

1. Utilisation de Papyrus_UML (éditeur UML2 eclipse)

(MDT) **Papyrus (UML)** est un **éditeur UML open source** qui fonctionne sous forme de **plugin** pour l'environnement de développement **ECLIPSE** (très utilisé pour le développement d'applications en JAVA). Le tout étant basé sur JAVA, **Papyrus_UML** est "**multi-plateforme**" et fonctionne entre autres sur **Windows**, **Unix/Linux**, **Mac**.

Historiquement, les premières versions de Papyrus ont été conçues par le **CEA**.

Un projet similaire "**Topcased UML**" avait été pris en charge par "**Airbus + écoles et universités proches de Toulouse**". En 2011/2012/2013/2014, ces deux produits sont actuellement en train de fusionner (en même temps que s'effectue une reprise officielle de la partie "éditeur UML papyrus" par la communauté "eclipse").

En tant que "(sous) projet eclipse officiel", l'éditeur "**Papyrus UML**" peut s'intégrer (par simple ajout) dans le tout dernier **eclipse** (ex : 4.3 / Kepler en 2013/2014).

En tant que "sous partie" des versions récentes de "**Topcased UML RCP**", "**Papyrus UML**" peut être rapidement utilisé dans **un environnement tout intégré** (avec générateur de code java et de documentation).

L'un des principaux intérêts des outils "Topcased UML" et "Papyrus UML" tient dans le fait qu'en tant que "plugins bien intégrés à eclipse", on peut les utiliser conjointement avec d'autres plugins importants (ex: accéléo_M2T, gendoc2, ...).

Ainsi à partir d'un même outil ECLIPSE, on peut:

- **créer/paramétrer des modèles UML** (diagrammes avec stéréotypes)
- **(re-)générer automatiquement de la documentation au format ".doc" ou ".odt"** (avec le plugin intégré gendoc2) pour produire des spécifications
- **(re-)générer automatiquement une bonne partie du code de l'application** (avec le plugin "accéléo_M2T" / MDA).

Ceci permet de travailler efficacement avec de bon atouts pour obtenir des éléments produits (spécifications, code, tests) bien **cohérents** entre eux.

NB: Pour bien utiliser cette plateforme de développement (et ses différents plugins), il faut savoir effectuer les **bons paramétrages** à chaque niveau:

- bien paramétrer les modèles UML (avec des stéréotypes adéquats)
- bien paramétrer la génération de documentation (avec des "templates" de "docs")
- bien paramétrer la génération de code (avec des "templates" de code)

Tout ceci correspond au final à **un investissement en "temps de mise au point"** qui peut se rentabiliser sur des projets importants.

1.1. Intégration/installation de Papyrus UML

Préalable : une machine virtuelle JAVA (idéalement JDK 1.6 ou 1,7) doit être installée sur le poste de développement.

Il y a au final deux grands modes d'intégration de "MDT Papyrus UML":

- intégration pré-établie (prête à être téléchargée)
---> **Topcased RCP** (*Rich Client Platform*)
- intégration spécifique (à construire soit même en partant d'ECLIPSE et en y ajoutant un à un tous les plugins jugés utiles : Papyrus_UML , accéléo_M2T , gendoc2 , ...)

NB:

En 2013, pour la version "Eclipse 4.3 / Kepler" , le mode opératoire pour installer le plugin "MDT Papyrus" est le suivant :

Menu Help / Software Update , ...

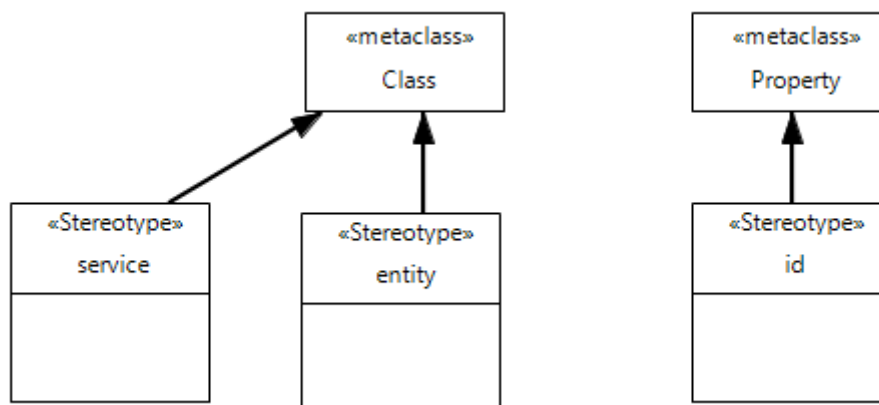
releases kepler / **Modeling** / (modeling components) / **papyrus**

1.2. Création d'un profil UML (avec stéréotypes)

Pour placer des stéréotypes (ex: <<entity>> , <<id>>) dans un modèle UML , il faut d'abord les créer dans un fichier de type "**UML profile**".

Mode opératoire:

- **New / Other ... / Papyrus / Papyrus model**
- **Select "Profile" et "UML profile diagram" (+include template "primitives types").**
- Créer de nouveaux **stéréotypes** (depuis la palette et en renseignant leurs **noms**).
- Placer et paramétrer des "**metaclass**" (ex: "Class" , "Property" , ...) .
- Relier les "**stéréotypes**" aux "**metaclass**" via des flèches d'**extension** pour indiquer "**sur quoi les stéréotypes seront applicables**".
- Bien sauvegarder le fichier généré (et choisir une **version**).



NB: un stéréotype peut éventuellement comporter des propriétés.

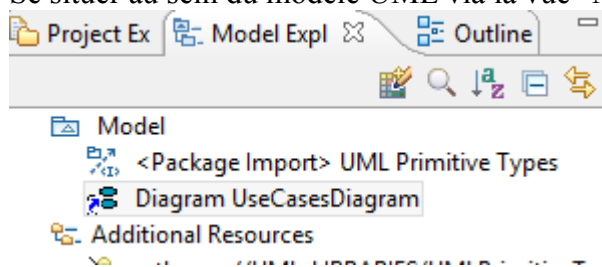
Quelques idées de stéréotypes:

<<entity>>	Entité persistante
<<id>>	Identifiant (proche de "clef primaire")
<<service>>	Service métier
<<requestScope>> , <<sessionScope>> , <<applicationScope>>	Scope / portée des objets de la partie "IHM_WEB"
<<stateful>> , <<stateless>>	Avec ou sans état (traitements ré-entrants et partagés ?)
<<facade>> , <<dao>> , <<dto>> , ...	Design pattern / pour la conception
<<in>> , <<out>> , <<inout>> , <<select>>	Pour paramétrer les fonctionnalités souhaitées coté IHM (entrée/saisie , sortie/affichage , sélection , ...)
<<transactional>> , <<CRUD>>	Fonctionnalités diverses attendues (transactionnel , ...)
<<module_web>> , <<module_services>> , <<database>>	Types de composants
...	
...	

1.3. Création et initialisation d'un modèle UML (pour application)

Mode opératoire:

- Créer un nouveau modèle via le menu "New / Other.../ Papyrus/ Papyrus Model"
- Sélectionner "UML" et (include template "primitives types")
- Sélectionner éventuellement le nom et le type de diagramme initial (ex : UsesCases ou "Class") (NB : d'autres diagrammes pourront être ajoutés ultérieurement au modèle).
- Se situer au sein du modèle UML via la vue "Model explorer" :



- Sélectionner la racine du modèle UML (Model) et ajuster la propriété "profiles" en "clicquant" sur "+" et en sélectionnant le fichier ".....profile.uml" .
- ...
- Bien (re-)sauvegarder le fichier du modèle UML.

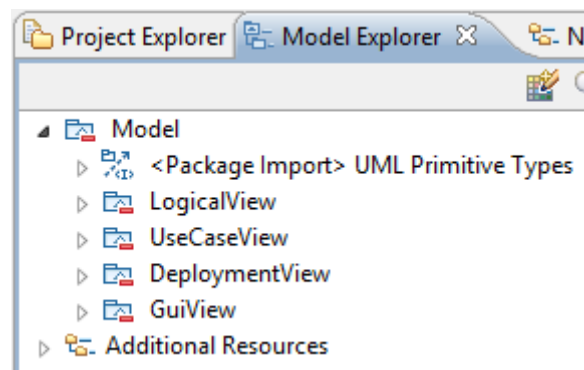
Remarque :

L'assistant de création de modèle "uml papyrus" ne crée pour l'instant qu'un point de départ très rudimentaire (rien ou un diagramme au choix à la racine du modèle).

Si l'on souhaite mieux organiser la structure interne d'un modèle papyrus UML , on peut éventuellement se placer à la racine "Model" et créer (via le menu contextuel "add Child / new Model") des sous modèles de type "UseCaseView" , "LogicalView" de façon à mieux ranger les

parties du modèle applicatif.

Exemple :

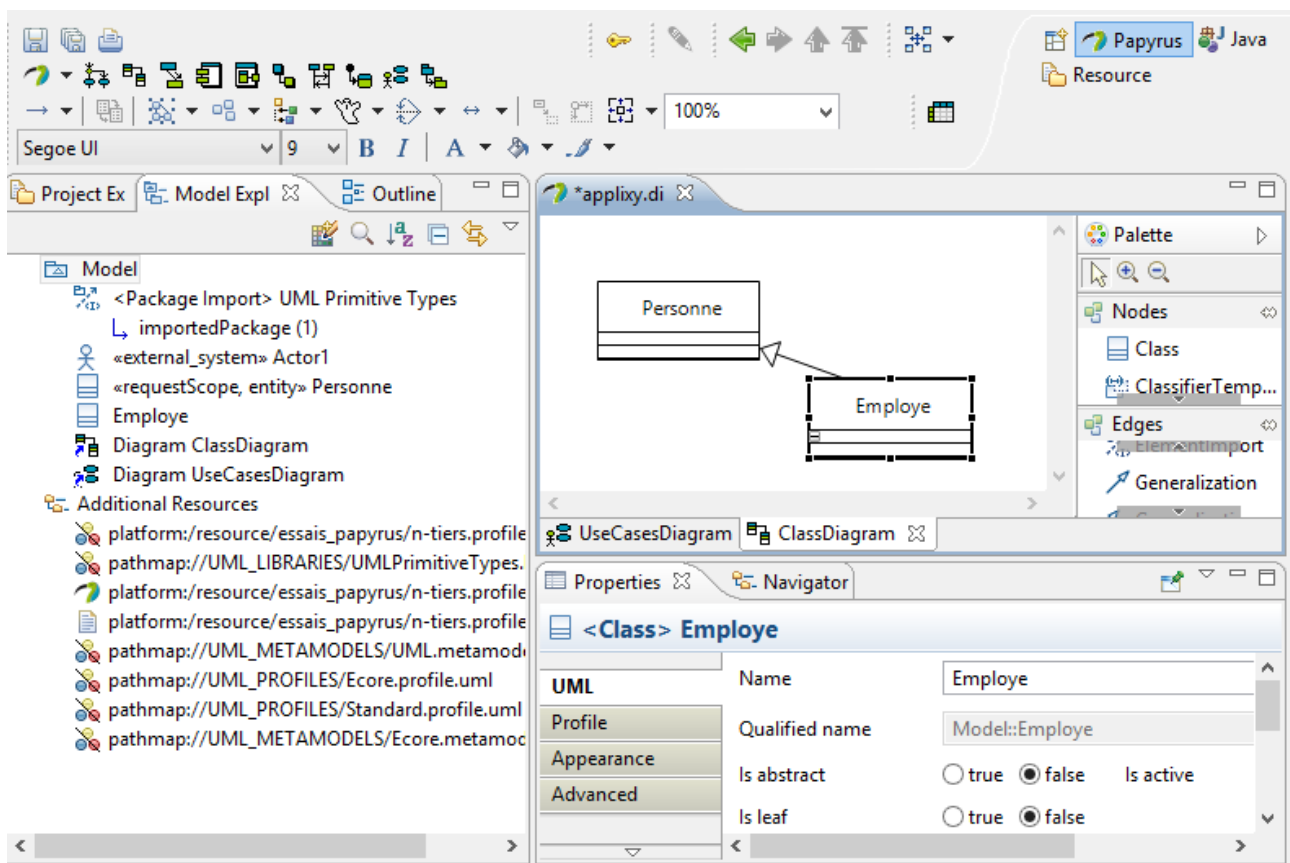


1.4. Utilisation générale de l'outil Papyrus UML

Chaque modèle UML est sauvegardé dans trois fichiers complémentaires:

- le fichier ".uml" comporte tous les éléments significatifs du modèle UML (packages , classes ,). C'est à partir de ce fichier que l'on peut extraire les informations essentielles du modèle UML pour générer du code (via un outil MDA tel qu'accéléo_m2t par exemple)
- les fichiers ".notation" et ".di" comportent les coordonnées (x,y,...) des éléments internes des diagrammes UML.

==> pour éditer graphiquement un modèle UML, il faut ouvrir (via un double-clic) le fichier ".di" (ou l'ensemble). Le fichier ".uml" de même nom sera alors automatiquement pris en charge et mis à jour.



De façon intuitive, la vue "Model Explorer" permet de naviguer dans l'arborescence interne du modèle UML et la vue "Properties" permet de fixer les propriétés de l'élément sélectionné. Une palette permet de choisir le type d'élément à ajouter au modèle.

- pas de menu contextuel "add attribute/property" ni "add operation/method" mais des éléments "**Property**" et "**Operation**" à récupérer dans la palette et à ajouter aux classes du diagramme.
- Souvent besoin de cacher (via **Filter/hide compartment** ou **Filter / ...**) certains éléments secondaires.

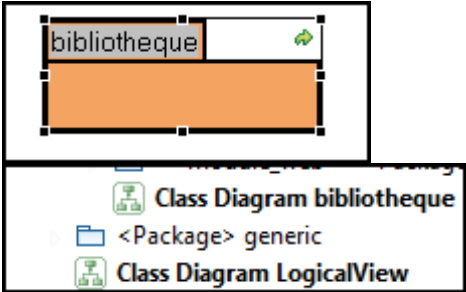
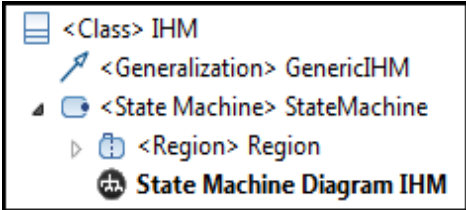
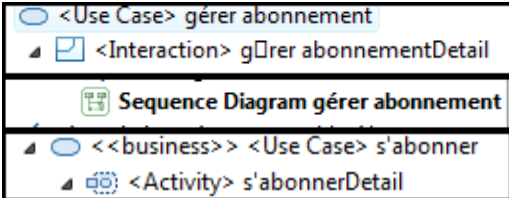
1.5. Organisation conseillée des packages et des diagrammes




L'un des principaux "**points forts**" des outils "**Topcased UML**" et "**Papyrus UML**" c'est de bien gérer la **cohérence** entre les **packages** et les **diagrammes** de façon à pouvoir naviguer de façon efficace dans l'arborescence d'un modèle UML.

Mode opératoire conseillé:

- Créer un élément UML (ex: package, classe, ...) dans un diagramme et le paramétrer.
- **Sélectionner un package existant (modélisé auparavant)** dans l'arborescence "**model explorer**", puis activer le menu "**add class diagram**" en donnant un nom explicite à ce diagramme (ex : packageXY_ClassDiagram), etc / etc
- Par la suite (une fois la création/association effectuée), un **double-clic** ultérieur sur le package au sein du diagramme permettra de **naviguer d'un niveau vers un sous niveau** (de détails).

Sous diagrammes classiques pour indiquer les détails:

Eléments du modèle UML	diagramme(s) classique(s) associé(s) pour les détails	Exemple(s)
Package	Diagramme de classes (ou ...)	
Classe	Diagramme d'états (state machine) ou diag. de structure composite	
Use Case	diagramme d'activités et/ou diagrammes de séquences	

		 Activity Diagram s'abonner
Composant	Diagramme de (sous)composants ou de structure composite , ...	 <Component> module_services_XY  Component Diagram module_services_XY







Remarques importantes:

- Pour bien organiser un modèle UML, il faut réfléchir le plus tôt possible à la décomposition en différents packages.
- Il est toujours possible de renommer ou déplacer à la souris un package (après coup / par la suite) via le "model_explorer".
- Cette "bonne organisation" des éléments du modèle UML est surtout utile pour que des scripts des templates (.docx/.odt) pour gendoc2 puissent efficacement retrouver les digrammes de façon à générer automatiquement une bonne documentation.

1.6. Edition d'un diagramme de classes (spécificités)

Après avoir sélectionner une association , on peut activer le menu contextuel "Filters/All-No-Managed connectors labels" ce qui fait apparaître la boîte de dialogue suivante :

Select the labels to display.

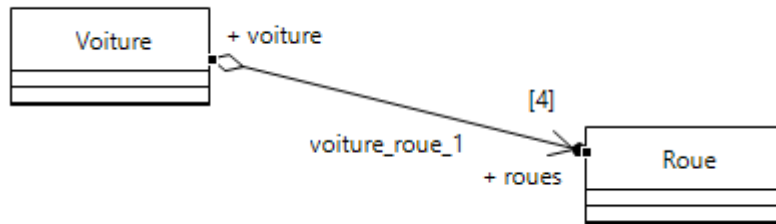
Label Role	Displayed Text
<input checked="" type="checkbox"/> / <Association> class1_class2_1	
<input type="checkbox"/>  Name	class1_class2_1
<input type="checkbox"/>  SourceMultiplicity	
<input type="checkbox"/>  SourceRole	+ class1
<input type="checkbox"/>  Stereotype	[No Text To Display]
<input type="checkbox"/>  TargetMultiplicity	
<input type="checkbox"/>  TargetRole	+ class2

Ceci est très pratique pour afficher ou cacher certains détails associés aux associations (rôles ,)

Dans la fenêtre des propriétés:

***Paramétrer les détails d'une d'association (navigabilité, agrégation, composition,)**

Member End		Member End	
Name	voiture	Name	roues
Owner	Association	Owner	Classifier
Navigable	<input type="radio"/> true <input checked="" type="radio"/> false	Navigable	<input checked="" type="radio"/> true <input type="radio"/> false
Aggregation	none	Aggregation	shared
Multiplicity	1	Multiplicity	4..4



* Paramétrer une opération (nom, type de retour, paramètres ,)

Name	rechercherExemplaireParNumero		
Is abstract	<input type="radio"/> true <input checked="" type="radio"/> false	Is leaf	<input type="radio"/> true <input checked="" type="radio"/> false
Is query	<input type="radio"/> true <input checked="" type="radio"/> false	Is static	<input type="radio"/> true <input checked="" type="radio"/> false
Concurrency	sequential	Visibility	public
Method	<div> <div>↑</div> <div>↓</div> <div>+</div> <div>×</div> <div>✎</div> </div> <div>Owned parameter</div> <div> <div>↔</div> <div><Parameter> num : Integer</div> <div> <div>↑</div> <div>↓</div> <div>+</div> <div>×</div> <div>✎</div> </div> </div>		

Create a new Parameter

Name

num

Is exception

☐ true ☒ false

Is stream

☐ true ☒ false

Direction

in

Visibility

public

Default value

<Undefined>

+

✎

×

Type

<Prim...teger

...

+

✎

×

Is ordered

Is unique

Effect

Multiplicity

UML

Profile

pour un paramètre d'entrée , laisser direction="in", **pour la valeur de retour , direction = return**

Name

return

Is exception

☐ true ☒ false

Is stream

☐ true ☒ false

Direction

return

Visibility

public

Default value

<Undefined>

+

Type

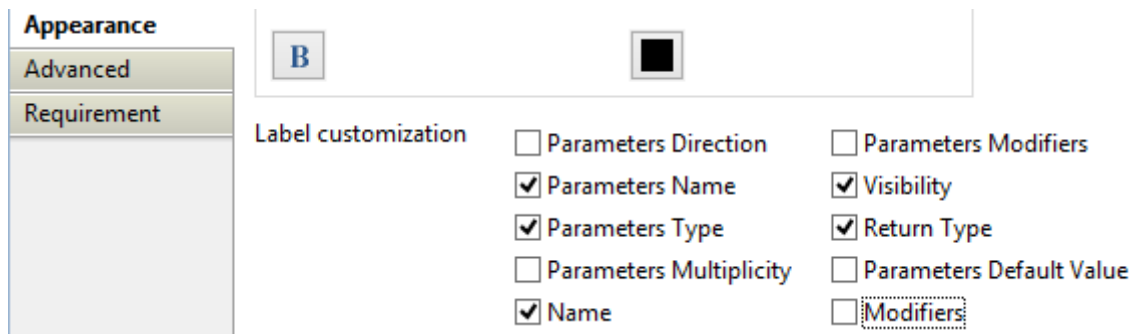
<<entity>> <Class> Exempleire

...

+

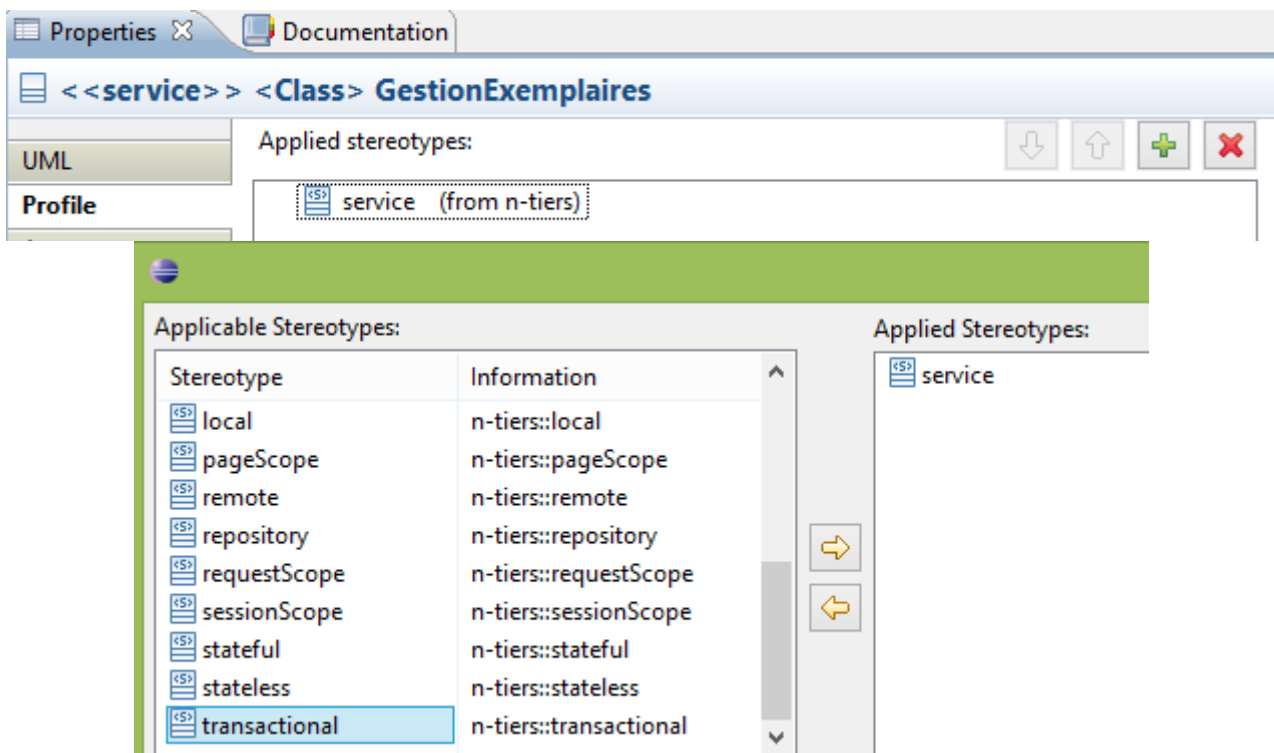
résultat → `rechercherExemplaireParNumero(num: Integer): Exemplaire`

et (dans le sous onglet "Appearance"), décocher "param direction" , "visibility" et "modifiers":



* Choix d'un (ou plusieurs) stéréotype(s) à appliquer:

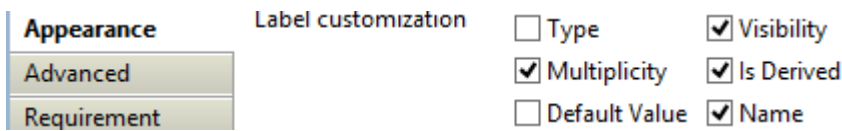
Sélectionner un des éléments du modèle (classe ou propriété ou package ou)
et sélectionner un stéréotype via le sous onglet "profile" :



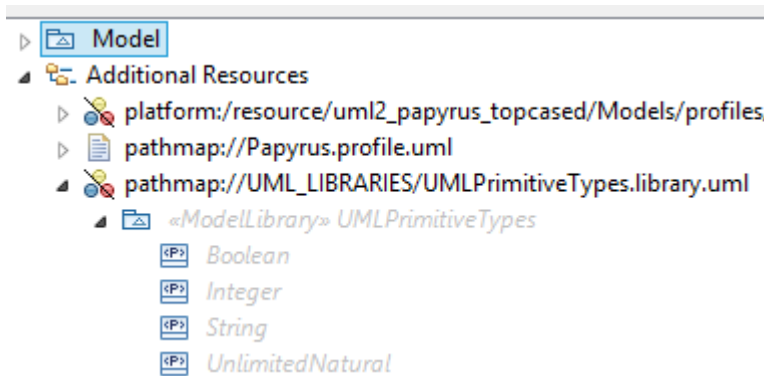
NB : Pour que certains stéréotypes applicables soient proposés, il faut qu'au préalable au moins un profile UML (autre fichier ".uml" comportant un paquet de stéréotypes) ait été associé à la racine du modèle via le sous onglet "profile" des propriétés.

* sélectionner un type de données pour une propriété ou le cacher :

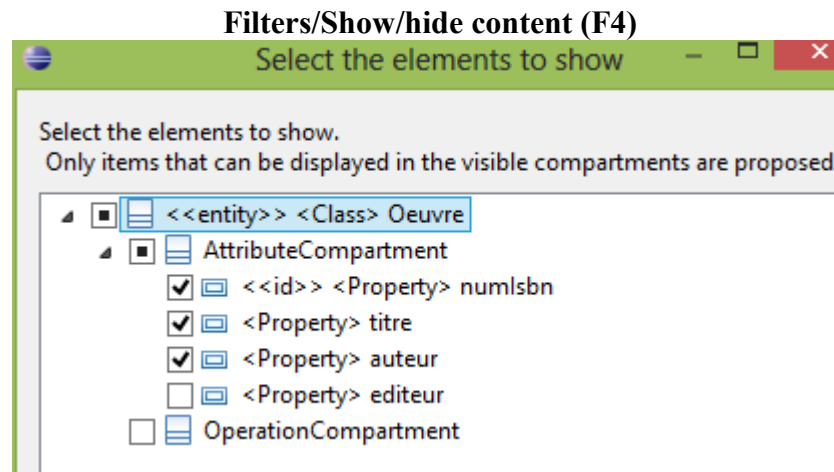
Sélectionner une propriété de la classe,
Si l'on souhaite (en analyse) cacher le type encore indéfini , il faut alors décocher "type" dans le sous onglet "appearance"



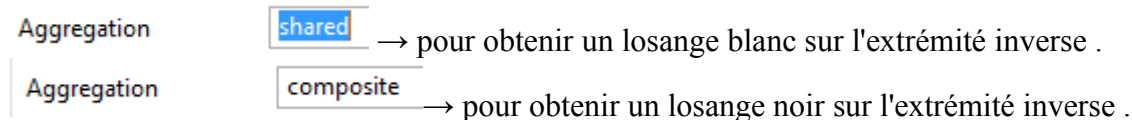
Si l'on souhaite (en conception détaillée), préciser un type de données parmi les types primitifs prédéfinis d'UML, il faut depuis le sous onglet "UML", le choisir via "..." en face "type :"



* Montrer ou cacher **Graphiquement** les différents éléments (propriétés/opérations) d'une classe:



* spécifier une **agrégation** ou une **composition** d'un coté d'une **association** :

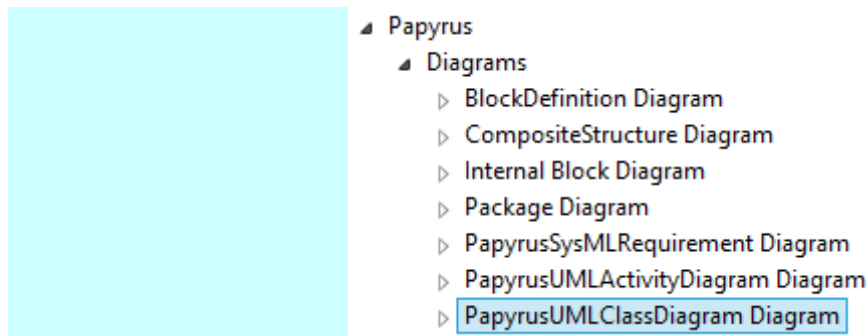


* spécifier une **classe d'association**

→ d'abord placer une association ordinaire et une classe ordinaire, relier ensuite par une liaison de type "AssociationClass" dans la sous palette "Edge" en partant de l'association et en pointant vers la classe. [Bug mi-2013 : les pointillés disparaissent lorsque l'on ferme et ré-ouvre le diagramme]

1.7. Préférences/options sur l'éditeur Papyrus UML

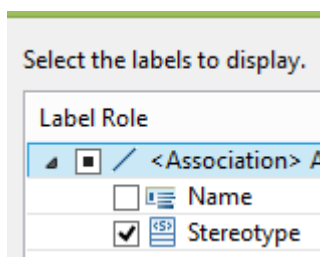
Windows/preferences/



1.8. Edition d'un diagramme de Use Cases (spécificités)

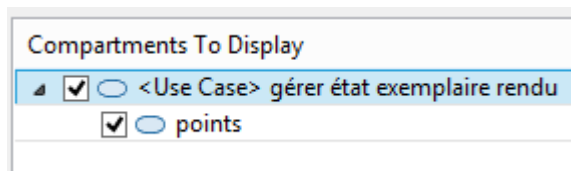
* Montrer ou cacher les différents éléments d'une association sélectionnée:

Filters / No Connector Label ou / Managed Connector Label



* Montrer ou cacher un point d'extension (lié à un <<extend>>) :

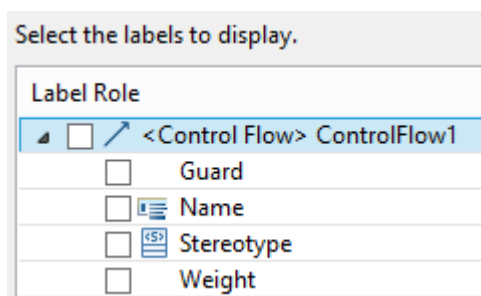
Filters/ Show/hide compartments



1.9. Edition d'un diagramme d'activités (spécificités)

* Montrer ou cacher les détails d'une liaison (controlFlow) :

Filters / No Connector Label ou / Managed Connector Label

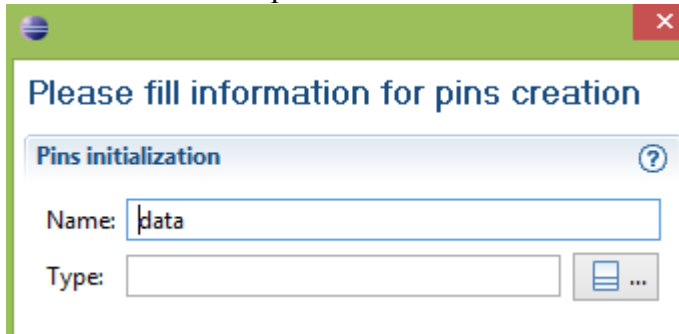


* Paramétrage des pins (output et input) autour d'un flot d'objet(s) :

Placer un "object flow" entre deux actions via la palette.

Au moment de l'établissement de la liaison, la boîte de dialogue suivante apparaît alors pour paramétrer le nom et/ou le type des objets (données/document) qui seront véhiculés d'une activité à

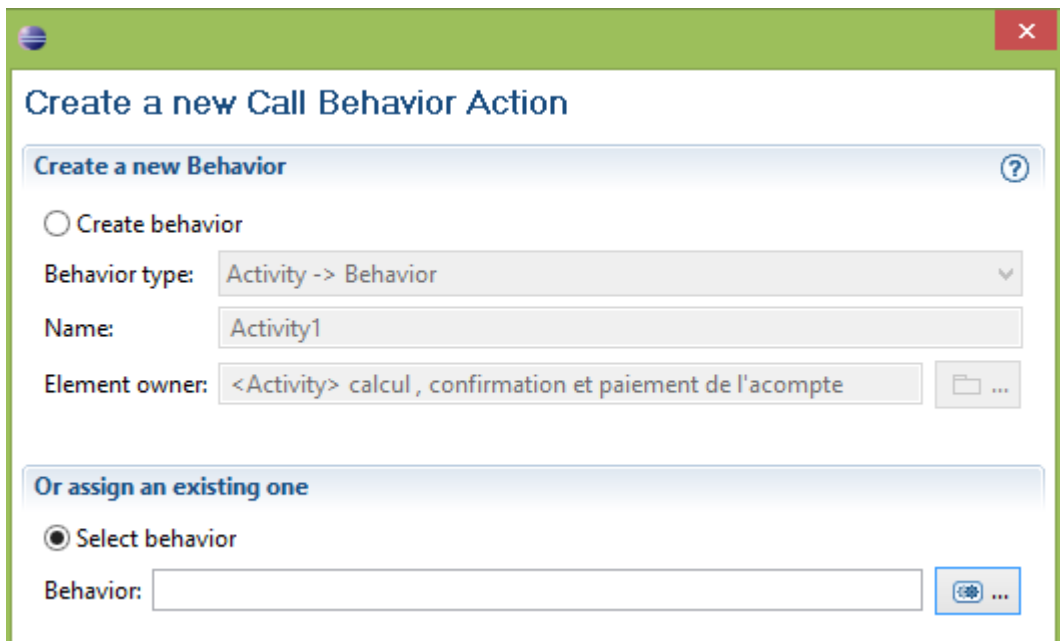
l'autre . A partir de ce paramétrage, l'éditeur "papyrus" va automatiquement construire des "pins" de même nature de chaque côté de la liaison.



* **Précautions à prendre pour bien paramétrer les liaisons d'entrées et de sorties** autour d'un "losange de décision" ou d'un "fork" :

→ De façon à contourner certains bugs temporaires de papyrus, il vaut mieux bien définir la (ou les) entrée(s) avant de définir la (ou les) sortie(s).

* **Paramétrage d'un "call behavior action" pour naviguer d'un diagramme d'activité à un sous autre :**

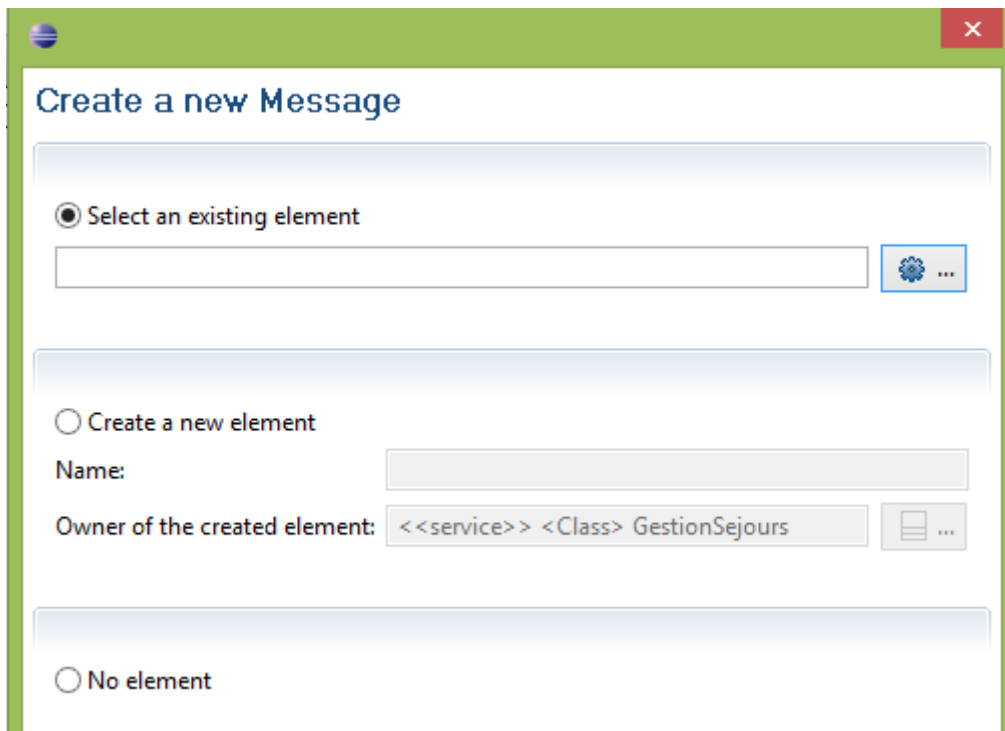


NB : de façon à bien contrôler la position du sous diagramme dans la hiérarchie du "model explorer" , on peut soit pré-crée le sous diagramme pour le sélectionner ensuite , soit bien paramétrer l'élément propriétaire ("owner") lors d'une création à la volée.

1.10. Edition d'un diagramme de séquences (spécificités)

Mode opératoire:

- créer un diagramme de séquence (idéalement en tant que détail d'un "use case")
- placer des **lignes de vie ("LifeLine")** et préciser les **types représentés** (acteurs, classes, ...) en effectuant des "glisser/poser" d'un élément (acteur ou classe) du "model explorer" vers l'entête du "LifeLine" et confirmer l'opération via un click sur "set represent ...".
- placer des **blocs d'exécution** sur les lignes de vie
- placer des **messages** entre un bloc d'exécution et un autre
- **paramétrer** les messages (*saisir un nom ou bien sélectionner une opération disponible au niveau de type d'objet qui reçoit le message*)













Remarque importante :

Lorsque (via l'option "create new element" de cette boîte de dialogue) l'on crée de nouvelles méthodes/opérations dans la classe de l'objet qui reçoit le message, celles-ci sont présentes dans le "model_explorer" mais n'apparaissent pas automatiquement dans les diagrammes de classes. Pour faire graphiquement apparaître les nouvelles opérations dans les classes d'un diagramme de classes, il faut activer le menu contextuel "filters/ show/hide contents" et sélectionner les nouvelles méthodes (supplémentaires) à afficher.

NB: On peut également placer des fragments combinés (avec mot clef "alt", "opt", "loop", ...) d'UML2.

1.11. Edition d'un diagramme d'états (spécificités)

* Paramétrage interne d'un état (do, exit, entry) :

State invariant	<Undefined>	  	Entry	<Undefined>
Do activity	 <Opaque Behavior> choix période, transport	  	Exit	<Undefined>
Submachine	<Undefined>	  		

1.12. Edition d'un diagramme de composants (spécificités)

RAS

1.13. Edition d'un diagramme de déploiement (spécificités)

RAS

1.14. Génération de code java (via le générateur par défaut de Topcased)

- Ouvrir (si besoin) la vue "**Navigator**"
- Sélectionner le fichier appXY.uml et activer le menu contextuel "**Code Generator / generateJava**".

→ le code généré apparaît alors dans le projet courant .