



Warning

The information contained in this document is the exclusive property of Aethera Networks. Except as specifically authorized in writing by Aethera Networks, the holder of this document shall keep all information contained herein confidential and shall protect it in whole or in part from disclosure and dissemination to all third parties

This is a controlled document.

Printed copies must have revision number verified prior to each use.

Product:		
Project: EA Code Engineering		
Subject: Tutorial		
Date: (day-month-year)		
Document Nr: DOC-XXXX	Revision:-AP1	Part Nbr: (optional)

Change History

Author	Date (day/month/year)	Revision	Description of change
Simon Rodrigue	11/04/2007	AP1	Creation

Approvals

Name (Originator)	Title	<input type="checkbox"/>	Signature	Date
Team Leader				
Name	Title	<input type="checkbox"/>	Signature	Date
Manager				
Name	Title	<input type="checkbox"/>	Signature	Date
 <input type="checkbox"/> 				
Name	Title	<input type="checkbox"/>	Signature	Date

Table of Contents

1. INTRODUCTION	2
1. ASSOCIATIONS ET ATTRIBUTES.....	2
1.1. Ajout d'association	2
1.2. Lien entre attributs et association	2
1.2.1. Cas 1 : Compositions ou agrégations simples.....	2
1.2.2. Cas 2 : Compositions ou agrégations particulières.....	2
1.3. Collection classes	2
1.4. "Member Type" dans "Association property"	2
1. SYNCHRONISATION DU CODE AVEC LE MODÈLE	2
1.1. Options par défaut pour la génération de code	2
1.2. Options particulières pour la génération de code	2
1.2.1. Element properties	2
1.3. Package == Namespace.....	2
1.4. « Synchronize » ou « Generate source code » ?	2
1.5. Déclaration à l'intérieur de la classe parfois embêtante	2
1.6. Stereotype et Tag.....	2
1.7. Effacé du diagramme != effacé du modèle	2
1.8. Options de visibilité	2
1.9. Extériorisation d'une Note d'un Élément.....	2
1.10. Génération de commentaires.....	2

1. Introduction

Enterprise Architect permet entre autres de faire de la modélisation logicielle avec UML, mais en plus d'assurer la synchronisation entre les modèles et le code. C'est un outil puissant qui peut sembler imposant mais qui est très intuitif. Par contre, au début, il peut être difficile de retrouver ce que l'on veut à travers la grande quantité d'options. Ce petit tutoriel souligne les principales connaissances requises pour faire la synchronisation entre un modèle UML et le code sans problème.

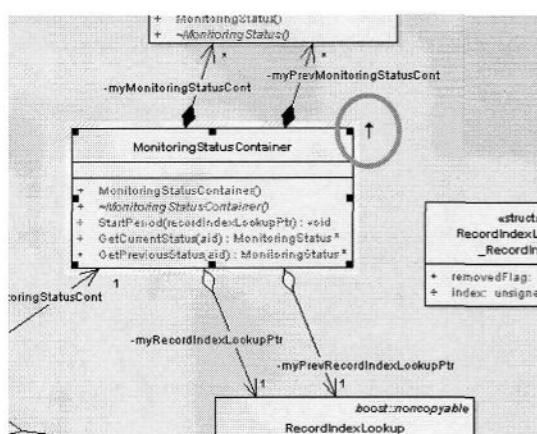
2. Associations et Attributs

L'association est la seule caractéristique de EA que j'ai trouvé moins intuitif. Voici les principaux points à savoir :

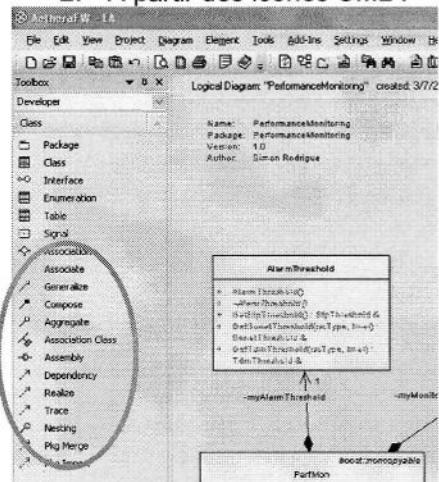
2.1. Ajout d'association

Il y a 2 façons de créer des associations :

1. À partir de l'icône contextuel, lorsqu'un élément UML est sélectionné (flèche sur le coin supérieur droit):



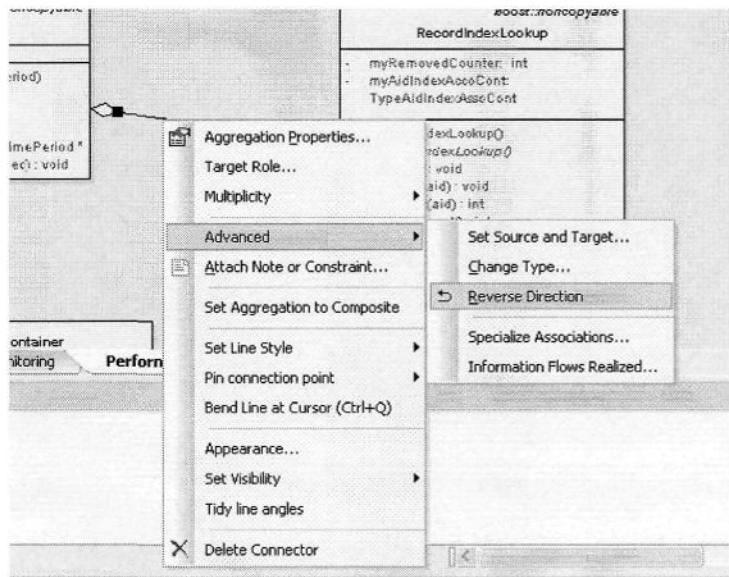
2. À partir des icônes UML :



ATTENTION : Ces deux méthodes ne fonctionnent pas de la même façon dans le cas d'agrégation et de composition.

- Pour la méthode 1, la classe à partir duquel vous créée l'association sera le *target* et la destination sera la *source*.
- Pour la méthode 2, la classe à partir duquel vous créée l'association sera la *source*, et la destination sera la *target*.

Généralement, la **source** devrait être l'objet parent qui contient l'autre objet. Donc, le **target** sera l'objet enfant. Par contre, il n'est pas nécessaire de suivre cette règle puisque les options pour la *source* ou le *target* sont identiques. Il y a même une option dans le menu contextuel qui permet de les échanger :



Dans le cas de généralisation (dérivation) les deux méthodes fonctionnent de la même façon.

2.2. Lien entre attributs et association

2.2.1. Cas 1 : Compositions ou agrégations simples

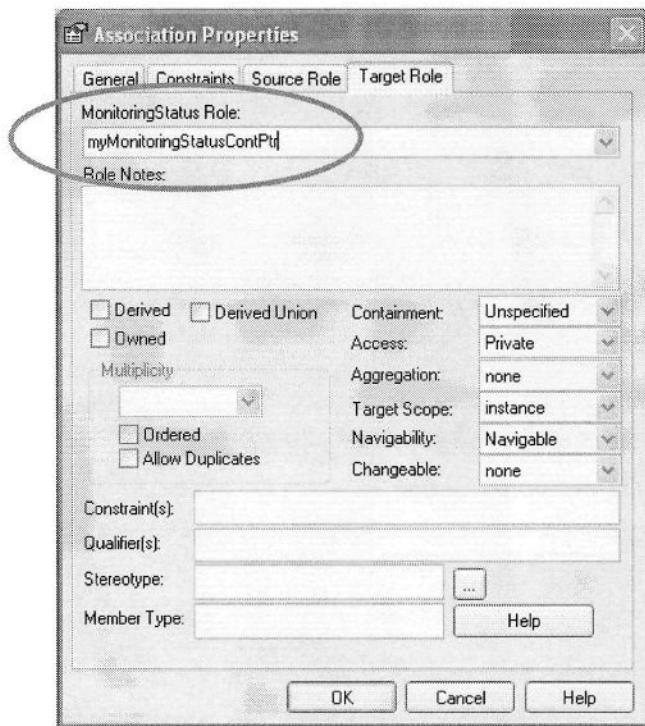
Lorsque l'on crée une association entre deux classes et que le code correspondant voulu est comme ceci :

```
class ParentClass
{
...
...
private:
    ChildClass1 myChildClass1; // composition simple
    ChildClass2 *myChildClass2; // agrégation simple
}
```

Ce cas est automatiquement bien géré par EA. La synchronisation du code avec le modèle ce fait dans les deux sens sans problème.

Association role == Attribute name

Si on veut changer le nom par défaut que EA va donner à l'attribut correspondant à l'association, il faut changer le champ rôle dans les propriétés de l'association :



Lorsqu'un attribut et une association ont le même nom, EA tient pour acquis qu'ils correspondent au même élément. Plusieurs options se recoupent pour l'attribut et l'association, s'il y a des conflits, les options de l'attribut seront considérées au détriment des options de l'association. Dans un tel cas, lors d'une validation du modèle (ctrl-alt-v), des warnings seront générés sur les incohérences entre l'attribut et l'association.

2.2.2. Cas 2 : Compositions ou agrégations particulières

Lorsque l'on crée une association entre deux classes et que le code correspondant voulu est semblable à ceci :

```
class ParentClass
{
...
private:
    // composition, cardinalité multiple
    std::vector<ChildClass1> myChildClass1Cont;

    // agrégation avec smart pointer
    boost::shared_ptr<ChildClass2> myChildClass2Ptr;
}
```

Ce cas est supporté par EA mais pas de façon automatique. Il est quand même facile de faire en sorte que la synchronisation entre le modèle et le code fonctionne bien pour ce genre de situation.

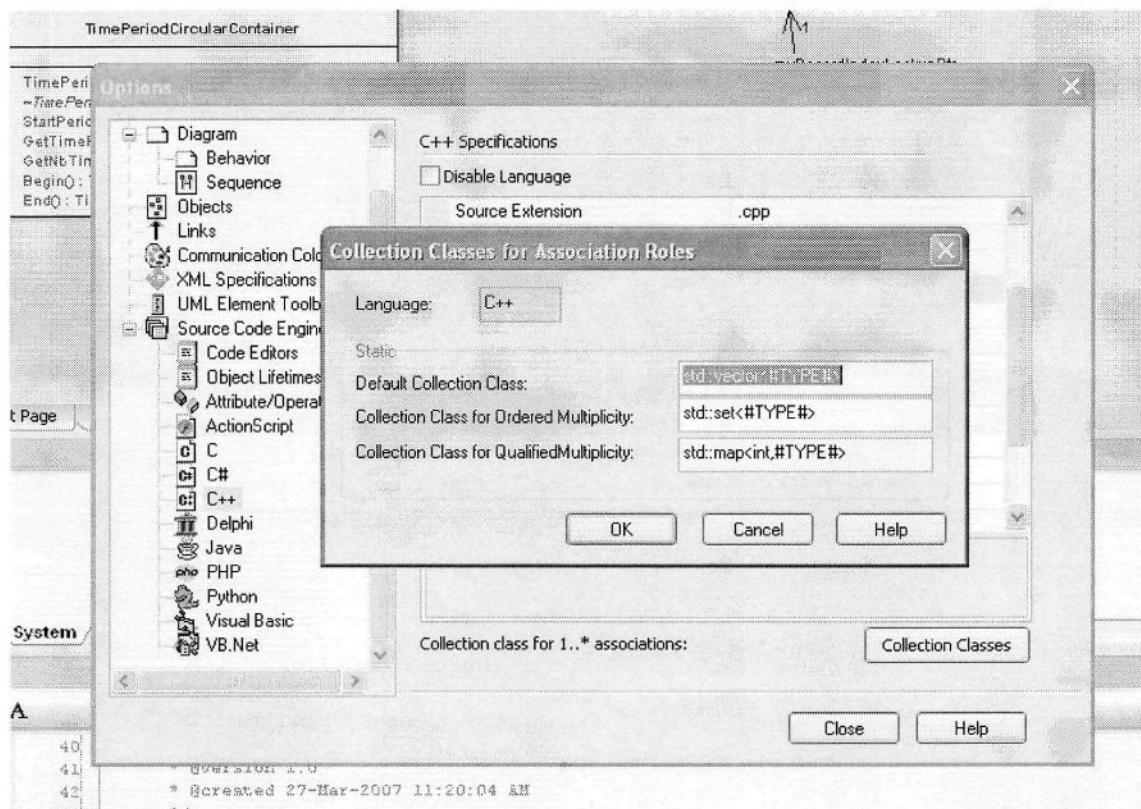
Association role == Attribute name

Il suffit d'ajouter une association au diagramme pour lequel on spécifiera un rôle qui aura le même nom que l'attribut correspondant.

Par contre, cette technique entraîne automatiquement la génération de warning lors de la génération du modèle. Il est possible de configurer la validation pour désactiver certaines règles pour ne plus voir ces warnings.

2.3. Collection classes

Il est possible de spécifier le container par défaut lorsqu'une classe aura une cardinalité supérieure à 1. L'identificateur #TYPE# sera remplacé par le nom de la classe.

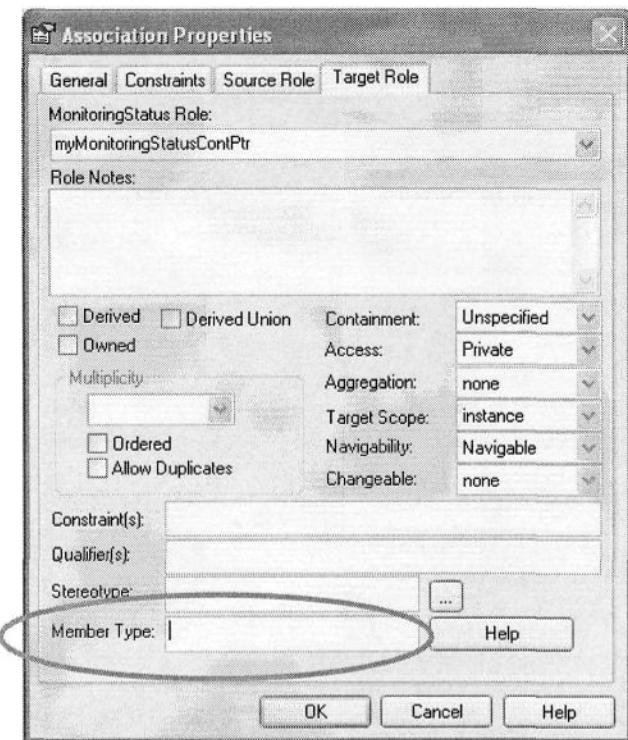


Il est aussi possible de spécifier des *collection classes* particulières pour chacune des classes dans les propriétés de celles-ci.

Cette caractéristique permet à EA de générer le code voulu lorsqu'il y a une cardinalité supérieur à 1.

2.4. “Member Type” dans “Association property”

Permet de spécifier un type particulier pour une association. ATTENTION : Ce type est considéré seulement si la cardinalité de l'association est supérieure à 1.

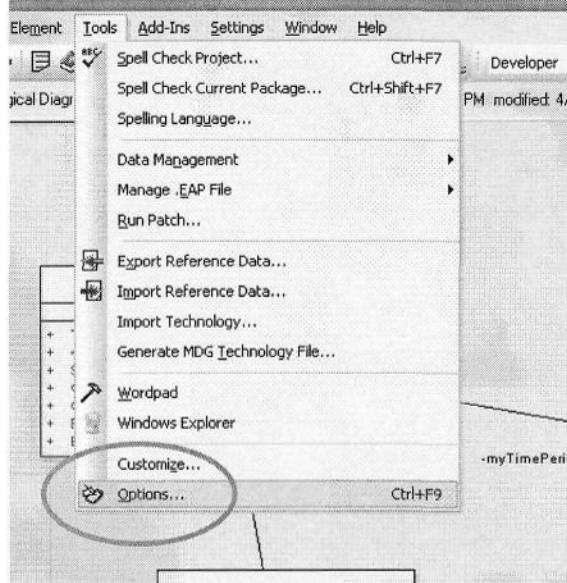


Cette caractéristique permet à EA de générer le code voulu lorsqu'il y a une cardinalité supérieur à 1.

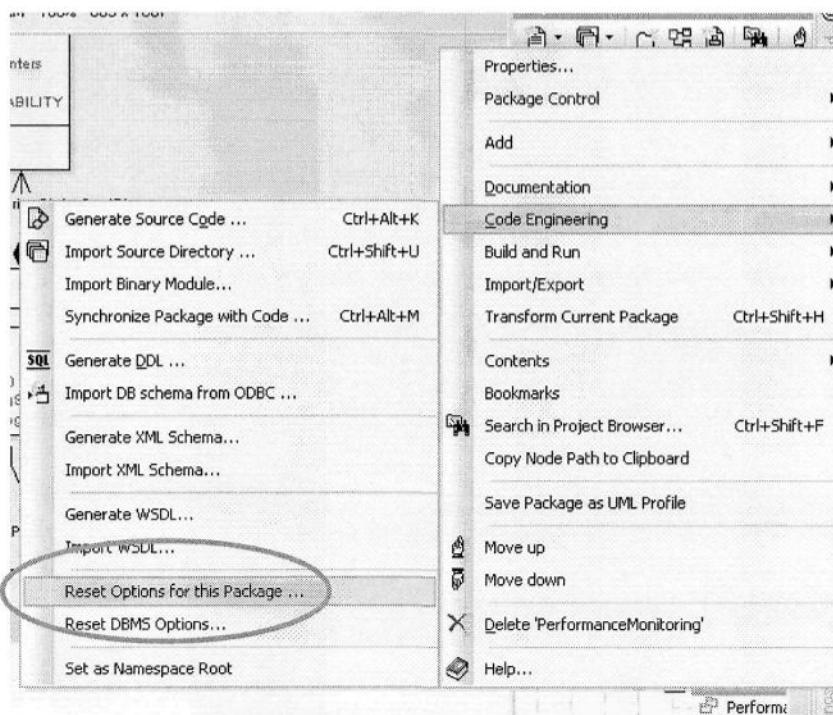
3. Synchronisation du code avec le modèle

3.1. Options par défaut pour la génération de code

Il est possible de spécifier les options par défaut pour le « code engineering »:

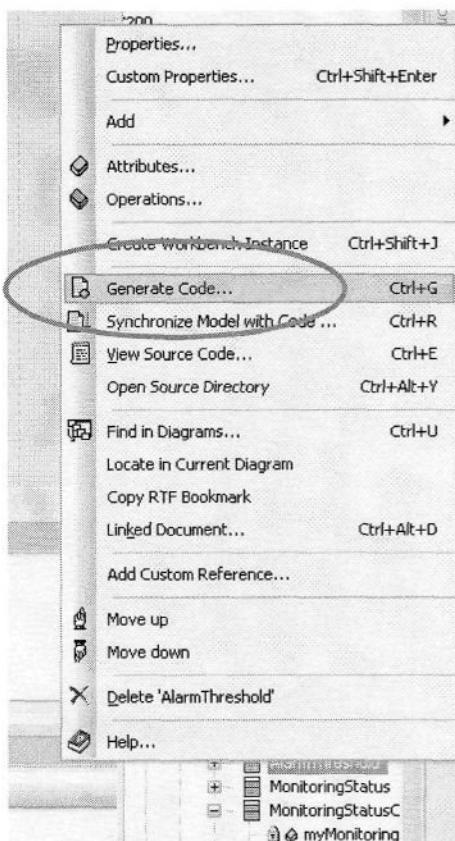


ATTENTION : Lorsque les options par défaut du « code engineering » sont changées, ils n'affecteront pas les éléments UML déjà présent dans le diagramme. Il est possible d'appliquer ces nouvelles configurations par le menu contextuel des packages :



3.2. Options particulières pour la génération de code

Il est aussi possible de changer les options de « code engineering » pour chaque élément de façon indépendante :

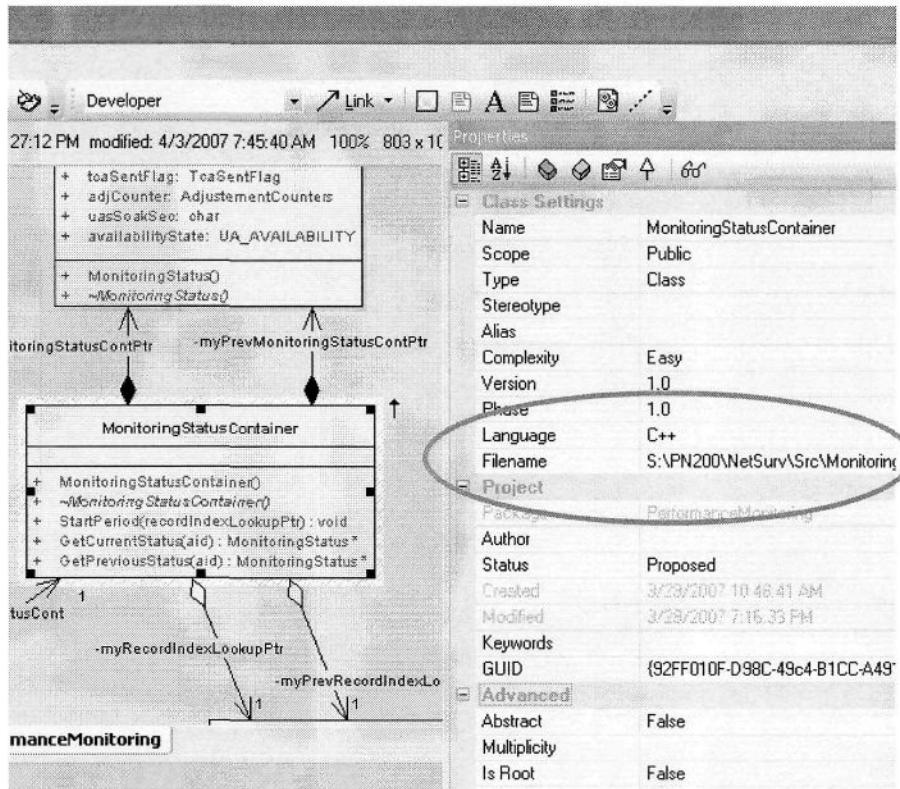


Le menu « generate code ... » a 2 principales fonctions :

- Configurer les options de génération de code et le chemin du fichier à synchroniser
- Effectuer la synchronisation du code

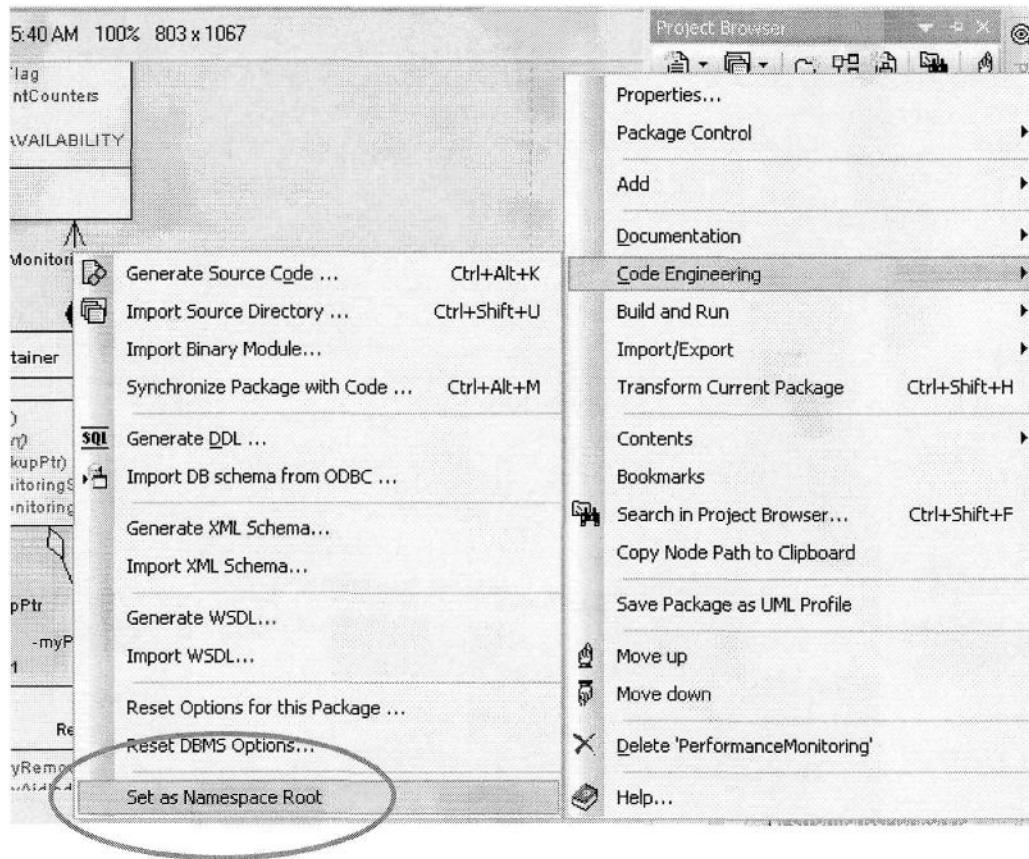
3.2.1. Element properties

Il est aussi possible de changer le chemin du fichier lié à un élément UML par l'onglet « properties » :



3.3. Package == Namespace

Pour chaque package, un *namespace* dans le code sera généré. Par contre, si il y a plusieurs packages imbriqués dans le modèle, le code généré risque d'être monstrueux si on ne limite pas la profondeur des *namespaces*. On peut faire cela en spécifiant un certain package comme étant un *namespace* racine :



3.4. « Synchronize » ou « Generate source code » ?

« Generate source code » devrait être utilisé initialement pour configurer des options particulière à l'élément, générer des fichiers sources inexistant, ou synchroniser avec des fichiers existants.

Lorsque les fichiers sources sont bien liés aux éléments du modèle et que les options sont configurées, le bouton « synchronize package with code » est plus rapide car il ne présente pas le dialogue de configuration.

3.5. Déclaration à l'intérieur de la classe parfois embêtante

Lorsqu'un `typedef` est déclaré dans le scope d'une classe, EA va toujours ajouter au diagramme ce `typedef` et ses associations lors de la synchronisation code → modèle. Cela n'est pas toujours souhaitable si on ne veut pas que ce `typedef` soit présent dans le diagramme. Une façon simple d'éviter cela est de mettre le `typedef` à l'extérieur du scope de la classe.

Exemple :

```
class ClassA
{
    typedef ClassB TypeClassB;
    ...
    TypeClassB myClassB;
}
```

Dans cet exemple, quand la classe `ClassA` sera synchronisée, le `typedef TypeClassB` sera ajouté ainsi qu'une association de `ClassA` vers `TypeClassB`. Si ce n'est pas souhaité, il ne suffit que de déplacer le `typedef` à l'extérieur de la déclaration de `ClassA`.

3.6. Stereotype et Tag

Les stéréotypes et tags permettent de spécifier la façon dont le code sera généré à partir d'un élément UML. Par exemple :

- Pour générer un enum, ajouter le stereotype « enum » à un élément de type `class`
- Pour générer un `typedef`, ajouter le stereotype « `typedef` » à un élément de type `class`
- etc.

Voici les stéréotypes et tags prédéfinis pour le C++ :

Stereotype	Applies to	Corresponds To
enumeration	Class	An enum type.
struct	Class	A struct type.
property get	Operation	A read property.
property set	Operation	A write property.
union	Class	A union type.
typedef	Class	A <code>typedef</code> statement, where the parent is the original type name.
friend	Operation	The friend keyword.

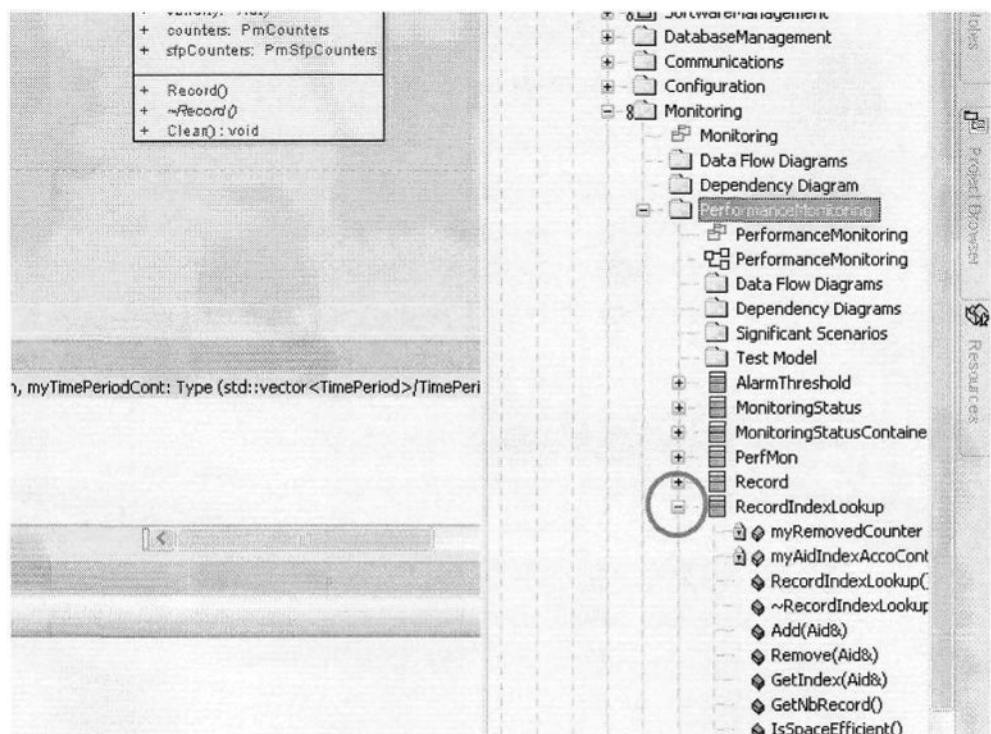
Tag	Applies to	Corresponds To
-----	------------	----------------

typedef	Class with stereotype other than typedef	This class being defined in a typedef statement.
anonymous	Class also containing the tagged value typedef	The name of this class being only defined by the typedef statement
attribute_name	Operation with stereotype "property get" or "property set"	The name of the variable behind this property.
mutable	Attribute	The mutable keyword.
inline	Operation	The inline keyword and inline generation of the method body.
explicit	Operation	The explicit keyword.
callback	Operation	A reference to the CALLBACK macro.
initializer	Operation	A constructor initialization list.
bodyLocation	Operation	The location the method body is generated to. Expected values are header, classDec or classBody.
typeSynonyms	Class	The typedef name and/or fields of this type.
throws	Operation	The exceptions that are thrown by this method.
afx_msg	Operation	The afx_msg keyword.
volatile	Operation	The volatile keyword.

3.7. Effacé du diagramme != effacé du modèle

Si vous effacez un élément UML d'un diagramme, cet élément est toujours présent dans le modèle et sera regénéré lors de la synchronisation du code. Si vous voulez voir cet élément disparaître, il faut aussi l'effacer du modèle (project browser).

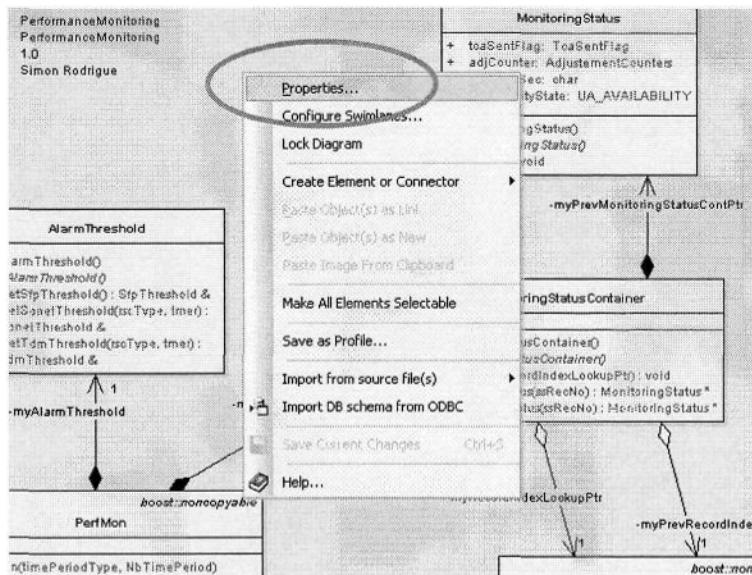
ATTENTION : Des éléments comme des *class* ou des *typedef* peuvent ce cacher sous les *class*, donc pas seulement sous les *package*. Cliquer sur le *plus* dans le *project browser* pour les découvrir.



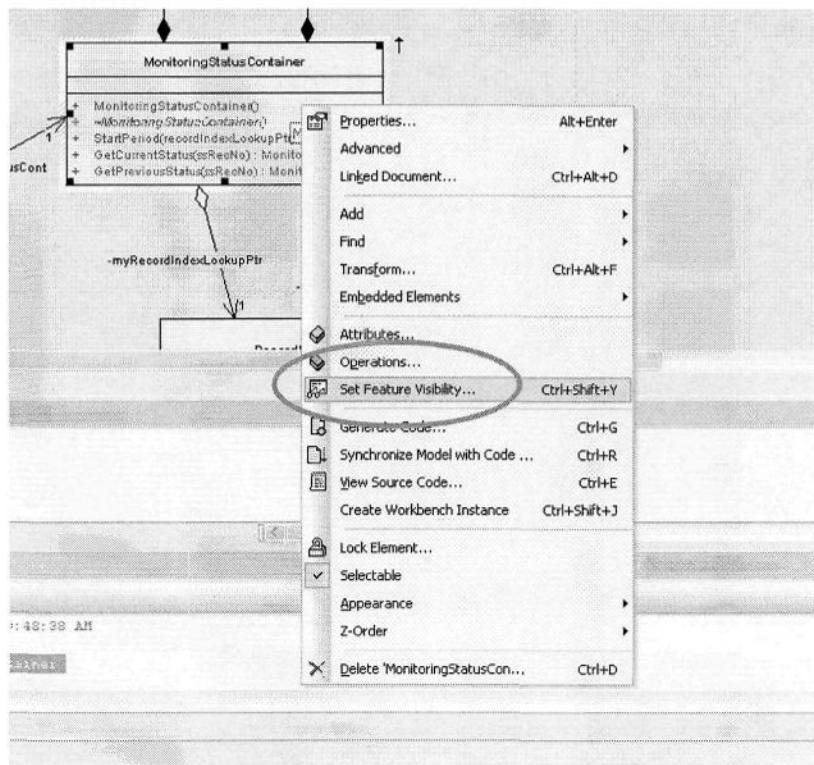
3.8. Options de visibilité

Il y a deux façons de contrôler les options de visibilité :

- Options par défaut pour le diagramme (menu contextuel du diagramme)



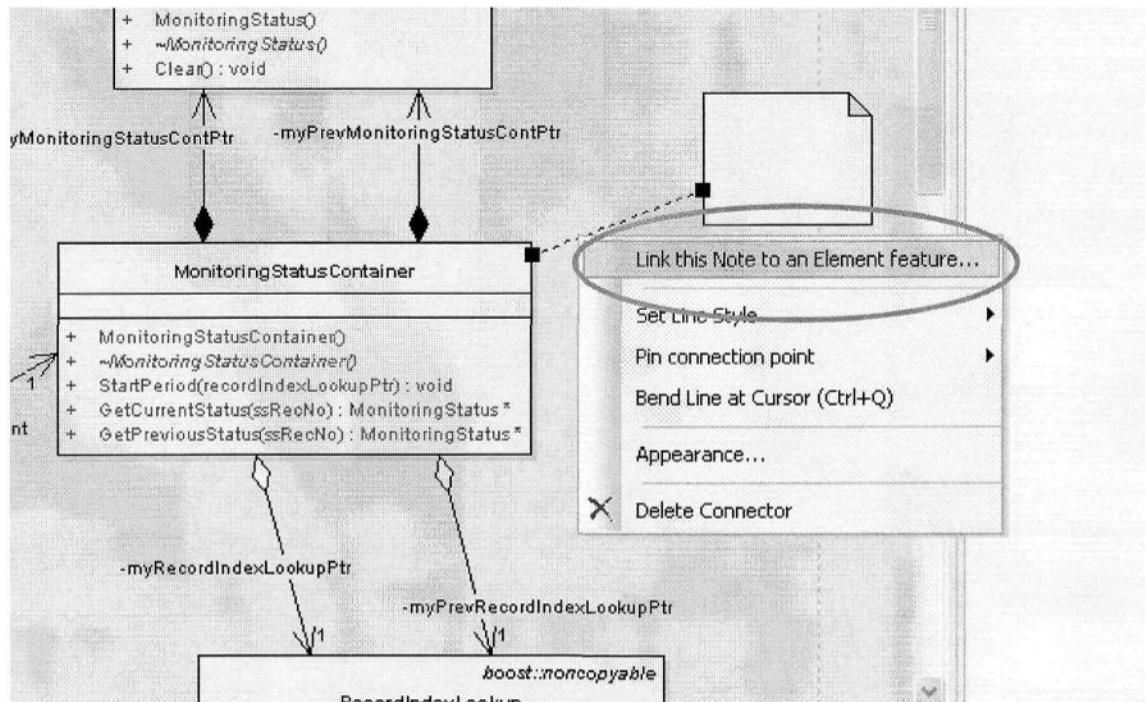
- Options particulières par élément (menu contextuel d'un élément)



3.9. Extériorisation d'une Note d'un Élément

Pour extérioriser une note d'un attribut, opération ou autres à partir d'un élément UML, il suffit de faire c'est étapes :

1. Ajouter un élément de type « note » au diagramme.
2. Faire une association entre la note et l'élément UML qui contient le texte à extérioriser.
3. Bouton de droite **sur le lien** et choisir l'option « Link this Note to an Element feature ».



3.10. Génération de commentaires

Dans EA, les commentaires du code source correspondent aux notes à l'intérieur des différents éléments UML.

Les commentaires peuvent être générés en plusieurs formats dont Javadoc. Les commentaires de méthodes peuvent être générés soit dans le fichier d'entête, soit dans le fichier d'implémentation, ou les deux.

Dans le cas où les commentaires de méthode sont générés dans les deux fichiers (.h et .cpp), les changements faits aux commentaires dans le .cpp seront ignorés lors de la synchronisation source → modèle. Lors de la synchronisation modèle → source, ces changements seront écrasés. Par conséquent, il faut toujours apporter les changements aux commentaires dans le fichier d'entête.

Pour faire disparaître un commentaire, il faut l'enlever du code et aussi enlever la note correspondante du modèle. Sinon, lors d'une synchronisation, les commentaires vont resurgir.

4. Options recommandés

Voici les options recommandées pour la génération de code.

