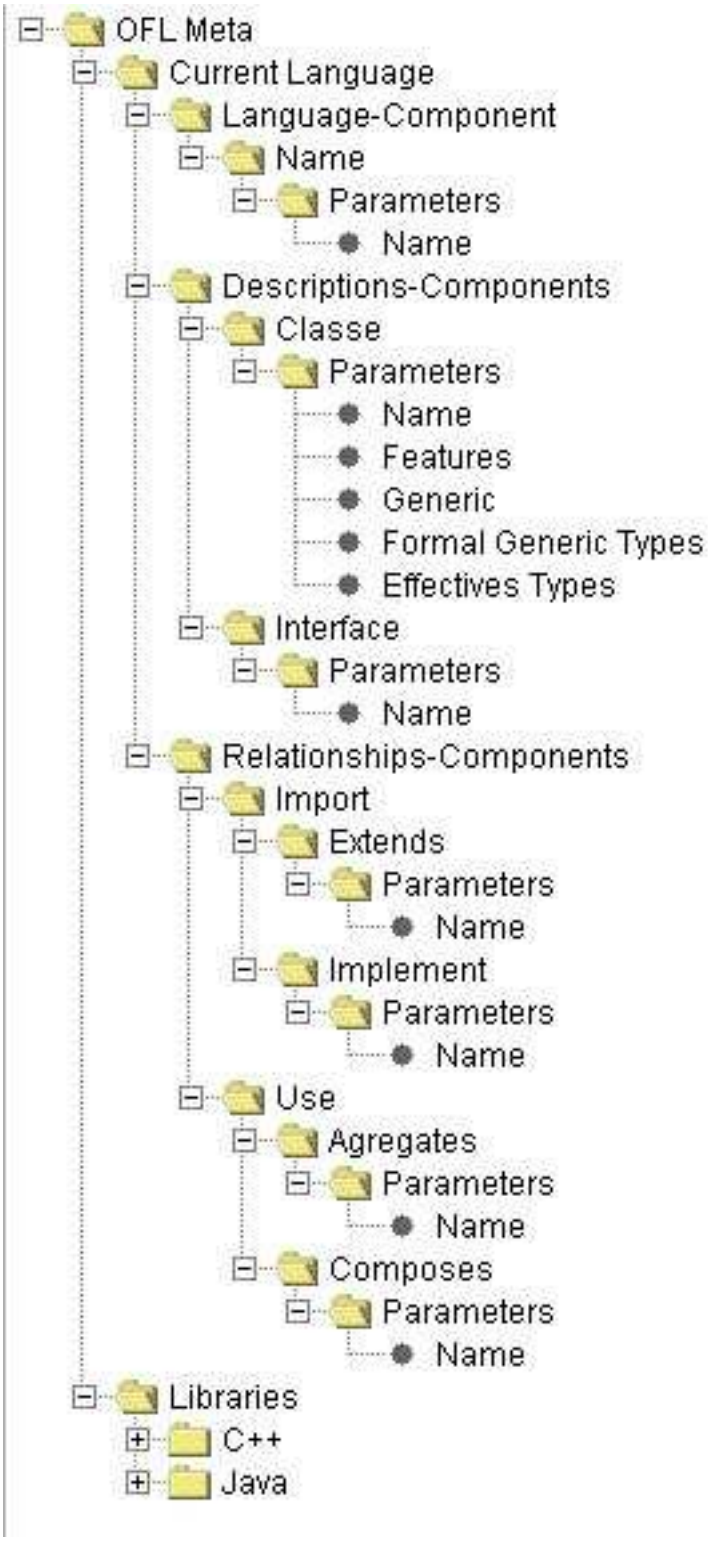
3.2.1 Fonctionnalité achevées ou en cours d'achèvement (captures d'écrans)

Capture d'écran générale du logiciel OFL-Meta (tel que nous le rendons à la fin de notre projet) :



On constate que l'interface générale ressemble bien à celui d'un explorateur de document classique sous Windows.

Capture d'écran de l'arbre ouvert :

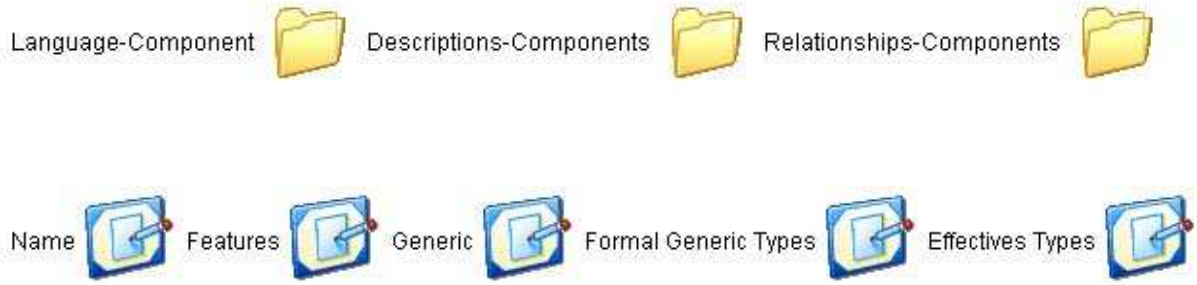


On peut remarquer que l'arbre de gauche a sa forme définitive conformément aux explications présentées dans la seconde partie de ce rapport. En effet, comme nous l'a demandé le responsable du projet, nous nous sommes principalement consacrés à la mise en place de cet arbre et de toutes les fonctionnalités qui y sont associées.
A noter qu'il est possible de renommer les nœuds, qui peuvent l'être, au travers d'une fenêtre indépendante s'ouvrant à l'exécution de la méthode « rename » :

Capture d'écran du menu contextuel (Menu popup) :

Un clic droit sur un élément de l'arbre nous ouvre un menu déroulant contextuel qui donne accès pour l'utilisateur à un certain nombre de fonctionnalités. Il convient de préciser que, bien entendu, toutes les fonctionnalités ne sont pas exécutables sur tous les éléments de l'arbre. Dans le cas ou une fonctionnalité n'est pas applicable à un élément sélectionné, celle-ci est grisée et n'est donc pas accessible par l'utilisateur. Toutes les fonctions présentes sur ce menu et reprises dans le menus général Option (voir ci-dessous) de l'application fonctionnent quand cela est nécessaire.

Capture d'écran de la partie droite de l'interface (classique) :

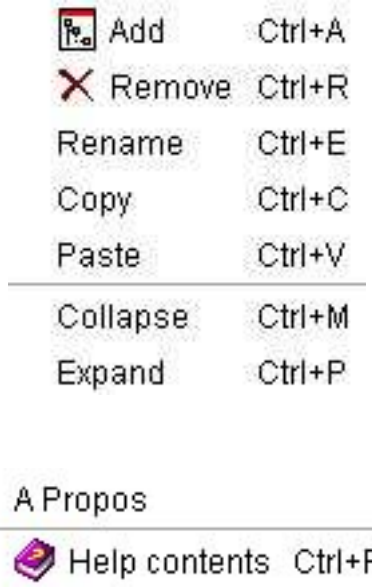


Cette partie de l'interface affiche exactement les informations (répertoires ou paramètres) qui composent le contenu du répertoire sélectionné dans l'arbre.
Toutefois, il faut préciser qu'elle n'est pas encore interactive (les éléments ne sont pas cliquables) et l'affichage n'est pas optimisé.

Capture d'écran de la partie droite de l'interface (saisie des données) :

Ce formulaire de saisie est, bien évidemment, affiché lorsque cela est nécessaire (si un paramètre est sélectionné dans l'arbre), mais nous n'avons pas mis en place les méthodes permettant de lister les valeurs possibles. L'affichage de la valeur actuelle du paramètre est néanmoins réalisé.

Capture d'écran des différents menus :



On constate la présence de raccourcis directs depuis le clavier vers les différentes fonctions des menus. Pour le menu « File » aucune fonction n'est implémentée, mais l'écouteur est déjà en place. En ce qui concerne le menu « Option » vous constaterez qu'il n'est pas encore contextuel (toutes les fonction sont disponible quelque soit la sélection dans l'arbre) mais que les fonctions sont bien exécutables depuis ce menu. Enfin nous avons simplement fait en sorte que les fonctions « A Propos » et « help contents » du menu « Help » affiche des fenêtres sommaires dont voici les captures d'écran.

3.2.2 Poursuite du travail et possibilités d'amélioration

Le logiciel OFL-Meta n'étant pas terminé, il est bien évident que de nombreux points sont susceptibles d'être améliorés. Nous souhaitons mettre en avant les finitions et les améliorations sur lesquelles nos successeurs dans le développement devront attacher la plus grande importance :

- Rendre contextuel le menu « Option »
- Améliorer l'affichage classique de la partie droite
- Travailler sur la saisie des données (non développé en l'état actuel du logiciel)

Ces opérations constitueraient la fin de notre phase d'intégration des fonctionnalités définies par notre méthodologie (voir paragraphe 2.3). Il reste encore bien évidemment à mettre en place l'ouverture et la sauvegarde des valeurs des paramètres des OFL-composants.

Conclusion

Le sujet de notre projet "réaliser un éditeur graphique permettant de définir facilement des composants de langages de programmation" s'est avéré être une mission longue car il nous a été, notamment, demandé de faire en sorte que les parties développées soient réutilisables. En fait, nous avons surtout travaillé sur l'arbre du logiciel avec ses fonctionnalités, délaissant du même coup la partie droite de l'interface.

Le logiciel OFL Meta, intégré au modèle OFL et à son environnement de développement, doit permettre de simplifier le travail du métaprogrammeur de par son interface simplifiée et intuitive. Nous avons donc apporté le plus grand soin à la qualité de l'interface graphique présenté tout en mettant en place le maximum des fonctionnalités demandées.

Même si elle n'était pas prévue dans la mission initiale du projet, la réalisation d'une interface contenant l'ensemble des chaînes de caractères sera selon nous utile pour les futurs développeurs du logiciel OFL-Meta. Elle s'intègre, de ce fait, dans la logique de réutilisation du code inhérent aux objectifs du projet.

La particularité de ce projet réside dans le fait d'utilisé le modèle OFL que nous avons ainsi pu découvrir. Le fait de travailler sur un des langages les plus évolués de nos jours nous a permis de passer un cap dans la maîtrise de celui-ci. Bien que nous n'ayons pas eu de contraintes de délai stricte, nous avons du apprendre à développer de manière intelligente et « propre ».

Ce projet a nécessité une implication rapide et constante, un travail à la fois autonome et en liaison régulière avec le responsable du projet, une méthodologie rigoureuse et formatrice.
Les acquis en matière de programmation ont été d'autant plus renforcés que la mission s'inscrivait dans un programme concret et ambitieux s'appuyant sur modèle novateur OFL. L'étude de ce modèle nous a permis de voir autrement les concepts des langages objets.

Contacts

Etudiants ayant travaillés sue le projet OFL-Meta :

M. Nicolas Coulomb : kift@ifrance.fr

M. Gasiglia Baptiste : baptiste.gasiglia@clicasso.org

Responsables du projet OFL-Meta :

M. Pierre Crescenzo : http://www.crescenzo.nom.fr

M. Philippe Lahire : http://verdon.unice.fr/élahire/

M. Robert Chignoli : http://www.i3s.unice.fr/~chignoli/