


# Présentation de StarUML

par Matthieu Brucher (<http://matthieu-brucher.developpez.com/>) (Blog)

Date de publication : 15/04/2008

Dernière mise à jour :

StarUML est un logiciel de modelage  **UML** qui est entré récemment dans le monde de l'OpenSource. Ecrit en Delphi, il est modulaire et propose plusieurs générateurs de code. Qu'en est-il de ses capacités ? Quels sont ses défauts ?

Introduction/Installation

I - Première utilisation

II - Création d'un projet simple

II-A - Use Case

II-B - Diagramme de classes

II-C - Diagramme d'implémentation

II-D - Mini-conclusion

III - Les générateurs et générateurs inverses

III-A - Génération de documents

III-B - La génération de code C++

III-C - Le lecteur de source C++

III-D - Le module patterns

Conclusion


Avantages

Inconvénients

Conclusion de la conclusion

## Introduction/Installation





Le site Internet de StarUML se trouve à **cette adresse**. Disponible uniquement sous Windows - pour cause d'utilisation d'objets COM -, le téléchargement s'effectue sur **Sourceforge**.

Outre l'exécutable, il peut être utile de télécharger l'un ou l'autre module, d'autres sont déjà installés, comme le générateur de code source C++, C#, Java, ... ou le module  **patterns**. Nous testerons le module de génération de code C++ ainsi que le module de patterns. Le code est aussi téléchargeable **ici**.

Pour finir l'introduction, le logiciel n'est disponible qu'en anglais, et l'aide en anglais ou en coréen.

## I - Première utilisation

Après l'installation, StarUML peut être directement lancé. En regardant dans la documentation, on se rend compte que StarUML supporte la version 1.4 d'UML ainsi que la notation de la version 2.0. L'approche MDA est mise en avant par le logiciel de part sa capacité à générer du code à partir d'un modèle UML pour une plateforme .Net ou J2EE par exemple.

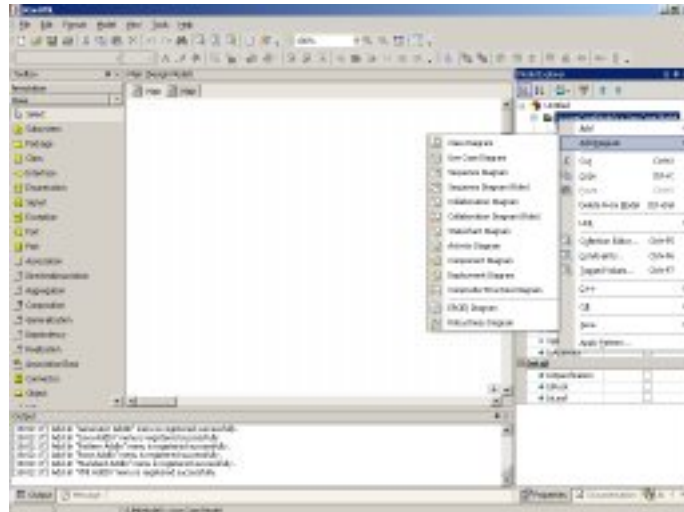
Au démarrage, StarUML propose plusieurs nouveaux patrons de projets. Par exemple, les approches Rationale, 4+1 view peuvent être utilisées, chaque approche ouvrant plusieurs diagrammes par défaut. A ce propos, l'approche par défaut comprend un diagramme des  **cas d'utilisation**, d'analyse, de  **classe**, d' **implémentation** (aussi appelé de composants) et de  **déploiement**.

Ceux qui ont l'habitude des Visual Studio ne se sentiront pas perdus non plus, l'interface graphique étant très proche du célèbre IDE de Microsoft.



## II - Création d'un projet simple

On va maintenant voir comment StarUML tient le choc dans l'utilisation courante avec le développement d'un projet. On va donc créer un projet avec l'approche de StarUML.



### II-A - Use Case

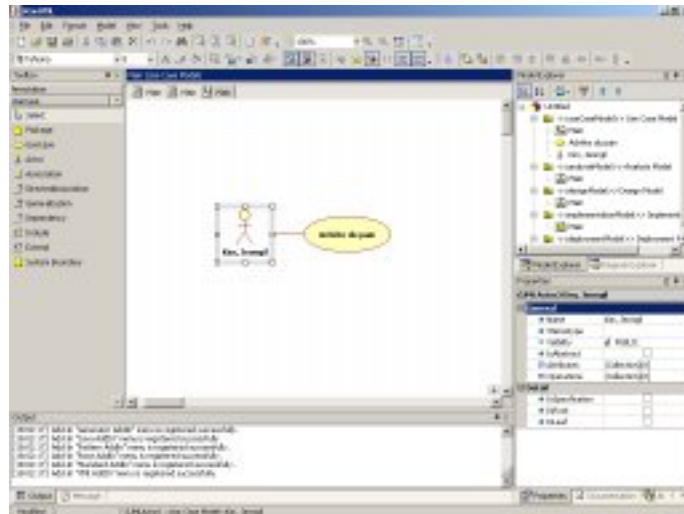


*Création d'un nouveau diagramme Use-Case*

Pour commencer, on va ouvrir la fenêtre des  **Use Cases**. Pour créer un nouveau  **diagramme**, il suffit d'appuyer avec le bouton droit sur le dossier dans lequel créer le nouveau diagramme et sélectionner "Add Diagram". De même, on peut aussi créer un nouveau diagramme dans le menu "Model"->"Add Diagram".

Il paraît qu'on peut activer le mode Drag & Drop pour placer des éléments, mais je n'ai pas trouvé... Enfin, donc on clique sur ce qu'on veut puis on clique sur le diagramme pour les placer. Les noms donnés aux éléments reflètent bien l'origine de StarUML, la Corée du Sud.

En revanche, ce qui est vraiment sympa, c'est la possibilité d'avoir des raccourcis en entrant le nom d'un nouvel élément. Par exemple, pour créer des  **Use Cases** reliés à un  **acteur**, il suffit de modifier le nom de l'acteur en ajoutant "-()" devant le nom du nouveau cas d'utilisation. Cela donne quelque chose de ce genre :



*Création d'un Use Case par raccourci*

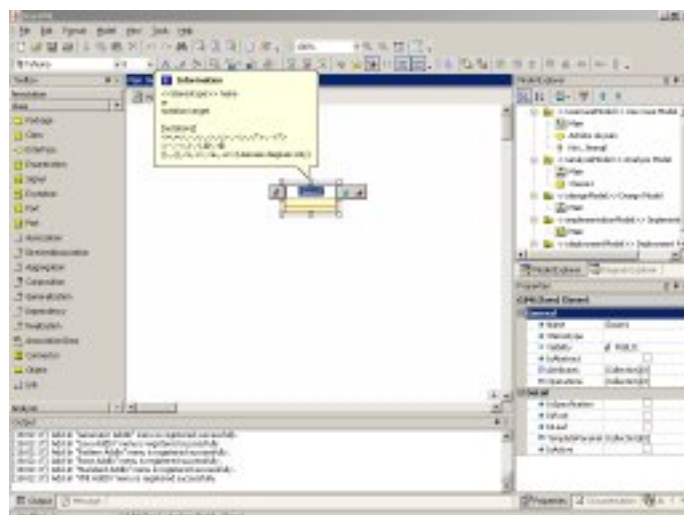
Bien évidemment, il y a d'autres raccourcis, dépendant des diagrammes à créer, mais il faut un peu d'habitude pour s'en servir. Tous ces raccourcis sont affichés sous forme de tooltips, c'est bien pensé, et ça ne surcharge pas trop l'affichage.

On peut aussi créer des **diaco** **associations** entre Use Cases et acteurs, on peut créer des généralisations de Use Cases, d'acteurs, des inclusions, des extensions, ...

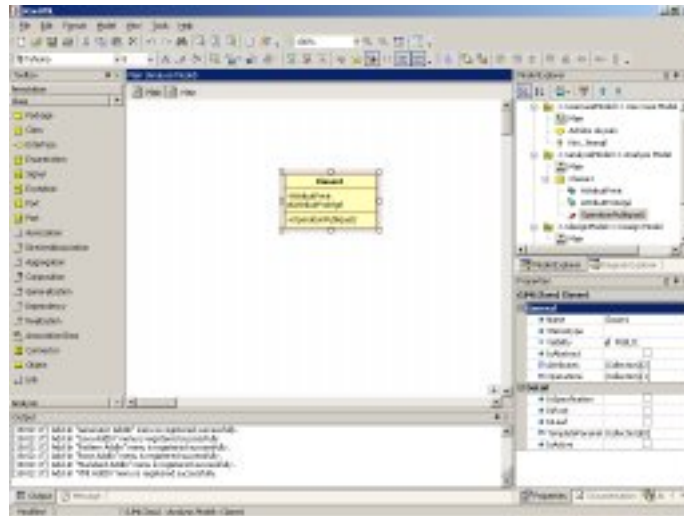
## II-B - Diagramme de classes

Il existe un type de diagramme que nous n'allons explorer, le diagramme d'analyse qui est une extension du diagramme des classes.

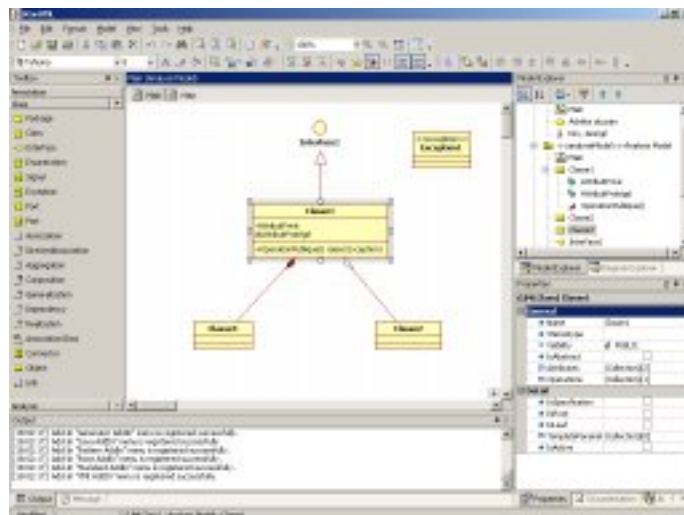
On va donc créer une nouvelle **diaco** **classe Class1**. Ensuite, pour ajouter des **diaco** **attributs** ou **diaco** **opérations**, on peut soit les gérer dans l'éditeur de collection - Ctr + F5 ou bouton droit -> éditeur de collection -, soit en double-cliquant sur la classe. Dans ce dernier cas, avec le boutons de gauche, on indique si l'attribut sera **private**, **protected**, ... tandis que les boutons de droite permettent de créer une propriété ou une méthode. Dans l'éditeur de collection, on constate qu'on peut ajouter des paramètres templates, on applaudit, même si c'est un requis d'UML.





### Créer des attributs d'une classe avec le double-clic

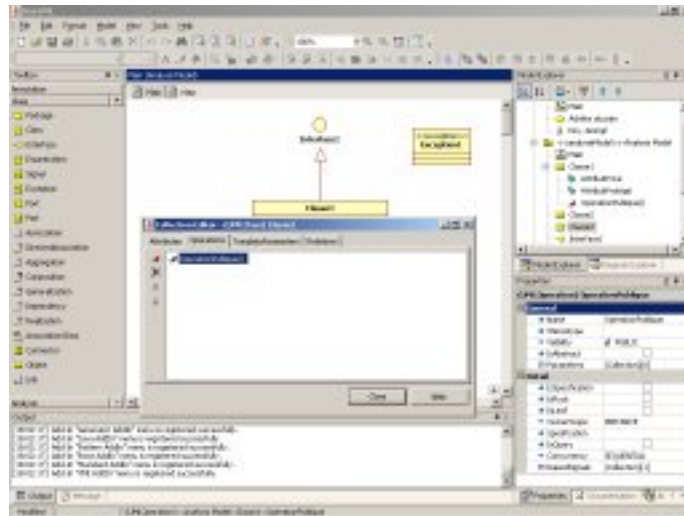


### Création d'attributs d'une classe avec le double-clic



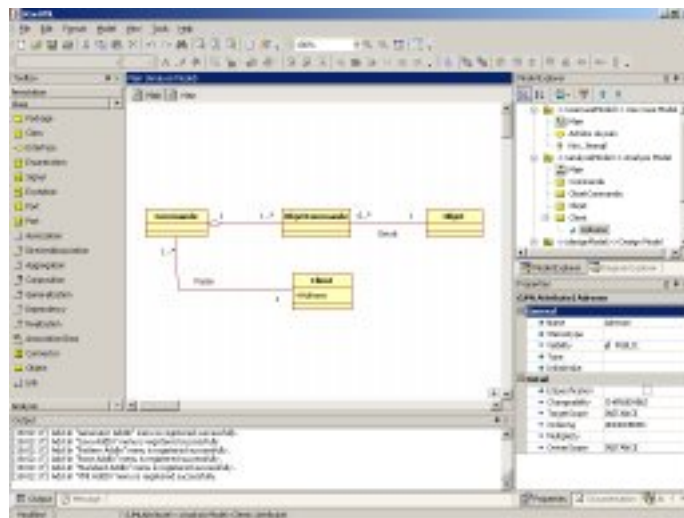
### Exemple avec plusieurs classes dans le diagramme

Le but n'étant pas de faire un cours d'UML, on observe simplement la richesse des diagrammes de StarUML. Outre les classiques  **associations**,  **dépendances**, ... on note la simplicité pour ajouter les spécifications d'exceptions, il suffit d'ajouter l'exception, de chercher l'opération dans le Collection Editor et on voit en bas à droite la ligne Raise Signal qui contient la liste des exceptions pouvant être levées.



*Le champ Raised Signals en bas à droite pour une opération*

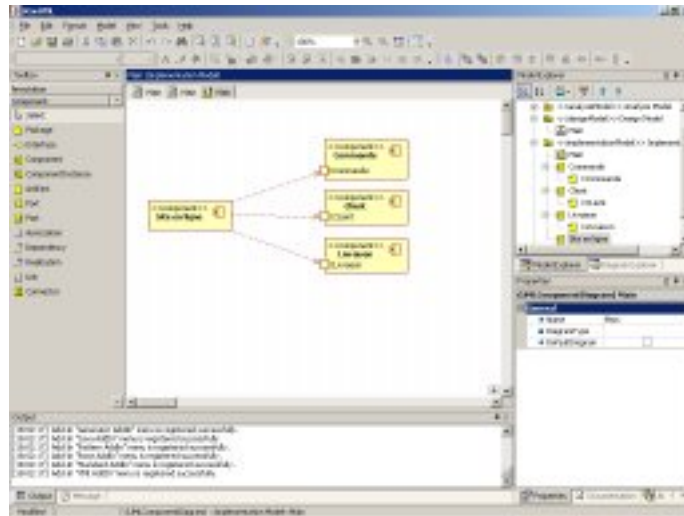
En revanche, pour accéder à ces spécifications, c'est un peu long... Sinon, on peut aussi générer des signaux, concept qui gagne du terrain en C++ grâce à Qt et Boost.



*Un diagramme plus complexe*

Sur cet exemple, on se rend compte que les directions pour les liens d'association ne sont pas visibles, il faut sélectionner DirectedAssociation pour en avoir. Pour ajouter les indications de **multiplicité** et autres, il faut double-cliquer aux extrémités du lien.

## II-C - Diagramme d'implémentation



*Le diagramme de composant*

On voit à nouveau que l'utilisation de ce diagramme est intuitive avec StarUML, à partir du moment où on sait ce qu'on fait.

## II-D - Mini-conclusion

Les autres diagrammes existent aussi, naturellement, il suffit d'en créer de nouveaux et de faire ce que l'on veut. On peut aussi ajouter des contraintes, celles-ci pourront être écrites dans chaque diagramme dans l'éditeur de contraintes - Ctrl + F6 ou bouton droit -> Constraints -. On note tout de même que l'aide est peu locale sur ces contraintes, à priori le langage utilisé est le Object Constraint Language.

Dans chaque diagramme, on peut directement utiliser certains éléments existants dans des diagrammes précédents. Par exemple, les classes des diagrammes de classes sont utilisables dans le diagramme d'implémentation, ou de séquence.

### III - Les générateurs et générateurs inverses

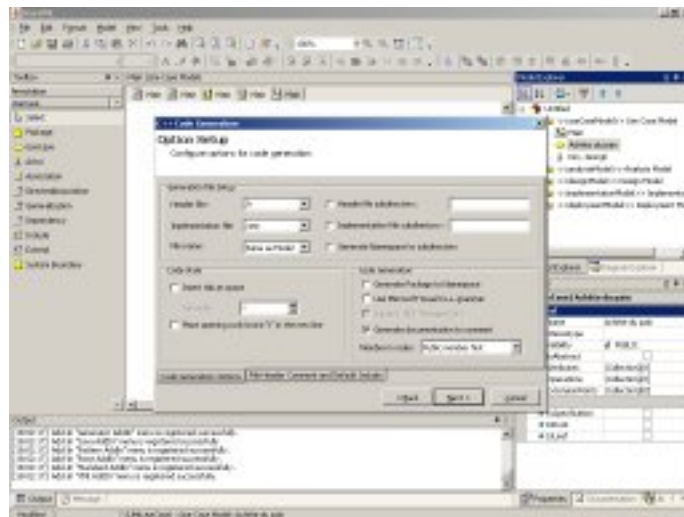
Il existe plusieurs type de générateurs, qu'il s'agisse des générateurs de code ou de documentation. Chacun répond à un besoin précis.

#### III-A - Génération de documents

Les générateurs de documents génèrent des documents à partir des diagrammes. Ces documents sont par exemple des fichiers .doc qui contiennent la description des Use Cases avec tous les détails qu'on a spécifiés à celui-ci - Ctrl + F7 ou bouton droit -> Tagged Values -.

Il faut donc suffisamment documenter ses diagrammes pour avoir des documents complets. Rien de nouveau sous le soleil.

#### III-B - La génération de code C++



*La boîte de dialogue de génération de code C++*

Pour pouvoir générer un code, il faut tout d'abord inclure le profil associé dans les modèles. Cela se fait dans le menu Model->Profiles...

On peut générer du code associé à chaque modèle généré. La boîte de dialogue de génération de code est très exhaustive, plusieurs paramètres peuvent être changés, certains négligeables - l'ordre des attributs/méthodes -, d'autres moins comme les noms de fichiers.


Des tags spéciaux permettent de particulariser le code, comme les attributs <<CppSourceFile>> à entrer avant le nom d'une classe - c'est ce qu'on appelle le stéréotype -. La liste des stéréotypes est disponible dans le fichier d'aide associé au générateur.

En ce qui concerne l'utilisation de la bibliothèque standard, je n'ai pas pu essayé, il faudrait sans doute utiliser les contraintes pour guider le choix des conteneurs.

#### III-C - Le lecteur de source C++

Le module d'importation de code source est efficace, les classes importées conservent les fonctions méthodes/variables, associations, dépendances, mais elles ne sont pas visibles dans le diagramme, dommage.

### III-D - Le module patterns

Ce module permet d'utiliser les  **Design Patterns** pour créer des diagrammes de classe automatiquement. Il connaît les 23 patterns du Gang Of Four, mais aussi d'autres. Il suffit d'indiquer lequel on veut, de spécifier quelques particularités, et le diagramme est créé. Facile. Naturellement, il s'agit juste d'un exemple de diagramme pour ces patterns, parfois un autre peut être plus judicieux.

## Conclusion

Ce logiciel est bien fait, facile à prendre en main, pour simplifier la conclusion, on va simplement présenter ses avantages et inconvénients immédiats.

## Avantages

Le premier avantage est le fait que tous les diagrammes UML 1.x peuvent être générés. Les petits trucs en plus sont appréciables pour créer des objets dans un diagramme, ajouter rapidement des attributs, ... Le fait que le code source soit disponible est aussi un avantage indéniable pour l'utilisateur afin d'assurer la pérennité du logiciel.

On critique parfois certains diagrammes de séquence dans certains logiciels. Apparemment, celui-ci de StarUML est conforme au standard.

## Inconvénients

Dans les inconvénients, on peut citer le fait qu'il ne soit disponible que sous Windows. De même, l'importation des sources n'est pas parfaite, lorsqu'on ajoute une classe importée dans un diagramme, les connexions avec les autres classes ne sont pas affichées. Enfin, la navigation dans les fenêtres nécessite une habitude qui peut être longue à prendre.

Un inconvénient plus dérangeant est la fâcheuse tendance des flèches à ne pas être droites. Ceci est dû au dimensionnement automatique, donc pour avoir des flèches droites, il faut le retirer, or des flèches non droites ne sont pas visuellement intéressantes, donc le dimensionnement automatique est inutile...

## Conclusion de la conclusion

Pour conclure, le logiciel a l'air plus stable que l'un de ses équivalents libres sous Linux, Umbrello, puisque ce dernier a planté plusieurs fois lors de tests, contrairement à StarUML. Pour plus de captures d'écran, vous pouvez aussi aller voir sur **le site Internet du logiciel**.

