# Ingénierie dirigée par les modèles

## Cours #03

## Outil de modélisation StarUML

Professeur: Abdelilah KAHLAOUI

25 octobre 2012

# Agenda

- C’est quoi StarUML
- Architecture StarUML
- Concepts de base
- Approche
- Extension
- Modules
- Frameworks

# C’est quoi StarUML

- StarUML™ est une plate-forme de modélisation logiciel qui supporte UML et MDA
- Il est basé sur UML 1.4 et accepte la notation UML 2.0
- Il supporte l’ approche MDA en implémentant le concept de profil UML
- Il offre une grande flexibilité en termes de personnalisation de l'environnement utilisateur et d’extensibilité de ses fonctionnalités
- StarUML™ permet de maximiser la productivité et la qualité des projets informatiques

# Architecture StarUML

# Concepts de base

- Les concepts de base de StarUML incluent :
  - modèles
  - Vues
  - diagrammes
  - projets
  - unités
  - approaches
  - Frameworks

# Modèle vue et diagramme

- Modèle : un élément qui contient les informations d'un modèle de logiciel
- Vue (*View*) : une expression visuelle de l'information contenue dans un modèle
- Diagramme : une collection d'éléments de vue qui représentent l'intention du concepteur

# Projets

- Project: unité de gestion de base de StarUML
- Un projet peut gérer un ou plusieurs modèles de logiciels
- C'est le package de plus haut niveau d'un modèle logiciel
- Un projet contient et gère les sous-éléments suivants:
  - Modèle
  - Sous-système
  - Package

# Projets

- Les projets sont enregistrés au format XML avec l'extension ".uml"
- Tous les modèles, vues et diagrammes créés dans StarUML™ sont enregistrées dans un fichier projet
- Un projet peut également être divisé et sauvegardé dans plusieurs unités
- Un fichier de projet contient les informations suivantes:
  - Profils UML utilisés dans le projet
  - Fichiers d'unités référencé par le projet
  - Information sur tous les modèles contenus dans le projet
  - Information sur tous les diagrammes et vues contenues dans le projet

# Unités

- Un projet peut être géré comme plusieurs unités
- Seulement les éléments package, sous-systèmes et modèle peuvent constituer une unité
- Une unité peut avoir une structure hiérarchique; il peut contenir des sous-unités
- Les unités sont enregistrés en tant que ".unt" et elles  sont référencés par des fichiers projet (.uml) ou par d'autres fichiers unités (.unt)

# Fragment de modèle

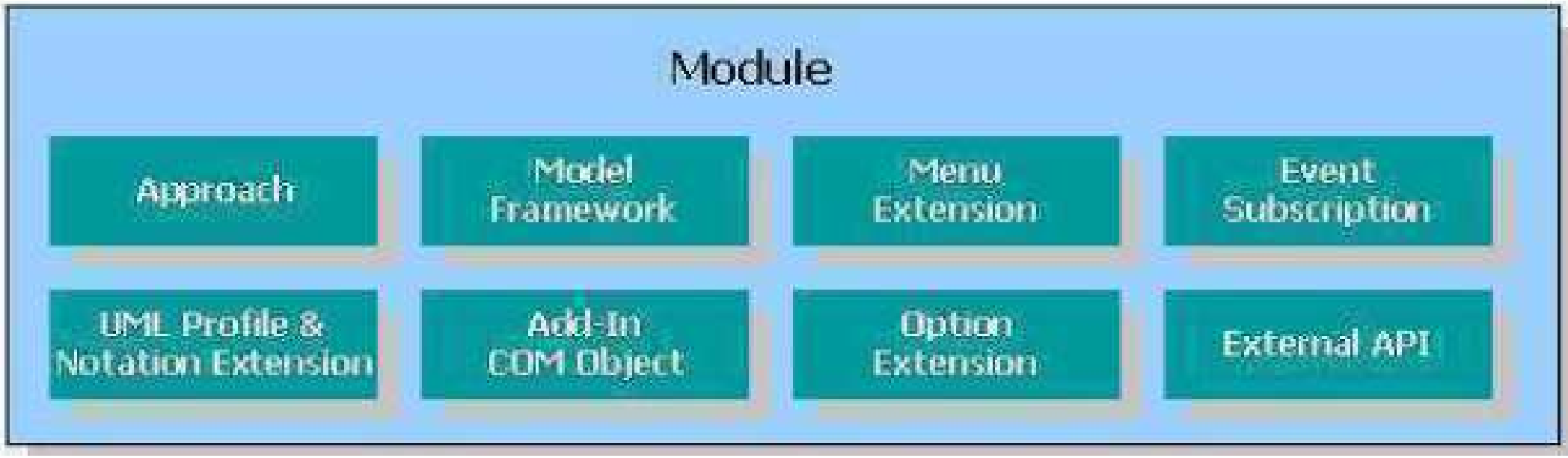
- Un fragment de modèle est une partie d'un projet enregistré dans un fichier séparé
- Les fragments de modèles sont enregistrés avec l'extension ".mfg"
- Seul les modèles, sous-systèmes ou packages peuvent constituer des fragments de modèles
- Les fragments de modèle peuvent être facilement inclus dans un projet à tout moment
- À la différence des unités, les fragments de modèle se fusionnent complètement avec le reste du projet une fois inclus dans le projet

# Document

- Un **Document** est une abstraction d'une partie sauvegardées dans un fichier
- L'objet **Document** offre les attributs et les méthodes nécessaires pour manipuler les fichiers « .uml » et « .unt »

# Module

- Un module est un package qui permet d'étendre fonctions et caractéristiques StarUML™
- Généralement, un module est constitué de divers mécanismes d'extension de StarUML™



# Application des Modules

- Les modules peuvent être développés à différentes fins
- Les modules peuvent être utilisés pour:
  - Prise en charge de processus spécifiques: UML Components, RUP, Catalysis, XP, ...
  - Prise en charge de langage de programmation spécifiques: C/C++, Python, C#, Visual Basic, Java, Perl, Object Pascal, ...
  - Intégration avec des outils spécifiques: Visual SourceSafe, CVS, MS Word, Eclipse, Visual Studio.NET, ...
  - Extension des fonctionnalités: Gestionnaire de Traçabilité, support de design patterns, vérification de règles, ...
  - Construction d'environnement specifique

# Éléments d'un Module

- Approach: une approche est appliquée au début du projet afin de déterminer la structure initiale du modèle
- Model Framework: un framework de modèle peut produire une bibliothèque de classes ou un Framework d'pplication
- UML Profile: les profils UML sont définis pour étendre l'expression UML ou pour utiliser des propriétés supplémentaires
- Menu Extension:utilisé pour étendre le menu principal ou le menu contextuel afin de permettre à l'utilisateur de sélectionner et d'exécuter les fonctions

# Éléments d’un Module

- Option Extension:permet d'utiliser des boîtes de dialogue d'options dans StarUML ™
- Add-In COM Object: En général, les objets COM sont utilisés pour des GUI ou des fonctionnalités complexes, et les
- Script: scripts sont utilisés pour des fonctions plus simples (JScript, VBScript, Python, ...)
- Help:Aide pour les Add-In peut être créée au format HTML

# Approche

- Une approche définit le modèle du projet et l'organisation de base des diagrammes
- Une approche consiste en:
  - Project Structure: spécifie la structure de base du projet
  - Profils importés : inclut automatiquement les profils UML par défaut dans le projet
  - Frameworks importés : charge et inclut automatiquement les frameworks par défaut dans le projet
  - fragments de modèle importés : charge et inclut automatiquement les fragments de modèle par défaut dans le projet

# Frameworks

- Frameworks en StarUML™ se réfèrent à des modèles logiciels qui expriment des bibliothèques de classes ou de Frameworks d'application
- Un Framework modèle permet d'utiliser des frameworks d'applications ou des bibliothèques de classes dans StarUML™
- Un framework est constitué d'une structure de fichier (.frw) et un ou plusieurs fichiers d'unité (.unt)
  - Fichier Framework : contient des informations sur le framework
  - Fichier unité : contient des informations sur les unités incluses et les profils UML utilisés

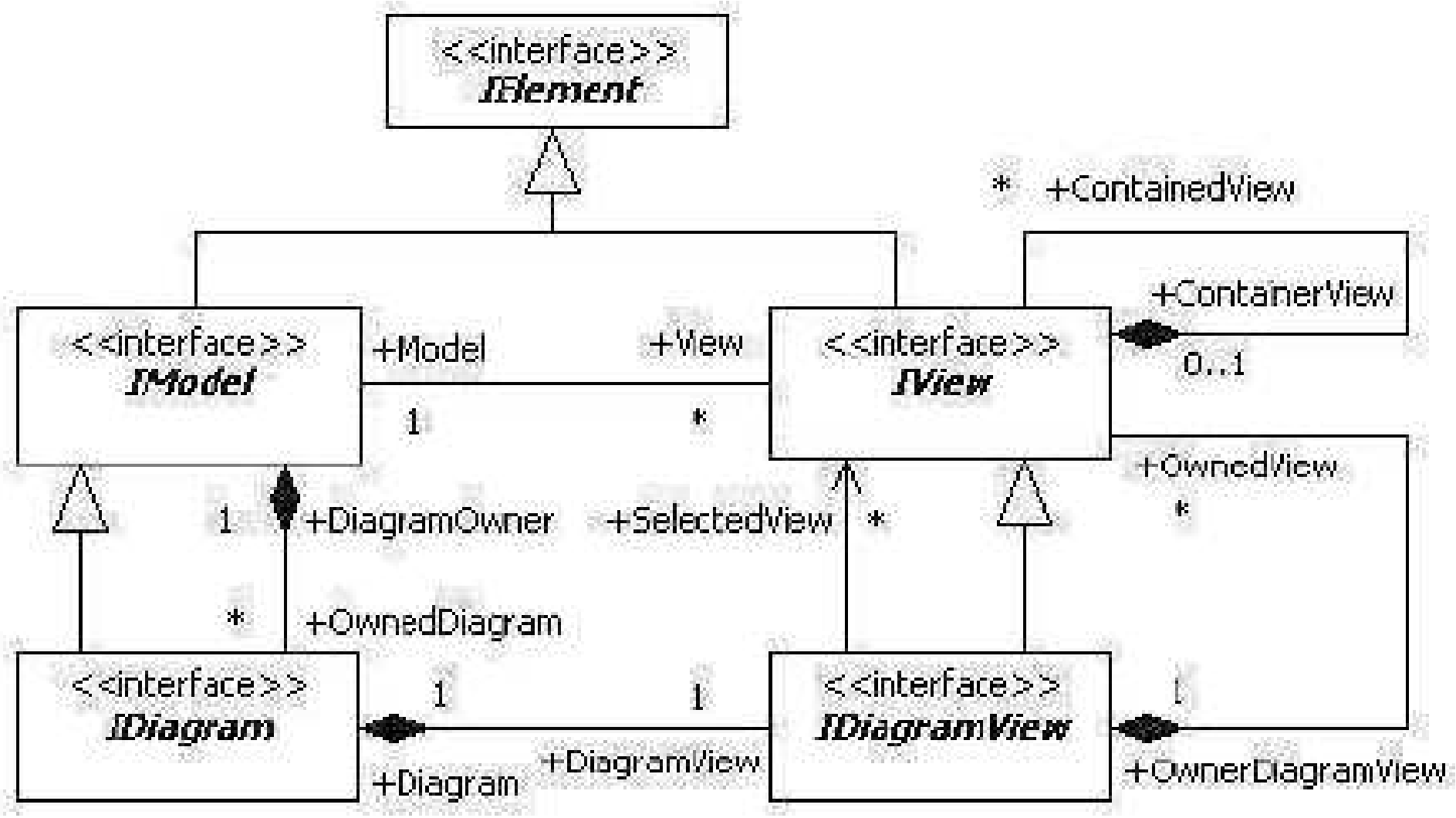
# UML Profile

- UML est si générale qu'elle peut être utilisée pour exprimer n'importe quel idée ou concepts
- StarUML ™ permet une extension facile d'UML en adoptant le concept de profils UML
- profiles UML sont utilisés pour :
  - Langages de programmation spécifiques: C/C++, Java, C#, …
  - Méthodes de développement spécifiques: RUP, XP, …
  - domaines Specifiques : EAI, CRM, SCM, ERP, …

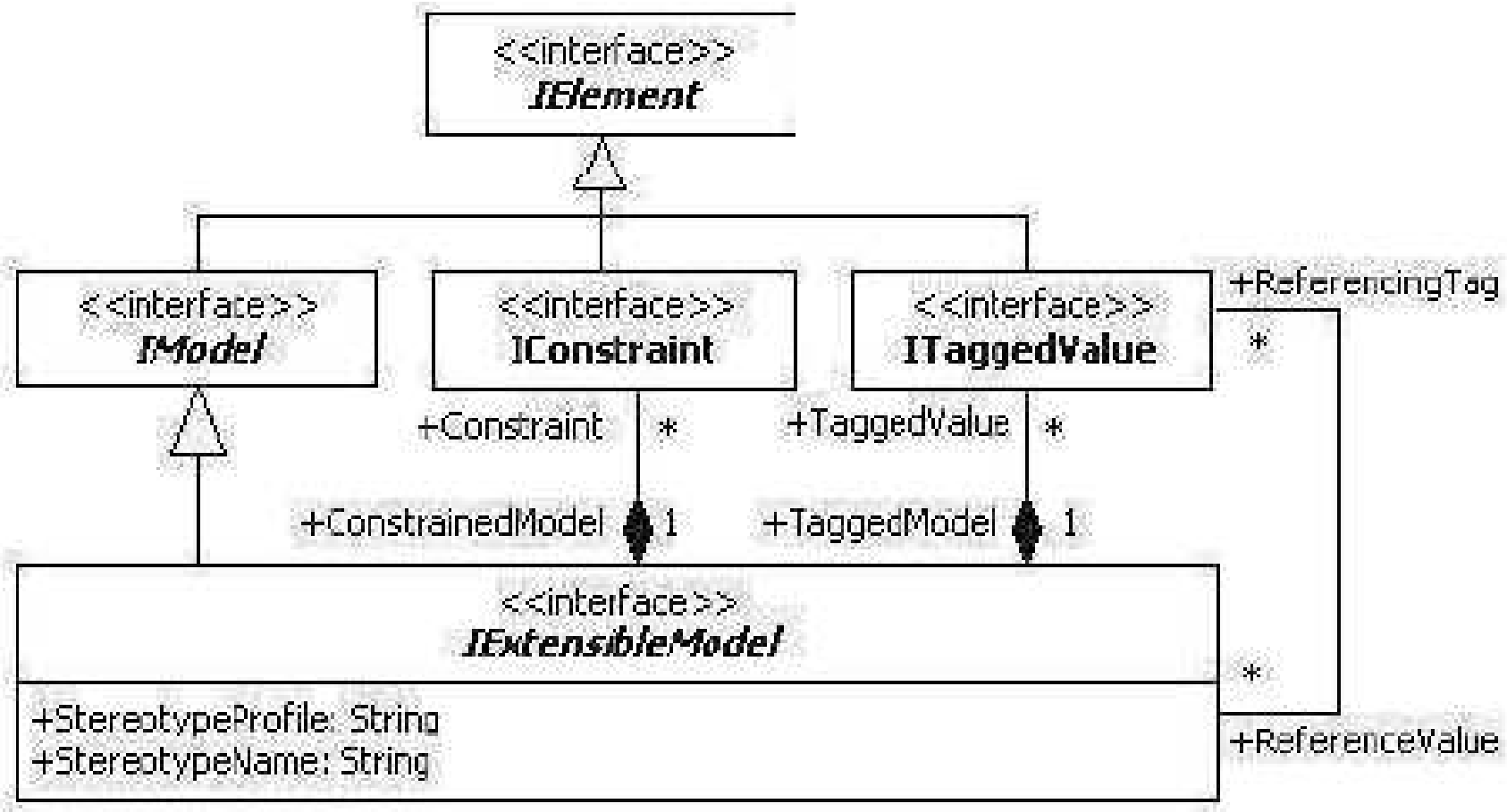
# Éléments de modélisation

- Les éléments de modélisation sont organisés dans les packages suivants
  - Core Elements: définit l'interface supérieure des éléments *model*, *view* et *diagram*
  - ExtCore Elements: définit l'interface commune des éléments extensibles du modèle
  - ViewCore Elements: définit les types de base pour les éléments de vue
  - UML Model Elements: Définit les éléments du modèle UML
  - UML View Elements: définit les éléments de vue (view elements) d'UML

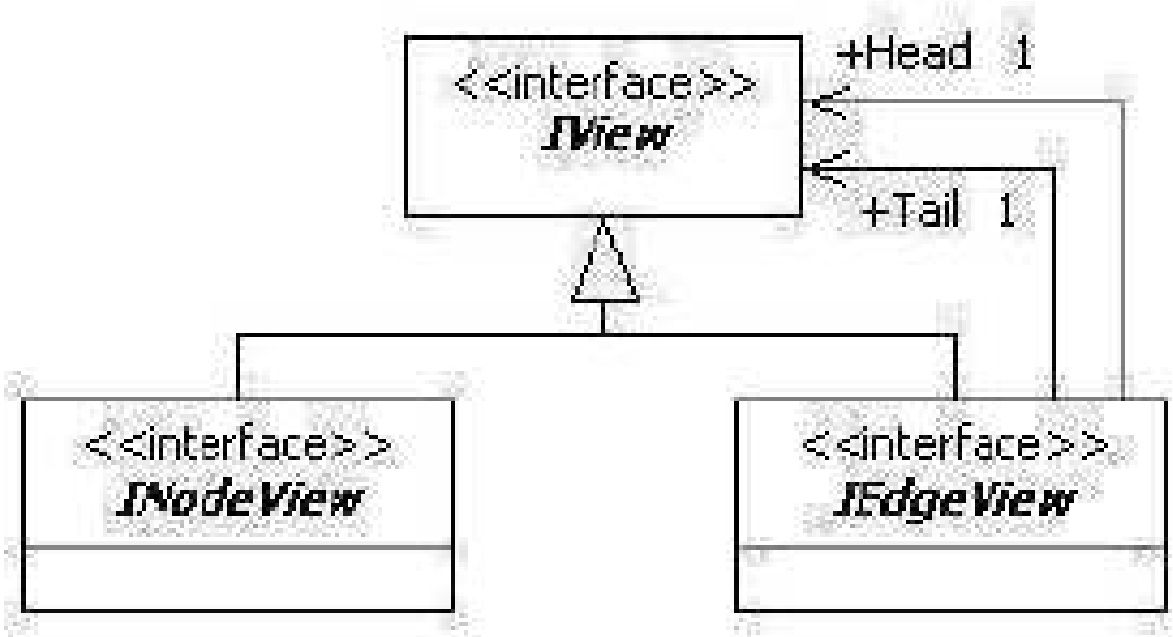
# Core Elements



# ExtCore Elements



# ViewCore Elements



# L'API StarUML

# StarUML API

- StarUML offre un API afin de permettre aux utilisateur d’accéder à ses fonctionnalités internes

# StarUMLApplication

- StarUMLApplication est une abstraction de l'application StarUML
- On obtient une référence à cette objet en utilisation l'instruction :

```
New ActiveXObject("StarUML.StarUMLApplication");
```

# ProjectManager

- Le gestionnaire de projet "ProjectManager" est un objet qui vous permet de gérer de vos projets StarUML
  - Création, ouverture, fermeture et sauvegarde de projets
  - Création, ouverture et sauvegarde d'unités
  - Import et export de fragments
  - Import de frameworks
  - …
- L'instruction suivante permet d'avoir une référence du gestionnaire de projet

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var prjmgr = app.ProjectManager;
```

# IUMLFactory

- L'objet UMLFactory offre des méthodes qui vous permettent de créer les éléments de votre modèle
  - Diagrammes, classes, associations, attributs, cas d'utilisation, …
- Exemple

```
var app = new
  ActiveXObject("StarUML.StarUMLApplication");
var factory = app.UMLFactory;
var prj = app.GetProject();
var newPackage = factory.CreatePackage(prj);
```

# UMLFactory-Exemple

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var facto = app.UMLFactory;
var pjt = app.GetProject();
//Création de modèle et package
var mdlElem = facto.CreateModel(pjt);
var pkgElem = facto.CreatePackage(mdlElem);
//Création de deux classes
var clsElem1 = facto.CreateClass(pkgElem);
var clsElem2 = facto.CreateClass(pkgElem);
//Création d'attributs et d'opération
var attrElem = facto.CreateAttribute(clsElem1);
var opElem = facto.CreateOperation(clsElem1);

var paramElem1 = facto.CreateParameter(opElem);
var paramElem2 = facto.CreateParameter(opElem);

paramElem1.TypeExpression = "String";
paramElem2.Type_ = clsElem2;
...
```

# MetaModel

- L'objet MetaModel permet d'accéder aux éléments du niveau Meta-Model

- Une application StarUML ne peut contenir plus qu'un objet MetaModel

- Exemple

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var meta = app.MetaModel;
var metaClass = meta.FindMetaClass("UMLClass");
// Trouver toutes les classe du modèle
for (var i = 0; i < metaClass.GetInstanceCount(); i++){
   var AClassElem = metaClass.GetInstanceAt(i);
}
```

# MetaClass

- L'objet MetaClass permet d'accéder à la définition des éléments de modélisation
- Exemple

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var meta = app.MetaModel;
var metaClassOfPackage = meta.FindMetaClass("UMLPackage");
var metaClassOfClass = meta.FindMetaClass("UMLClass");
var metaClassOfAttr = meta.FindMetaClass("UMLAttribute");
...
```

# MetaAttribute

- L'objet MetaAttribute permet d'accéder à la spécifications des attributs des éléments de modélisation

- Exemple

```
var app = new ActiveXObject("StarUML.StarUMLApplication");
var metaClass = app.MetaModel.FindMetaClass("UMLClass");
for (var i=0; i < metaClass.GetMetaAttributeCount(); i++){
    var metaAttr = metaClass.GetMetaAttributeAt(i);
    ...
}
```

# Personnalisation StarUML

# Création d'approches

- La création d'une approche consiste en:
  - Création d'un fichier «.apr» qui contient la définition de l'approche
  - Placer le fichier créé dans le répertoire «module» de StarUML

- Structure du fichier «.apr»

```
<?xml version="1.0" encoding="..." ?>
<APPROACH version="...">
    <HEADER>
        ...
    </HEADER>
    <BODY>
        ...
    </BODY>
</APPROACH>
```

# Création de modèle Framework

- Un modèle Framework permet l'utilisation de Frameworks et de librairies de classes dans StarUML
- La liste des modèles Framework installés est accessible via le menu [File]-[Import]-[Framework]
- La création d'un modèle Framework consiste en:
  - Création des fichiers unités contenant les information sur le modèle Framework
  - Création d'un fichier «.frw» qui définit le modèle Framework
  - Placer tous les fichier dans le répertoire module de StarUML

# Structure du fichier Framework

```
<?xml version="1.0" encoding="UTF-8" ?>
<FRAMEWORK version="1.0">
   <HEADER>
        <NAME>J2SE1.3</NAME>
        <DISPLAYNAME>Java2 Standard 1.3</DISPLAYNAME>
        <DESCRIPTION>J2SE 1.3 Framework. </DESCRIPTION>
   </HEADER>
   <BODY>
        <IMPORTPROFILES>
              <PROFILE>...</PROFILE>
              ...
        </IMPORTPROFILES>
        <FRAMEWORKMODELS>
              <UNIT>J2SE13 (java).pux</UNIT>
              <UNIT>J2SE13 (javax).pux</UNIT>
              <UNIT>J2SE13 (org).pux</UNIT>
        </FRAMEWORKMODELS>
   </BODY>
</FRAMEWORK>
```

# Création de profils UML

- Un profil est un package de mécanismes d'extensions
- En plus des mécanismes d'extension standards, StarUML offre les mécanismes d'extension suivants:
  - Diagram Type: définit un nouveau type de diagrammes
  - Element Prototype: définit un prototype d'élément avec des propriétés prédéfinis
  - Model Prototype: définit un prototype de modèle
  - Palette Extension: définit une nouvelle palette

# Création de profils UML

- La création d'une profil consiste en:
  - Création d'un fichier «.prf» qui contient la définition du profil
  - Placer le fichier créé dans le répertoire «module» de StarUML
- Structure d'un fichier profil

```
<?xml version=”1.0” encoding=”...” ?>
<PROFILE version="...">
    <HEADER>
        ...
    </HEADER>
    <BODY>
        ...
    </BODY>
</PROFILE>
```

# Extension du menu

- L'extension de menu consiste en:
  - Création d'un fichier «.mnu» qui contient la définition du menu
  - Placer le fichier créé dans le répertoire «module» de StarUML

# Structure du fichier d'extension menu

```
<?xml version="1.0" encoding="..."?>
<ADDINMENU addInID="...">
   <HEADER>
        <NAME>...</NAME>
        <VERSION>...</VERSION>
        <DESCRIPTION>...</DESCRIPTION>
        <COMPANY>...</COMPANY>
        <COPYRIGHT>...</COPYRIGHT>
   </HEADER>
   <BODY>
        <MAINMENU>
              <MAINITEM>...</MAINITEM>
              <MAINITEM>...</MAINITEM>
        </MAINMENU>
        <POPUPMENU>
              <POPUPITEM>...</POPUPITEM>
              <POPUPITEM>...</POPUPITEM>
        </POPUPMENU>
   </BODY>
</ADDINMENU>
```

# Génération de code

# Générateur StarUML

- Un générateur StarUML est un module capable de générer un ou plusieurs artefacts
  - Document Word
  - Document Excel
  - Présentation PowerPoint
  - Fichiers texte
- Les générateurs StarUML utilisent principalement les templates

# Générateur – structure du répertoire

- La structure du dossier d'un générateur ressemble à:

```
⇒ Modules\
    ⇒ staruml-generator\
        ⇒ templates\
            ⇒ template1\
                template1.tdf
                template1.doc
            ⇒ template2\
                ...
        ⇒ Batches\
            batch1.btf
            ...
```

# Éléments d'un template

- Un template se compose généralement de deux parties
  - Style : définit la forme générale d code à générer
  - Commandes : ensemble d'instructions ayant une signification spéciale pour le moteur de template (*Template engine*)

# Commandes

- Dans une template on trouve des commandes
  - REPEAT … ENDREPEAT
  - IF … ENDIF
  - DISPLAY
  - SCRIPT
- Templates texte

# REPEAT ... ENDREPEAT

**Syntaxe**

```
<@REPEAT PathName;TypeFilter;CollectionName;Condition @>
...
<@ENDREPEAT@>
```

| Argument | Description |
|---|---|
| `PathName` | Répèter pour les éléments sous `pathName` |
| `TypeFilter` | Répèter pour les éléments ayant le type `TypeFilter` |
| `CollectionName` | parcourir les éléments de la collection nommée par `CollectionName` des éléments qui sont sélectionnés par `pathName` et `TypeFilter` |
| `Condition` | Répèter pour les éléments satisfaisant `Condition` |

# REPEAT ... ENDREPEAT

- Exemples

```
<@REPEAT {R}::Design Model;UMLClass;;@>
class <@DISPLAY ;current().Name@>
<@ENDREPEAT@>
```

# IF ... ENDIF

**Syntaxe**

```
<@IF Condition @>
...
<@ENDIF@>
```

| Argument | Description |
|---|---|
| `Condition` | Condition exprimée en JScript |

# DISPLAY

**Syntaxe**

```
<@DISPLAY PathName;Expression @>
```

| Argument | Description |
|---|---|
| `PathName` | Path de l'élément à sélectionner |
| `Expression` | Expression de la valeur à écrire |

# SCRIPT

- **Syntaxe**

```
<@SCRIPT
   JScript Statements
@>
```

- **Exemple**

```
<@SCRIPT
   function fct(a, b) {
      ...
   }
@>
```

# Built in functions (1)

| Signature | Description |
|---|---|
| `StarUMLProject(): IUMLProject` | Retourne un objet de type projet |
| `findByFullpath(Path): IElement` | Retourne l'élément qui se trouve dans l'argument `Path` |
| `findByLocalpath (RootElem,Path): IElement` | Retourne l'élément qui se trouve dans le path relatif `Path` au path `RootElem` |
| `itemCount(RootElem, CollectionName): int` | Retourne la nombre d'éléments dans la collection `CollectionName` |
| `item(RootElem, CollectionName, Index):IElement` | Retourne l'element qui existe à l'index `Index` de la collection `CollectionName` |
| `attr(Elem,AttrName): Value` | Retourne la valeur de l'attribut `AttrName` de l'élément `Elem` |

# Built in functions (2)

| Signature | Description |
|---|---|
| `current(): IElement` | Retourne le dernier élément sélectionné |
| `createFile(path): TextStream` | Crée un fichier et retourne un objet fichier |
| `deleteFile(path)` | Supprime un fichier |
| `fileExists(path): Boolean` | Vérifie l'existence d'un fichier |
| `fileBegin(path)` | Crée un fichier. Toutes les ouputs seront écrites dans ce fichier tant que la fonction `fileEnd(path)` n'est pas appelée |
| `fileEnd(path)` | Arrete l'ecriture sur un fichier créé par `fileBegin(path)` |

# Built in functions (3)

| Signature | Description |
|---|---|
| `createFolder(path): Folder` | Crée un dossier et retourne un objet `Folder` |
| `deleteFolder(path)` | Supprime un dossier |
| `folderExists(path): Boolean` | Vérifie l'existence d'un dossier |
| `getTarget(): String` | Retourne le chemin de génération |

# Exemple

```
<@REPEAT {R}::Design Model;UMLClass;;@>

<@SCRIPT
  fileBegin(getTarget()+"\\"+current().Name+".java");
@>
class <@DISPLAY ;current().Name@> {
  // <@DISPLAY ;current().Documentation@>
}
<@SCRIPT
  fileEnd();
@>

<@ENDREPEAT@>
```