Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Filière Systèmes industriels

Orientation Infotronics

Diplôme 2007

Philippe Pralong

Format XML optimisé pour la traduction de UML en Code

Professeur	Medard Rieder
Expert	Régis Chevrel

Sion, le 23 novembre 2007

HES-SO Valais

SI	TV	EE	IG	EST
X	X	X	X	

Données du travail de diplôme
Daten der Diplomarbeit

FO.0.2.02.07.DB
DD/18/05/2006

Filière / *Studiengang* : Systèmes industriels

Confidentiel / *Vertraulich* ☐

Etudiant / *Student* Philippe Pralong	**Année scolaire / *Schuljahr*** 2006/07	**No TD / *Nr. DA*** SI/2007/9
Proposé par / *vorgeschlagen von* HES-SO Valais, UIT		**Lieu d'exécution / *Ausführungsort*** HES-SO Valais, DSI **Expert / *Experte*** Philippe Soulard, Sodius SA France

Titre / *Titel*:

Format XML optimisé pour la traduction de UML en Code

Description / *Beschreibung*:

Les différentes couches de la modélisation (M1 - M4) – qui étaient jusqu'à présent plutôt des termes théoriques – sont devenues aujourd'hui très concrètes grâce à des outils qui deviennent disponibles.

La modélisation spécifique à un domaine est un thème de recherche situé au coeur de cette problématique.

Objectifs / *Ziele:*

Le but de ce travail est de mettre en oeuvre une chaîne de ces outils qui permette alors de produire du code à partir d'un modèle d'un système représenté par un modèle spécifique à ce domaine.

Les outils suivants sont à utiliser :

- TOPCASED pour la création des Meta modèles
- TOPCASED pour la modélisation d'un système dans un langage spécifique
- MDWorkbench pour la transformation de modèles en code.

La chaîne d'outils pour la production de code doit être le plus possible automatisée. Un démonstrateur doit être réalisé et un rapport final doit être établi.

Signature ou visa / *Unterschrift oder Visum*	**Délais / *Termine***
Resp. de l'orientation infotronics	Attribution du thème / Ausgabe des Auftrags: 03.09.2007
Professeur/*Dozent*: Medard Rieder	Remise du rapport / Abgabe des Schlussberichts: 23.11.2007
Etudiant/*Student*:	Exposition publique / Ausstellung Diplomarbeiten: 30.11.2007
	Défenses orales / Mündliche Verfechtungen Semaine 49

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats:* .....

Pralong Philippe 2007

Format XML optimisé
pour la traduction de UML en code

Objectif

En premier lieu, faire découvrir les possibilités de l'outil Topcased, au travers d'un tutorial de démonstration. Il permet de réaliser des métamodèles et d'en modéliser des systèmes dans le langage défini au préalable.

En second, mettre en place une transformation de modèles de GME (Generic Modeling Environment, un éditeur de métamodèles et modèles) vers UML 2.1 au format XMI.

Résultats

Le tutorial de Topcased a été étudié et présenté dans le rapport. La transformation de modèle a occupé la majeure partie du travail. Elle réalise tous les objectifs demandés, jusqu'à la transformation de diagrammes de classes et de diagrammes d'objet UML.

Mots-clés

Model Transformation, Metamodel, Domain Specific Modelling, Model Driven Architecture, UML

Ziel

Als erstes sollen die Möglichkeiten des Programms Topcased mit Hilfe eines Tutorials erkundet werden. Topcased erlaubt es, Metamodelle zu realisieren und Systeme in der zuvor definierten Sprache zu modellieren.

*Als zweites soll eine Transformation von GME-Modellen (*Generic Modeling Environment, ein Editor für Metamodelle sowie Modelle*) nach UML 2.1 im Format XMI realisiert werden.*

Resultate

Das Topcased Tutorial wurde studiert und ist im Bericht präsentiert. Die Modeltransformation stellte den grössten Teil der Arbeit dar. Die Transformation realisiert die Zielvorgaben sowie die Transformation von UML Klassen- und Objektdiagrammen.

Schlüsselwörter

Model Transformation, Metamodel, Domain Specific Modelling, Model Driven Architecture, UML

Table des matières

Table des figures

Table des codes

1 Introduction

1.1 Objectifs

Ce travail de diplôme entre dans le cadre d'un projet nommé ADS[1]. ADS pour « Ad Hoc Design Studio » est un projet initié par l'Infotronic de la HES-SO.

Avec l'avancée technologique, les systèmes embarqués ne sont plus indépendants mais peuvent appartenir à un système d'information global. De ce fait, il est de plus en plus nécessaire de modéliser ces systèmes d'information plutôt que les nœuds séparément.

De plus, lorsque ces nœuds varient dans le temps, le système dans lequel ils sont utilisés est nommé réseau « ad hoc ».

Un réseau ad hoc propriétaire peut très bien régler un problème particulier sans se soucier de tous les autres points que traitent les réseaux standard (Zigbee, Bluetooth).

Enfin, la personne concevant un réseau ad hoc doit s'inquiéter de trois complexités :
- Complexité des applications, relativement spécifique à chaque noeud
- Complexité du moyen de communication
- Complexité de l'implémentation sur des environnements matériels différents

Le projet ADS a pour ambition de mettre en œuvre des outils permettant de mieux maîtriser cette triple complexité.

Le travail de diplôme, quant à lui, traite une des parties de ce projet ADS. Il intervient au niveau de la transformation de modèles.

La modélisation d'un réseau ad hoc est faite premièrement sous le programme « Generic Modeling Environment », de l'institut de logiciel pour systèmes intégrés à l'université Vanderbilt[2], qui permet d'avoir une vision assez simple et nette de ce que ce réseau doit représenter. Afin d'en générer du code par la suite, ce modèle GME va être transformé en UML afin d'être repris par « Rhapsody », de la firme Telelogic, qui offre une possibilité de conception au niveau modèle bien plus complète et avancée.

Cette transformation se fait à l'aide d'un outil développé par la société Sodius, qui est « MDWorkbench », basé sur l'environnement de programmation Eclipse.

[1] Description du projet ADS inspirée du document « Ad Hoc Design Studio », n°-17223-, de la réserve stratégique de la HES-SO.
[2] Référence : http://en.wikipedia.org/wiki/Generic_Modeling_Environment

1.2 Etapes

Afin de se familiariser avec l'outil MDWorkbench, la première étape consistera à faire le tutorial de transformation de modèles livré avec le logiciel. En parallèle à cela, un autre tutorial, basé sur l'outil Topcased, développé par un consortium[3], a été expérimenté. Celui-ci permet de construire ses propres métamodèles et modèles.

A l'origine du travail de diplôme, Topcased devait être utilisé pour refaire le métamodèle et le modèle du réseau ad hoc initialement créés sous GME, mais finalement ce dernier a été largement étudié et il est resté la base principale de modélisation. Cependant, Topcased deviendra certainement l'éditeur de modèles dans la suite du projet ADS, étant lié à Eclipse comme plug in, tout comme MDWorkbench.

Après avoir découvert brièvement les possibilités qu'offre MDWorkbench, la transformation de GME à Rhapsody commencera. Chaque étape sera décrite de la manière la plus explicite possible, avec des exemples afin de permettre au lecteur une visualisation précise de ce qui a été fait.

Enfin, quelques notions seront ajoutées sous le chapitre « 4.6 Transformation intelligente » car il y a des éléments qui, n'existant pas dans le modèle GME d'origine, sont ajoutés lors de la transformation en UML.

La partie du rapport concernant les tests sera très brève, puisque le « test » effectué est celui de la transformation, il n'y a pas lieu d'avoir des tests spécifiques en parallèle, de plus aucun matériel n'a été développé.

[3] Référence : http://fr.wikipedia.org/wiki/TOPCASED

2 Outils de développement

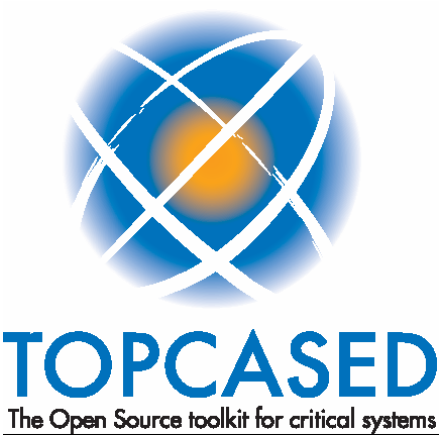
2.1 Topcased

2.1.1 Introduction

Topcased est un acronyme pour Toolkit in Open Source for Critical Applications & Systems Development[4] (boîte à outils open-source pour le développement d'applications critiques et de systèmes).

C'est un logiciel d'ingénierie assistée par ordinateur. Il contient un IDE basé sur le framework de la plateforme de développement Eclipse, à laquelle il ajoute des fonctionnalités essentiellement liées à la mise en œuvre de la première branche du cycle en V (voir Figure 1) pour l'ingénierie du logiciel, du matériel ou de systèmes mixtes logiciel/matériel.

Les moyens mis à disposition sont de la modélisation, implémentation, rétro-ingénierie, gestion de projet, etc…



[4] Référence : http://fr.wikipedia.org/wiki/TOPCASED

2.1.2 Tutorial

Un tutorial disponible avec le logiciel permet de découvrir dans les grandes lignes les possibilités de l'outil Topcased.

Par ce tutorial il est montré la manière de créer un métamodèle EMF (format .ecore) et d'en instancier un modèle par après. De plus, l'éditeur de modèle EMF par défaut étant sous forme d'arborescence, le tutorial propose un moyen de créer son propre éditeur graphique de modèles.

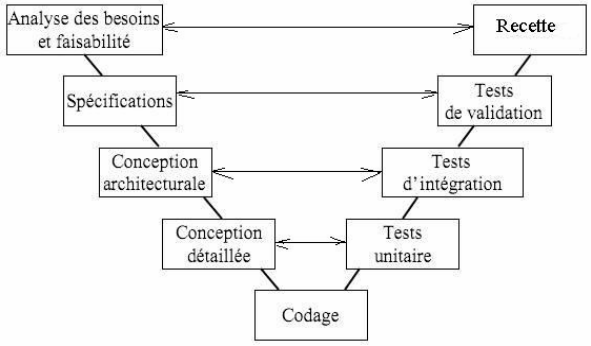


Figure 1 : Cycle en V[5] pour l'ingénierie logicielle

[5] Source : http://fr.wikipedia.org/wiki/Image:Cycle_de_developpement_en_v.jpg

2.1.3 Déroulement du tutorial

Pour débuter, voici le métamodèle « Network » qui va être créé. Il s'agit d'une architecture simpliste d'un réseau pouvant contenir des nœuds par lesquels sont reliés divers ordinateurs et Internet.

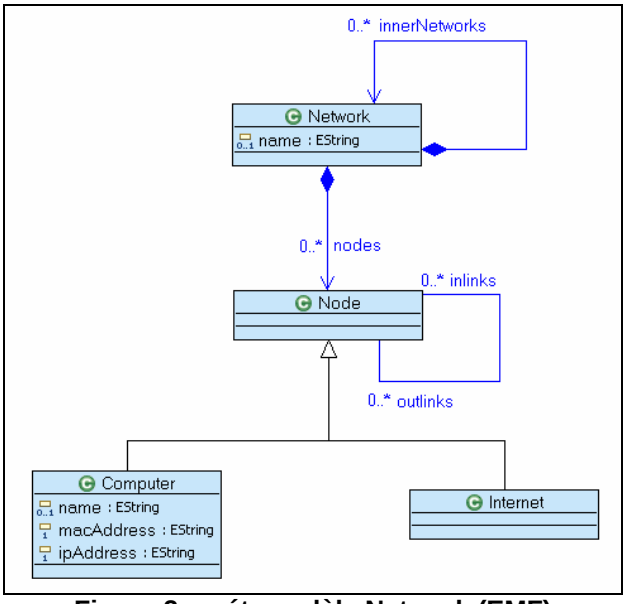


Figure 2 : métamodèle Network (EMF)

Dans l'arborescence de Topcased, il apparaît deux fichiers : network.ecoredi et network.ecore. Le premier est le diagramme du métamodèle dessiné avec l'éditeur d'Eclipse, le second est l'arborescence au format Ecore.

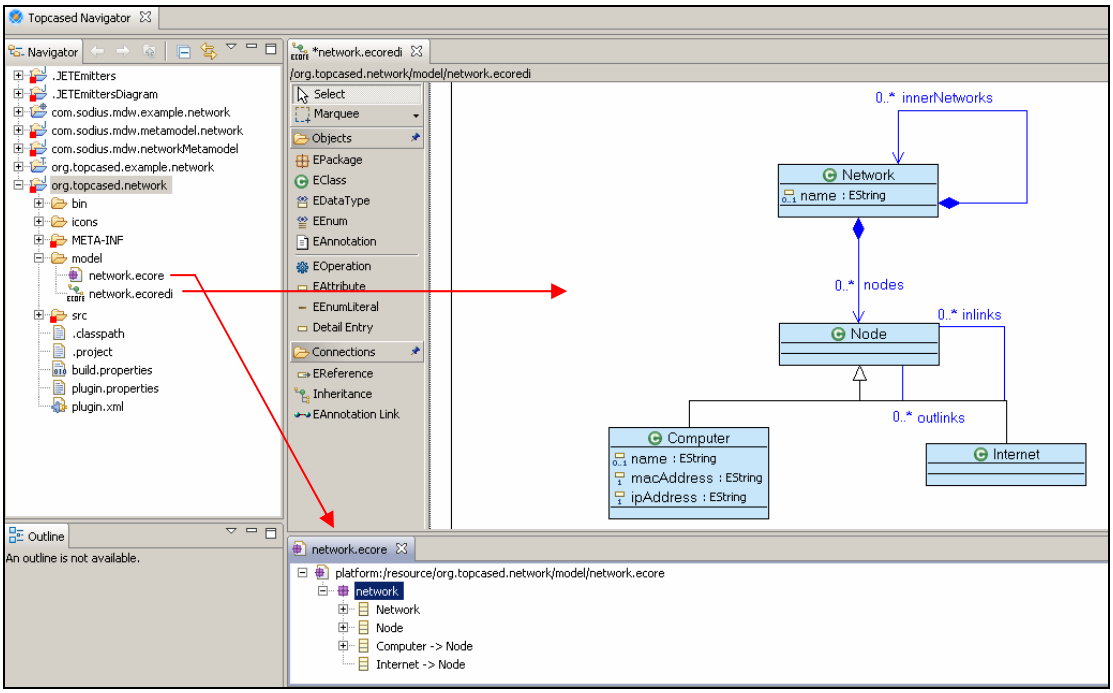


Figure 3 : network.ecore et network.ecoredi

⚠ Les deux vues ne sont pas forcément liées, le fait de modifier des éléments dans l'arborescence du fichier .ecore ne modifiera rien dans le fichier .ecoredi ! (Cependant, dans l'autre sens les modifications sont prises en compte)

Ce métamodèle est enregistré au format XMI[6] 2.0.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="network"
    nsURI="http://www.Topcased.org/network/1.0" nsPrefix="network">
  <eClassifiers xsi:type="ecore:EClass" name="Network">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="innerNetworks" upperBound="-1"
        eType="#//Network" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="nodes" upperBound="-1"
        eType="#//Node" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Node">
    <eStructuralFeatures xsi:type="ecore:EReference" name="inlinks" upperBound="-1"
        eType="#//Node" eOpposite="#//Node/outlinks"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="outlinks" upperBound="-1"
        eType="#//Node" eOpposite="#//Node/inlinks"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Computer" eSuperTypes="#//Node">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="macAddress" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="ipAddress" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Internet" eSuperTypes="#//Node"/>
  <eClassifiers xsi:type="ecore:EClass" name="Test"/>
</ecore:EPackage>
```

Code 1 : network.ecore au format XML (XMI2.0)

La prochaine étape consiste à créer le fichier .genmodel qui contiendra toute la configuration nécessaire à la génération des différents outils d'édition de modèle basé sur ce métamodèle.

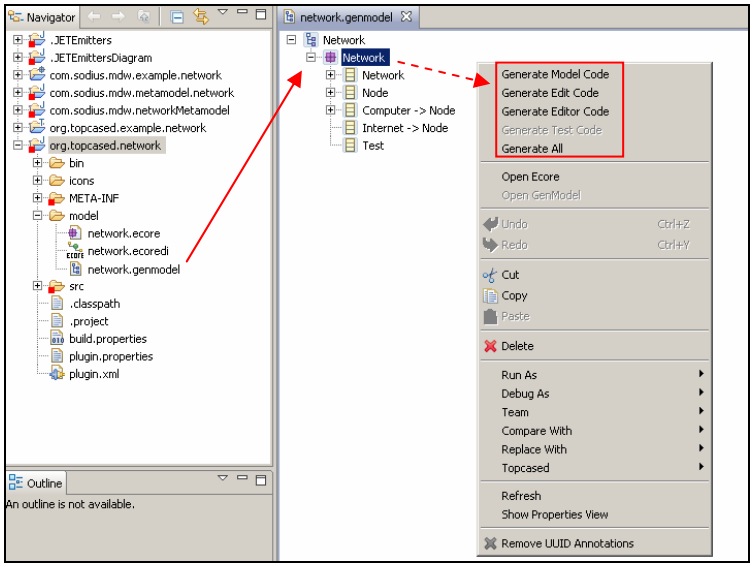


Figure 4 : network.genmodel

[6] XMI (XML Metadata Interchange) est un standard créé par l'OMG. Il sert à l'échange d'informations de métadonnées UML basé sur XML (source : http://fr.wikipedia.org/wiki/XMI).

- Generate Model Code : génère l'implémentation Java des entités du métamodèle.
- Generate Edit Code : crée le code permettant l'édition du modèle.
- Generate Editor Code : crée le code d'une petite application d'édition du modèle. Elle a besoin du Edit Code pour fonctionner.

On retrouve ensuite dans l'explorateur les classes créées après la génération de network.genmodel.

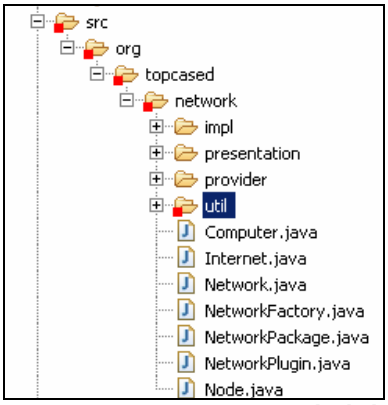


Figure 5 : Fichiers java générés

A ce moment, afin de pouvoir créer un modèle dérivé de notre métamodèle « Network », il est nécessaire d'enregistrer le projet sous forme de plug in pour qu'Eclipse le reconnaisse en tant que tel.

Il faut exporter le plug in, et d'une fois qu'il est créé, il doit être copié dans le répertoire « …/eclipse/plugins/ »

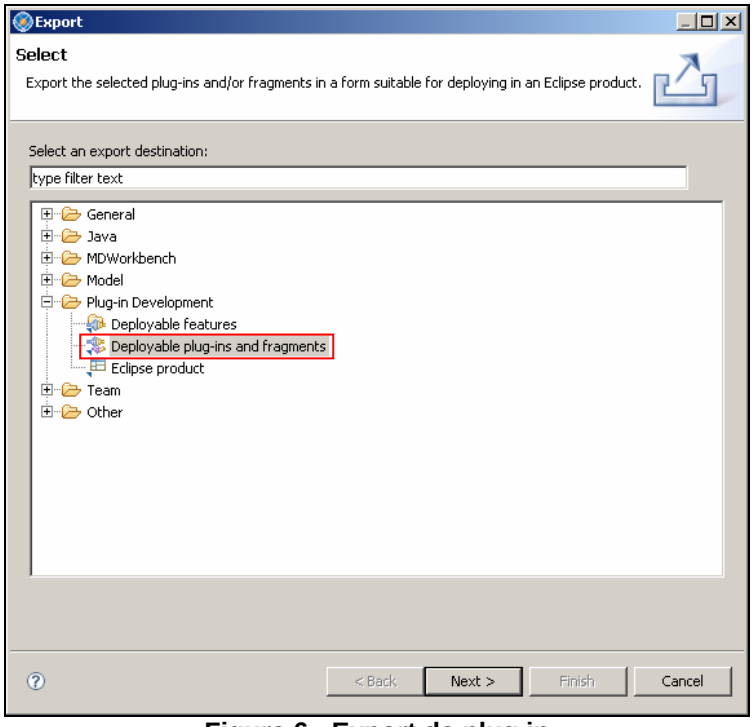


Figure 6 : Export de plug in

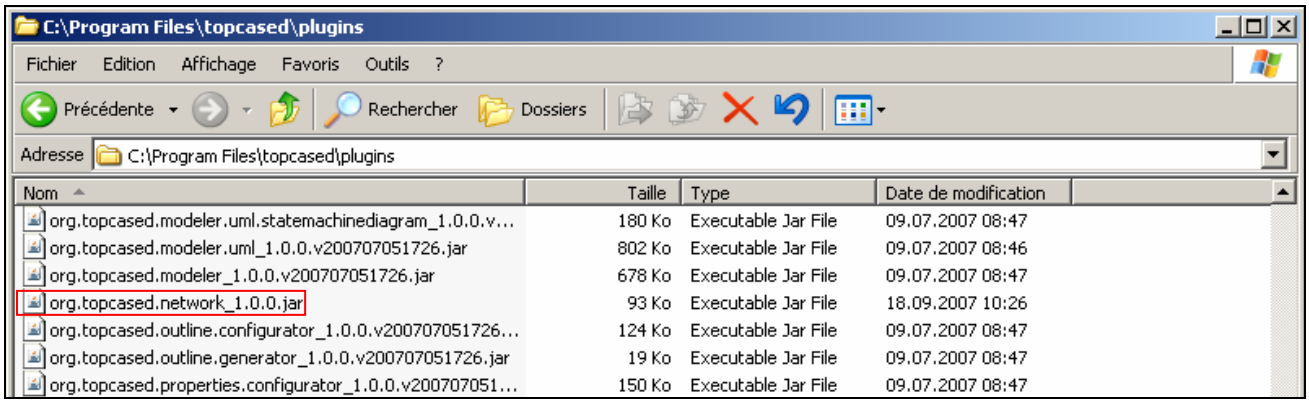


Figure 7 : Plug in créé

Après avoir redémarré Eclipse, le plug in est initialisé et il est dorénavant possible d'instancier le métamodèle « Network ». Cela se voit lors de la création d'un fichier, sous l'option « Example EMF Model Creation Wizard », le type « Network Model» a été rajouté par le plug in.

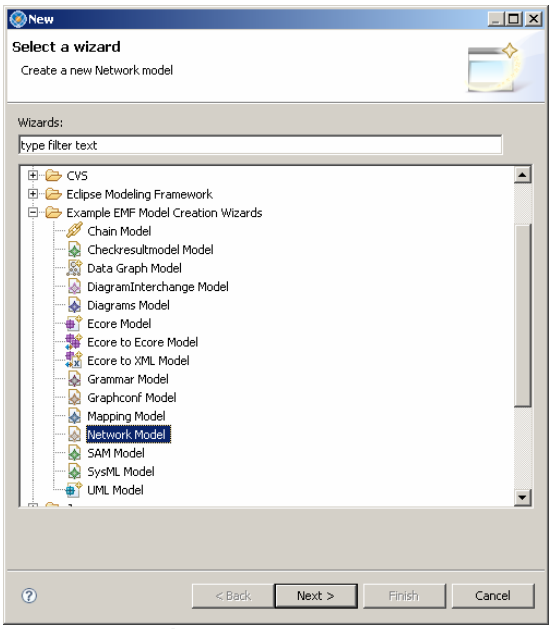


Figure 8 : Nouveau modèle de type « Network Model » disponible

L'éditeur existant nous permet de développer le modèle sous forme d'arborescence. Les propriétés de chaque entité, précédemment définies dans le métamodèle, peuvent être modifiées via l'onglet « Properties »

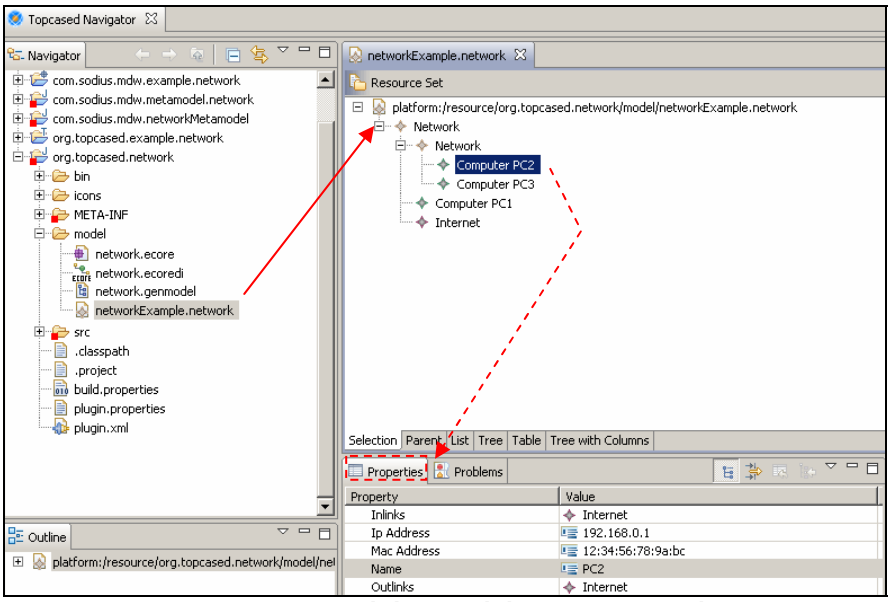


Figure 9 : Propriétés des éléments du modèle

Comme cet éditeur demeure assez basique, le tutorial propose de créer son propre éditeur graphique avec des images propres au métamodèle.

Pour cela, il faut créer deux nouveaux fichiers du projet Topcased, le premier étant « network.diagramconfigurator » qui définit tous les outils dont on va se servir dans notre éditeur graphique, comme les images, les liens, etc…, le second étant « network.editorconfigurator » qui permet la configuration des ressources utiles au modèle, comme le métamodèle *network.ecore* et l'outil de génération du modèle *network.genmodel.*

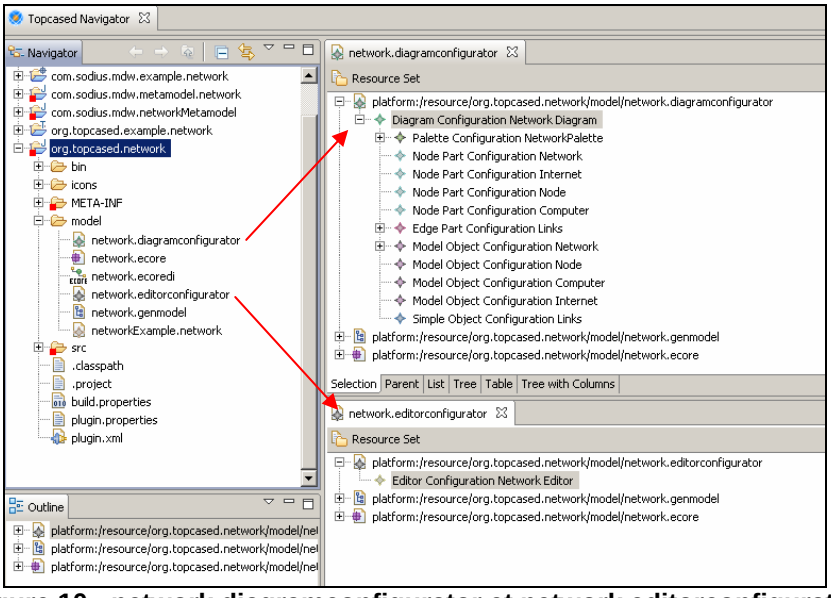


Figure 10 : network.diagramconfigurator et network.editorconfigurator

Dès que ces fichiers ont été configurés, il suffit de générer l'éditeur (il faut le générer sur les deux fichiers !)

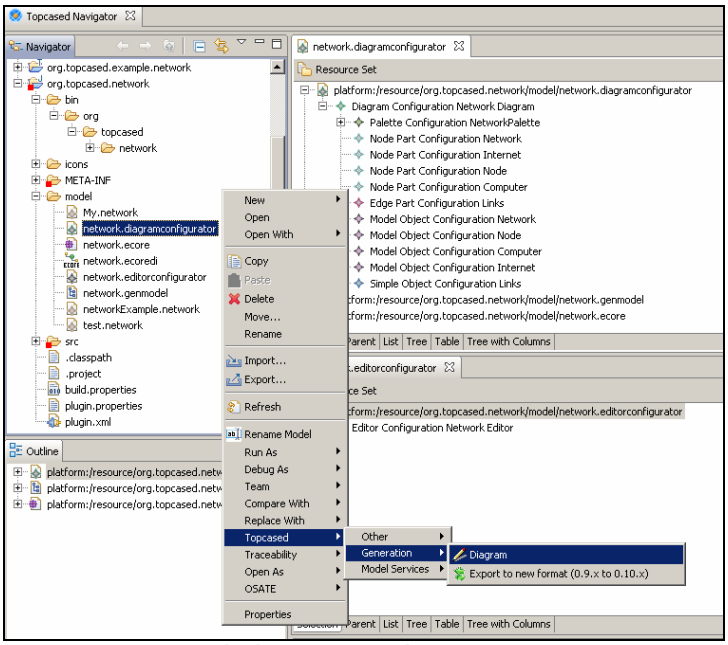


Figure 11 : Génération de l'éditeur de diagramme

Cela crée deux nouveaux packages :

- org.Topcased.modeler.network.netdiagram
- org.Topcased.modeler.network

Dans le package du .netdiagram, on va trouver le fichier du modèle précréé, sous le nom « %name%.network » et sa version graphique « %name%.networkdi ». C'est dans ce dernier que l'on va retrouver les options graphiques que nous avons définies précédemment dans le *network.editorconfigurator*, et ces outils permettent de dessiner le modèle afin d'en tirer une vue plus accessible et intuitive que sous la forme d'une arborescence.

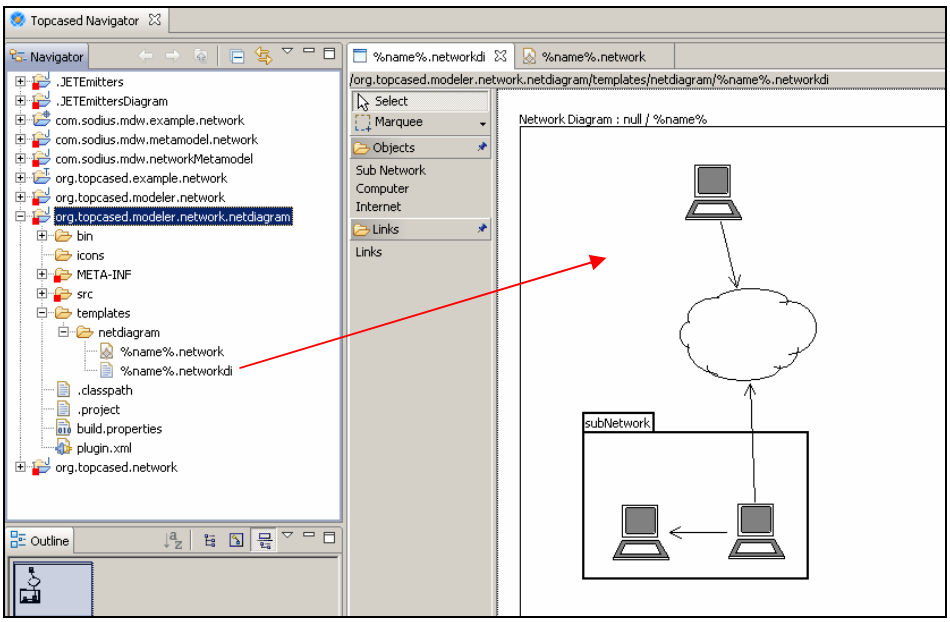


Figure 12 : Editeur de diagramme créé

2.2 MDWorkbench

2.2.1 Introduction

Comme Topcased, MDWorkbench[7] est outil de développement basé sur la plateforme Eclipse. Créé par la société Sodius, il sert à la génération de code et aux transformations de modèles suivant le principe du MDA/MDE (Model Driven Architecture / Model Driven Engineering)

Il peut générer du code via des templates, il manipule des modèles avec des langages tels que ATL, Java, règles prédéfinies et il possède des connecteurs préexistants (XMI 1.0 à 2.1, XML, Hibernate,…).

[7] Référence : http://www.mdworkbench.com/

2.2.2 Tutorial

La libraire MDWorkbench propose trois tutoriaux :
- Tutorial de génération
- Tutorial de documentation
- Tutorial de transformation

Nous allons nous intéresser seulement au tutorial de transformation, car c'est sur ce sujet que porte le travail de diplôme.

Nous découvrirons ici comment transformer un modèle UML 2.0 en modèle Relational. Cela donnera les bases pour travailler sur le sujet du diplôme, qui concerne la transformation d'un modèle GME (Generic Modeling Environment) en UML 2.1.

Le passage de modèle à modèle est guidé par des *ruleset*, ou ensembles de règles qui régissent chaque étape du mécanisme de transformation.

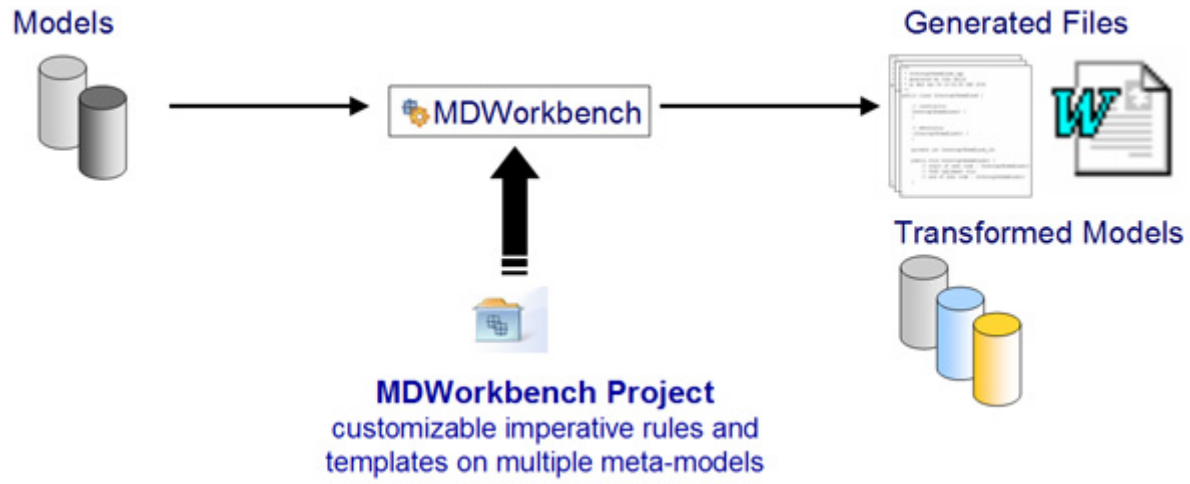


Figure 13 : Vue[8] d'ensemble de MDWorkbench

[8] Source : documentation d'aide de MDWorkbench

2.2.3 Déroulement du tutorial

Dans cet exemple les modèles sont fournis avec le tutorial, le but n'étant pas d'en construire.

Voici le modèle à partir duquel le code va être généré :

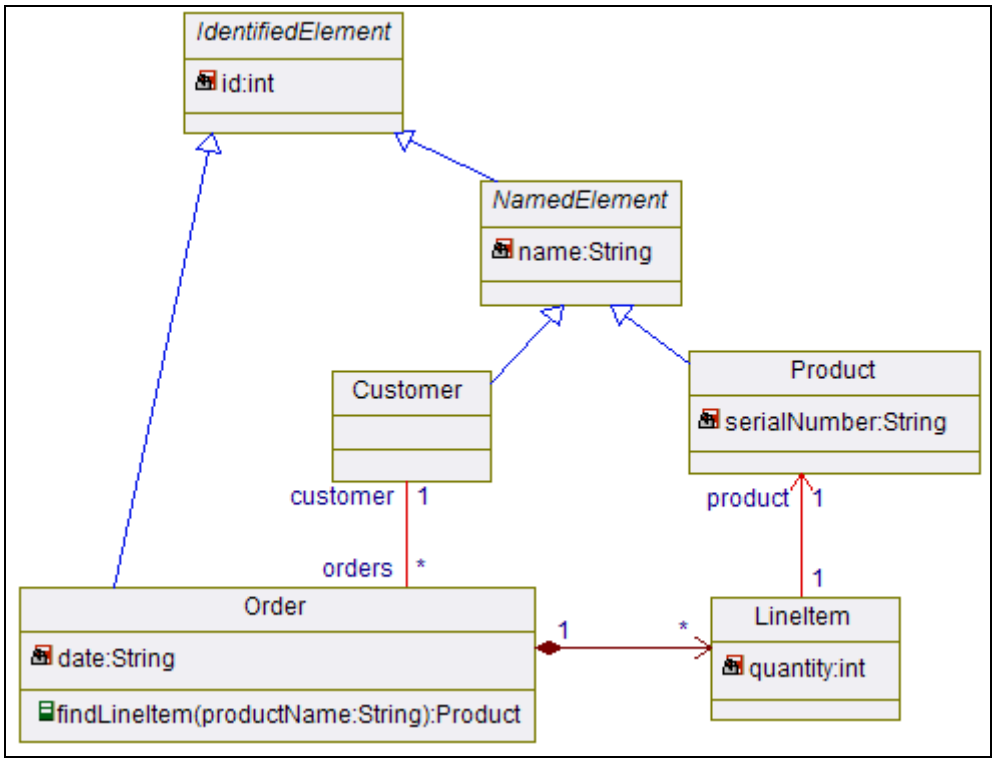


Figure 14 : Modèle d'une librairie

Le projet se crée automatiquement en passant par la page de bienvenue d'Eclipse

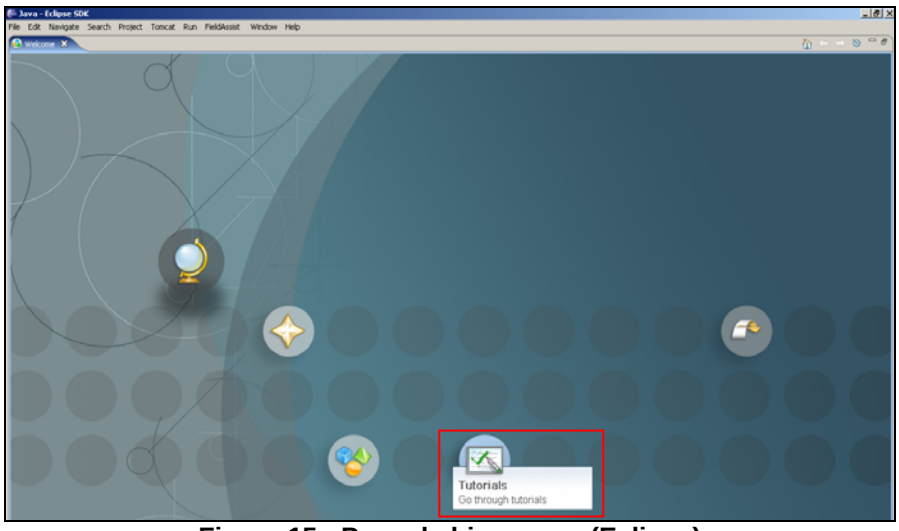


Figure 15 : Page de bienvenue (Eclipse)

Et dans ce projet, figure le modèle cité plus haut « *product20.xmi* ».

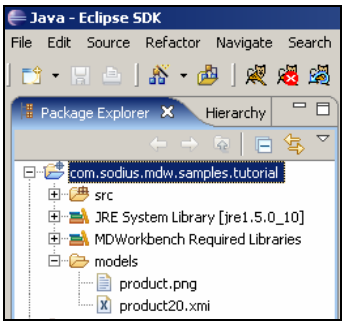


Figure 16 : Package du tutorial

En passant à la perspective MDWorkbench, une petite fenêtre apparaît. Depuis cette fenêtre il y a tous les métamodèles livrés avec MDW permettant la lecture de modèles. Celui qui nous intéresse est l'UML 2.0.

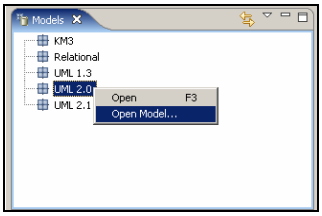


Figure 17 : Vue des métamodèles

MDWorkbench possède un lecteur de modèle qui nous montre tous les types du métamodèle, à gauche, et les objets dérivés de ces types créés dans le modèle, à droite.

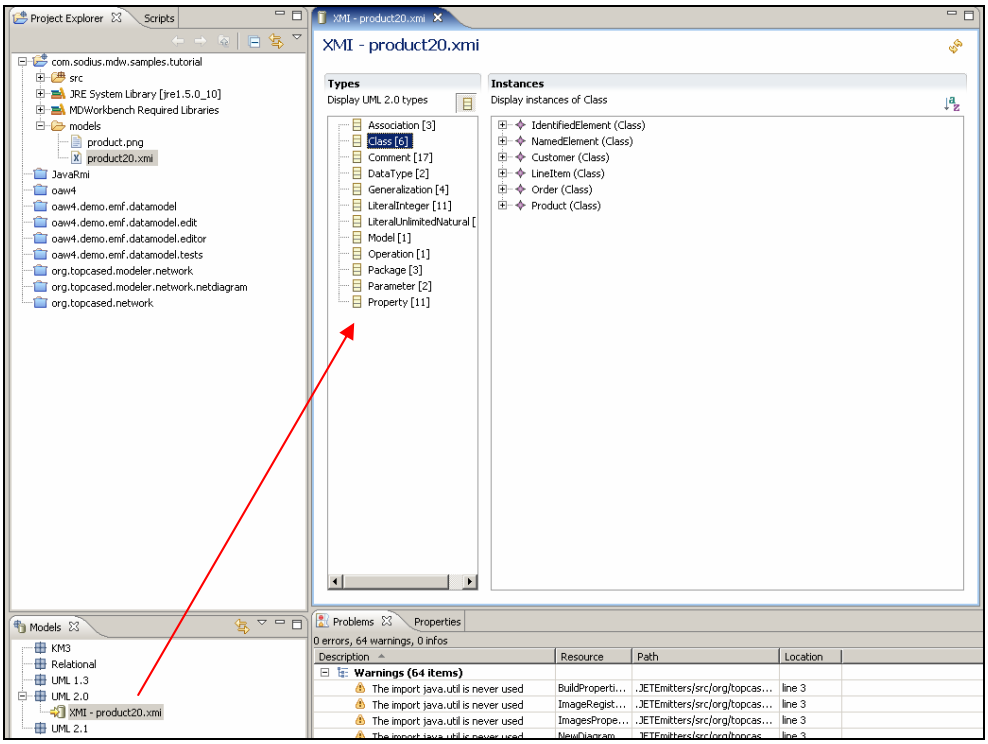


Figure 18 : Vue du modèle avec le « Model Viewer » de MDWorkbench

Comme le tutorial propose une transformation d'UML à Relationnel, nous pouvons inspecter la vue du métamodèle Relationnel, en ouvrant simplement celui-ci.

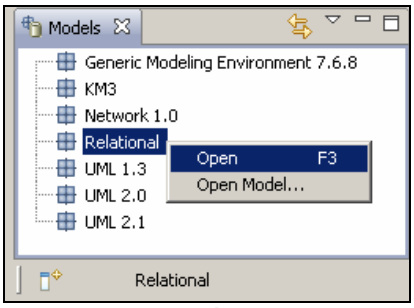


Figure 19 : Modèle Relationnel

Comme pour la vue précédente, voici celle du métamodèle Relationnel.

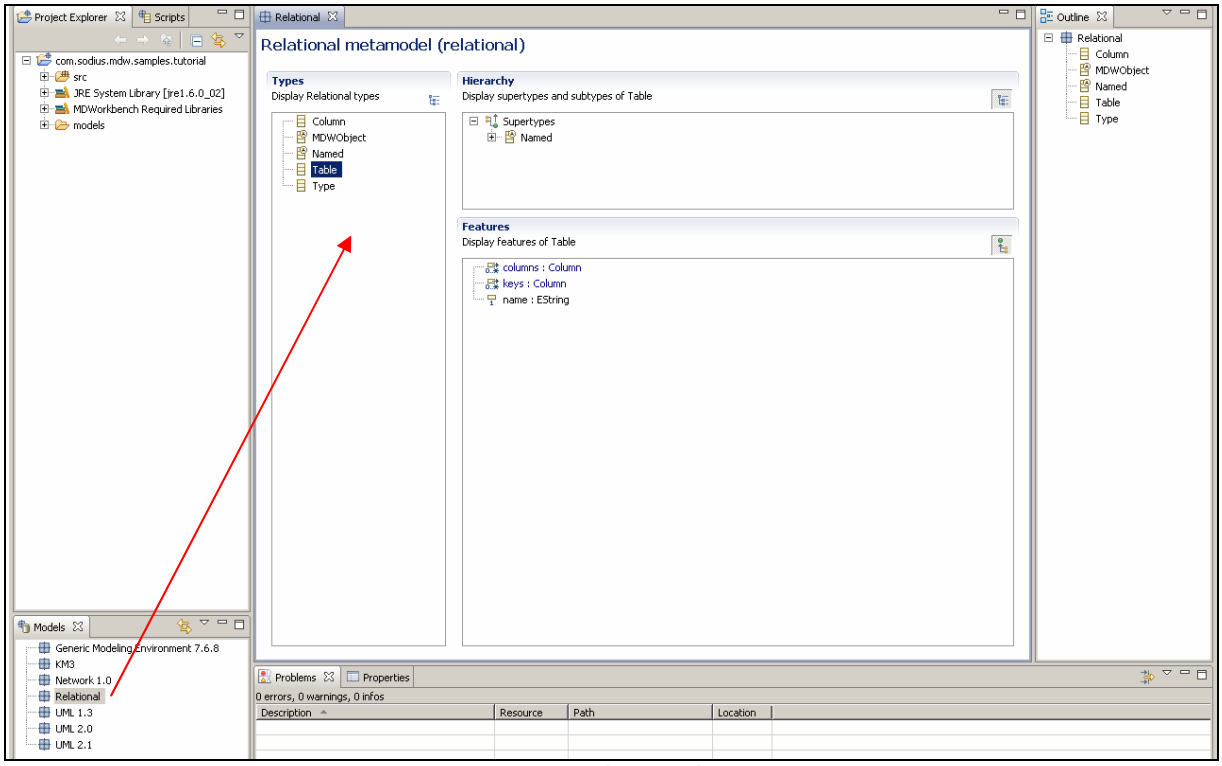


Figure 20 : Vue du métamodèle Relationnel

Pour transformer un type de modèle en un autre, l'élément majeur se trouve être le « ruleset de transformation ».

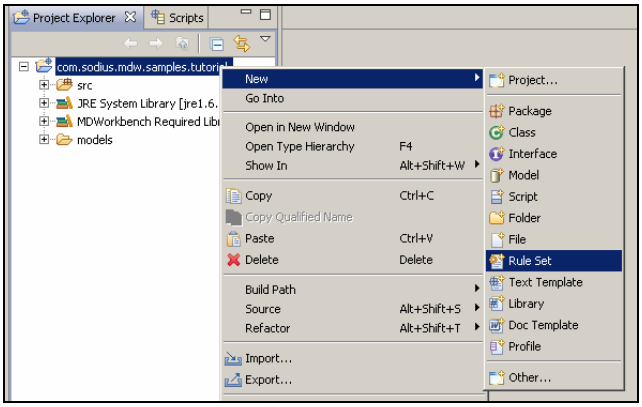


Figure 21 : Création d'un « Ruleset »

Cela crée un fichier « UML »Relational.mqr » dans le dossier *src*.

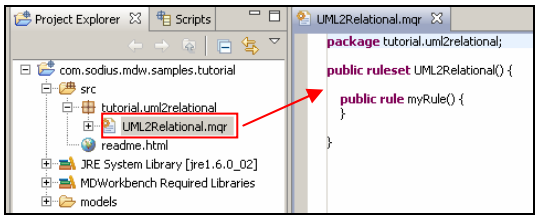


Figure 22 : Relational.mqr

Puisque le but est de faire la transformation d'un type de modèle en un autre, il faut rajouter les paramètres « in » et « out » au *ruleset*.

```
package tutorial.uml2relational;

public ruleset UML2Relational(in source : uml20, out target : relational)
{
   public rule myRule()
   {

   }
}
```

Code 2 : Paramètres « in » et « out »

Ce *ruleset* étant vide, il est nécessaire d'y rajouter les méthodes qui permettent de rapporter les éléments UML au format Relationnel. La première étape consiste à créer pour chaque classe UML une table dans le modèle Relationnel. Tout d'abord il faut renommer le méthode « myRule() » en « main() » et y rajouter le code suivant.

```
package tutorial.uml2relational;

public ruleset UML2Relational(in source : uml20, out target : relational) {

    public rule main()
   {
      // create a Table for each Class
      foreach (class : uml20.Class in source.getInstances("Class"))
      {
         @createTable(class, target.create("Table"));
      }
   }

   private rule createTable(class : uml20.Class, table : relational.Table)
   {
      // set the name
      table.name = class.name;
   }

}
```

Code 3 : Création d'une table

Le type **foreach** permet de parcourir chaque élément précisé dans les paramètres, soit ici toutes les classes du modèle UML2.0.

Afin de faire un premier essai de transformation, il faut créer un nouveau démarrage (Run…) de type Workbench, indiquer que l'élément principal est une règle (1) et lui donner le modèle d'entrée *in* (2) et créer un fichier pour la sortie *out* (3).

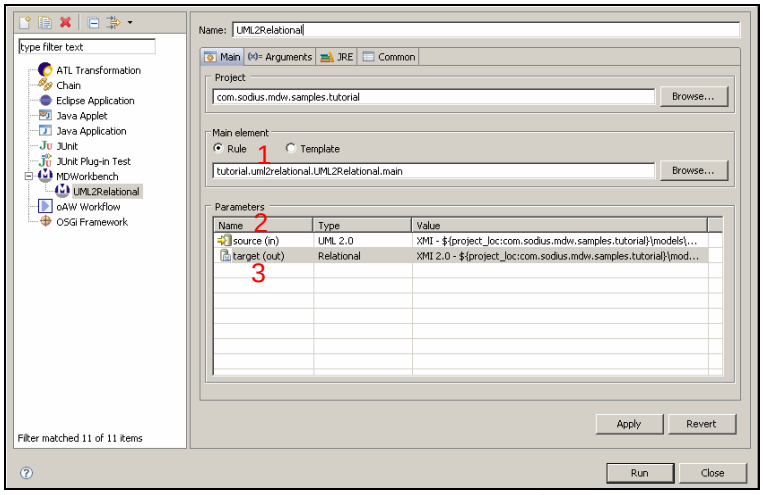


Figure 23 : Configuration de l'exécutable

A l'exécution, la console nous indique que la transformation s'est bien exécutée,

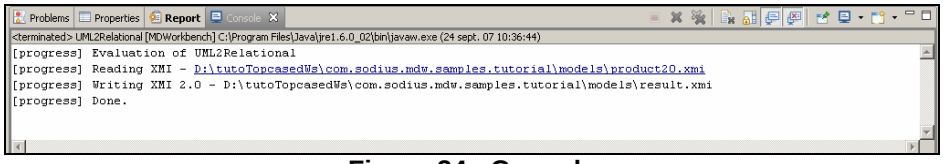


Figure 24 : Console

et le fichier result.xmi a bel et bien été créé, avec les tables qui représentent les classes du modèle UML.

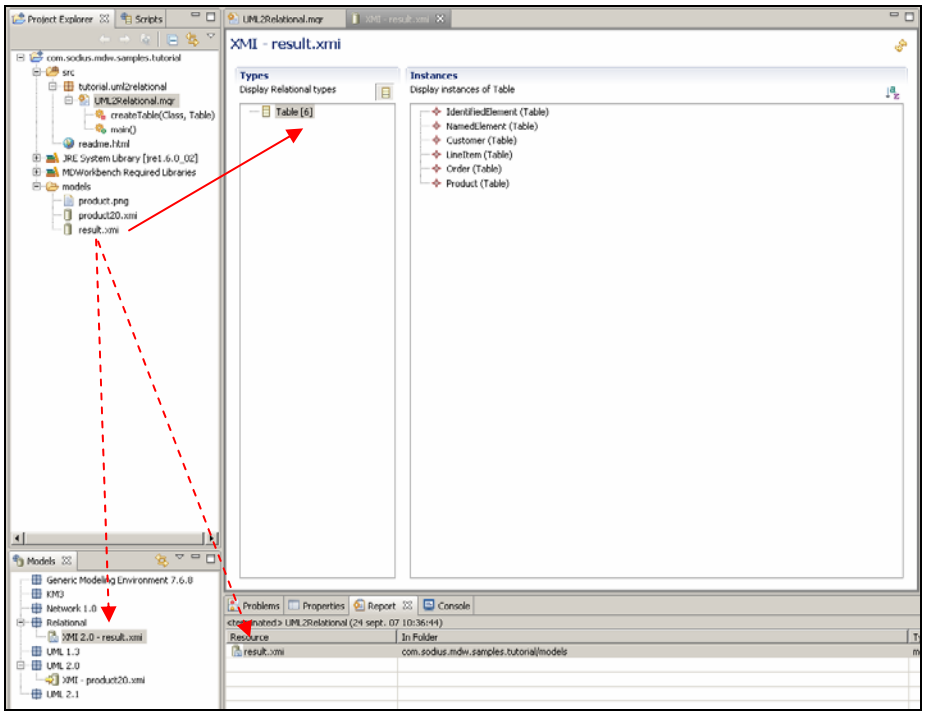


Figure 25 : result.xmi

Cependant, toutes ces tables sont vides, c'est comme si les classes ne contenaient aucun attribut, alors pour remédier à cela, il faut rajouter la création de colonnes pour chaque table, colonnes qui sont à l'image des attributs UML.

```
package tutorial.uml2relational;

public ruleset UML2Relational(in source : uml20, out target : relational)
{
   public rule main()
   {
      // create a Table for each Class
      foreach (class : uml20.Class in source.getInstances("Class"))
      {
         @createTable(class, target.create("Table"));
      }
   }

   private rule createTable(class : uml20.Class, table : relational.Table)
   {
      // set the name
      table.name = class.name;

      // create a column for each attribute
      foreach (attribute : uml20.Property in class.attribute)
      {
         @createColumn(attribute, target.create("Column"));
      }

   private rule createTable::createColumn(attribute : uml20.Property,
                                          column : relational.Column)
   {
      // set the name
      column.name = attribute.name;
      // set the owner of the column
      column.owner = table;
   }
}
```

Code 4 : Création des colonnes

La méthode « createColumns() » est une sous-règle de « createTable() », elle n'est appelée que dans celle-ci et pas directement dans le « main() » et de plus elle possède des accès directs aux paramètres de « createTable() ».

Lors de l'exécution, le modèle Relationnel s'enrichit.

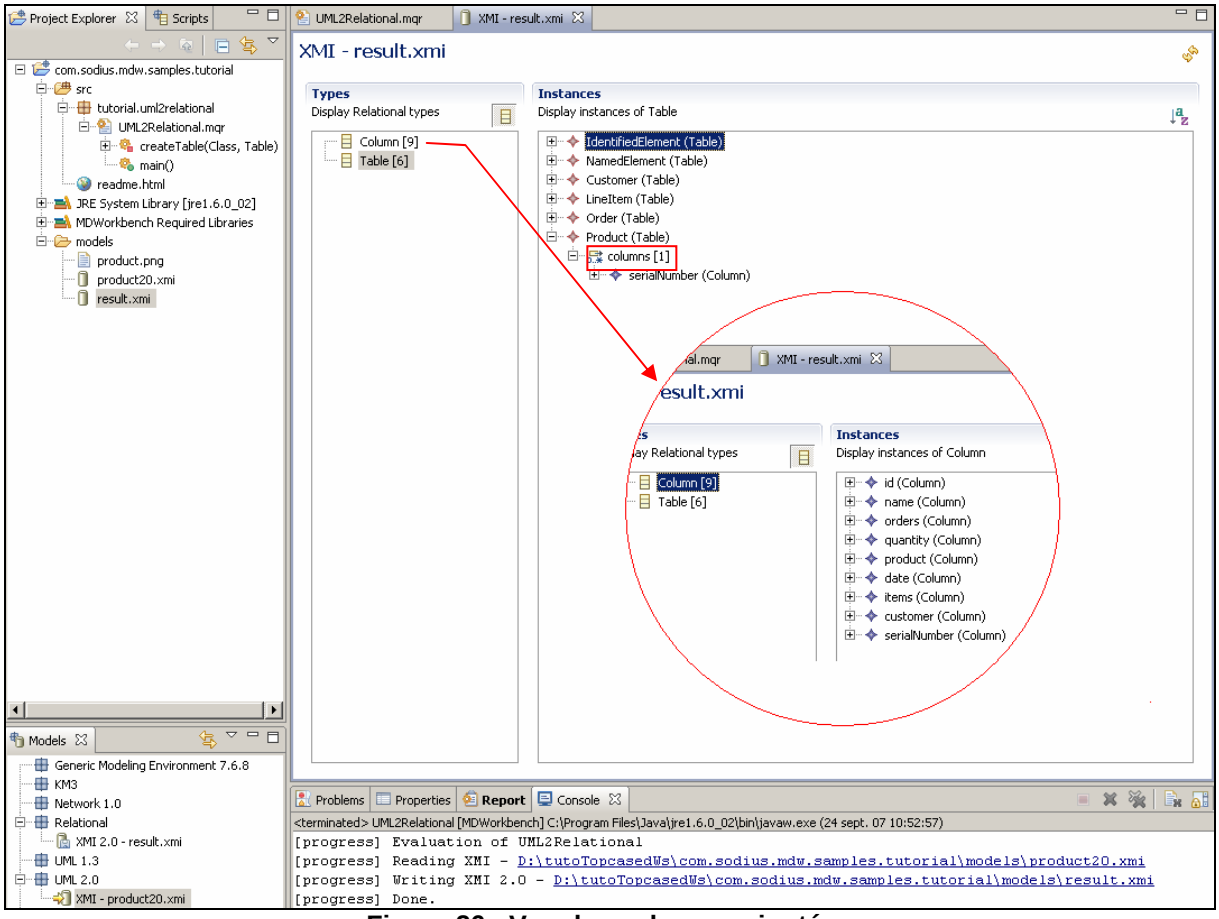


Figure 26 : Vue des colonnes ajoutées

Maintenant que les attributs ont été ajoutés sous forme de colonne, il est nécessaire d'indiquer le type de ces attributs,

```
package tutorial.uml2relational;

public ruleset UML2Relational(in source : uml20, out target : relational)
{
    public rule main()
   {
      @createTypes();

      // create a Table for each Class
      foreach (class : uml20.Class in source.getInstances("Class"))
      {
         @createTable(class, target.create("Table"));
      }
   }

   private rule createTypes()
   {
      // create a Relational Type for each UML DataType
      foreach (dataType : uml20.DataType in source.getInstances("DataType"))
      {
         var type : relational.Type = target.create("Type");

         if (dataType.name.equalsIgnoreCase("int"))
            type.name = "INT";
         else if (dataType.name.equalsIgnoreCase("string"))
            type.name = "VARCHAR(255)";
         else
            type.name = dataType.name;

         dataType#coref.add(type);
      }
   }
...
```

et dans la méthode « createColumn() » il faut rajouter le « set » pour le type.

```
...

   private rule createTable::createColumn(attribute : uml20.Property,
                                          column : relational.Column)
   {
         // set the name
         column.name = attribute.name;
         // set the owner of the column
         column.owner = table;
         // set the type
         column.type = attribute.type#coref.first();
   }
...
```

Code 5 : Ajout des types des colonnes

Le modèle est à nouveau mis à jour.

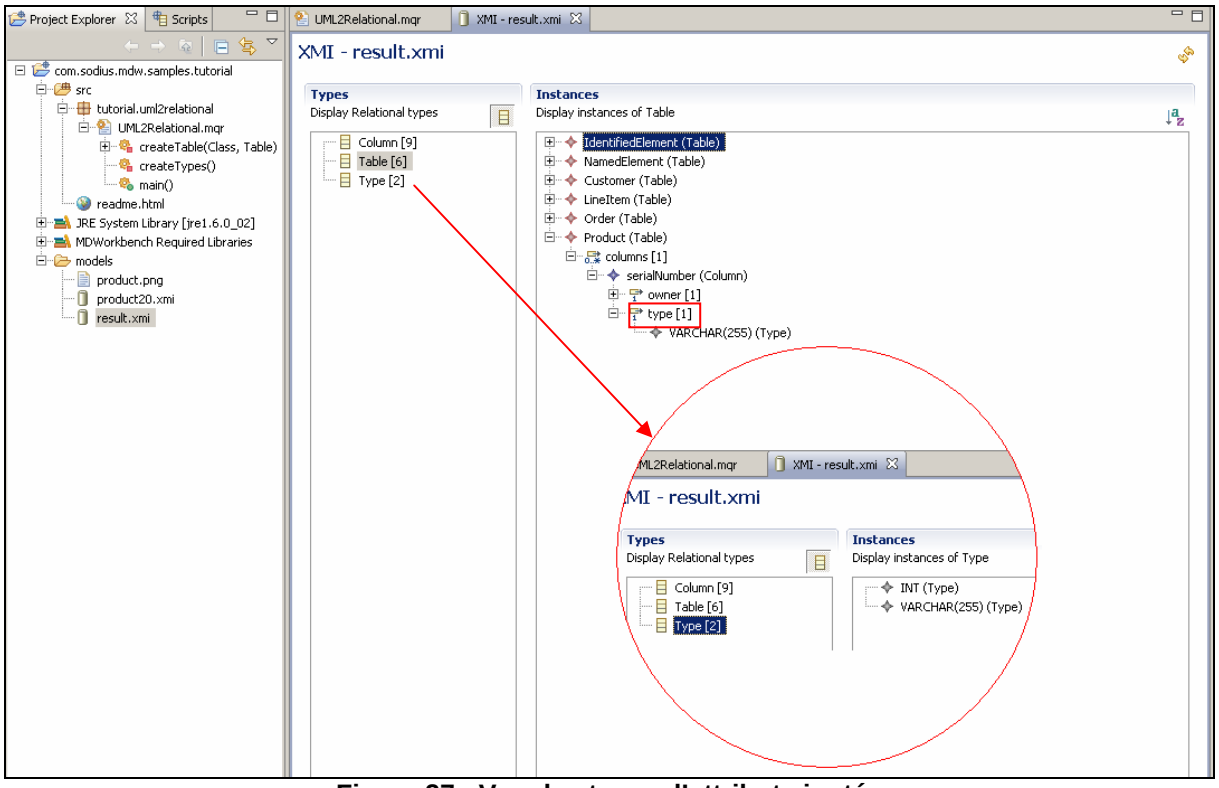


Figure 27 : Vue des types d'attribut ajoutés

Voici le principe de la transformation de modèle à modèle. Il existe aussi la possibilité de créer des templates pour la génération de script SQL, dès que l'on veut travailler avec des bases de données.

3 Projet AdHocNetwork

3.1 Introduction

AdHocNetwork est l'idée d'un réseau tirée du projet ADS. Le but serait de pouvoir déployer des réseaux ad hoc virtuellement sur n'importe quel type de système.

Un réseau ad hoc (en latin : « qui va vers ce vers quoi il doit aller », c'est-à-dire « formé dans un but précis ») sont des réseaux indépendant de la physique de communication capables de s'organiser sans infrastructure définie préalablement.

GME est l'outil de modélisation pour cet AdHocNetwork. Le métamodèle y est construit, de là on en crée un paradigme, une base pour modeler son réseau.

3.2 Métamodèle

Sous GME, le métamodèle de ces réseaux ad hoc se définit comme tel :
(Voir Annexes pour une vue complète du métamodèle)

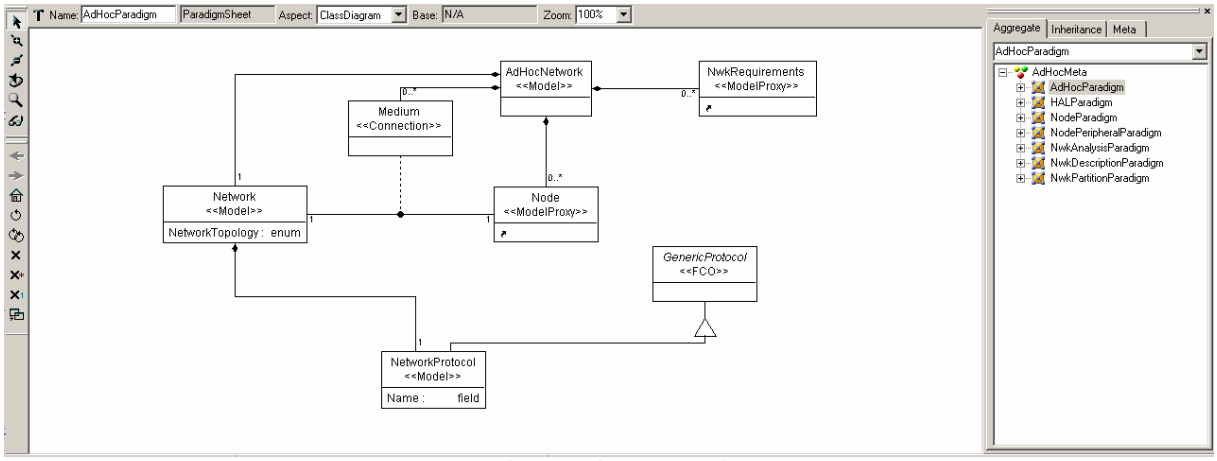


Figure 28 : Métamodèle d'un réseau ad hoc

Il s'agit d'un réseau, contenu dans un environnement, qui peut posséder des descriptions, des prérequis et divers nœuds, ces nœuds étant eux-mêmes soumis au protocole du réseau en question.

Il est important de noter que la notation utilisée pour la création du métamodèle est un sous ensemble de UML.

3.3 Modèle

Voici la représentation du modèle :

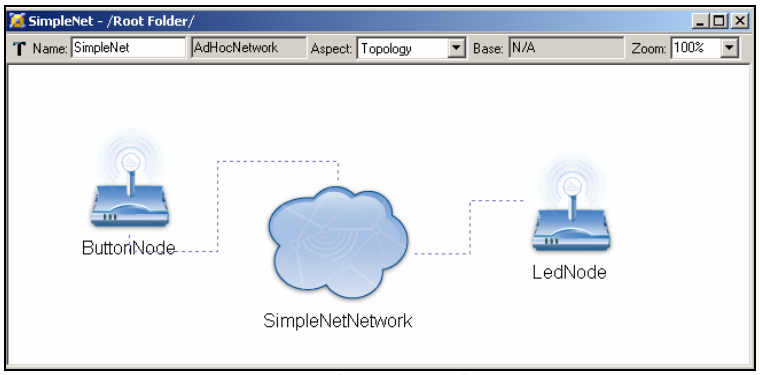


Figure 29 : Modèle d'un réseau ad hoc

Il s'agit d'un système très simple composé d'un bouton et d'une led reliés l'un à l'autre à travers un réseau ad hoc. En pratique, on peut imaginer deux cartes, comportant chacune un bouton et une led. Le bouton de la première carte doit pouvoir commander la led de la seconde carte, et vice versa.

3.3.1 Topologie

Si nous reprenons le métamodèle, ces deux éléments, bouton et led, représentent chacun un nœud. En rapprochant la vue sur un de ceux-ci, il est possible de distinguer ce qui le compose. On y retrouve l'entrée (ou sortie dans le cas de la led) périphérique (1), la couche application (2), la couche réseau (3) et le périphérique de communication (4).

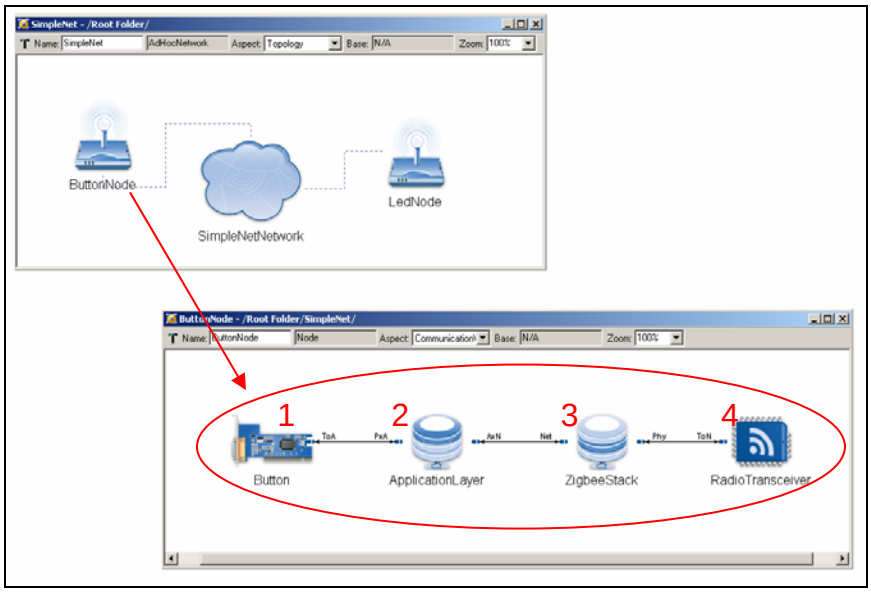


Figure 30 : Topologie d'un noeud

Ceci représente le côté « Topologie » de notre modèle.

3.3.2 Analyse

Du côté de l'analyse, nous pouvons distinguer une autre approche scindée en deux parties.

La première représente les possibilités que possède l'utilisateur avec les entrées de la carte, comme d'allumer, éteindre ou appuyer sur le bouton

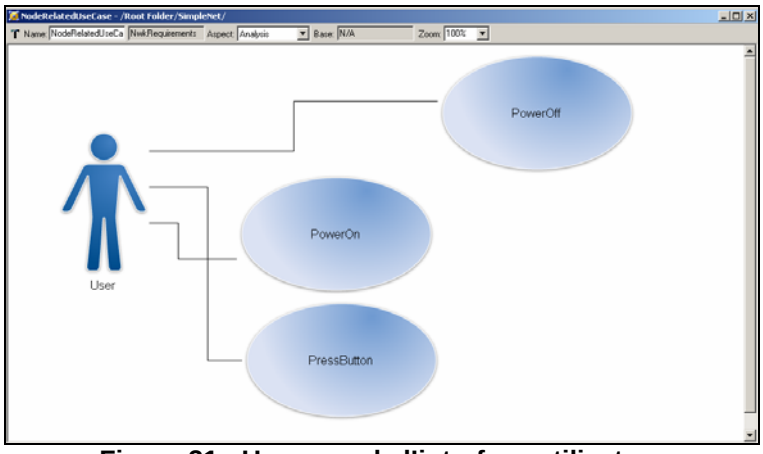


Figure 31 : Use case de l'interface utilisateur

et la seconde les cas d'utilisation liés aux périphériques directement.

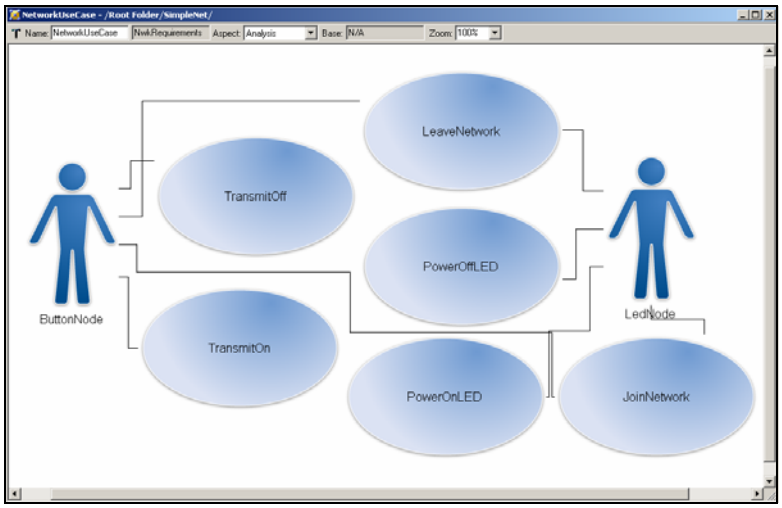


Figure 32 : Use case des périphériques

3.3.3 Profiles

Une troisième approche nous permet de découvrir l'instauration de profiles pour les services.

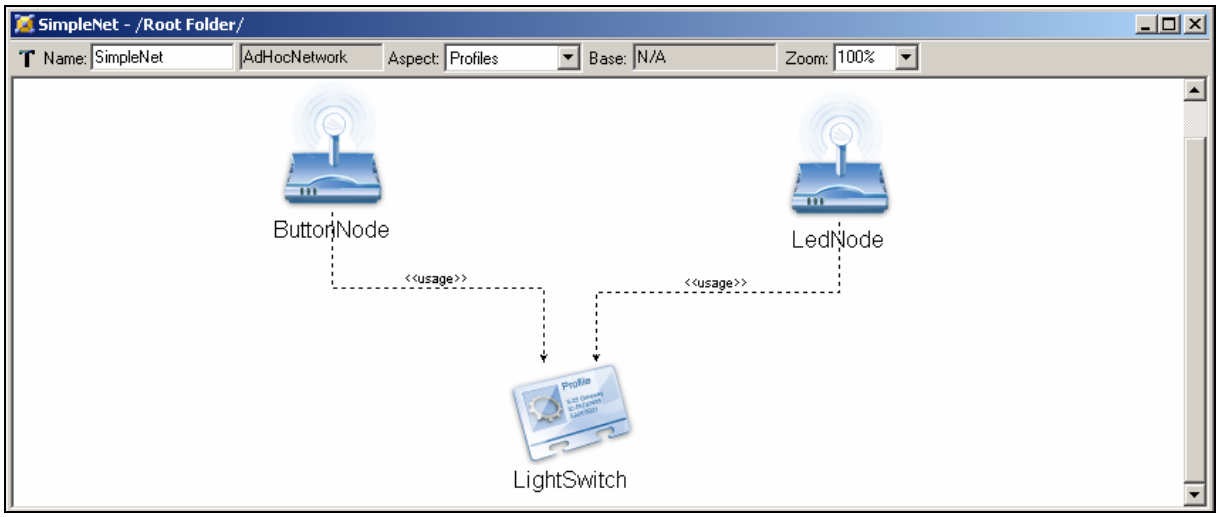


Figure 33 : Profiles

Le profile « LightSwitch » dans le modèle contient un service appelé « switch ». C'est le service qui permet d'allumer ou d'éteindre la led à l'aide du bouton.

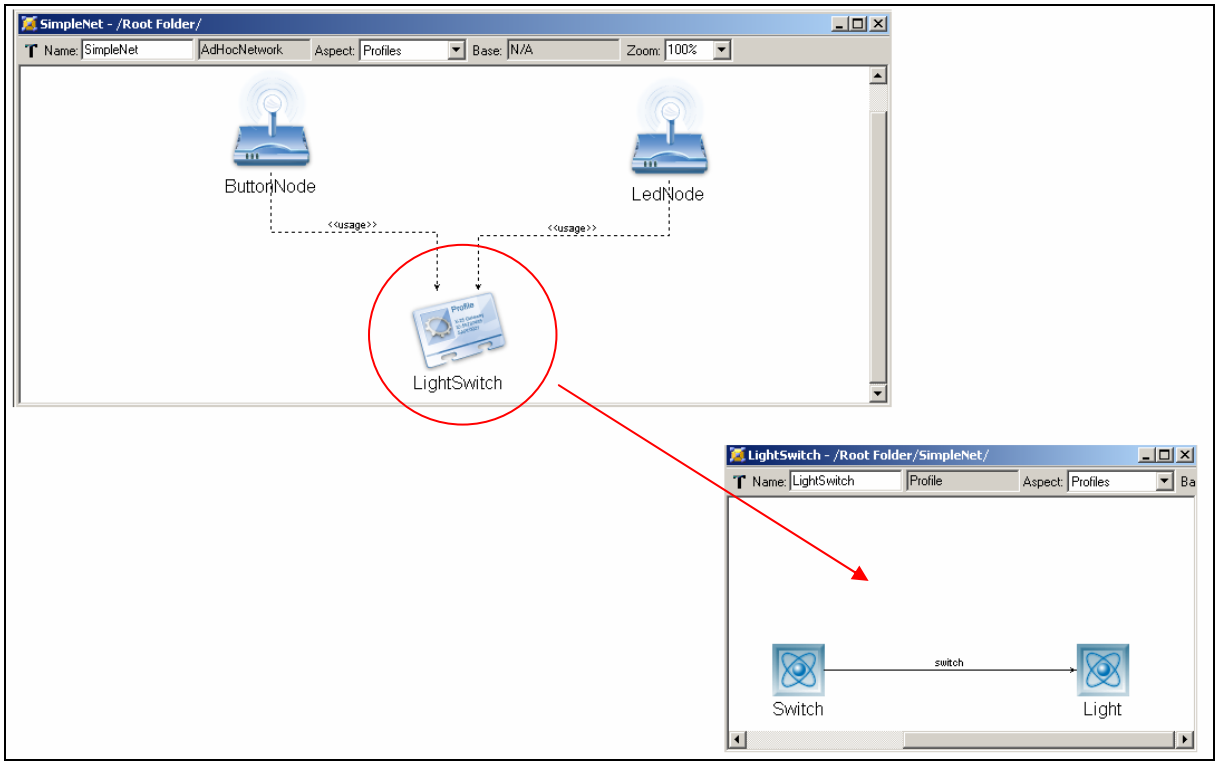


Figure 34 : Service

Voilà les 3 vues que l'ont peut distinguer à la racine du modèle.

3.3.4 Model Viewer

Ensuite, ce modèle créé sous GME s'exporte dans un fichier XML. Il possède des balises qui sont conformes au métamodèle Ad Hoc. La tâche de ce travail de diplôme est de récupérer ce modèle sous MDWorkbench, et de le transformer en modèle UML 2.1.

Pour cela, un plugin faisant office de lecteur de modèle GME est importé dans les ressources de MDWorkbench.

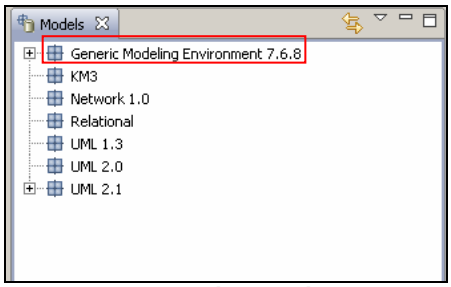


Figure 35 : Métamodèle GME

Lors de l'ouverture du modèle, nous pouvons voir son arborescence qui affiche les types de données liées au métamodèle AdHocNetwork :

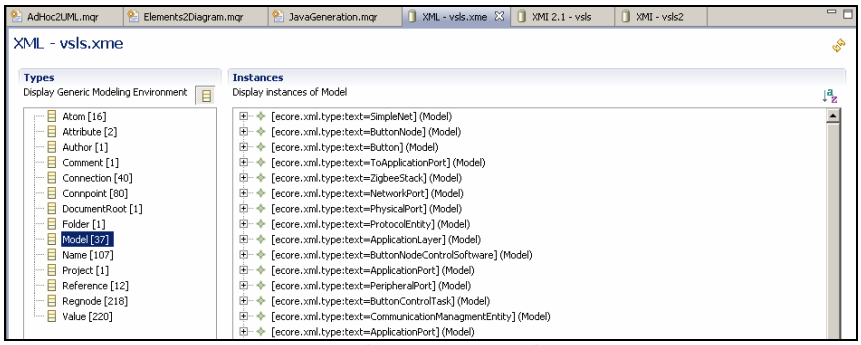


Figure 36 : Modèle GME du réseau ad hoc

Et l'objectif du travail et de le transformer au format XMI 2.1 (UML), tel que celui-ci, où l'on y retrouve les types spécifiques à UML :

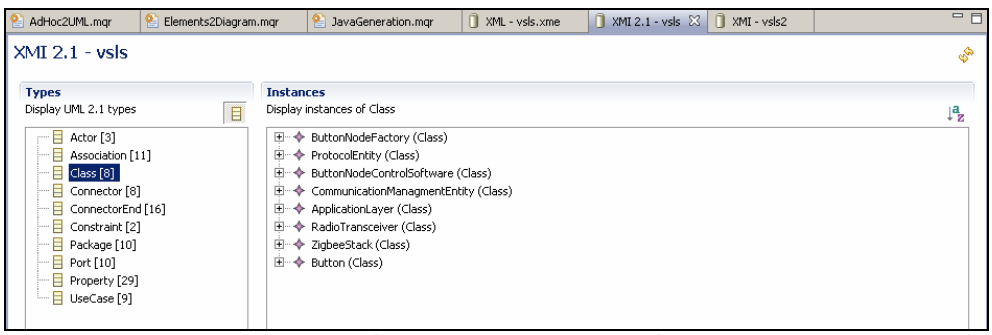


Figure 37 : Modèle UML (XMI 2.1) du réseau ad hoc

4 Transformation

4.1 Introduction

Nous avons découvert le modèle GME sous différents aspects, il est temps de reprendre tout ce qui le compose et de le transformer en format UML 2.1 afin d'être reconnu par Rhapsody, le programme de modélisation UML et génération de code de Telelogic.

L'outil qui permet l'import est « XMI Tool » de Sodius, un importateur/exportateur de fichier XMI. C'est lui qui va lire le modèle UML pour que Rhapsody puisse le reconnaître en tant que tel.

Tout au long de cette transformation, des exemples visuels seront montrés afin de se faire une idée très concrète de ce qui est expliqué.

4.2 Remarques concernant les exemples de code

4.2.1 Remarques générales

Certains exemples de code feront leur apparition, il est nécessaire de définir quelques points à leur sujet :

Les exemples montrés indiquent la manière la plus générale de créer des éléments pour le modèle UML. Il ne s'agit pas du code du projet, qui lui est disponible en annexe. Ces exemples ont pour but de donner une marche à suivre quant à la manière de créer des éléments UML à partir des éléments GME.

Le paramètre *layer* du type « gme.Model » est pris à titre indicatif comme une couche quelconque du modèle. Une couche peut être le périphérique, l'application, le réseau, la couche principale, une sous-couche de l'application, etc…

Les recherches d'éléments du modèle GME se font à l'aide de la routine « foreach » (« pour chaque… ») et chaque élément trouvé peut être filtré selon son « Kind » (~type, sorte) par exemple.

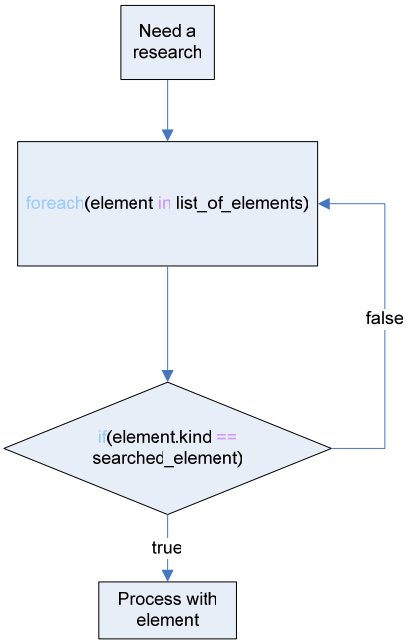


Figure 38 : Routine foreach…if

4.2.2 Transient links

Dans MDWorkbench, il existe des « transient links ». Il s'agit de liens créés et détruits dans la mémoire lors de l'exécution du code uniquement, servant à lier deux objets quelconques entre eux. La plupart du temps un élément du modèle GME est lié à son correspondant UML, ce qui permet de simplifier grandement certaines recherches d'éléments répétitives par la suite.

Ces liens se repèrent facilement, à cause de l'utilisation du caractère '#' dans la ligne de code.

Un exemple simple serait la liaison entre l'Id du bouton GME et sa classe en UML :

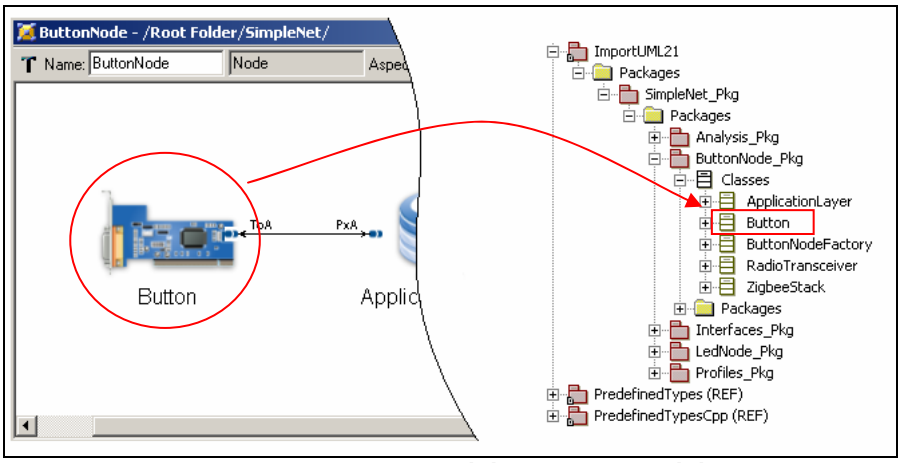


Figure 39 : Transient link entre élément GME et élément UML

Le code sera :

```
var button : gme.Model ;
var classButton : uml21.Class ;

button.getId()#classReference.add(classButton) ;
```

Code 6 : Création d'un transient link

Et pour récupérer cette classe, il suffira de faire une recherche de l'élément choisi, de récupérer son ID et d'appeler la méthode « first() » sur le *transient link* :

```
var newClass: uml21.Class;

newClass = button.getId()#classReference.first();
```

Code 7 : Utilisation d'un transient link

En prenant l'Id, qui est unique pour chaque élément du modèle, on s'assure de lier la classe au bon élément GME.

4.2.3 Scripts

Il est également possible de créer des scripts, rendant l'utilisation et la lecture du code moins pénible.

Les scripts sont des petits bouts de code créés à part et s'appliquant sur un type particulier (sur un package, une classe, un port,…).

Prenons comme exemple un package. Pour le type uml21.Package (« com.sodius.mdw.metamodel.uml21.Package ») il existe toute une liste de méthodes prédéfinies dans la documentation de MDWorkbench, sur la référence du métamodèle UML21.

Cependant, si l'utilisateur cherche à récupérer toutes les interfaces d'un package, aucune méthode n'a été faite pour. Alors il devient pratique d'écrire un petit script que l'on appellera « getInterfaces() » qui retournera une liste des interfaces contenues dans ce package.

En code, voilà comment le script peut être représenté :

```
import com.sodius.mdw.core.model.DefaultMDWList;

metatype uml21.Package;

/*StaticMDWList extends DefaultMDWList to create a new MDWList*/
public class StaticMDWList extends DefaultMDWList
{
   public static DefaultMDWList create()
   {
      return new DefaultMDWList();
   }

}

/*Start of script, based on uml21.Package*/
public script getInterfaces(): MDWList
{
   //Create the List
   var interfacesList: MDWList = StaticMDWList.create();

   //Search all elements of package...
   foreach(interface: uml21.Element in self.getMembers())
   {
      //...and keep only interfaces.
      if(interface instanceof uml21.Interface)
      {
         interfacesList.add(interface);
      }
   }

   return interfacesList;
}
```

Code 8 : Création d'un script

Grâce à cela, dans le code on pourra obtenir les interfaces d'un package quelconque en appelant simplement :

```
package.getInterfaces();
```

Code 9 : Utilisation d'un script

4.3 Architecture UML

4.3.1 Principes généraux

L'architecture des Packages, Classes, etc, du modèle a été prédéfinie sous Rhapsody. Au sommet de l'arborescence se trouvent les Packages principaux du modèle, soit un pour l'analyse (contenant les UseCases), un par nœud, bouton et led, un pour les interfaces et le dernier pour les profiles.

Cette architecture est encore au niveau prototype du « Node-configurator » qui est un outil de conception, ou *wizard*, automatisé assisté par ordinateur. Dans l'avenir, cet outil devrait permettre à l'utilisateur de choisir quel type d'architecture, parmi un choix plus large, est nécessaire pour son système.

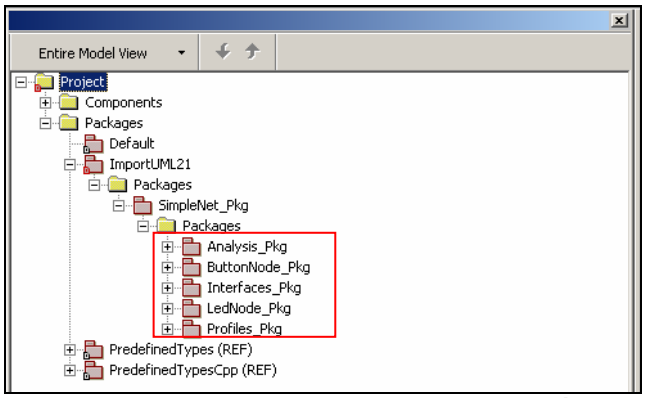


Figure 40 : Principaux packages du modèle

Création d'un package

```
public ruleset GME2UML(in source: gme,  out target: uml21)
{
   public rule main()
   {
      //create new package in uml model, and set a name
      var newPackage: uml21.Package;

      newPackage = target.create("Package");
      newPackage.name = "Nom du package";
   }
}
```

Code 10 : Création d'un package

4.3.2 Package analyse

Dans le package « Analysis_Pkg »[9] on y retrouve les cas d'utilisation avec leurs éléments comme les Actors, Associations et UseCases.

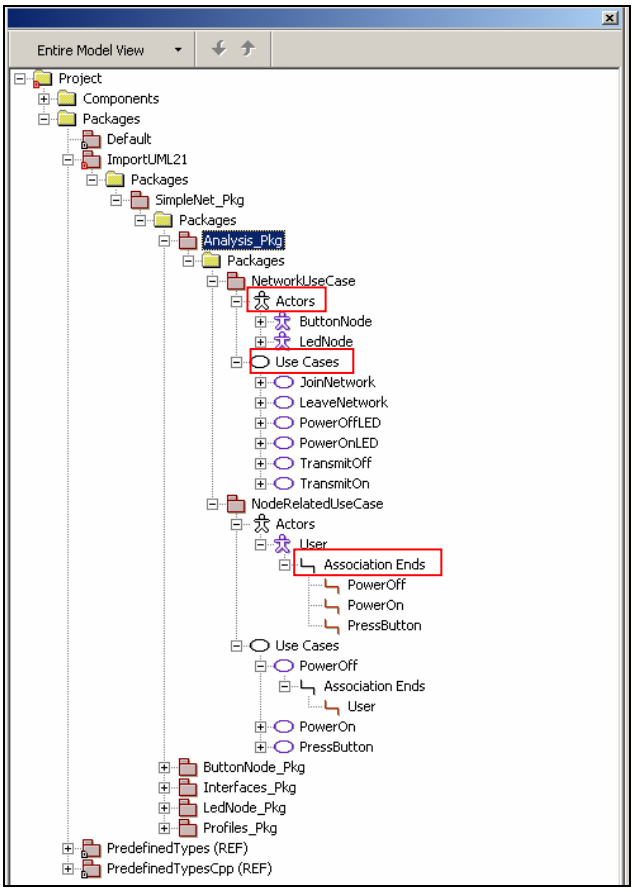


Figure 41 : Acteurs, Cas d'utilisation et Associations

Recherche d'un acteur et création de cet acteur

```
public ruleset GME2UML(in source: gme,  out target: uml21)
{
   public rule main(layer: gme.Model)
   {
      var newActor: uml21.Actor;

      //search each Atom in layer model...
      foreach(actor: gme.Atom in layer.getAtom())
      {  //...with Kind property "Actor"
         if(actor.getKind() == "Actor")
         {
            newActor = target.create("Actor");
            newActor.setName(actor.getName());
            actor.getId()#useCaseReferences.add(newActor);
         }
      }
   }
}
```

Code 11 : Création d'un acteur

[9] Cette partie de la transformation, l'analyse, avait déjà été faite par Rico Steiner, assistant à la HEVS. Elle est reprise ici dans le but d'être expliquée.

Recherche d'un use case et création de ce use case

```
public ruleset GME2UML(in source: gme,  out target: uml21)
{
   public rule main(layer: gme.Model)
   {
      var newUseCase: uml21.UseCase;

      //search each Atom in layer model...
      foreach(useCase: gme.Atom in layer.getAtom())
      {  //...with Kind property "UseCase"
         if(useCase.getKind() == "UseCase")
         {
            newUseCase = target.create("UseCase");
            newUseCase.setName(useCase.getName());
            useCase.getId()#useCaseReferences.add(newUseCase);
         }
      }
   }
}
```

Code 12 : Création d'un use case

Recherche d'une association et création de cette association

Pour créer une association, il faut en premier lieu déterminer les deux extrémités de la connexion (1), qui sont la source et la destination. De là, l'objet « Association » est créé (2) et les deux extrémités lui sont ajoutées (3). Il est intéressant de voir que premièrement ce sont les éléments GME source et destination qui sont cherchés (1) et qu'ensuite, par les transient links, leur correspondant UML (2) sont pris comme source et destination de la connexion.

```
public ruleset GME2UML(in source: gme,  out target: uml21)
{
   public rule main(layer: gme.Model)
   {
      var newAssoc: uml21.Association;
      var srcID: String = "";
      var dstID: String = "";

      foreach(connection: gme.Connection in layer.getConnection())
      {
         srcID = "";
         dstID = "";

         foreach(connTarget: gme.Connpoint in connection.getConnpoint())
         {
            if(connTarget.getRole() == "src")
               srcID = connTarget.getTarget();
            else if(connTarget.getRole() == "dst")
               dstID = connTarget.getTarget();
         }

         if(connection.getKind() == "AnalysisAssociation")
            @addAssociations(srcID, dstID);
      }
   }

   private rule addAssociations(srcID: String, dstID: String)
   {
      var association: uml21.Association;
      var srcObj: uml21.Type = srcID#useCaseReferences.first();
      var dstObj: uml21.Type = dstID#useCaseReferences.first();

      association = target.create("Association");
      association.createNavigableOwnedEnd(srcObj.getName(), srcObj);
      association.createNavigableOwnedEnd(dstObj.getName(), dstObj);
   }
}
```

1 1 (GME)

2 2 (UML)

3

Code 13 : Création d'une association

Ces exemples ci-dessus ont été repris tels quels des annexes, afin de représenter concrètement la recherche d'éléments dans le modèle et de les lier par les transient links afin de les retrouver lors de la création d'association entre eux.

Les prochains exemples de code suivent la consigne expliquée dans les principes généraux, c'est-à-dire qu'ils ne sont qu'à but informatif de comment créer les éléments UML.

4.3.3 Packages des noeuds

Dans les packages « ButtonNode_Pkg » et « LedNode_Pkg, chaque nœud y est développé avec toute sa configuration selon le métamodèle AdHocNetwork. En regardant plus en détail un de ces nœuds, le bouton par exemple, on y distingue les couches principales qui le composent, comme cité précédemment. Chacun de ces couches est représentée par une classe dans le modèle UML.

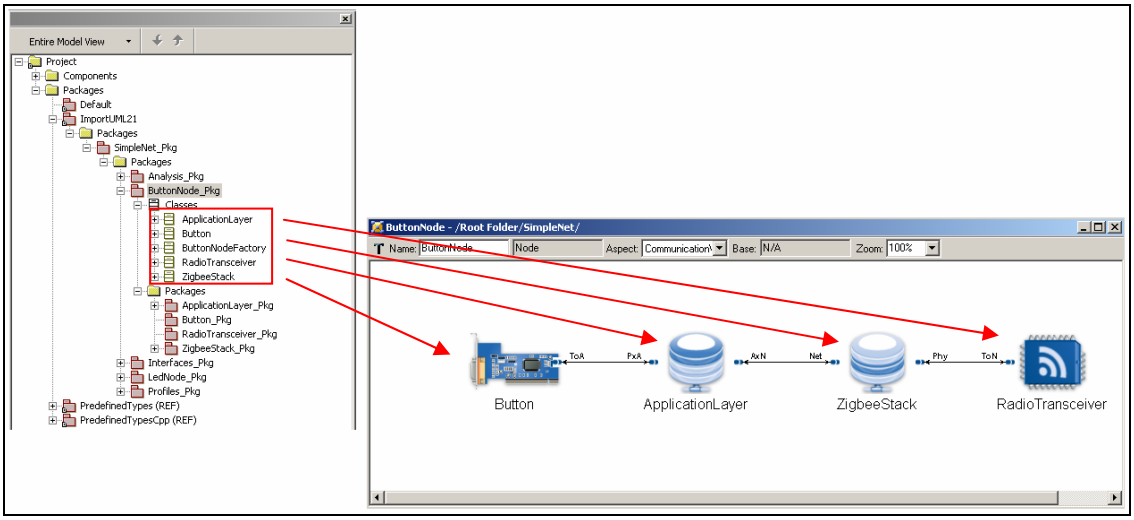


Figure 42 : Couche principale d'un noeud

Une cinquième classe est apparente, « ButtonFactory ». Il s'agit de la classe *factory* comportant les objets composites des autres classes.

Création d'une classe

```
public ruleset GME2UML(in source: gme,  out target: uml21)
{
   public rule main(layer: gme.Model)
   {
      var newPackage: uml21.Package;
      var newClass: uml21.Class;

      //a Class is created within a package
      //parameters are: createOwnedClass(String name, boolean isAbstract)
      newPackage = target.create("Package");
      newClass = newPackage.createOwnedClass("NewClass", false);
   }
}
```

Code 14 : Création d'une classe

Les packages *Interfaces* et *Profiles* seront expliqués plus loin, dans le chapitre « 4.6 Transformation intelligente ».

4.4 Couches principales

Observons maintenant de plus près ces couches.

4.4.1 Couche périphérique

La couche périphérique ne comporte que le port « PxA » (Peripheral to Application) symbolisé par l'élément « ToApplicationPort ».

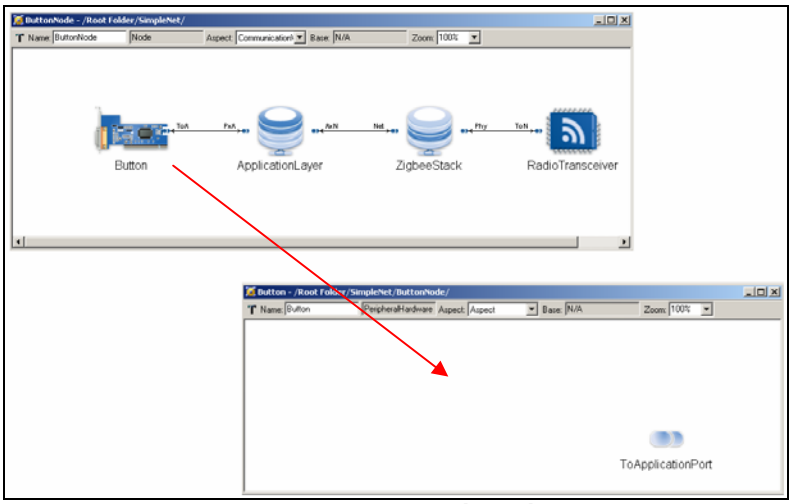


Figure 43 : Couche périphérique

La création d'un port est détaillée dans la partie réseau (4.4.3 Couche Réseau).

4.4.2 Couche application

La couche application est plus intéressante, c'est elle le centre névralgique du système. On y retrouve des ports, « PxA » relié au bouton, et « AxN » qui pointe vers la couche réseau. Entre ces ports se trouve toute la logique de l'application, notamment par le centre de contrôle du bouton « ButtonNodeControlSoftware ».

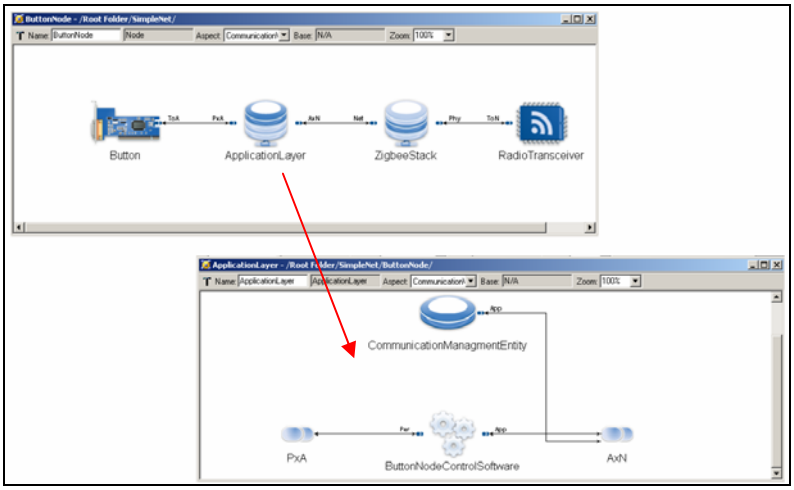


Figure 44 : Couche application

Il n'est pas nécessaire de regarder ce que contiennent les deux modèles « CommunicationEntityManager » et « ButtonNodeControlSoftware ». Il n'y a que les ports visibles sur l'image ci-dessus, du même principe que le port de la couche périphérique.

4.4.3 Couche réseau

La couche réseau, ZigbeeStack, comporte le protocole qui va trier les informations entrantes et sortantes, ainsi que les deux ports créant cette liaison.

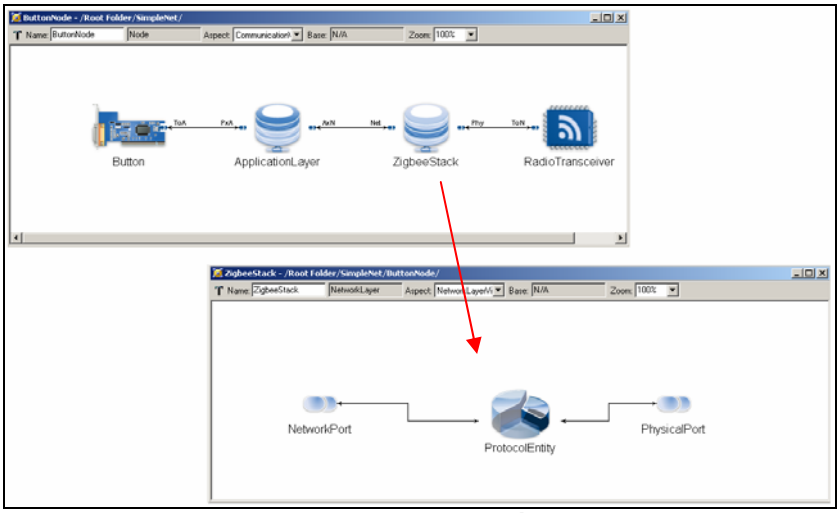


Figure 45 : Couche réseau

Création d'un port et d'interfaces

La manière simple de créer un port ressemble à ceci :

```
public ruleset GME2UML(in source: gme,  out target: uml21)
{
   public rule main(layer: gme.Model)
   {
      var newPackage: uml21.Package;
      var newClass: uml21.Class;
      var newPort: umkl21.Port;

      //a Port is created within a Class
      //parameters are: createOwnedPort(String name, Type type)
      newPackage = target.create("Package");
      newClass = newPackage.createOwnedClass("NewClass", false);
      newPort = newClass.createOwnedPort("NewPort", newClass);
   }
}
```

Code 15 : Création d'un port et d'interfaces

Cependant, la plupart du temps des interfaces composent ces ports. Certaines sont « provided », d'autres « required », cela dépend des requêtes qui sont faites hors ou dans la classe contenant le port.

En UML on distingue facilement la différence par leur forme. Les interfaces « provided » sont représentées par une forme en demi-cercle ou réceptacle, elles indiquent qu'une requête peut être faite de l'environnement (objets extérieurs) vers la classe du port. Celles « required » prennent la forme d'un cercle ou plus communément d'une "sucette", elles indiquent que la requête peut être faite depuis la classe du port vers l'environnement.

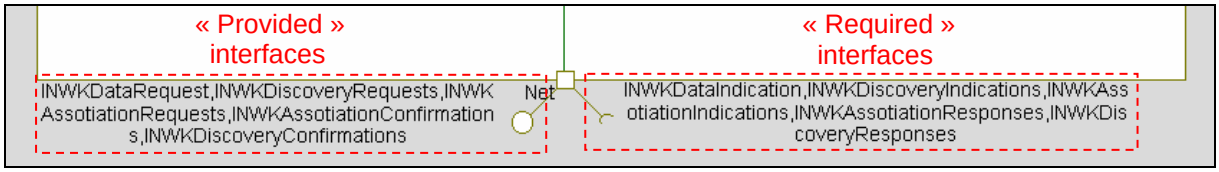


Figure 46 : Interfaces d'un port

Au niveau code, une interface « provided » est du type *uml21.InterfaceRealization* (1) et une interface « required » est du type *uml21.Usage* (2).

Le dernier point à respecter est que lorsque l'on crée le port, on lui assigne un type qui est généralement la classe qui le contient. En faisant cela, les interfaces que l'on va lui ajouter vont se répéter à tous les autres ports de la même classe. Alors il est plus convenable de créer une « ContractClass » du port désiré, et c'est cette « ContractClass » qu'il prendra comme type (3).

Enfin, il est nécessaire d'ajouter une ligne indiquant où placer le « Usage », pour qu'il se situe au même endroit que les « InterfaceRealization » (4).

```
public ruleset GME2UML(in source: gme,  out target: uml21)
{
   public rule main(layer: gme.Model)
   {

      var newPackage: uml21.Package = target.create("Package");
      var newClass: uml21.Class;
      var contractClass: uml21.Class;
      var newPort: umkl21.Port;

      //create Port with type contractClass
      newClass = newPackage.createOwnedClass("NewClass", false);
      contractClass = newClass.createNestedClassifier("ContractClassOfNewPort");
      newPort = newClass.createOwnedPort("NewPort", contractClass);
                                                                    3
      //create Interfaces
      var interfaceProvided: uml21.Interface;
      var interfaceRequired: uml21.Interface;
      interfaceProvided = newPackage.createOwnedInterface("InterfaceProvided");
      interfaceRequired = newPackage.createOwnedInterface("InterfaceRequired");

      //interface provided
      var realization: uml21.InterfaceRealization;
      realization = target.create("InterfaceRealization");
      realization.setContract(interfaceProvided);                  1
      realization.setImplementingClassifier(contractClass);

      //interface required
      var usage: uml21.Usage;
      usage = target.create("Usage");              2
      usage.supplier.add(interfaceRequired);
      usage.client.add(interfaceRequired);

      //add Usage to package of Port, to list dependencies in the contract Class
      newPackage.packagedElement.add(usage);
                                                   4
   }
}
```

Code 16 : Ajout des interfaces sur un port

Dans Rhapsody, on retrouvera le port sous cette forme :

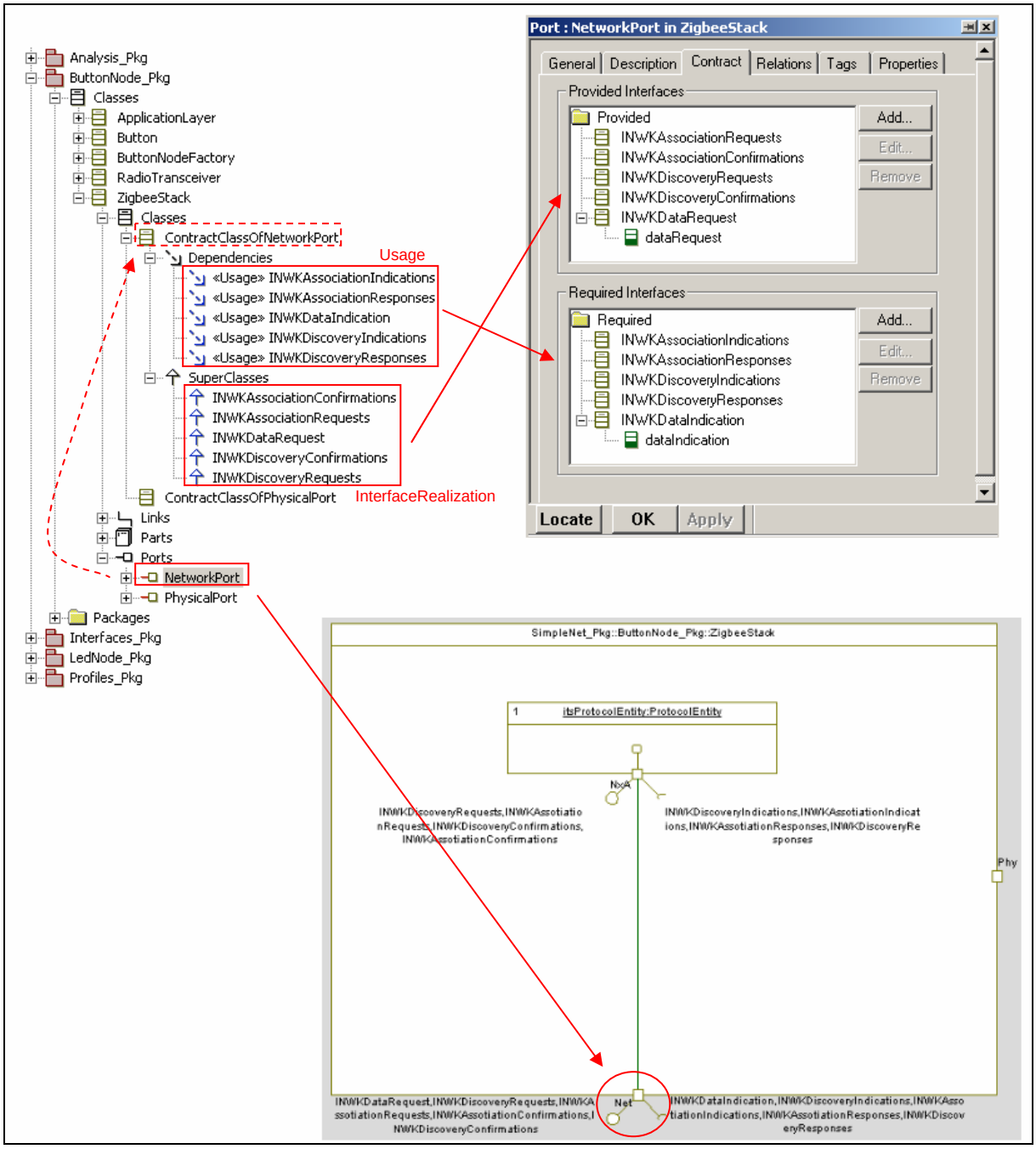


Figure 47 : Vue d'un port et de ses interfaces « provided » et « required »

4.4.4 Couche hardware

Pour finir, la couche hardware, contenant le port « ToNetwork », comparable à la couche périphérique, puisqu'à elles deux, elles représentent les extrémités de la chaîne de travail.

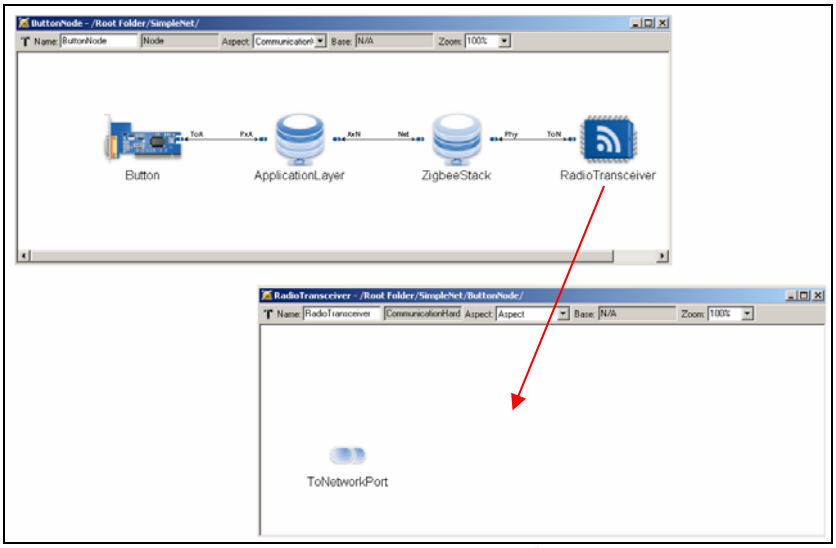


Figure 48 : Couche matérielle

4.4.5 Packages complémentaires

Comme il a été expliqué précédemment, chacune de ces parties est représentée par une classe dans le modèle UML, avec une cinquième classe *factory* regroupant les objets composés par les quatre autres classes du modèle. Cependant, les deux couches centrales, respectivement l'application et le réseau, possèdent des éléments internes différents des simples ports.

Afin de représenter ces éléments par des classes, des packages ont été créés dans ce but précis dont voici la preuve par l'image :

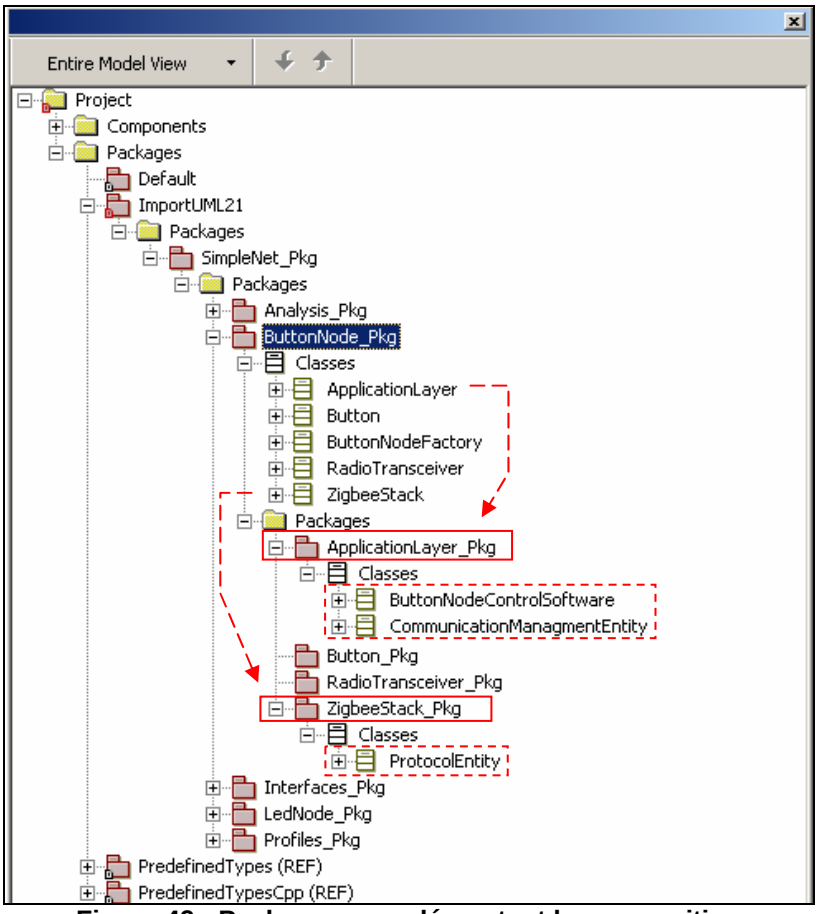


Figure 49 : Packages complémentant la composition des classes de la couche principale

Les packages sont là pour compléter en quelque sorte les classes dont ils dépendent.

Voici l'exemple de ce que contient le package de la couche application :

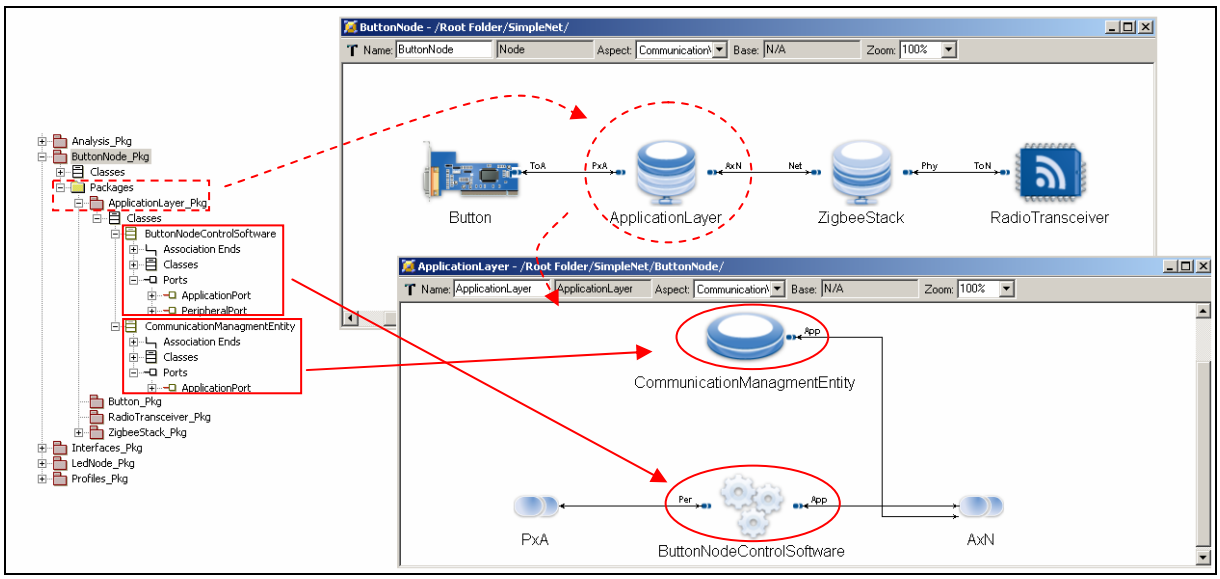


Figure 50 : Contenu du package de la couche application

4.5 Connexions

Lorsque de l'information transite entre deux objets, ou qu'un appel de fonction d'une classe est faite par une autre, il est nécessaire d'établir des connexions entre eux.

Observons les canaux de transmissions principaux, ceux reliant les couches mères.

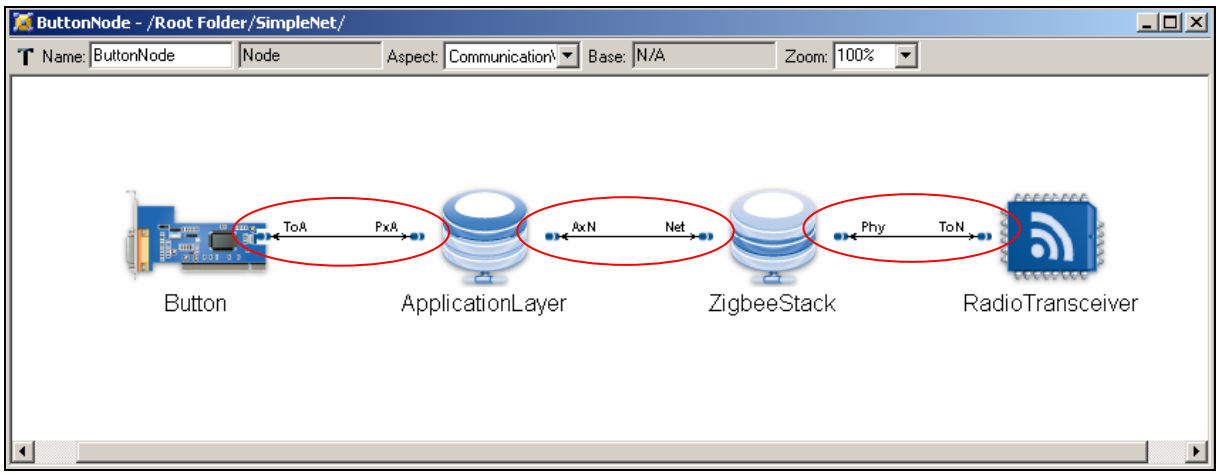


Figure 51 : Connexions dans la couche principale

Ces liens vont se retrouver sur le modèle UML, sous la forme d'un diagramme d'objets, tel que celui-ci :

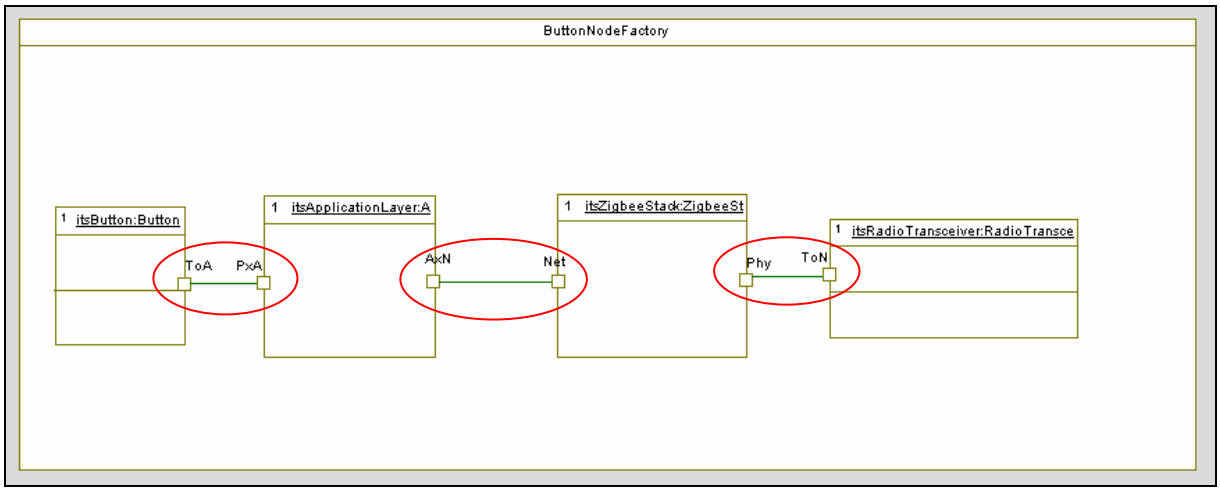


Figure 52 : Liens des objets de la couche principale

Et on remarque que la classe contenant ces objets et liens est bien la classe « ButtonNodeFactory », créée dans ce but là.

La création des connexions se fait sur le même principe que les associations entre *Acteurs* et *UseCases*. Mais avant la connexion, voyons comment créer des objets, afin de les interconnecter :

Création d'un objet

```
public ruleset GME2UML(in source: gme,  out target: uml21)
{
   public rule main(layer: gme.Model)
   {
      var newClass: uml21.Class;
      var compositeObj: uml21.Property;

      compositeObj = self.createOwnedAttribute("NewCompositeObject", newClass);
      compositeObj.setIsComposite(true);
   }
}
```

Code 17 : Création d'un objet

Création d'une connexion

```
public ruleset GME2UML(in source: gme,  out target: uml21)
{
   public rule main()
   {
      var newPackage: uml21.Package = target.create("Package");
      var newFactory: uml21.Class = newPackage.createOwnedClass("Factory", false);
      var newClass1: uml21.Class = newPackage.createOwnedClass("Class1", false);
      var newClass2: uml21.Class = newPackage.createOwnedClass("Class2", false);

      //create Objects
      var compositeObj1: uml21.Property =
                              newFactory.createOwnedAttribute("CompObject1",newClass1);
      var compositeObj2: uml21.Property =
                              newFactory.createOwnedAttribute("CompObject2",newClass2);
      compositeObj1.setIsComposite(true);
      compositeObj2.setIsComposite(true);

      //create Connector
      var newConnector: uml21.Connector = target.create("Connector");
      var end1: uml21.ConnectorEnd;
      var end2: uml21.ConnectorEnd;
      var end1Name: String = compositeObj1.name;
      var end2Name: String = compositeObj2.name;

      //add Connector to class Factory and create Ends
      newFactory.ownedConnector.add(newConnector);
      end1 = newConnector.createEnd();
      end2 = newConnector.createEnd();

      //set Ends of Connector
      end1.setRole(compositeObj1);
      end2.setRole(compositeObj2);
      end1.setPartWithPort(compositeObj1);
      end2.setPartWithPort(compositeObj2);

      newConnector.setName(end1Name+"_"+end2Name);
   }
}
```

Code 18 : Création d'une connexion

La représentation d'un lien d'objets en UML est symbolise par un trait vert, et généralement on lui donne le nom des deux objets reliés.

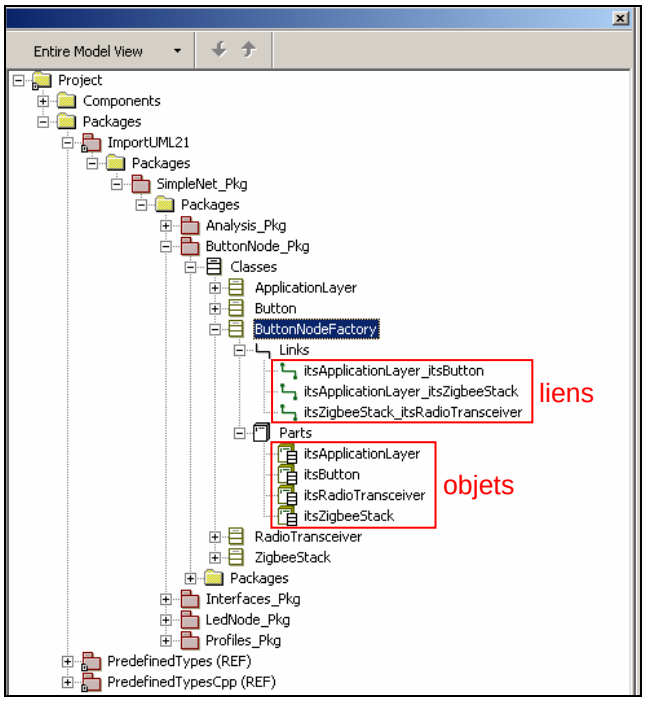


Figure 53 : Liens et objets

Le même principe est appliqué dans les sous-couches contenant elles aussi des connexions, comme la couche application ou la couche réseau.

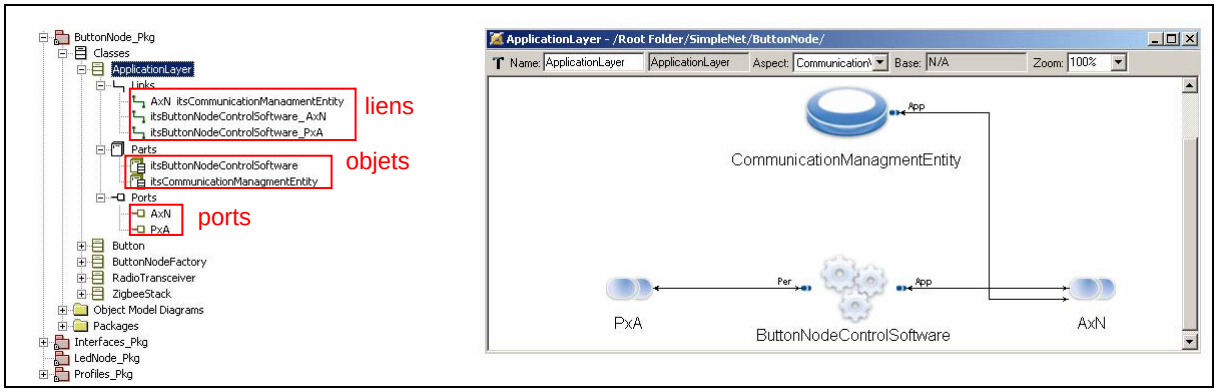


Figure 54 : Liens dans la couche application

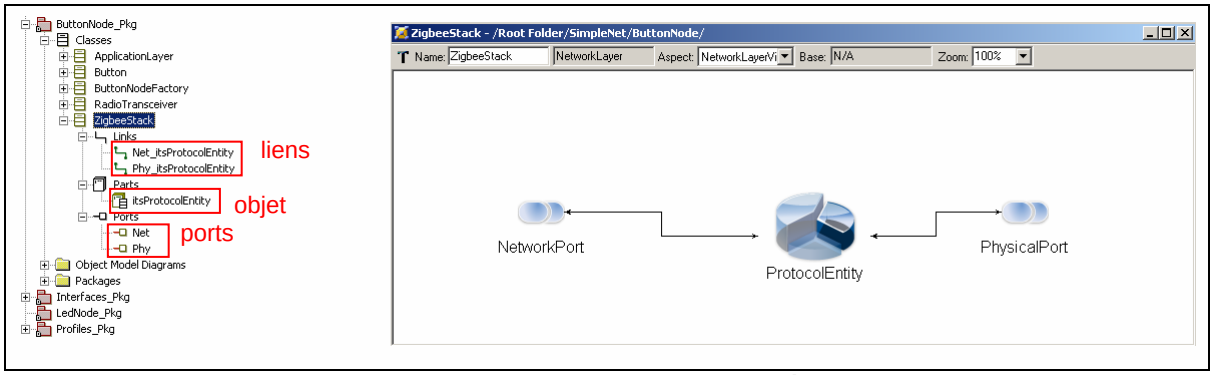


Figure 55 : Liens dans la couche réseau

4.6 Transformation intelligente

Voici l'idée générale de la transformation GME vers UML. Toutefois cela ne suffit pas, en plus de transformer les éléments existants il est indispensable d'ajouter d'autres éléments qui ne sont pas visibles dans le modèle. C'est une sorte d'intelligence que l'on rajoute à la transformation.

Ces éléments nouveaux seront notamment des interfaces pour les ports, des fonctionnalités de base pour les profiles, des blocs de traitement des données entrantes et sortantes dans la couche application,…

4.6.1 Package Profiles

Tout d'abord, il convient de parler du profile « LightSwitch », pour rappel :

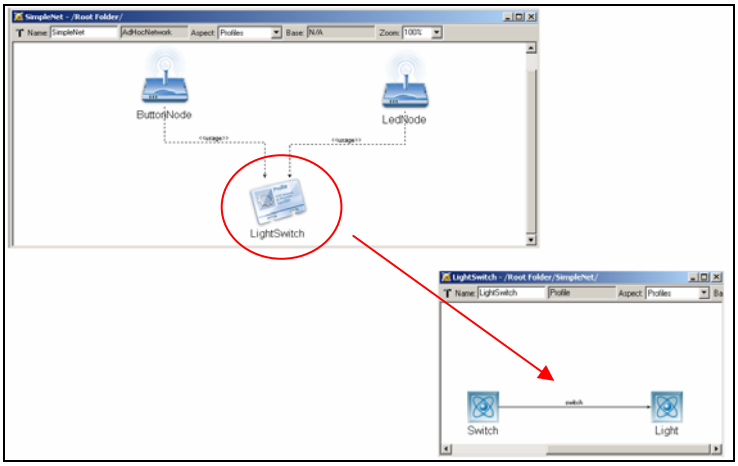


Figure 56 : Profile « LightSwitch »

Pour instaurer des profiles il paraît judicieux de créer une classe de base de laquelle vont dépendre les futurs profiles par polymorphisme. On l'appelle « baseProfile », et celui existant dans notre modèle va en prendre l'héritage.

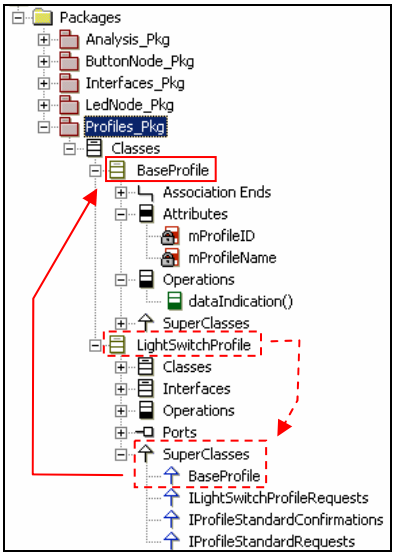


Figure 57 : Profile de base et profile« LightSwitch »

Ce profile de base possède également un ID et un nom, ainsi qu'une opération d'indication de données.

Création de l'héritage

```
public ruleset GME2UML(in source: gme,  out target: uml21)
{
   public rule main(layer: gme.Model)
   {
      var newPackage: uml21.Package = target.create("Package");
      var newInterface: uml21.Interface;
      var newClass: uml21.Class;

      newInteface = newPackage.createOwnedInterface("NewInterface");
      newClass = newPackage.createOwnedClass("NewClass", false);

      //create generalization
      newClass.createGeneralization(newInterface);
   }
}
```

Code 19 : Création de l'héritage

4.6.2 Package Interfaces

De plus, des interfaces font leur apparition.

Tout d'abord les interfaces du profile « LightSwitch » pour le service « switch ». Si le service est confirmé, il possèdera les quatre méthodes, du *request* à la *confirmation*. En étant non confirmé il possède que la *request* et *l'indication*.

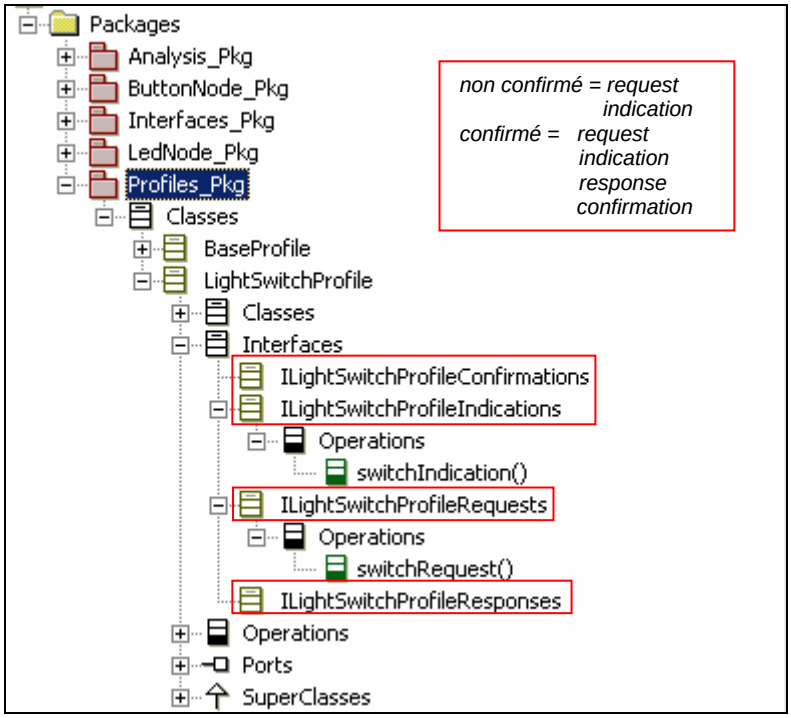


Figure 58 : Interfaces du profile « LightSwitch »

Ensuite, il existe aussi des interfaces décrivant le protocole du système, visible dans le modèle d'origine :

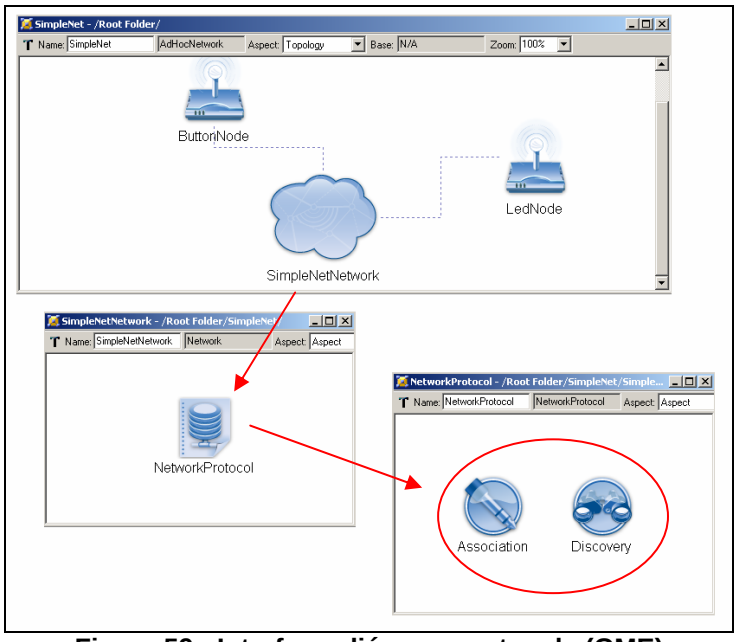


Figure 59 : Interfaces liées au protocole (GME)

Ces deux aspects, « Association » et « Discovery », du protocole vont mener à la création de leur interfaces correspondantes au quatre états habituels, soit *Request, Indication, Response* et *Confirmation*.

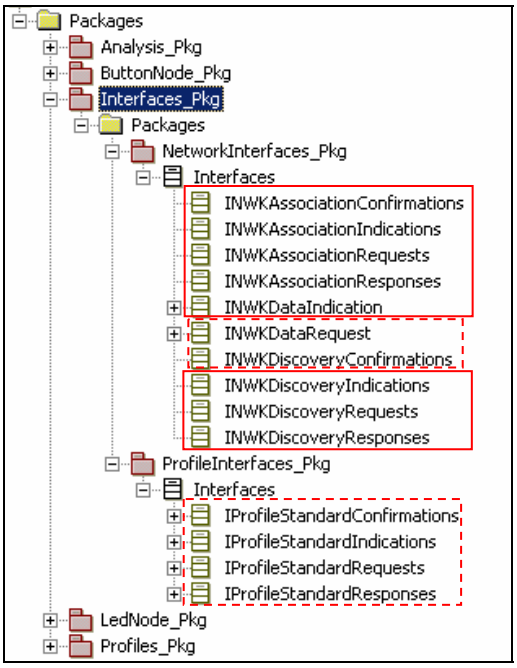


Figure 60 : Interfaces du protocole (UML)

On remarque aussi deux interfaces dans le package « Network_Pkg » network pour la requête et l'indication des données. De plus, un groupe d'interfaces dans « ProfileInterfaces_Pkg » est créé comme base d'utilisation des profiles, d'où leur suffixe « standard ». Elles sont à un niveau général pour le modèle.

Parallèlement à cela, il est nécessaire de revoir la manière dont sont traitées les informations qui transitent dans la couche application.

Imaginons une requête sur le service « switch » est faite. L'information part du périphérique (le bouton) pour atteindre la couche application. Là elle est traité par un nouvel objet, « LightSwitchProfile », qui se trouve être respectivement le profile correspondant au service appelé. De là, elle est envoyée vers un autre objet, le « DataDispatcher » qui se charge de composer les données afin de les envoyer vers la couche réseau.

C'est là qu'on distingue cette "intelligence" de transformation. Pourquoi n'apparaît-elle pas dans le modèle GME ? Car le but du modèle GME est de présenter le système sous une forme plus simple et facile à comprendre dès le premier regard.

Mais rien ne vaut l'explication par l'image. Sur le dessin, on voit qu'il s'agit de la connexion entre « ButtonNodeControlSoftware » et le port « AxN » qui est transformée en UML par le profile *LightSwichProfile* et le *DataDispatcher* :

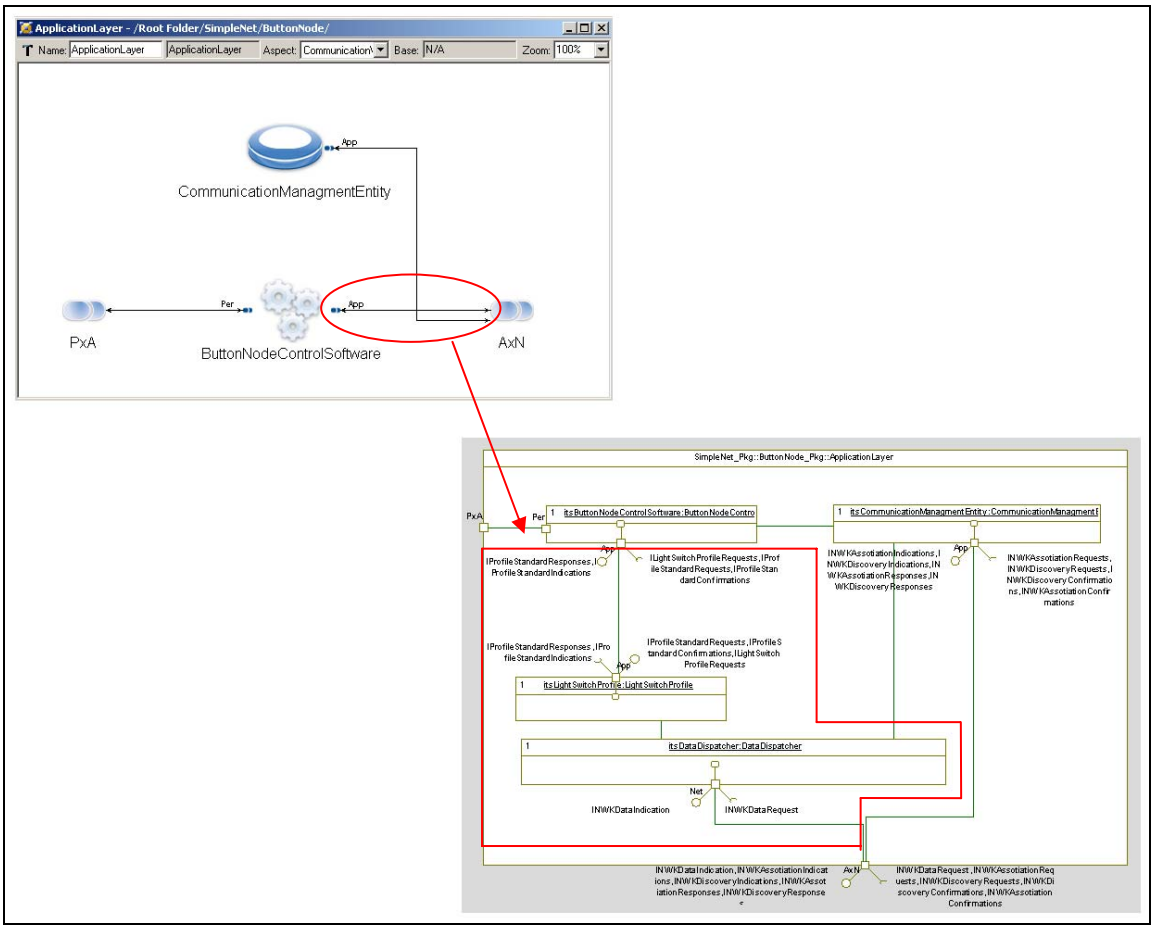


Figure 61 : Vue détaillée en UML d'une simple connexion en GME

Ces éléments sont bien évidemment visibles dans l'arborescence UML :

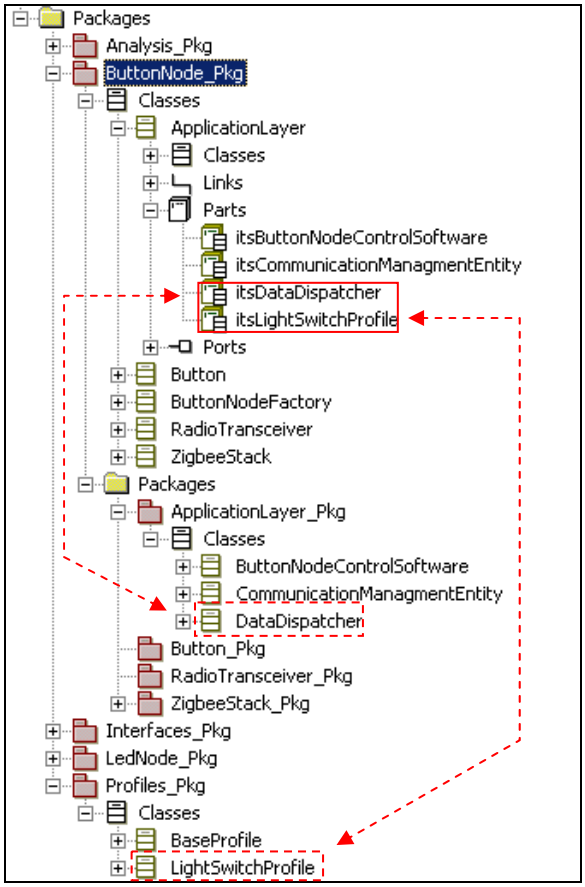


Figure 62 : Eléments ajoutés en UML pour la connexion GME

4.6.3 Associations

Un diagramme de classe est créé conjointement avec la couche application, afin de mettre en place des associations entre les classes utilisées, se référant aux liens entre les objets.

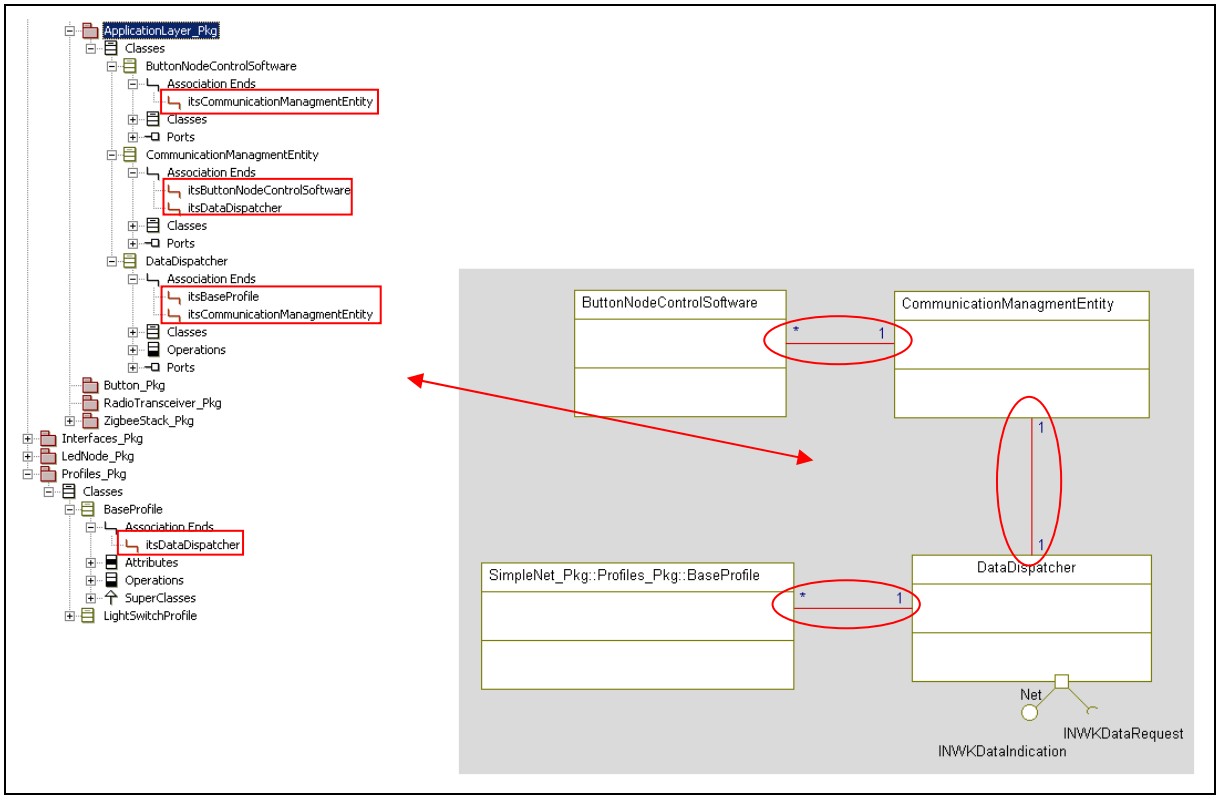


Figure 63 : Associations entre classes

Création d'une association

```
public ruleset GME2UML(in source: gme,  out target: uml21)
{
   public rule main(layer: gme.Model)
   {
      var newPackage: uml21.Package = target.create("Package");
      var newClass1: uml21.Class = newPackage.createOwnedClass("Class1", false);
      var newClass2: uml21.Class = newPackage.createOwnedClass("Class2", false);

      //create Association
      var newAssociation: uml21.Association = target.create("Association");
      var srcName: String = newClass1.name;
      var dstName: String = newClass2.name;

      //set Ends of Association
      newAssociation.createNavigableOwnedEnd(srcName, newClass1);
      newAssociation.createNavigableOwnedEnd(dstName, newClass2);
   }
}
```

Code 20 : Création d'une association

Enfin, le dernier ajout ayant été fait se situe dans la couche réseau. C'est le port de l'entité protocole, n'apparaissant pas dans le modèle, qui est créé lors de la transformation :

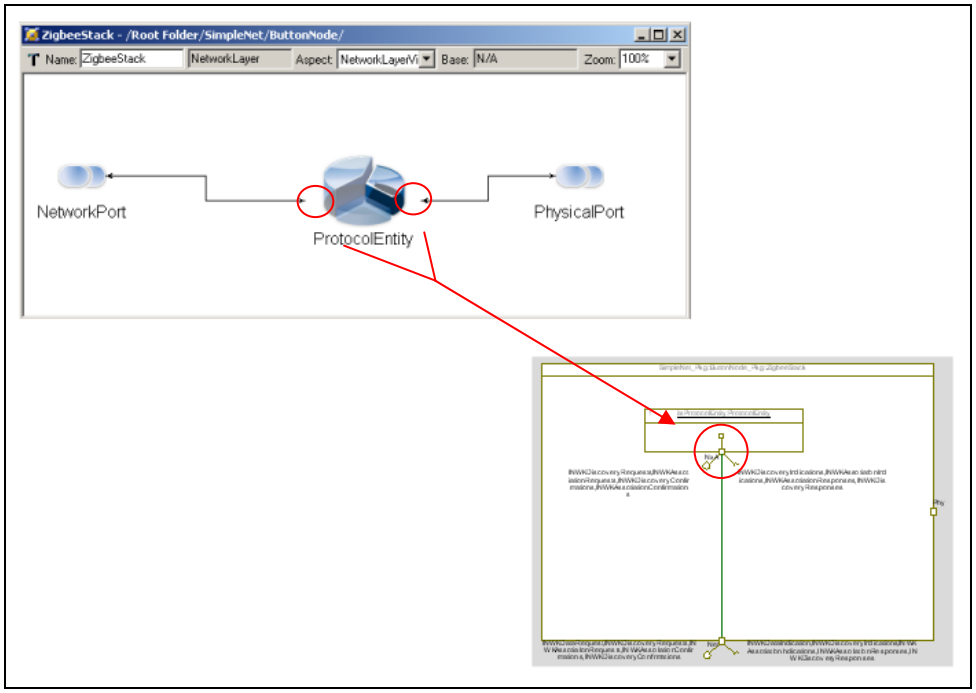


Figure 64 : Ajout d'un port de la couche réseau n'existant pas sous GME

4.7 Diagrammes de classes et d'objets

Dans ce chapitre, nous allons aborder la transformation en diagrammes de classes ou d'objets. Ces diagrammes donnent des indications sur les associations entre les classes ou les liens entre objets, quels ports sont utilisés pour tel lien, etc.

Le programme XMI Importer, utilisé jusqu'à maintenant pour la transformation de modèle, n'est pas configuré pour créer des diagrammes. C'est pourquoi un second outil intervient, « RulesComposer » de Sodius, qui lui permet de réaliser cela.

Il s'exécute depuis Rhapsody, sous Tools/RulesComposer.

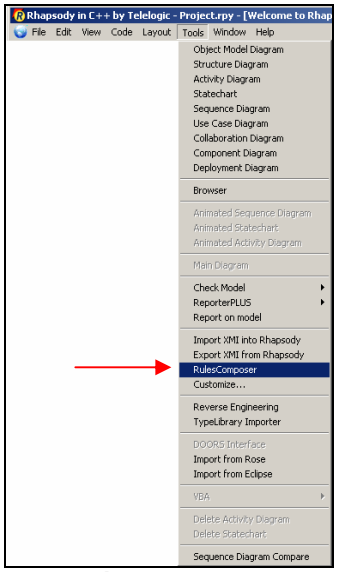


Figure 65 : Exécutable RulesComposer

4.7.1 RulesComposer

Cet outil peut se baser sur une instance de Rhapsody en cours d'exécution, y lire le contenu d'un projet afin d'y opérer des modifications.

La tâche consiste à aller lire les éléments existants de notre modèle UML et d'y récupérer seuls ceux servant aux diagrammes de classes et objets.

Pour cela, un nouveau plug in, spécialement conçu pour travailler avec les modèles UML de Rhapsody, est visible dans la liste des métamodèles.

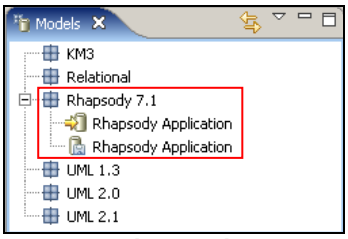


Figure 66 : Métamodèle rhapsody

4.7.2 Diagramme d'objet

Le projet compte trois diagrammes d'objets. Le premier étant celui de la couche principale d'un nœud et les deux suivants composent les couches application et réseau.

Prenons pour rappel celui de la couche principale :

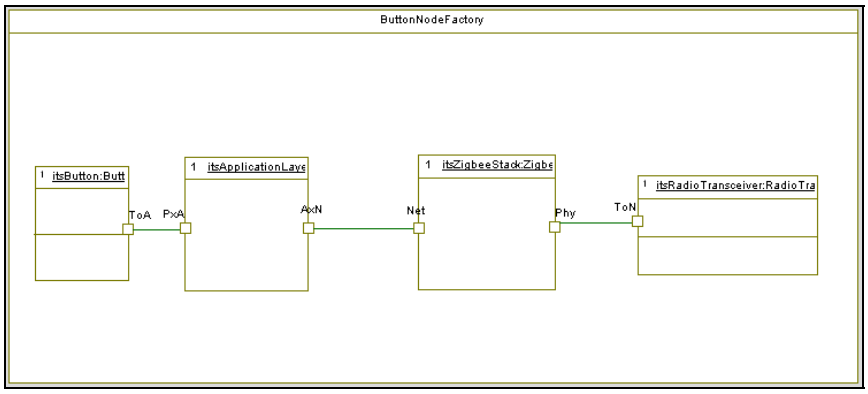


Figure 67 : Diagramme d'objets de la couche principale (fait main)

RulesComposer permet donc de recréer ces diagrammes par le code. Cependant la tâche n'est pas aussi simple. Ajouter un élément relève d'une simplicité plutôt évidente, mais le mettre en forme au niveau graphique est bien plus complexe.

C'est là le problème majeur de cette transformation/création. Dès que l'on veut créer un style particulier, une mise en page précise pour chaque diagramme, il faudrait créer un code spécifique avec énormément de calcul de coordonnées graphiques.

De plus, lorsque les éléments du modèle sont parcourus dans la recherche, ils ne sont pas forcément pris dans l'ordre souhaité pour la mise en place sur le diagramme. Ca relève du casse-tête si l'on cherche à faire quelque chose de "beau" visuellement.

C'est pourquoi l'accent n'a pas été trop fort sur l'aspect présentation du diagramme, ceci n'étant pas une priorité majeure de ce travail de diplôme.

Voyons d'abord comment créer simplement des éléments sur un graphique UML.

Création d'un diagramme

```
public ruleset UML2Diagram(inout rhp: rhapsody)
{
   public rule main()
   {
      var diagram: rhapsody.ObjectModelDiagram;
      var pkg: rhapsody.Package;

      //create diagram and add it in a package
      diagram = rhp.create("ObjectModelDiagram");
      diagram.setName("NewObjectDiagram");
      pkg.objectModelDiagrams.add(diagram);
   }
}
```

Code 21 : Création d'un diagramme UML

Création d'un noeud (classe, objet, …)

```
public ruleset UML2Diagram(inout rhp: rhapsody)
{
   public rule main()
   {
      //create diagram...

      //an object will be added
      var newObject: rhapsody.Instance;

      //GraphNode is the type for all node in diagram
      var graphNode: rhapsody.GraphNode;
      graphNode = diagram.eModel().create("GraphNode");

      //set what is the node ('newObject' here) and add it to diagram
      graphNode.setModelObject(newObject);
      diagram.graphicalElements.add(graphNode);
   }
}
```

Code 22 : Création d'un noeud

Création d'un lien (association entre classes, lien entre objets, …)

```
public ruleset UML2Diagram(inout rhp: rhapsody)
{
   public rule main()
   {
      //create diagram...

      //create two objects and the link between them
      var object1: rhapsody.Instance;
      var object2: rhapsody.Instance;
      var link: rhapsody.Link;

      //GraphEdge is the type for edges in diagram
      var graphEdge: rhapsody.GraphEdge;
      graphEdge = diagram.eModel().create("GraphEdge");

      //set what is the edge ('link' here) and add it to diagram
      graphEdge.setModelObject(link);
      diagram.graphicalElements.add(graphEdge);

      //add ends of edge
      ge.source = object1;
      ge.target = object2;
   }
}
```

Code 23 : Création d'un lien entre objets

Cela reste évidemment toujours des exemples simples. Il faut garder à l'esprit que ces nœuds existent dans le modèle et qu'un lien, par exemple, ne porte pas forcément sur un objet à un autre, mais peut provenir d'un port. Le code réel et complet est toujours disponible en annexe.

Le résultat de ces transformations donne des diagrammes ou les objets sont distinctement visibles, mais le plus complexe relève de la création des liens, qui apparaissent tous superposés, leur position géométrique n'ayant pas été définie. Rhapsody les trace à sa manière.

Voici le diagramme de la couche principale d'un nœud, après transformation avec le RulesComposer :

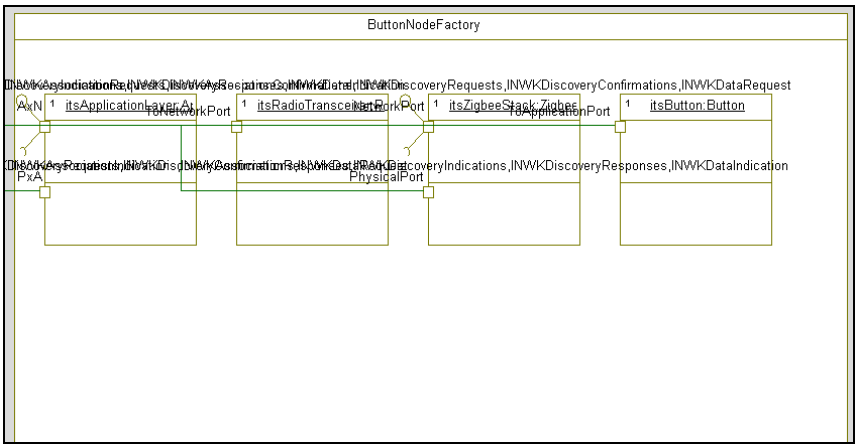


Figure 68 : Diagramme d'objets de la couche principale (généré)

On distingue clairement un brouillon au niveau des liens et des interfaces de ports qui se superposent et rendent le schéma encore loin d'une esthétique voulue.

Le niveau d'importance sur le visuel de ces diagrammes est encore à définir par le chef de projet.

4.7.3 Diagramme de classe

Afin de clarifier chaque couche du modèle nécessitant un diagramme d'objet, un diagramme de classe y est créé conjointement.

Prenons cette fois le schéma de la couche application.

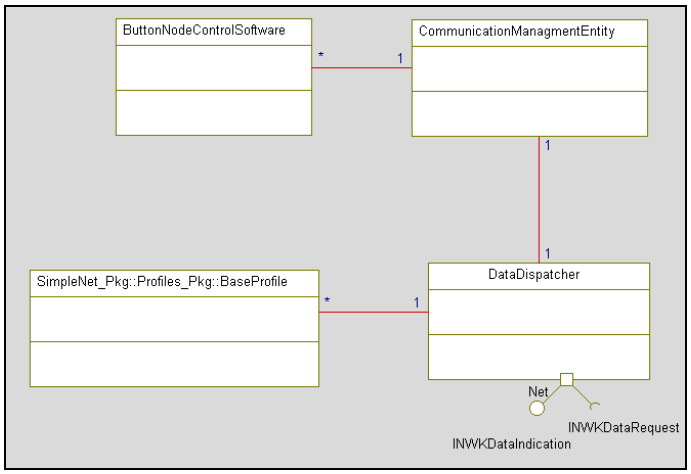


Figure 69 : Diagramme de classes de la couche application (fait main)

Fait à la main, on voit nettement les classes utilisées et leur relation entre elles, les ports que l'on souhaite afficher ou non, etc…

Produit avec RulesComposer, on retrouve le même type de disposition des nœuds avec des relations et des noms d'interfaces superposés :

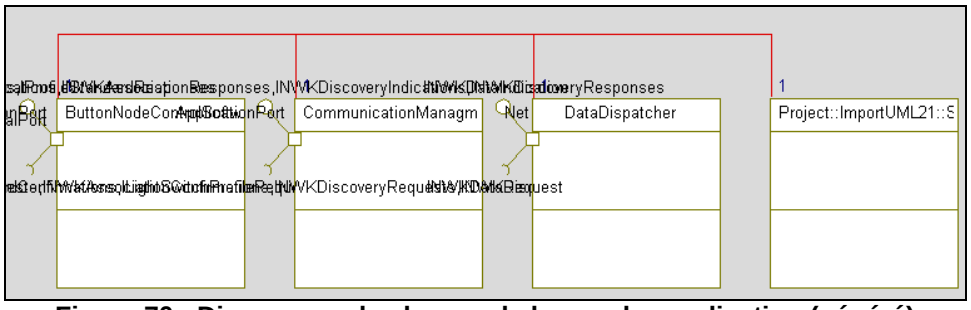


Figure 70 : Diagramme de classes de la couche application (généré)

Voilà le principe de base de la création d'éléments graphiques en UML sous Rhapsody.

5 Tests

En considérant la transformation du modèle comme un test, il est possible d'affirmer que le test donne un résultat positif. Tous les éléments sont bien pris et transformés en UML selon les besoins pour la future implémentation et génération de code.

Les éléments ajoutés par la « transformation intelligente » sont eux aussi présents dans le résultat UML.

Les diagrammes UML ont été la dernière tâche expérimentée du projet. Leur création fonctionne sans problème, leur aspect reste cependant un sujet à étudier.

A côté de cela, aucun matériel hardware n'a été fait ou utilisé. Le test se limite donc au code développé.

6 Conclusion

Le projet ADS, brièvement défini en introduction, arrive au terme de sa première année d'étude et de développement. Ce travail de diplôme en a été une partie et servira pour la seconde année d'ADS.

La transformation de modèles m'est apparue comme une nouvelle manière de traiter la modélisation, elle brise les limites entre les différents types de modèles et ouvre un nouvel horizon sur le développement MDA[10].

Autant la transformation du modèle GME originel paraît simpliste après avoir compris les bases, autant la complexité se fait sentir dès l'ajout d'éléments inexistants durant la transformation dite « intelligente ». Chaque classe, objet, port,…, doit être référencé afin d'y créer des associations, liens, dépendances, ...

Au niveau de l'utilisation, MDWorkbench est un outil clair et didactique, très optimisé pour la transformation de modèle. Les tutoriaux sont également très bien expliqués, en détail, et de les parcourir fait gagner un temps précieux sur la découverte du programme.

Son plus est également la possibilité de travailler avec le format XMI 2.1, ce qui permet d'utiliser la dernière version de Rhapsody en parallèle avec toutes les fonctionnalités, si l'on veut travailler en UML 2.1 évidemment.

Il ne restera qu'à découvrir plus en profondeur les atouts du plug in Topcased pour Eclipse, afin de travailler les modèles et métamodèles. Cela limitera l'aire de travail de modélisation et transformation à un seul outil.

En conclusion, je dirai que le travail de semestre sur le MDA m'a été très profitable pour démarrer ce travail de diplôme, ayant acquis grâce à celui-là de bonnes connaissances de base sur les métamodèles, modèles et leur utilisation dans la globalité d'un projet.

Je tiens aussi à remercier particulièrement M. Rico Steiner pour son aide précieuse qui m'a permis de réaliser ce travail jusqu'à sa finalité.

[10] MDA, ou Architecture dirigée par des modèles est une démarche de création de logiciel. Le modèle du système souhaité est créé indépendamment de la plateforme, puis ensuite traduit selon la plateforme utilisée pour finalement être implémenté (source : Travail de semestre, « Model Driven Architecture », Philippe Pralong).

7 Références

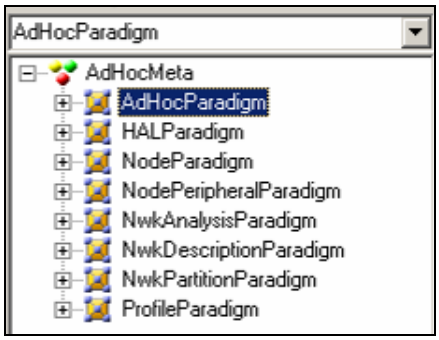
- M. Rico Steiner, assistant à la HES-SO Valais
- M. Philippe Soulard, Sodius SA France
- M. Philippe Pralong, "MDA", Travail de semestre, 2007
- http://fr.wikipedia.org, définition générale de Topcased, GME
- http://www.google.ch, base de recherche pour divers problèmes de programmation
- Documentation Topcased sous forme numérique
- Documentation MDWorkbench sous forme numérique
- Documentation Rhapsody sous forme numérique, pour RulesComposer

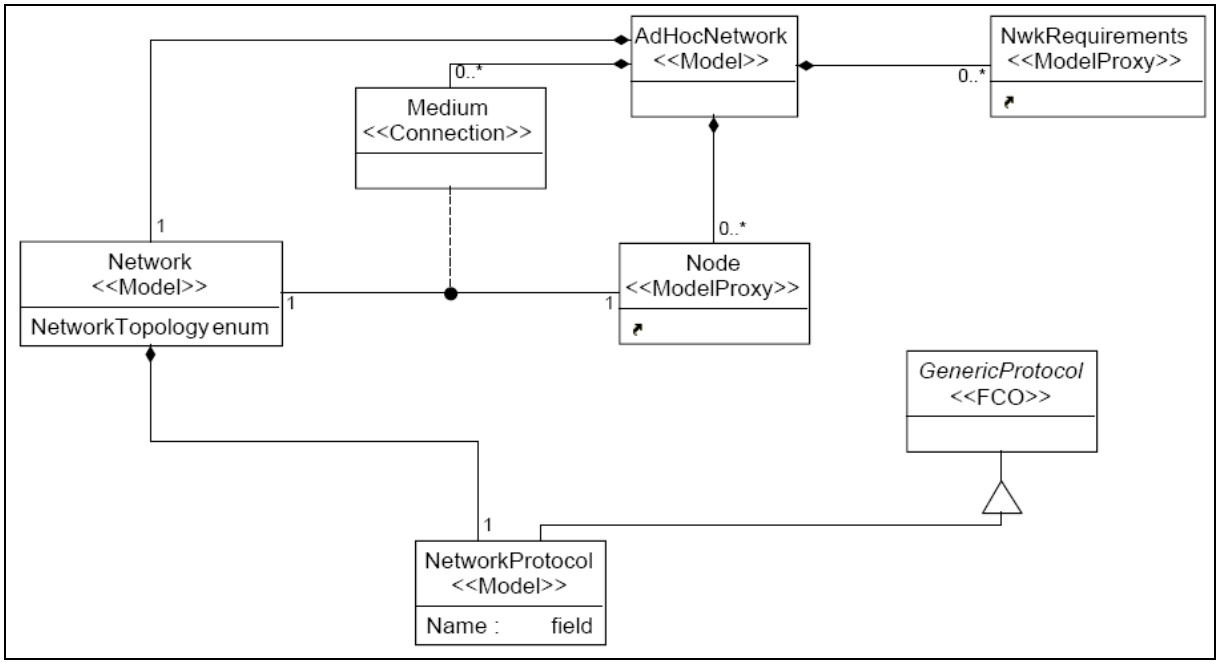
8 Annexes

A Métamodèle « AdHocMeta »

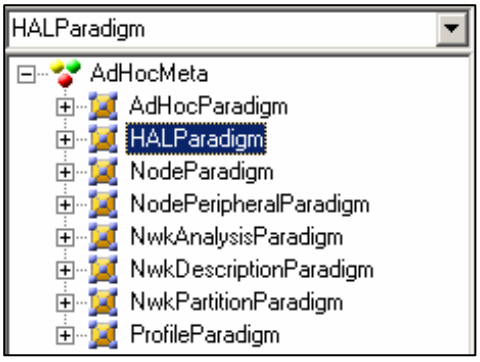
L'annexe A présente toutes les parties qui composent le métamodèle AdHocMeta servant à l'élaboration du modèle du réseau ad hoc. Ce métamodèle a été créé sous GME.

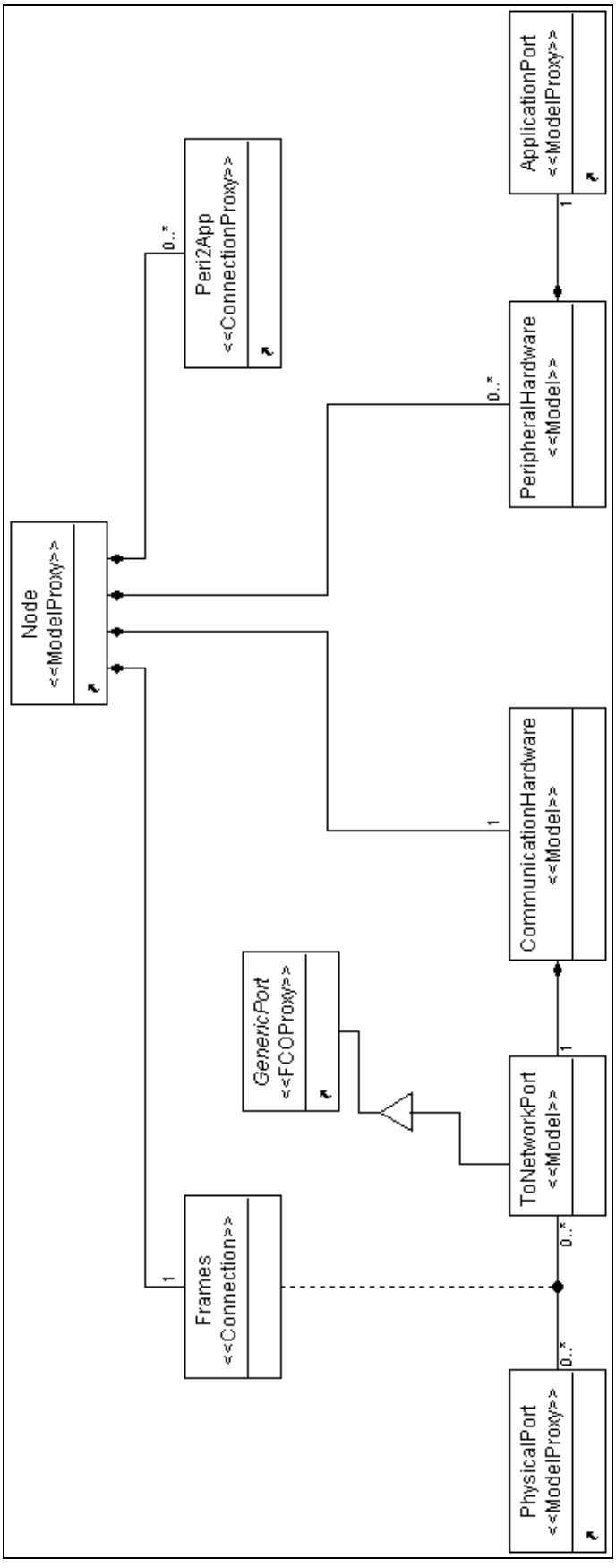
A.1 AdHocParadigm



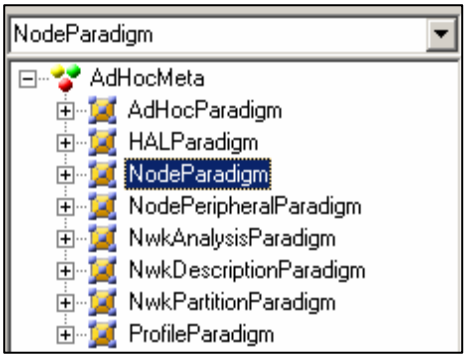


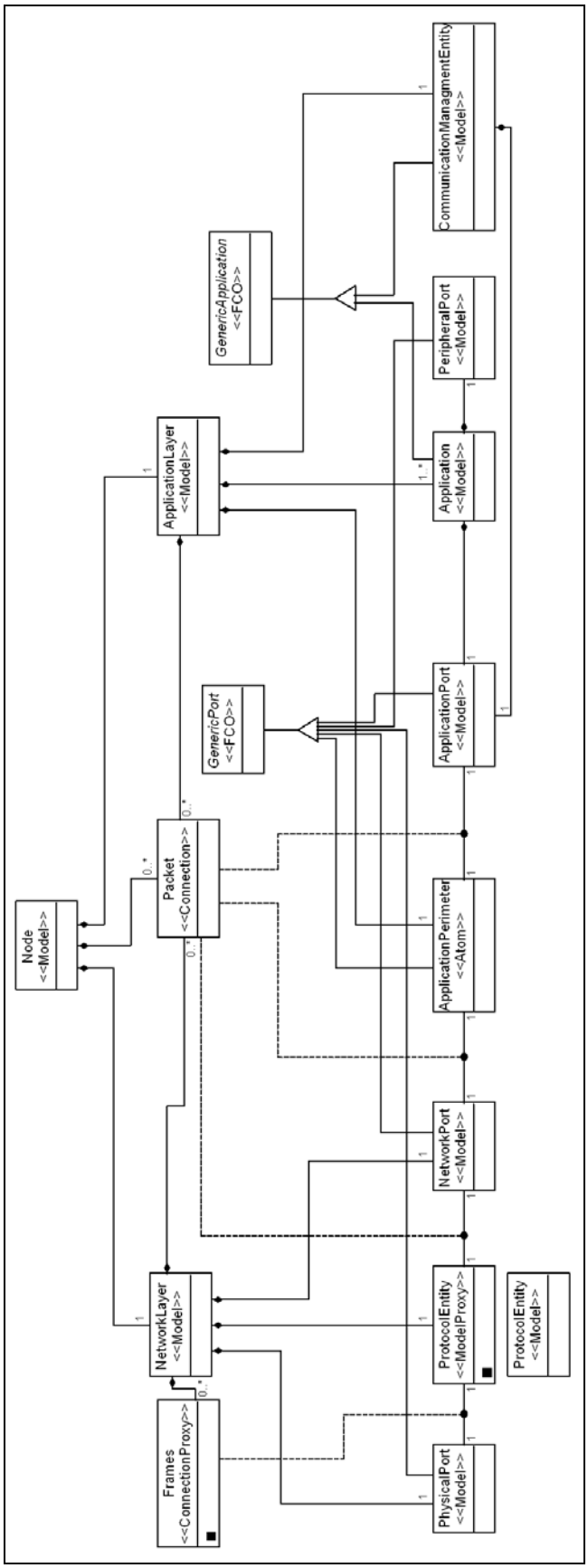
A.2 HALParadigm



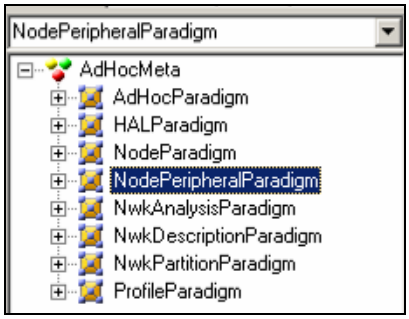


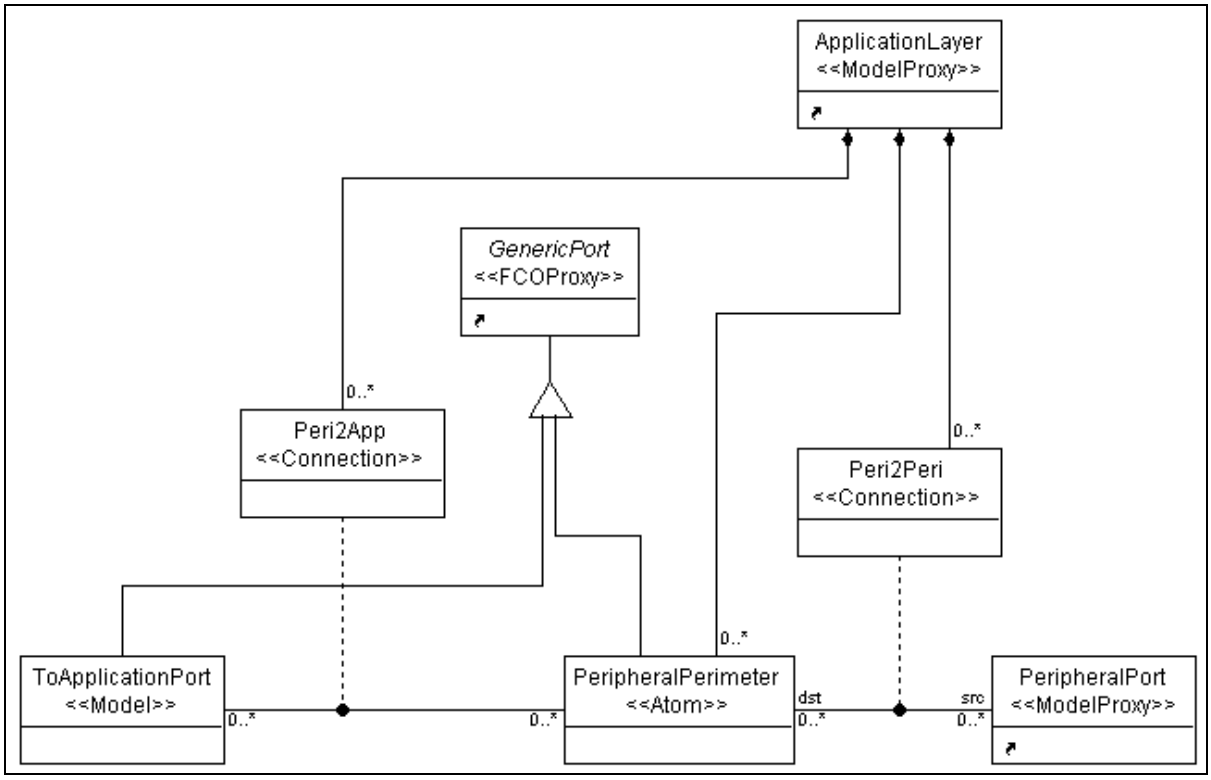
A.3 NodeParadigm



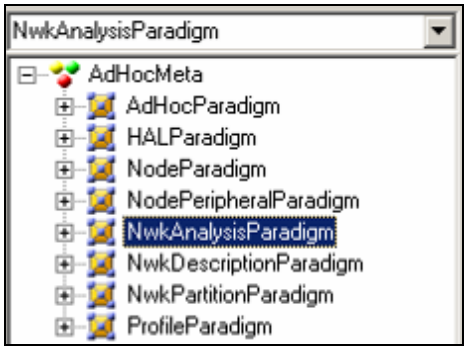


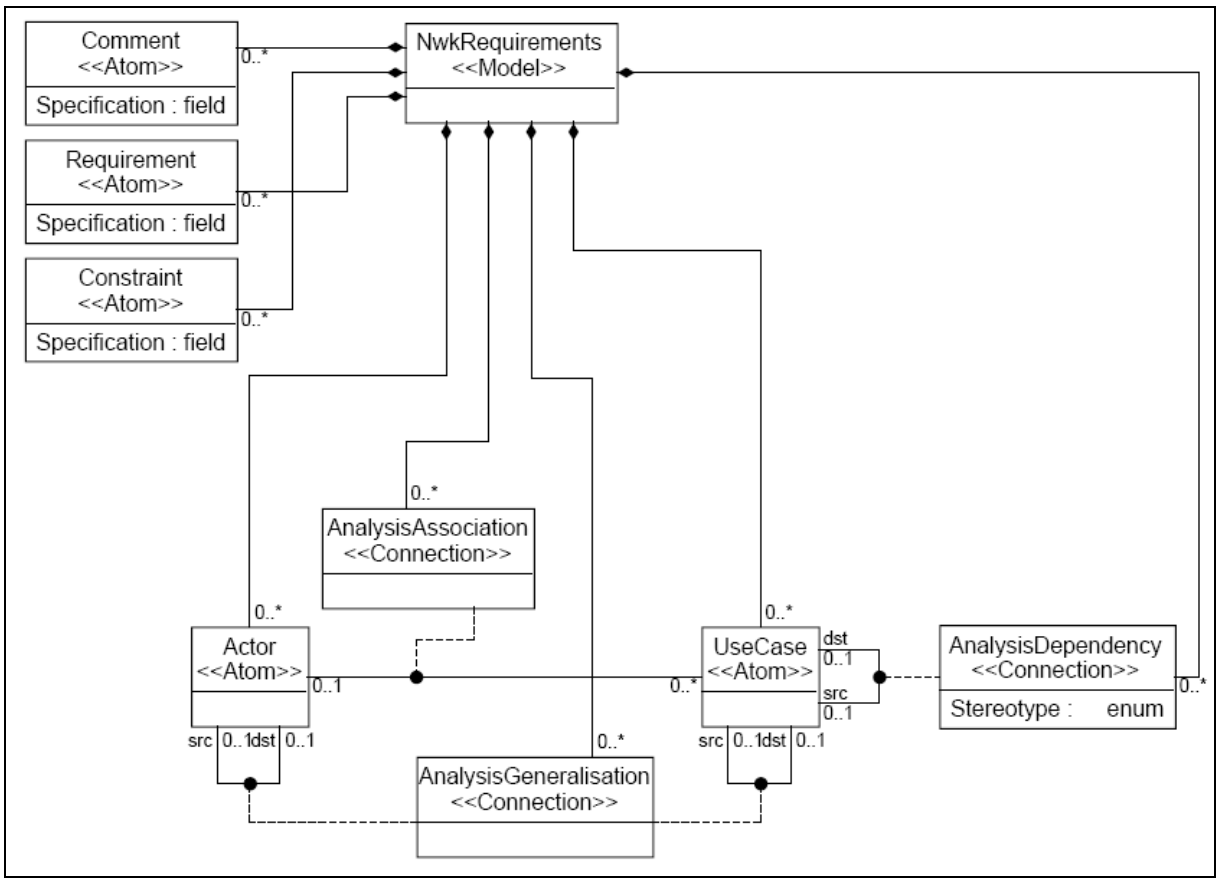
A.4 NodePeripheralParadigm



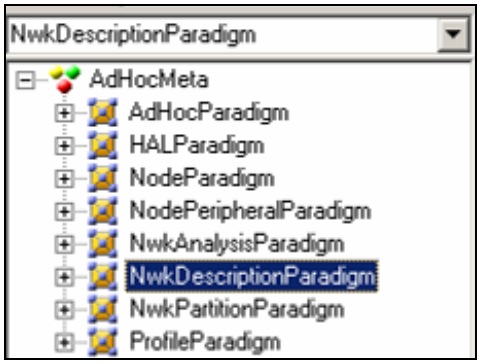


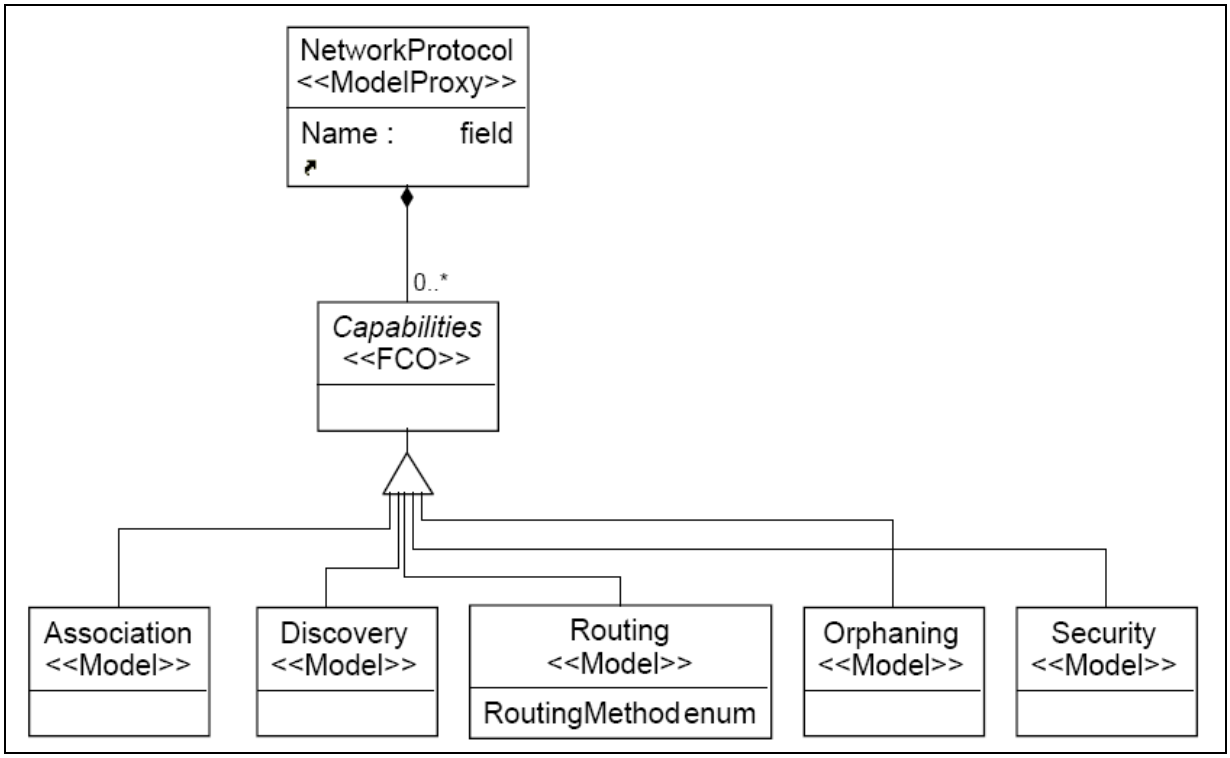
A.5 NwkAnalysisParadigm



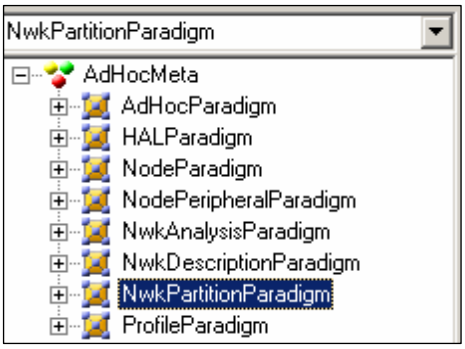


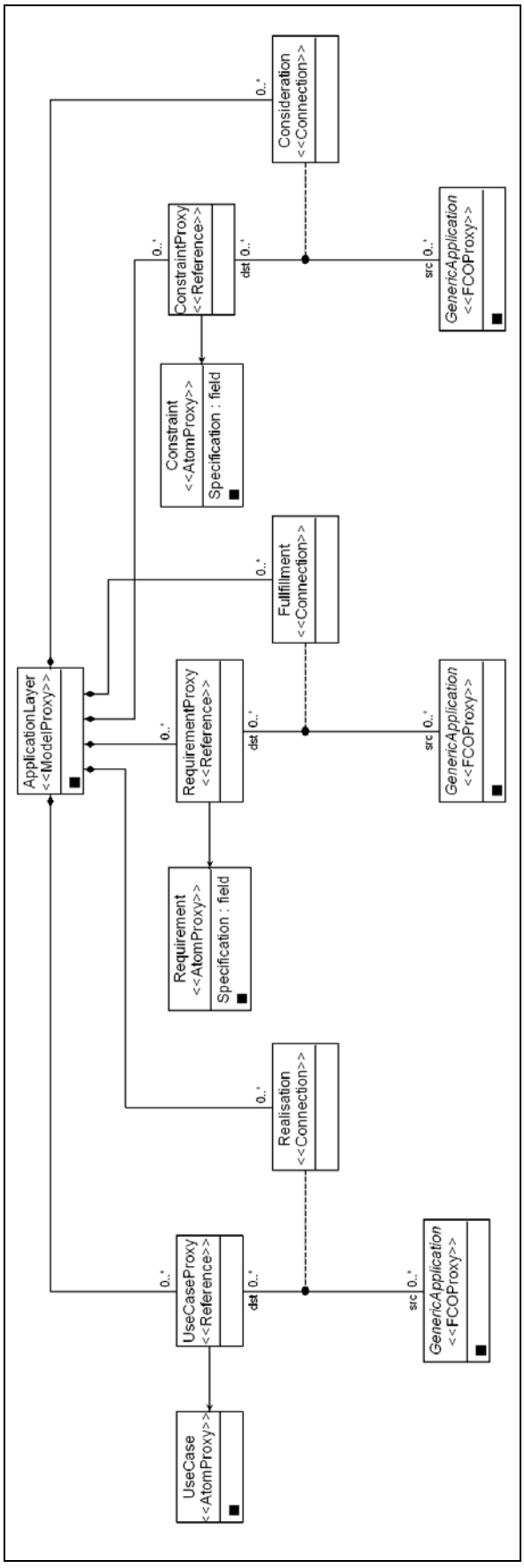
A.6 NwkDescriptionParadigm



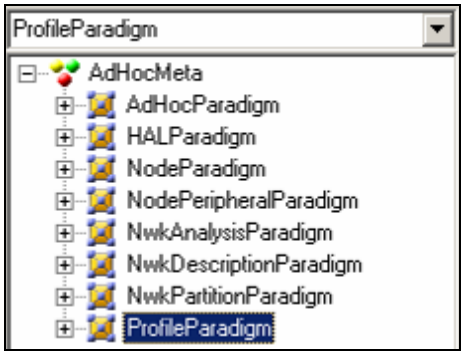


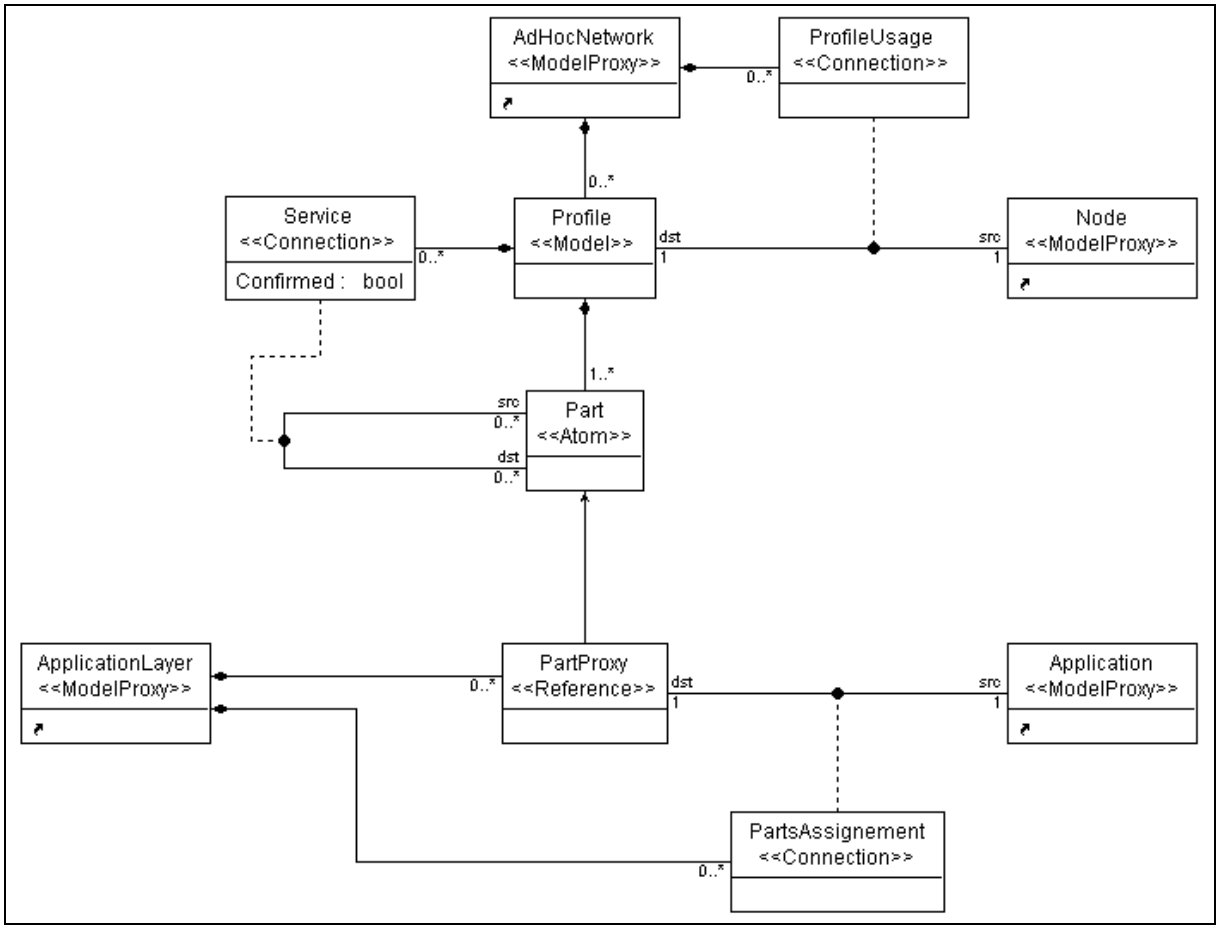
A.7 NwkPartitionParadigm





A.8 ProfileParadigm





B Code de transformation GME -> UML

B.1 AdHoc2UML

```
/**
*AdHoc2UML is the main file of transformation
*It calls main rulesets of each part of transformation GME to UML
*written by Philippe Pralong
*/

package org.ads;

public ruleset AdHoc2UML(in source: gme,  out target: uml21)
{
   public rule main()
   {
      // define main package
      var pkgProject: uml21.Package = target.create("Package");
      var adHocProject: gme.Project = source.getInstances("Model").first();
      pkgProject.name = adHocProject.getName().getAsString()+"_Pkg";


      foreach(adHocNetwork: gme.Model in source.getInstances("Model"))
      {
         if(adHocNetwork.getKind() == "AdHocNetwork")
         {
            // define Analysis package
            var pkgAnalysis: uml21.Package =
                                    pkgProject.createNestedPackage("Analysis_Pkg");
            @AnalysisTransformation(pkgAnalysis, target).main(adHocNetwork);

            //define Interfaces package
            var pkgInterfaces: uml21.Package =
                                    pkgProject.createNestedPackage("Interfaces_Pkg");
            @InterfaceTransformation(pkgInterfaces, target).main(adHocNetwork);

            //define Profiles package
            var pkgProfiles: uml21.Package =
                                    pkgProject.createNestedPackage("Profiles_Pkg");
            @ProfileTransformation(pkgProfiles, target,
                                    pkgInterfaces).main(adHocNetwork);

            // define nodes package
            foreach(node: gme.Model in adHocNetwork.getModel())
            {
               if(node.getKind() == "Node")
               {
                  var pkgNode: uml21.Package =
                  pkgProject.createNestedPackage(node.getName().getAsString()+"_Pkg");

                  @NodeTransformation(pkgNode, adHocNetwork, target, pkgInterfaces,
                                       pkgProfiles).main(node);
               }
            }
         }
      }
   }
}
```

B.2 AnalysisTransformation

```
/**
*Code written by Rico Steiner
*Concern all Analysis part of AdHoc Network
*/

package org.ads;

public ruleset AnalysisTransformation(target: uml21.Package, in model: uml21)
{
   /* main rule */
   public rule main(parentNet: gme.Model )
   {
      foreach(nwkReq: gme.Model in parentNet.getModel())
      {
         if(nwkReq.getKind() == "NwkRequirements")
         {
            var pkgNwkReq: uml21.Package =
                        target.createNestedPackage(nwkReq.getName().getAsString());

            //add all elements
            @addActors(nwkReq, pkgNwkReq);
            @addUseCases(nwkReq, pkgNwkReq);
            @addComments(nwkReq, pkgNwkReq);
            @addConstraintsAndRequirements(nwkReq, pkgNwkReq);
            @addConnections(nwkReq, pkgNwkReq);
         }
      }
   }

   /* addNetworkLayer rule */
   private rule addNetworkLayer(nwkReq: gme.Model, targetPkg: uml21.Package)
   {
      var newNetworkLayer: uml21.Model;

      foreach(networkLayer: gme.Model in nwkReq.getModel())
      {
         if(networkLayer.getKind() == "NetworkModel")
         {
            newNetworkLayer = model.create("NetworkLayer");
            newNetworkLayer.setName(networkLayer.getName().getAsString());
         }
      }
   }

   /* addActors rule */
   private rule addActors(nwkReq: gme.Model, targetPkg: uml21.Package)
   {
      var newActor: uml21.Actor;

      foreach(actor: gme.Atom in nwkReq.getAtom())
      {
         if(actor.getKind() == "Actor")
         {
            newActor = model.create("Actor");
            newActor.setPackage(targetPkg);
            newActor.setName(actor.getName().getAsString());
            actor.getId()#useCaseReferences.add(newActor);
         }
      }
   }

   /* addUseCases rule */
   private rule addUseCases(nwkReq: gme.Model, targetPkg: uml21.Package)
   {
      var newUseCase: uml21.UseCase;

      foreach(useCase: gme.Atom in nwkReq.getAtom())
      {
         if(useCase.getKind() == "UseCase")
         {
            newUseCase = model.create("UseCase");
            newUseCase.setPackage(targetPkg);
            newUseCase.setName(useCase.getName().getAsString());
            useCase.getId()#useCaseReferences.add(newUseCase);
```

```
        }
    }
}

/* addConnections rule */
private rule addConnections(nwkReq: gme.Model, targetPkg: uml21.Package)
{
    var newAssoc: uml21.Association = null;
    var srcID: String = "";
    var dstID: String = "";

    //search each connection...
    foreach(connection: gme.Connection in nwkReq.getConnection())
    {
        srcID = "";
        dstID = "";

        //... and set source and target
        foreach(connTarget: gme.Connpoint in connection.getConnpoint())
        {
            if(connTarget.getRole() == "src")
            {
                srcID = connTarget.getTarget();
            }
            else if(connTarget.getRole() == "dst")
            {
                dstID = connTarget.getTarget();
            }
        }

        // check the type of connection
        if(connection.getKind() == "AnalysisAssociation")
        {
            @addAssociations(targetPkg, srcID, dstID);
        }
        else if(connection.getKind() == "AnalysisDependency")
        {
            @addDependency(targetPkg, srcID, dstID);
        }
        else if(connection.getKind() == "AnalysisGeneralisation")
        {
            @addGeneralisation(targetPkg, srcID, dstID);
        }
    }
}

/* addAssociations rule */
private rule addAssociations(targetPkg: uml21.Package, srcID: String,
                             dstID: String)
{
    var associataion: uml21.Association = model.create("Association");
    var srcObj: uml21.Type = srcID#useCaseReferences.first();
    var dstObj: uml21.Type = dstID#useCaseReferences.first();

    associataion.setPackage(targetPkg);
    associataion.createNavigableOwnedEnd(srcObj.getName(), srcObj);
    associataion.createNavigableOwnedEnd(dstObj.getName(), dstObj);
}

/* addDependency */
private rule addDependency(targetPkg: uml21.Package, srcID: String,
                           dstID: String)
{
    var dependency: uml21.Dependency = null;
    var srcObj: uml21.UseCase = srcID#useCaseReferences.first();
    var dstObj: uml21.UseCase = dstID#useCaseReferences.first();

    // Stereotypes are missing
    dependency = srcObj.createDependency(dstObj);
}

/* addGeneralisation */
private rule addGeneralisation(targetPkg: uml21.Package, srcID: String,
                               dstID: String)
{
    var generalization: uml21.Generalization = null;
    var srcObj: uml21.Classifier = srcID#useCaseReferences.first();
```

```
      var dstObj: uml21.Classifier = dstID#useCaseReferences.first();

      generalization = srcObj.createGeneralization(dstObj);
   }

   /* addComments */
   private rule addComments(nwkReq: gme.Model, targetPkg: uml21.Package)
   {
      var newComment: uml21.Comment = null;

      foreach(comment: gme.Atom in nwkReq.getAtom())
      {
         if(comment.getKind() == "Comment")
         {
            newComment = targetPkg.createOwnedComment();
            foreach(attribute: gme.Attribute in comment.getAttribute())
            {
               if(attribute.getKind() == "Specification")
               {
                  newComment.setBody(attribute.getValue().getAsString());
               }
            }
         }
      }
   }

   /* addConstraintsAndRequirements rule */
   private rule addConstraintsAndRequirements(nwkReq: gme.Model,
                                              targetPkg: uml21.Package)
   {
      var newConstraint: uml21.Constraint = null;
      var tmpConstraint: uml21.Constraint = model.create("Constraint");
      var newSpec: uml21.LiteralString = null;

      foreach(constraint: gme.Atom in nwkReq.getAtom())
      {
         if((constraint.getKind() == "Constraint")
            || (constraint.getKind() == "Requirement"))
         {
            newConstraint = targetPkg.createPackagedElement(constraint.getKind(),
                                                            tmpConstraint.eClass());

            foreach(attribute: gme.Attribute in constraint.getAttribute())
            {
               if(attribute.getKind() == "Specification")
               {
                  newSpec = model.create("LiteralString");
                  newSpec.setValue(attribute.getValue().getAsString());
                  newConstraint.setSpecification(newSpec);
               }
            }
         }
      }

      tmpConstraint.destroy();
   }
}
```

B.3 ConnectionTransformation

```
/**
*ConnectionTransformation works on all type of connections (links, associations, ...)
*
*-main() : search existing  source and target of connections
*-addConnection(Class, String, String) : called in main(), add the existing connection
*-createConnection(Port, Port) : some elements are added during transformation, we use
createConnection to make connection between them
*-createLinks(Class, Class) : called when we want a classes' association; we make first link
between objects, and then the classes' association
*-createAssociation(Class, Class) : called in createLinks(Class, Class)
*written by Philippe Pralong
*/

package org.ads;

public ruleset ConnectionTransformation(node: gme.Model, targetClass: uml21.Class,
                                        in model: uml21)
{
   /* main rule*/
   public rule main()
   {
      var srcID: String;
      var dstID: String;

      foreach(connection: gme.Connection in node.getConnection())
      {
         srcID = "";
         dstID = "";

         //avoid Realisation and PartsAssignement types of connections
         if(connection.getName().getAsString() != "Realisation"
            & connection.getName().getAsString() != "PartsAssignement")
         {
            foreach(connTarget: gme.Connpoint in connection.getConnpoint())
            {
               //search Source and Destination
               if(connTarget.getRole() == "src")
                  srcID = connTarget.getTarget();

               else if(connTarget.getRole() == "dst")
                  dstID = connTarget.getTarget();
            }
            //add associations
            @addConnection(targetClass, srcID, dstID);
         }
      }
   }

   /* addConnection rule */
   private rule addConnection(target: uml21.Class, srcID: String, dstID: String)
   {
      //Create Connectors (~Links) between Composite Objects
      var connector: uml21.Connector = model.create("Connector");
      var end1: uml21.ConnectorEnd = connector.createEnd();
      var end2: uml21.ConnectorEnd = connector.createEnd();
      var srcPort: uml21.Port = srcID#portReferences.first();
      var dstPort: uml21.Port = dstID#portReferences.first();

      //search name of source and target
      var owningClassSrc: uml21.Class = srcPort.getClass_();
      var owningClassDst: uml21.Class = dstPort.getClass_();
      var srcName: String = owningClassSrc#compObjRef.first().getName();
      var dstName: String = owningClassDst#compObjRef.first().getName();;


      //add Connector to class, set ends, partWithPorts and name
      target.ownedConnector.add(connector);
      end1.setRole(srcPort);
      end2.setRole(dstPort);

      //check if Port exists in the same Class as Connector,
      //if true, the param "partWithPort" mustn't be set,
      //and name is the port, not the object
      if(srcPort == target.getOwnedPort(srcPort.getName(),srcPort.getType()))
```

```
      {
         srcName = srcPort.getName();
         end1.setPartWithPort(null);
      }
      else
         end1.setPartWithPort(srcPort.getClass_()#compObjRef.first());

      if(dstPort == target.getOwnedPort(dstPort.getName(),dstPort.getType()))
      {
         dstName = dstPort.getName();
         end2.setPartWithPort(null);
      }
      else
         end2.setPartWithPort(dstPort.getClass_()#compObjRef.first());


      connector.setName(srcName+"_"+dstName);
   }

   /* createConnection rule */
   public rule createConnection(src: uml21.Port, dst: uml21.Port)
   {
      //Create Connectors (~Links) between Composite Objects
      var connector: uml21.Connector = model.create("Connector");
      var end1: uml21.ConnectorEnd = connector.createEnd();
      var end2: uml21.ConnectorEnd = connector.createEnd();
      var srcPort: uml21.Port = src;
      var dstPort: uml21.Port = dst;
      var owningClassSrc: uml21.Class = srcPort.getClass_();
      var owningClassDst: uml21.Class = dstPort.getClass_();
      var srcName: String = owningClassSrc#compObjRef.first().getName();
      var dstName: String = owningClassDst#compObjRef.first().getName();;


      //add Connector to class, set ends, partWithPorts and name
      targetClass.ownedConnector.add(connector);
      end1.setRole(srcPort);
      end2.setRole(dstPort);

      //check if Port exists in the same Class as Connector,
      //if true, the param "partWithPort" mustn't be set,
      //and name is the port, not the object
      if(srcPort == targetClass.getOwnedPort(srcPort.getName(),srcPort.getType()))
      {
         srcName = srcPort.getName();
         end1.setPartWithPort(null);
      }
      else
         end1.setPartWithPort(srcPort.getClass_()#compObjRef.first());

      if(dstPort == targetClass.getOwnedPort(dstPort.getName(),dstPort.getType()))
      {
         dstName = dstPort.getName();
         end2.setPartWithPort(null);
      }
      else
         end2.setPartWithPort(dstPort.getClass_()#compObjRef.first());


      connector.setName(srcName+"_"+dstName);
   }

   /* createLinks rule */
   /* 1. Create links between  objects, and then                          */
   /* 2. call createClassAssociations, for associations between classes */
   public rule createLinks(src: uml21.Class, dst: uml21.Class)
   {
      //Create Connectors (~Links) between Composite Objects
      var connector: uml21.Connector = model.create("Connector");
      var end1: uml21.ConnectorEnd = connector.createEnd();
      var end2: uml21.ConnectorEnd = connector.createEnd();
      var srcPort: uml21.Port = src#compObjRef.first();
      var dstPort: uml21.Port = dst#compObjRef.first();
      var srcName: String = src#compObjRef.first().getName();
      var dstName: String = dst#compObjRef.first().getName();

      //add Connector to class, set ends, partWithPorts and name
```

```
      targetClass.ownedConnector.add(connector);
      end1.setRole(srcPort);
      end2.setRole(dstPort);

      //check if Port exists in the same Class as Connector,
      //if true, the param "partWithPort" mustn't be set,
      //and name is the port, not the object
      if(srcPort == targetClass.getOwnedPort(srcPort.getName(),srcPort.getType()))
      {
         srcName = srcPort.getName();
         end1.setPartWithPort(null);
      }
      else
         end1.setPartWithPort(srcPort.getClass_()#compObjRef.first());

      if(dstPort == targetClass.getOwnedPort(dstPort.getName(),dstPort.getType()))
      {
         dstName = dstPort.getName();
         end2.setPartWithPort(null);
      }
      else
         end2.setPartWithPort(dstPort.getClass_()#compObjRef.first());


      connector.setName(srcName+"_"+dstName);
      @createAssociations(src, dst);
   }

   /* createAssociations rule */
   public rule createAssociations(src: uml21.Class, dst: uml21.Class)
   {
      //create Associations between Classes
      var association: uml21.Association = model.create("Association");
      var srcName: String = src#compObjRef.first().getName();
      var dstName: String = dst#compObjRef.first().getName();
      var profileBase: uml21.Class;

      //for profiles, create association with superClass of profile
      if(src.getName().contains("Profile"))
      {
         foreach(pkg: uml21.Package in model.getInstances("Package"))
         {
            if(pkg.getName() == "Profiles_Pkg")
            {
               dst = pkg.getClass(dst.getName()).getSuperClass("BaseProfile");
               srcName = "its"+src.getName();
            }
         }
      }
      else if(dst.getName().contains("Profile"))
      {
         foreach(pkg: uml21.Package in model.getInstances("Package"))
         {
            if(pkg.getName() == "Profiles_Pkg")
            {
               dst = pkg.getClass(dst.getName()).getSuperClass("BaseProfile");
               dstName = "its"+dst.getName();
            }
         }
      }

      //create ends of association
      association.createNavigableOwnedEnd(srcName, src);
      association.createNavigableOwnedEnd(dstName, dst);
   }
}
```

B.4 InterfaceTransformation

```
/**
*InterfaceTransformation create all interfaces used for AdHoc Network
*written by Philippe Pralong
*/

package org.ads;
import org.ads.tools.CommonMacros;

public ruleset InterfaceTransformation(target: uml21.Package, in model: uml21) {

   /* main rule */
   public rule main(parentNet: gme.Model)
   {
      //create packages
      var networkPkg: uml21.Package =
                                    target.createNestedPackage("NetworkInterfaces_Pkg");
      var profilePkg: uml21.Package =
                                    target.createNestedPackage("ProfileInterfaces_Pkg");
      var interface: uml21.Interface;

      //create network interfaces according to protocol
      foreach(node: gme.Model in parentNet.getModel())
      {
         if(node.getKind() == "Network")
         {
            foreach(subNode: gme.Model in node.getModel())
            {
               if(subNode.getKind() == "NetworkProtocol")
               {
                  foreach(interfaceState: gme.Model in subNode.getModel())
                  {
                     foreach(state: String in CommonMacros.UML_INTERFACES)
                     {
                        interface =
                                 networkPkg.createOwnedInterface(CommonMacros.INetwork
                                 +interfaceState.getName().getAsString()+state);

                        //add Interfaces' references
                        interfaceState#iRef.add(interface);
                     }
                  }
               }
            }
         }
      }

      //service data request and indication
      var dataRequest: uml21.Interface =
            networkPkg.createOwnedInterface(CommonMacros.INetwork+"DataRequest");
      var dataIndication: uml21.Interface =
            networkPkg.createOwnedInterface(CommonMacros.INetwork+"DataIndication");
      var dataRequestOp: uml21.Operation =
            dataRequest.createOwnedOperation("dataRequest", null, null);
      var dataIndicationOp: uml21.Operation =
            dataIndication.createOwnedOperation("dataIndication", null, null);
      dataRequestOp.setIsAbstract(true);
      dataIndicationOp.setIsAbstract(true);

      //add Interfaces' references
      model#iRef.add(dataRequest);
      model#iRef.add(dataIndication);

      //interfaces for Profile
      foreach(state: String in CommonMacros.UML_INTERFACES)
      {
         interface =
   profilePkg.createOwnedInterface(CommonMacros.IProfileStandard+state);


   interface.createOwnedOperation("getProfileID"+interface.getName().substring(CommonMacros.IP
rofileStandard.length(),interface.getName().length()-1), null, null);

   interface.createOwnedOperation("getProfileName"+interface.getName().substring(CommonMacros.
IProfileStandard.length(),interface.getName().length()-1), null, null);
```

```
         foreach(operation: uml21.Operation in interface.getAllOperations())
         {
            operation.setIsAbstract(true);
         }

         //add Interfaces' references
         model#iRef.add(interface);
      }
   }
}
```

B.5 NodeTransformation

```
/**
*NodeTransformation creates all elements in a node
*It works on each layer of the model (peripheral, application, network, hardware)
*written by Philippe Pralong
*/

package org.ads;

import org.ads.tools.CommonMacros;

public ruleset NodeTransformation(target: uml21.Package, mainModel: gme.Model, in model:
uml21, pkgInterfaces: uml21.Package, pkgProfiles: uml21.Package) {


   /* main rule */
   public rule main(node: gme.Model)
   {
      var pkgNode: uml21.Package;

      //create Factory
      var factory: uml21.Class =
   target.createOwnedClass(node.getName().getAsString()+"Factory", false);

      //create package for each model element
      foreach(subNode: gme.Model in node.getModel())
      {
         pkgNode =
                  target.createNestedPackage(subNode.getName().getAsString()+"_Pkg");

         //Package references
         subNode.getId()#pkgReferences.add(pkgNode);
      }

      //add model elements as Classes
      @addApplicationLayer(node, target, factory);
      @addCommunicationHardware(node, target, factory);
      @addNetworkLayer(node, target, factory);
      @addPeripheralHardware(node, target, factory);
      @ConnectionTransformation(node, factory, model).main();
   }


   /* addPeripheralHardware rule */
   private rule addPeripheralHardware(node: gme.Model, targetPkg: uml21.Package,
                                          classFac: uml21.Class)
   {
      var newClass: uml21.Class;

      foreach(peripheralHardware: gme.Model in node.getModel())
      {
         if(peripheralHardware.getKind() == "PeripheralHardware")
         {
            //create Class and Object
            newClass =
      targetPkg.createOwnedClass(peripheralHardware.getName().getAsString(), false);

            //global  mapping
            peripheralHardware.getId()#globalMap.add(newClass);
            newClass.createObj(newClass, classFac);

            //add ports
```

```
            @addPort(peripheralHardware, newClass);
         }
      }
   }

   /* addApplicationLayer method */
   private rule addApplicationLayer(node: gme.Model, targetPkg: uml21.Package,
                                    classFac: uml21.Class)
   {
      var newClass: uml21.Class;
      var appClass: uml21.Class;
      var cmeClass: uml21.Class;
      var dataDisClass: uml21.Class;
      var ccDataDisClass: uml21.Class;
      var profileClass: uml21.Class;
      var compositeObj: uml21.Property;
      var neastedPkg: uml21.Package;
      var newPort: uml21.Port;
      var netPort: uml21.Port;

      foreach(applicationLayer: gme.Model in node.getModel())
      {
         if(applicationLayer.getKind() == "ApplicationLayer")
         {
            //create Class and Object
            newClass =
         targetPkg.createOwnedClass(applicationLayer.getName().getAsString(), false);

            //global  mapping
            applicationLayer.getId()#globalMap.add(newClass);
            newClass.createObj(newClass, classFac);

            //add ports
            @addPort(applicationLayer, newClass);

            // Submodel ------------------------------------------------------------/

            //search corresponding Package to this Class, and create subClasses
            neastedPkg = applicationLayer.getId()#pkgReferences.first();
            foreach(subApplicationLayer: gme.Model in applicationLayer.getModel())
            {
               if(subApplicationLayer.getKind() == "Application")
               {
                  //create Class and Object
                  appClass =
            neastedPkg.createOwnedClass(subApplicationLayer.getName().getAsString(),
                                        false);
                  //global  mapping
                  subApplicationLayer.getId()#globalMap.add(appClass);
                  newClass.createObj(appClass);
                  //add ports
                  @addPort(subApplicationLayer, appClass);
               }
               else if(subApplicationLayer.getKind()=="CommunicationManagmentEntity")
               {
                  //create Class and Object
                  cmeClass =
            neastedPkg.createOwnedClass(subApplicationLayer.getName().getAsString(),
                                        false);
                  //global  mapping
                  subApplicationLayer.getId()#globalMap.add(cmeClass);
                  newClass.createObj(cmeClass);
                  //add ports
                  @addPort(subApplicationLayer, cmeClass);
               }
            }

            //create additional classes for services, ports and connections
            dataDisClass = neastedPkg.createOwnedClass("DataDispatcher", false);
            dataDisClass.createOwnedOperation("dataRequest", null, null);
            newClass.createObj(dataDisClass);

            ccDataDisClass =
                        dataDisClass.createNestedClassifier("ContractClassOfNetPort",
                        dataDisClass.eClass());
```

```
netPort = dataDisClass.createOwnedPort("Net", ccDataDisClass);
netPort#ccReferences.add(ccDataDisClass);

@ConnectionTransformation(applicationLayer, newClass,
            model).createConnection(netPort,
            newClass.getPort(applicationLayer,  "ApplicationPerimeter"));

foreach(profileNode: gme.Model in mainModel.getModel())
{
   if(profileNode.getKind() == "Profile")
   {
      newClass.createObj(profileNode.getId()#globalMap.first());
      profileClass = profileNode.getId()#globalMap.first();
   }
}

// Connections ------------------------------------------------------------/
//create links between existing objects
@ConnectionTransformation(applicationLayer, newClass, model).main();
//but if profiles exist,
//delete connections that are modified during transformation
foreach(profile: gme.Model in mainModel.getModel())
{
   if(profile.getKind() == "Profile")
   {
      foreach(port: gme.Atom in applicationLayer.getAtom())
      {
         if(port.getKind() == "ApplicationPerimeter")
         {
newClass.connectionDestroy(port.getId()#portReferences.first().getName()
                           +"_"+appClass#compObjRef.first().getName());
         }
      }
   }
}
foreach(profileNode: gme.Model in mainModel.getModel())
{
   if(profileNode.getKind() == "Profile")
   {
      var profileC: uml21.Class = profileNode.getId()#globalMap.first();
      var srcPort: uml21.Port;
      var dstPort: uml21.Port;

      foreach(subApplicationLayer: gme.Model
               in applicationLayer.getModel())
      {
         if(subApplicationLayer.getKind() == "Application")
         {
            foreach(profilePort: uml21.Port in profileC.getPorts())
            {
               if(profilePort.name == "App")
               {
                  @ConnectionTransformation(applicationLayer, newClass,
            model).createConnection(profilePort,
            appClass.getPort(subApplicationLayer, "ApplicationPort"));
               }
            }
         }
      }
   }
}

// Associations -----------------------------------------------------------/

//create (1) links between created objects and then inside
//(2) associations between related classes
//LedNode hasn't been defined yet, associations are limited to
//buttonNode, this is reason of <foreach, if> statement
foreach(class: uml21.Class in neastedPkg.getClasses())
{
   if(class == cmeClass)
   {
      @ConnectionTransformation(applicationLayer, newClass,
   model).createLinks(appClass, cmeClass);
      @ConnectionTransformation(applicationLayer, newClass,
   model).createLinks(neastedPkg.getClass("DataDispatcher"), cmeClass);
```

```
            foreach(pkg: uml21.Package in model.getInstances())
            {
               if(pkg instanceof uml21.Package
                  & pkg.getName() == "Profiles_Pkg")
               {
                  @ConnectionTransformation(applicationLayer, newClass,
                                    model).createLinks(dataDisClass, profileClass);
               }
            }
         }
      }

      // Port's interfaces ------------------------------------------------/
      var port: uml21.Port;

      //ApplicationPerimeter Port
      foreach(portPerimeter: gme.Atom in applicationLayer.getAtom())
      {
         if(portPerimeter.getKind() == "ApplicationPerimeter")
         {
            port = portPerimeter.getId()#portReferences.first();

            foreach(interfaceState: gme.Model in mainModel.getProtocolAspects())
            {
               foreach(interface: uml21.Interface in interfaceState#iRef)
               {
                  if(interface.name.contains("Indication")
                     || interface.name.contains("Response"))
                  {
                     port.setProvided(interface);
                  }
                  else
                  {
                     port.setRequired(interface);
                  }
               }
            }
            foreach(interface: uml21.Interface in model#iRef)
            {
               if(interface.name.contains("DataIndication"))
               {
                  port.setProvided(interface);
               }
               else if(interface.name.contains("DataRequest"))
               {
                  port.setRequired(interface);
               }
            }
         }
      }

      //Button...Software/ApplicationPort Port
      foreach(subApplicationLayer: gme.Model in applicationLayer.getModel())
      {
         if(subApplicationLayer.getKind() == "Application")
         {
            foreach(portPerimeter: gme.Model in subApplicationLayer.getModel())
            {
               if(portPerimeter.getKind() == "ApplicationPort")
               {
                  port = portPerimeter.getId()#portReferences.first();

                  foreach(interface: uml21.Interface in model#iRef)
                  {
                     if(interface.name ==
                        (CommonMacros.IProfileStandard+"Indications")
                        || interface.name ==
                        (CommonMacros.IProfileStandard+"Responses"))
                     {
                        port.setProvided(interface);
                     }
                     else if(interface.name ==
                              (CommonMacros.IProfileStandard+"Requests")
                              || interface.name ==
                              (CommonMacros.IProfileStandard+"Confirmations"))
                     {
                        port.setRequired(interface);
```

```
                        }
                    }
                    foreach(profileNode: gme.Model in mainModel.getModel())
                    {
                        if(profileNode.getKind() == "Profile")
                        {
                            foreach(interface: uml21.Interface in profileNode#iRef)
                            {
                                if(interface.name.contains("Request"))
                                {
                                    port.setRequired(interface);
                                }
                            }
                        }
                    }
                }
            }

            //Community...Entity/ApplicationPort Port
            else if(subApplicationLayer.getKind()=="CommunicationManagmentEntity")
            {
                foreach(portPerimeter: gme.Model in subApplicationLayer.getModel())
                {
                    if(portPerimeter.getKind() == "ApplicationPort")
                    {
                        port = portPerimeter.getId()#portReferences.first();


                        foreach(interfaceState: gme.Model
                                in mainModel.getProtocolAspects())
                        {
                            foreach(interface: uml21.Interface in interfaceState#iRef)
                            {
                                if(interface.name.contains("Indication")
                                    || interface.name.contains("Response"))
                                {
                                    port.setProvided(interface);
                                }
                                else
                                {
                                    port.setRequired(interface);
                                }
                            }
                        }
                    }
                }
            }

            //+DataDispatcher/Net Port
            foreach(interface: uml21.Interface in model#iRef)
            {
                if(interface.name.contains("DataIndication"))
                {
                    netPort.setProvided(interface);
                }
                else if(interface.name.contains("DataRequest"))
                {
                    netPort.setRequired(interface);
                }
            }
        }
    }
}


/* addNetworkLayer rule */
private rule addNetworkLayer(node: gme.Model, targetPkg: uml21.Package,
                             classFac: uml21.Class)
{
    var newClass: uml21.Class;
    var protocolClass: uml21.Class;
    var ccNxAClass: uml21.Class;
    var ccNxPClass: uml21.Class;
    var compositeObj: uml21.Property;
    var neastedPkg: uml21.Package;
```

```
      var nxaPort: uml21.Port;
      var nxpPort: uml21.Port;

      foreach(nwkLayer: gme.Model in node.getModel())
      {
         if(nwkLayer.getKind() == "NetworkLayer")
         {
            //create Class and Object
            newClass = targetPkg.createOwnedClass(nwkLayer.getName().getAsString(),
                                                  false);
            //global  mapping
            nwkLayer.getId()#globalMap.add(newClass);
            newClass.createObj(newClass, classFac);

            //add ports
            @addPort(nwkLayer, newClass);

            // Submodel ------------------------------------------------------------/
            //seek corresponding Package to this Class, and create subClasses
            neastedPkg = nwkLayer.getId()#pkgReferences.first();
            foreach(subNwkLayer: gme.Model in nwkLayer.getModel())
            {
               if(subNwkLayer.getKind() == "ProtocolEntity")
               {
                  protocolClass =
                     neastedPkg.createOwnedClass(subNwkLayer.getName().getAsString(),
                     false);

                  //global  mapping
                  subNwkLayer.getId()#globalMap.add(protocolClass);
                  newClass.createObj(protocolClass);

                  //add ports
                  ccNxAClass =
                           protocolClass.createNestedClassifier("ContractClassOfNxA",
                           protocolClass.eClass());
                  nxaPort = protocolClass.createOwnedPort("NxA", ccNxAClass);
                  subNwkLayer.getId()#portReferences.add(nxaPort);
                  subNwkLayer.getId()#classReferences.add(ccNxAClass);
                  nxaPort#ccReferences.add(ccNxAClass);

                  ccNxPClass =
                           protocolClass.createNestedClassifier("ContractClassOfNxP",
                           protocolClass.eClass());
                  nxpPort = protocolClass.createOwnedPort("NxP", ccNxPClass);
                  subNwkLayer.getId()#portReferences.add(nxpPort);
                  subNwkLayer.getId()#classReferences.add(ccNxPClass);
                  nxpPort#ccReferences.add(ccNxPClass);
               }
            }

            // Connections ---------------------------------------------------------/
            foreach(port: gme.Model in nwkLayer.getModel())
            {
               if(port.getKind() == "NetworkPort")
               {
                  @ConnectionTransformation(nwkLayer, newClass,
                           model).createConnection(port.getId()#portReferences.first(),
                           nxaPort);
               }
               else if(port.getKind() == "PhysicalPort")
               {
                  @ConnectionTransformation(nwkLayer, newClass,
                           model).createConnection(port.getId()#portReferences.first(),
                           nxpPort);
               }
            }

            // Port's interfaces ---------------------------------------------------/
            var port: uml21.Port;
            var realization: uml21.InterfaceRealization;
            var usage: uml21.Usage;

            //NetworkPort Port
            foreach(portPerimeter: gme.Model in nwkLayer.getModel())
            {
               if(portPerimeter.getKind() == "NetworkPort")
```

```
            {
               port = portPerimeter.getId()#portReferences.first();

               foreach(interfaceState: gme.Model in mainModel.getProtocolAspects())
               {
                  foreach(interface: uml21.Interface in interfaceState#iRef)
                  {
                     if(interface.getName().contains("Request")
                        || interface.getName().contains("Confirmation"))
                     {
                        port.setProvided(interface);
                     }
                     else if(interface.getName().contains("Indication")
                              || interface.getName().contains("Response"))
                     {
                        port.setRequired(interface);
                     }
                  }
               }
               foreach(interface: uml21.Interface in model#iRef)
               {
                  if(interface.name.contains("DataRequest"))
                  {
                     port.setProvided(interface);
                  }
                  else if(interface.name.contains("DataIndication"))
                  {
                     port.setRequired(interface);
                  }
               }
            }

            //+ProtocolEntity/NxA Port
            else if(portPerimeter.getKind() == "ProtocolEntity")
            {
               port = portPerimeter.getId()#portReferences.first();

               foreach(interfaceState: gme.Model in mainModel.getProtocolAspects())
               {
                  foreach(interface: uml21.Interface in interfaceState#iRef)
                  {
                     if(interface.getName().contains("Request")
                        || interface.getName().contains("Confirmation"))
                     {
                        port.setProvided(interface);
                     }
                     else if(interface.getName().contains("Indication")
                              || interface.getName().contains("Response"))
                     {
                        port.setRequired(interface);
                     }
                  }
               }
            }
         }
      }
   }


   /* addCommunicationHardware rule */
   private rule addCommunicationHardware(node: gme.Model, targetPkg: uml21.Package,
                                         classFac: uml21.Class)
   {
      var newClass: uml21.Class;
      foreach(comHardware: gme.Model in node.getModel())
      {
         if(comHardware.getKind() == "CommunicationHardware")
         {
            //create Class and Object
            newClass =
                     targetPkg.createOwnedClass(comHardware.getName().getAsString(),
                     false);

            //global  mapping
            comHardware.getId()#globalMap.add(newClass);
            newClass.createObj(newClass, classFac);
```

```
            //add ports
            @addPort(comHardware, newClass);
        }
    }
}


/* addPort rule */
private rule addPort(node: gme.Model, targetClass: uml21.Class)
{
    var newPort: uml21.Port;
    var ccOfTargetClass: uml21.Class;

    //search port and create it, set reference and contractClass for interfaces
    foreach(port: gme.Model in node.getModel())
    {
        if(port.getKind() == "ToApplicationPort")
        {
            ccOfTargetClass =
                            targetClass.createNestedClassifier("ContractClassOf"
                            +port.getName().getAsString(), targetClass.eClass());

            newPort =
                     targetClass.createOwnedPort(port.getName().getAsString(),
                     ccOfTargetClass);
            port.getId()#portReferences.add(newPort);
            port.getId()#classReferences.add(targetClass);
            newPort#ccReferences.add(ccOfTargetClass);
        }
        else if(port.getKind() == "PeripheralPort")
        {
            ccOfTargetClass =
                            targetClass.createNestedClassifier("ContractClassOf"
                            +port.getName().getAsString(), targetClass.eClass());
            newPort =
                     targetClass.createOwnedPort(port.getName().getAsString(),
                     ccOfTargetClass);
            port.getId()#portReferences.add(newPort);
            port.getId()#classReferences.add(targetClass);
            newPort#ccReferences.add(ccOfTargetClass);
        }
        else if(port.getKind() == "ApplicationPort")
        {
            ccOfTargetClass =
                            targetClass.createNestedClassifier("ContractClassOf"
                            +port.getName().getAsString(), targetClass.eClass());
            newPort =
                     targetClass.createOwnedPort(port.getName().getAsString(),
                     ccOfTargetClass);
            port.getId()#portReferences.add(newPort);
            port.getId()#classReferences.add(targetClass);
            newPort#ccReferences.add(ccOfTargetClass);
        }
        else if(port.getKind() == "PhysicalPort")
        {
            ccOfTargetClass =
                            targetClass.createNestedClassifier("ContractClassOf"
                            +port.getName().getAsString(), targetClass.eClass());
            newPort =
                     targetClass.createOwnedPort(port.getName().getAsString(),
                     ccOfTargetClass);
            port.getId()#portReferences.add(newPort);
            port.getId()#classReferences.add(targetClass);
            newPort#ccReferences.add(ccOfTargetClass);
        }
        else if(port.getKind() == "NetworkPort")
        {
            ccOfTargetClass =
                            targetClass.createNestedClassifier("ContractClassOf"
                            +port.getName().getAsString(), targetClass.eClass());
            newPort =
                     targetClass.createOwnedPort(port.getName().getAsString(),
                     ccOfTargetClass);
            port.getId()#portReferences.add(newPort);
            port.getId()#classReferences.add(targetClass);
            newPort#ccReferences.add(ccOfTargetClass);
```

```
         }
         else if(port.getKind() == "ToNetworkPort")
         {
            ccOfTargetClass =
                              targetClass.createNestedClassifier("ContractClassOf"
                              +port.getName().getAsString(), targetClass.eClass());
            newPort =
                     targetClass.createOwnedPort(port.getName().getAsString(),
                     ccOfTargetClass);
            port.getId()#portReferences.add(newPort);
            port.getId()#classReferences.add(targetClass);
            newPort#ccReferences.add(ccOfTargetClass);
         }
      }

      //port can be an Atom instead of Model
      foreach(port: gme.Model in node.getAtom())
      {
         if(port.getKind() == "PeripheralPerimeter")
         {
            ccOfTargetClass =
                              targetClass.createNestedClassifier("ContractClassOf"
                              +port.getName().getAsString(), targetClass.eClass());
            newPort =
                     targetClass.createOwnedPort(port.getName().getAsString(),
                     ccOfTargetClass);
            port.getId()#portReferences.add(newPort);
            port.getId()#classReferences.add(targetClass);
            newPort#ccReferences.add(ccOfTargetClass);
         }
         else if(port.getKind() == "ApplicationPerimeter")
         {
            ccOfTargetClass =
                              targetClass.createNestedClassifier("ContractClassOf"
                              +port.getName().getAsString(), targetClass.eClass());
            newPort =
                     targetClass.createOwnedPort(port.getName().getAsString(),
                     ccOfTargetClass);
            port.getId()#portReferences.add(newPort);
            port.getId()#classReferences.add(targetClass);
            newPort#ccReferences.add(ccOfTargetClass);
         }
      }
   }
}
```

B.6 ProfileTransformation

```
/**
*ProfileTransformation creates profiles for services in network
*written by Philippe Pralong
*/

package org.ads;

import org.ads.tools.CommonMacros;
import com.sodius.mdw.metamodel.uml21.VisibilityKind;

public ruleset ProfileTransformation(target: uml21.Package, in model: uml21,
                                     pkgInterfaces: uml21.Package)
{

   public rule main(parentNet: gme.Model)
   {
      //create base profile
      var baseProfile: uml21.Class = target.createOwnedClass("BaseProfile", false);
      var interfaceForProfile: uml21.Interface;
      var iType: uml21.Interface = model.create("Interface");

      //profileM Model used to accelerate research of profile's Interfaces
      var profileM: gme.Model;
      foreach(profileNode: gme.Model in parentNet.getModel())
      {
         if(profileNode.getKind() == "Profile")
         {
            profileM = profileNode;
         }
      }

      foreach(node: gme.Model in parentNet.getModel())
      {
         if(node.getKind() == "Profile")
         {
            //create profile for service in model
            var profile: uml21.Class =
                                target.createOwnedClass(node.getName().getAsString()
                                +"Profile", false);
            node.getId()#globalMap.add(profile);

            //add Interfaces
            foreach(state: String in CommonMacros.UML_INTERFACES)
            {
               interfaceForProfile =
                                   profile.createNestedClassifier("I"
                                   +profile.name+state, iType.eClass());

               node#iRef.add(interfaceForProfile);

            }

            //add methods to Interfaces (check if service is confirmed or not)
            foreach(service: gme.Connection in node.getConnection())
            {
               foreach(attr: gme.Attribute in service.getAttribute())
               {
                  if(attr.getKind() == "Confirmed")
                  {
                     profile.createInterfaceOperation(service,
                                                      attr.value.getAsBoolean());
                  }
               }
            }

            //add Attributes
            var intType: uml21.Type = target.createOwnedPrimitiveType("int");
            var OMStringType: uml21.Type =
                                          target.createOwnedPrimitiveType("OMString");
            var mProfileID: uml21.Property =
                     baseProfile.createOwnedAttribute("mProfileID", intType);
            var mProfileName: uml21.Property =
                     baseProfile.createOwnedAttribute("mProfileName", OMStringType);
            mProfileID.setIntegerDefaultValue(0);
```

```
mProfileID.setIsReadOnly(true);
mProfileID.setVisibility(VisibilityKind.PRIVATE_LITERAL);
mProfileName.setIsReadOnly(true);
mProfileName.setVisibility(VisibilityKind.PRIVATE_LITERAL);

//add Generalizations
profile.createGeneralization(baseProfile);

foreach(interface: uml21.Interface in profileM#iRef)
{
   if(interface.name.contains("Request"))
   {
      profile.createGeneralization(interface);
   }
}

foreach(interface: uml21.Interface in model#iRef)
{
   if(interface.name == (CommonMacros.IProfileStandard+"Requests"))
   {
      profile.createGeneralization(interface);
      baseProfile.createGeneralization(interface);
   }
   else if(interface.name
            == (CommonMacros.IProfileStandard+"Confirmations"))
   {
      profile.createGeneralization(interface);
   }
   else if(interface.name == (CommonMacros.IProfileStandard+"Responses"))
   {
      baseProfile.createGeneralization(interface);
   }
}


//add Operations
var op: uml21.Operation;

baseProfile.createOwnedOperation("dataIndication", null, null);

foreach(interface: uml21.Interface in profileM#iRef)
{
   if(!interface.ownedOperation.isEmpty())
   {
      foreach(operation: uml21.Operation in interface.getOperations())
      {
         if(operation.name.contains("Request"))
         op = profile.createOwnedOperation(operation.name, null, null);
         op.setIsAbstract(true);
      }
   }
}


//add Ports
var ccProfile: uml21.Class =
               profile.createNestedClassifier("ContractClassOfProfilePort",
               profile.eClass());
var profilePort: uml21.Port = profile.createOwnedPort("App", ccProfile);
profilePort.setIsBehavior(true);
profilePort#ccReferences.add(ccProfile);


//add port's interfaces
foreach(interface: uml21.Interface in profileM#iRef)
{
   if(interface.name.contains("Request"))
   {
      profilePort.setProvided(interface);
   }
}

foreach(interface: uml21.Interface in model#iRef)
{
   if(interface.name == (CommonMacros.IProfileStandard+"Requests")
      || interface.name==(CommonMacros.IProfileStandard+"Confirmations"))
   {
```

```
                profilePort.setProvided(interface);
            }
            else if(interface.name==(CommonMacros.IProfileStandard+"Indications")
                    ||interface.name==(CommonMacros.IProfileStandard+"Responses"))
            {
                profilePort.setRrequired(interface);
            }
        }
    }
    iType.destroy();
  }
}
```

B.7 CommonMacro

```
/**
*CommonMacro used to call global variables
*written by Philippe Pralong and Rico Steiner
*/

package org.ads.tools;

public class CommonMacros
{
   //GME2UML global variables
   public static final String[] UML_INTERFACES     = {"Requests", "Indications", "Responses",
                                                      "Confirmations"};
   public static final String IProfileStandard     = "IProfileStandard";
   public static final String INetwork             = "INWK";
}
```

B.8 StaticMDWList

```
/**
*StaticMDWList used as list of elements
*written by Rico Steiner
*/

package org.ads.tools;

import com.sodius.mdw.core.model.DefaultMDWList;

public class StaticMDWList extends DefaultMDWList
{
   private static final long serialVersionUID = 1L;

   public static DefaultMDWList create()
   {
      return new DefaultMDWList();
   }
}
```

B.9 Scripts (Transformation)

B.9.1 Script pour gme.Model

```
/**
*Script for metatype gme.Model
*written by Philippe Pralong
*/

package org.ads;

import org.ads.tools.*;
import com.sodius.mdw.core.model.MDWList;

metatype gme.Model;

/* getProtocolAspects(): MDWList */
public script getProtocolAspects(): MDWList
{
   var protocolAspects: MDWList = StaticMDWList.create();

   foreach(networkNode: gme.Model in self.getModel())
   {
      if(networkNode.getKind() == "Network")
      {
         foreach(protocol: gme.Model in networkNode.getModel())
         {
            if(protocol.getKind() == "NetworkProtocol")
            {
               foreach(interfaceState: gme.Model in protocol.getModel())
               {
                  protocolAspects.add(interfaceState);
               }
            }
         }
      }
   }
   return protocolAspects;
}
```

B.9.2 Script pour gme.Name

```
/**
*Script for metatype gme.Name
*written by Rico Steiner
*/

package org.ads;

metatype gme.Name;

/* getAsString() */
public script getAsString()
{
   var retVal: String = "";

   foreach(a: Object in self.getMixed())
   {
      retVal = retVal + a.getValue();
   }

   return retVal;
}
```

B.9.3 Script pour gme.Value

```
/**
*Script for metatype uml21.Class
*written by Philippe Pralong
*/

package org.ads;

metatype gme.Value;

/* getAsString() */
public script getAsString()
{
   var retVal: String = "";

   foreach(a: Object in self.getMixed())
   {
      retVal = retVal + a.getValue();
   }

   return retVal;
}

/* getAsBoolean */
public script getAsBoolean()
{
   if(self.getAsString() == "true")
      return true;
   else if(self.getAsString() == "false")
      return false;
}
```

B.9.4 Script pour uml21.Package

```
/**
*Script for metatype uml21.Package
*written by Philippe Pralong
*/

package org.ads;

import org.ads.tools.*;
import com.sodius.mdw.core.model.MDWList;

metatype uml21.Package;

/*getInterfaces(): MDWList */
public script getInterfaces(): MDWList
{
   var interfacesList: MDWList = StaticMDWList.create();

   foreach(interface: uml21.Interface in self.getMembers())
   {
      if(interface instanceof uml21.Interface)
      {
         interfacesList.add(interface);
      }
   }
   return interfacesList;
}

/* getPackages(): MDWList*/
public script getPackages(): MDWList
{
   var packagesList: MDWList = StaticMDWList.create();

   foreach(pkg: uml21.Package in self.getMembers())
   {
      if(pkg instanceof uml21.Package)
      {
         packagesList.add(pkg);
      }
   }
   return packagesList;
```

```
}

/* getClasses(): MDWList */
public script getClasses(): MDWList
{
   var classesList: MDWList = StaticMDWList.create();

   foreach(class: uml21.Class in self.getMembers())
   {
      if(class instanceof uml21.Class)
      {
         classesList.add(class);
      }
   }
   return classesList;
}

/* getClass(String): uml21.Class */
public script getClass(name: String): uml21.Class
{
   var classesList: MDWList = self.getClasses();

   foreach(class: uml21.Class in classesList)
   {
      if(class.getName() == name)
      {
         return class;
      }
   }
   return null;
}
```

B.9.5 Script pour uml21.Class

```
/**
*Script for metatype uml21.Class
*written by Philippe Pralong
*/

package org.ads;

import org.ads.tools.*;
import com.sodius.mdw.core.model.MDWList;

metatype uml21.Class;

/* getNbrOfAttr(): int */
public script getNbrOfAttr(): int
{
   return self.attribute.size();
}

/* createObj(uml21.Class, uml21.Class) */
public script createObj(class: uml21.Class, factory: uml21.Class)
{
   var classToCompose: uml21.Class;
   var compositeObj: uml21.Property;

   classToCompose = class;
   compositeObj =
                  factory.createOwnedAttribute(class.setNameComposite(),
                  classToCompose);
   compositeObj.setIsComposite(true);
   classToCompose#compObjRef.add(compositeObj);
}

/* createObj(uml21.Class) */
public script createObj(class: uml21.Class)
{
   var classToCompose: uml21.Class;
   var compositeObj: uml21.Property;

   classToCompose = class;
   compositeObj =
                  self.createOwnedAttribute(class.setNameComposite(),
                  classToCompose);
```

```
   compositeObj.setIsComposite(true);
   classToCompose#compObjRef.add(compositeObj);
}

/* setNameComposite() */
public script setNameComposite(): String
{
   return "its"+self.getName();
}

/* getInterfaces(): MDWList */
public script getInterfaces(): MDWList
{
   var interfacesList: MDWList = StaticMDWList.create();

   foreach(interface: uml21.Interface in self.getMembers())
   {
      if(interface instanceof uml21.Interface)
      {
         interfacesList.add(interface);
      }
   }
   return interfacesList;
}

/* getPorts() */
public script getPorts(): MDWList
{
   var portsList: MDWList = StaticMDWList.create();

   foreach(port: uml21.Port in self.getMembers())
   {
      if(port instanceof uml21.Port)
      {
         portsList.add(port);
      }
   }
   return portsList;
}

/* getPort(gme.Model, String): uml21.Port */
public script getPort(node: gme.Model, portKind: String): uml21.Port
{
   foreach(element: gme.Model in node.getModel())
   {
      if(element.getKind() == portKind)
      {
         return element.getId()#portReferences.first();
         System.out.println(element.getId()#portReferences.first());
      }
   }
   foreach(element: gme.Atom in node.getAtom())
   {
      if(element.getKind() == portKind)
      {
         return element.getId()#portReferences.first();
      }
   }
   return null;
}

/* createInterfaceOperation(gme.Connection, Boolean) */
public script createInterfaceOperation(service: gme.Connection, val: Boolean)
{
   var iOperation: uml21.Operation;

   foreach(interface: uml21.Interface in self.getInterfaces())
   {
      if(val)
      {
         if(interface.getName().contains("Request"))
            iOperation =
                        interface.createOwnedOperation(service.getName().getAsString()
                        +"Request", null, null);
         else if(interface.getName().contains("Indication"))
            iOperation =
                        interface.createOwnedOperation(service.getName().getAsString()
```

```
                        +"Indication", null, null);
         else if(interface.getName().contains("Response"))
            iOperation =
                        interface.createOwnedOperation(service.getName().getAsString()
                        +"Response", null, null);
         else if(interface.getName().contains("Confirmation"))
            iOperation =
                        interface.createOwnedOperation(service.getName().getAsString()
                        +"Confirmation", null, null);
      }
      else
      {
         if(interface.getName().contains("Request"))
            iOperation =
                        interface.createOwnedOperation(service.getName().getAsString()
                        +"Request", null, null);
         else if(interface.getName().contains("Indication"))
            iOperation =
                        interface.createOwnedOperation(service.getName().getAsString()
                        +"Indication", null, null);
      }
      iOperation.setIsAbstract(true);
   }
}

/* connectionDestroy(String) */
public script connectionDestroy(name: String)
{
   var connecToDestroy: uml21.Connector;

   foreach(connection: uml21.Connector in self.getMembers())
   {
      if(connection instanceof uml21.Connector)
      {
         if(connection.name == name)
         {
            connecToDestroy = connection;
         }
      }
   }
   connecToDestroy.destroy();
}
```

B.9.6 Script pour uml21.Port

```
/**
*Script for metatype uml21.Port
*written by Philippe Pralong
*/

package org.ads;

metatype uml21.Port;

/* setProvided(uml21.Interface) */
public script setProvided(interface: uml21.Interface)
{
   var realization: uml21.InterfaceRealization = self.eModel().create("InterfaceRealization");
   realization.setContract(interface);
   realization.setImplementingClassifier(self#ccReferences.first());
}

/* setRequired(uml21.Interface) */
public script setRequired(interface: uml21.Interface)
{
   var usage: uml21.Usage = self.eModel().create("Usage");
   usage.supplier.add(interface);
   usage.client.add(self#ccReferences.first());

   //Add Usage to package of Port, to list dependencies in the contract Class
   var pkg: uml21.Package = self.owner.owner;
   pkg.packagedElement.add(usage);
}
```

B.10 Diagram Populater

Code concernant la création des diagrammes UML.

B.10.1Elements2Diagram

```
/**
*Elements2Diagram is the main file of diagrams creation
*written by Rico Steiner and Philippe Pralong
*/

package org.ads;

public ruleset Elements2Diagram(inout rhp: rhapsody)
{
   public rule main()
   {
      var pkgProfile: rhapsody.Package;
      var diagram: rhapsody.ObjectModelDiagram;
      var appClass: rhapsody.Class;
      var netClass: rhapsody.Class;

      foreach(pkg: rhapsody.Package in rhp.getInstances("Package"))
      {
         if(pkg.getName() == "Profiles_Pkg")
         {
            pkgProfile = pkg;
         }
         else if(pkg.getName() == "ButtonNode_Pkg")
         {
            //ObjectModelDiagram (ButtonNode)
            diagram = rhp.create("ObjectModelDiagram");
            diagram.setName("OMD_"+pkg.getName());
            pkg.objectModelDiagrams.add(diagram);

            foreach(class: rhapsody.Class in pkg.classes)
            {
               if(class.isComposite)
               {
                  if(class.name == "ApplicationLayer")
                  {
                     appClass = class;
                  }
                  else if(class.name == "ZigbeeStack")
                  {
                     netClass = class;
                  }
                  else if(class.name == "ButtonNodeFactory")
                  {
                     diagram.addFactory(class, true, false);

                     foreach(relation: rhapsody.Relation in class.getRelations())
                     {
                        diagram.addObject(relation, true);
                     }

                     foreach(link: rhapsody.Link in class.getLinks())
                     {
                        diagram.addLink(link);
                     }
                  }
               }
            }

            //ClassModelDiagram (ButtonNode)
            diagram = rhp.create("ObjectModelDiagram");
            diagram.setName("CMD_"+pkg.getName());
            pkg.objectModelDiagrams.add(diagram);

            foreach(class: rhapsody.Class in pkg.classes)
            {
               if(class.name == "ButtonNodeFactory")
               {}
               else
               {
```

```
                diagram.addClass(class, false, false, true);
            }
        }

        foreach(subPkg: rhapsody.Package in pkg.getPackages())
        {
            if(subPkg.getName() == "ApplicationLayer_Pkg")
            {
                //ObjectModelDiagram (ApplicationLayer)
                diagram = rhp.create("ObjectModelDiagram");
                diagram.setName("OMD_"+subPkg.getName());
                subPkg.objectModelDiagrams.add(diagram);

                diagram.addFactory(appClass, true, true);

                foreach(relation: rhapsody.Relation in appClass.getRelations())
                {
                    diagram.addObject(relation, true);
                    subPkg#relations.add(relation);
                }

                foreach(link: rhapsody.Link in appClass.getLinks())
                {
                    diagram.addLink(link);
                    subPkg#links.add(link);
                }

                //ClassModelDiagram (ApplicationLayer)
                diagram = rhp.create("ObjectModelDiagram");
                diagram.setName("CMD_"+subPkg.getName());
                subPkg.objectModelDiagrams.add(diagram);

                foreach(class: rhapsody in subPkg.getClasses())
                {
                    diagram.addClass(class, false, false, true);
                }

                foreach(profileClass: rhapsody.Class in pkgProfile.getClasses())
                {
                    if(profileClass.getName() == "BaseProfile")
                    {
                        diagram.addClass(profileClass, false, false, true);
                    }
                }
                foreach(class: rhapsody.Class in subPkg.getClasses())
                {
                    foreach(relation: rhapsody.Relation in class.getRelations())
                    {
                        diagram.addAssociation(relation);
                    }
                }
            }

            if(subPkg.getName() == "ZigbeeStack_Pkg")
            {
                //ObjectModelDiagram (ZigbeeStack)
                diagram = rhp.create("ObjectModelDiagram");
                diagram.setName("OMD_"+subPkg.getName());
                subPkg.objectModelDiagrams.add(diagram);

                diagram.addFactory(netClass, true, true);

                foreach(relation: rhapsody.Relation in netClass.getRelations())
                {
                    diagram.addObject(relation, true);
                    subPkg#relations.add(relation);
                }

                foreach(link: rhapsody.Link in netClass.getLinks())
                {
                    diagram.addLink(link);
                    subPkg#links.add(link);
                }

                //ClassModelDiagram (ZigbeeStack)
                diagram = rhp.create("ObjectModelDiagram");
                diagram.setName("CMD_"+subPkg.getName());
```

```
                    subPkg.objectModelDiagrams.add(diagram);

                    foreach(class: rhapsody in subPkg.getClasses())
                    {
                       diagram.addClass(class, false, false, true);
                    }
                }
            }
        }
    }
}
```

B.10.2Scripts pour rhapsody.Diagram

```
/**
*Script for metatype rhapsody.Diagram
*written by Rico Steiner and Philippe Pralong
*/

package org.ads;

metatype rhapsody.Diagram;

public script beginNewDiagram()
{
   self#mGraphNodes.clear();
   self#mGraphEdges.clear();
}

public script addFactory(class: rhapsody.Class, showAsCompsite: boolean, showPorts: boolean)
{
   var node: rhapsody.GraphNode = self.addNode(class);

   var xPos: int = self.getNodeInitialX();
   var yPos: int = self.getNodeInitialY();
   var xCnt: int = self.graphicalElements.size()%(self.getMaxNodePerRow()-1);
   var yCnt: int = Integer.valueOf(self.graphicalElements.size()/self.getMaxNodePerRow());

   xPos = 20;
   yPos = 20;

   node.addGraphicalProperty("Height", Integer.toString(self.getNodeHeightFac()));
   node.addGraphicalProperty("Width", Integer.toString(self.getNodeWidthFac()));
   node.addGraphicalProperty("Position", Integer.toString(xPos) + ","
                             +Integer.toString(yPos));

   if(showAsCompsite && class.isComposite)
   {
      node.addGraphicalProperty("StructureView", "true");
   }

   if(showPorts)
   {
      foreach(p: rhapsody.Port in class.ports)
      {
         System.out.println("found object port\n------------------");
         var port: rhapsody.GraphNode = self.addNode(p);
         node#portRef.add(port);

         var yCntPort: int = node#portRef.size();

         yPos = yPos + (yCntPort-1) * (self.getPortSize() + self.getPortSpacingY());

         port.addGraphicalProperty("Height", Integer.toString(self.getPortSize()));
         port.addGraphicalProperty("Width", Integer.toString(self.getPortSize()));
         port.addGraphicalProperty("Position", Integer.toString(xPos) + ","
                                   +Integer.toString(yPos));
      }
   }
}

public script addClass(class: rhapsody.Class, showAsCompsite: boolean, showBig: boolean,
showPorts: boolean)
{
   var node: rhapsody.GraphNode = self.addNode(class);
```

```
   self#classRef.add(class);

   var xPos: int = self.getNodeInitialX();
   var yPos: int = self.getNodeInitialY();
   var xCnt: int = (self.getNbrOfClasses()-1)%self.getMaxNodePerRow();
   var yCnt: int = Integer.valueOf((self.getNbrOfClasses()-1)/self.getMaxNodePerRow());

   xPos = xPos + xCnt * (self.getNodeWidth() + self.getNodeSpacingX());
   yPos = yPos + yCnt * (self.getNodeHeight() + self.getNodeInitialY());

   if(showAsCompsite && class.isComposite)
   {
      node.addGraphicalProperty("StructureView", "true");
   }

   if(showBig)
   {
      node.addGraphicalProperty("Height", Integer.toString(self.getNodeHeightBig()));
      node.addGraphicalProperty("Width", Integer.toString(self.getNodeWidthBig()));
      node.addGraphicalProperty("Position", Integer.toString(xPos) + ","
                                +Integer.toString(yPos));
   }
   else
   {
      node.addGraphicalProperty("Height", Integer.toString(self.getNodeHeight()));
      node.addGraphicalProperty("Width", Integer.toString(self.getNodeWidth()));
      node.addGraphicalProperty("Position", Integer.toString(xPos) + ","
                                +Integer.toString(yPos));
   }

   if(showPorts)
   {
      foreach(p: rhapsody.Port in class.ports)
      {
         var port: rhapsody.GraphNode = self.addNode(p);
         port.addGraphicalProperty("Height", Integer.toString(self.getPortSize()));
         port.addGraphicalProperty("Width", Integer.toString(self.getPortSize()));
         port.addGraphicalProperty("Position", Integer.toString(xPos) + ","
                                   +Integer.toString(yPos));
      }
   }
}

public script addObject(instance: rhapsody.Instance, showPorts: boolean)
{
   var node: rhapsody.GraphNode = self.addNode(instance);
   self#objRef.add(instance);

   var xPos: int = self.getNodeInitialX();
   var yPos: int = self.getNodeInitialY();
   var xCnt: int = (self.getNbrOfObjects()-1)%self.getMaxNodePerRow();
   var yCnt: int = Integer.valueOf((self.getNbrOfObjects()-1)/self.getMaxNodePerRow());

   xPos = xPos + xCnt * (self.getNodeWidth() + self.getNodeSpacingX());
   yPos = yPos + yCnt * (self.getNodeHeight() + self.getNodeSpacingY());

   node.addGraphicalProperty("Height", Integer.toString(self.getNodeHeight()));
   node.addGraphicalProperty("Width", Integer.toString(self.getNodeWidth()));
   node.addGraphicalProperty("Position", Integer.toString(xPos) + ","
                             +Integer.toString(yPos));

   if(showPorts)
   {
      foreach(p: rhapsody.Port in instance.otherClass.ports)
      {
         System.out.println("found object port\n------------------");
         var port: rhapsody.GraphNode = self.addNode(p);
         node#portRef.add(port);

         var yCntPort: int = node#portRef.size();

         yPos = yPos + (yCntPort-1) * (self.getPortSize() + self.getPortSpacingY());

         port.addGraphicalProperty("Height", Integer.toString(self.getPortSize()));
         port.addGraphicalProperty("Width", Integer.toString(self.getPortSize()));
         port.addGraphicalProperty("Position", Integer.toString(xPos) + ","
                                   +Integer.toString(yPos));
```

```
      }
   }
}


private script addNode(node: rhapsody.ModelElement): rhapsody.GraphNode
{
// if(node#mGraphNodes.isEmpty())
// {
      System.out.println("ok\n------------------");
      var gn: rhapsody.GraphNode = self.eModel().create("GraphNode");
      gn.setModelObject(node);
      node#mGraphNodes.add(gn);

      self.graphicalElements.add(gn);
      return gn;
// }
/* else
   {  System.out.println("not ok\n------------------");
      return node#mGraphNodes.first();
   }
*/
}

public script addAssociation(association: rhapsody.Relation)
{
   if(association.inverse#mGraphEdges.isEmpty() && association.eClass().getName() ==
"Relation")
   {
      self.addEdge(association, association.ofClass, association.otherClass);
   }
}

public script addLink(link: rhapsody.Link)
{
   if(link#mGraphEdges.isEmpty())
   {
      var source: rhapsody.Instance = null;
      var target: rhapsody.Instance = null;

      if(link.fromPort instanceof rhapsody.Port)
      {
         source = link.fromPort;
      }
      else
      {
         source = link.from;
      }

      if(link.toPort instanceof rhapsody.Port)
      {
         target = link.toPort;
      }
      else
      {
         target = link.to;
      }

      self.addEdge(link, source, target);
   }
}

private script addEdge(edge: rhapsody.ModelElement, source: rhapsody.ModelElement,
                       target: rhapsody.ModelElement): rhapsody.GraphEdge
{
   var srcX: int = 0;
   var srcY: int = 0;
   var dstX: int = 0;
   var dstY: int = 0;
   var point1: String = "";
   var point2: String = "";
   var srcPosition: String;
   var dstPosition: String;
   var yedgeCnt: int = self.getNbrOfEdges();
   //var yEdgeCnt: int = Integer.valueOf((self.getNbrOfEdges())/self.getMaxEdgePerRow());
   var ge: rhapsody.GraphEdge  = self.eModel().create("GraphEdge");
   ge.setModelObject(edge);
```

```
   edge#mGraphEdges.add(ge);

   self.graphicalElements.add(ge);
   self#edgeRef.add(edge);

   //take the "last", because some port can be used in several diagrams
   ge.source = source#mGraphNodes.last();
   ge.target = target#mGraphNodes.last();

   //search X and Y coord of source and destination, to redraw edge
   foreach(gp: rhapsody.GraphicalProperty in ge.source.graphicalProperties)
   {
      if(gp.key == "Position")
      {
         srcPosition = gp.value;
         srcX = Integer.parseInt(gp.value.substring(0,gp.value.indexOf(",")));
         srcY = Integer.parseInt(gp.value.substring(gp.value.indexOf(",")+1));
      }
   }
   foreach(gp: rhapsody.GraphicalProperty in ge.target.graphicalProperties)
   {
      if(gp.key == "Position")
      {
         dstPosition = gp.value;
         dstX = Integer.parseInt(gp.value.substring(0,gp.value.indexOf(",")));
         dstY = Integer.parseInt(gp.value.substring(gp.value.indexOf(",")+1));
      }
   }

   //Compose intermediate points of edge
   srcY = srcY + yedgeCnt * self.getPortSize();
   point1 = Integer.toString(srcX)+","+Integer.toString(srcY);
   dstY = srcY;
   point2 = Integer.toString(dstX)+","+Integer.toString(dstY);

// ge.addGraphicalProperty("LineStyle","Straight");
   ge.addGraphicalProperty("Polygon","4,"+srcPosition+","+point1+","+point2+","+dstPosition);

   return ge;
}

private script getNbrOfClasses(): int
{
   return self#classRef.size();
}
private script getNbrOfObjects(): int
{
   return self#objRef.size();
}
private script getNbrOfPorts(): int
{
   return self#portRef.size();
}
private script getNbrOfEdges(): int
{
   return self#edgeRef.size();
}
private script getNodeWidth(): int
{
   return 150;
}
private script getNodeHeight(): int
{
   return 150;
}
private script getNodeWidthBig(): int
{
   return 300;
}
private script getNodeHeightBig(): int
{
   return 225;
}
private script getNodeWidthFac(): int
{
   return 825;
}
```

```
private script getNodeHeightFac(): int
{
   return 725;
}
private script getNodeSpacingX(): int
{
   return 40;
}
private script getNodeSpacingY(): int
{
   return 40;
}
private script getNodeInitialX(): int
{
   return 50;
}
private script getNodeInitialY(): int
{
   return 100;
}
private script getMaxNodePerRow(): int
{
   return 4;
}
private script getMaxEdgePerRow(): int
{
   return 8;
}
private script getPortSize(): int
{
   return 60;
}
private script getPortSpacingY(): int
{
   return 5;
}
```

B.10.3Script pour rhapsody.GraphElement

```
/**
*Script for metatype rhapsody.GraphElement
*written by Rico Steiner

package org.ads;

metatype rhapsody.GraphElement;

public script addGraphicalProperty(key: String, value: String)
{
   var gp: rhapsody.GraphicalProperty = self.eModel().create("GraphicalProperty");
   gp.key = key;
   gp.value = value;
   self.graphicalProperties.add(gp);
}
```