République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Aboubekr BELKAÏD - TLEMCEN
Faculté des Sciences
Département d' Informatique

TITRE

# MODÉLISATION AVEC LE LANGAGE UML

Adressé aux étudiants niveau : Licences ( L 3)

Domaine : MI
Filière : Systèmes Informatiques
Spécialité : Informatique

Etabli Par :

Amal HALFAOUI ép. GHERNAOUT

Année 2021/2022

# Table des matières

# Liste des Figures

# Liste des Tableaux

# Avant-propos

Ce polycopié s'adresse aux étudiants de licence de la filière informatique. Les séries de travaux pratiques proposées concernent la matière Génie Logiciel et en particulier la modélisation avec le langage UML (*Unified Modeling Language)*. Cette matière est enseignée au département informatique pour la promotion troisième année de la filière systèmes informatiques de l'université de Tlemcen.

UML est un standard adopté par OMG[1], c'est un langage de modélisation graphique à base de pictogrammes, il est utilisé pour visualiser la conception d'un système. Il est souvent utilisé en conception orientée objet et en développement logiciel. De nombreux outils destinés à la modélisation graphique du langage UML existent. Nous utilisons dans ce polycopié Modelio[2] qui est un outil de modélisation UML open source qui supporte la plupart des diagrammes spécifiés par UML 2.0. Modelio propose aussi des extensions qui offrent des fonctionnalités comme des générateurs de code (Java, C++, C#) et générateurs de documentations.

La modélisation avec UML est basée sur des diagrammes et des pictogrammes, il est important de proposer dans les séries de TP des tutoriaux qui permettent, à chaque fois, de réaliser le diagramme pas à pas. En effet, ceci permettra aux étudiants de se familiariser avec l'environnement Modelio. Pour cela, nous proposons dans chaque série de Tp un tutoriel qui est, ensuite, suivi d'un exercice d'application qui permettra de tester les connaissances acquises des étudiants. Chaque série de Tp contient aussi, à son début, un petit rappel de cours accompagné d'un aide-mémoire (mémento) qui présente les différents éléments du diagramme étudié.

UML présente 14 diagrammes dans la version actuelle 2.5[3]. Nous aborderons dans ce polycopié :

- Le diagramme de cas d'utilisation ;
- Le diagramme de séquence avec ses deux versions d'utilisation :
    i. Digramme de séquence système (boite noire) ;

[1] Object Management Group https://www.omg.org/
[2] https://www.modelio.org/
[3] La dernière version publiée par OMG (Décembre 2017) : https://www.omg.org/spec/UML/2.5.1/

ii. Diagramme de séquence de conception (boite blanche);

- Le diagramme de classe avec ses deux versions d'utilisation :
  i. Diagramme de classes métiers ;
  ii. Diagramme de classes participantes;
- Le diagramme d'activité.

Le polycopié présente 6 séries de TP :

TP 1 : Prise en main de l'outil Modelio ;

TP 2 : Diagrammes de cas d'utilisation et la description textuelle ;

TP 3 : Description des cas d'utilisation : Diagramme de Séquence Système ;

TP 4 : Description de l'aspect statique : Diagramme de classes métier ;

TP 5 : Diagramme de séquence (boite blanche) et son Diagramme de classes participantes;

TP 6 : Diagramme d'activité.

Notons que ce polycopié est accompagné d'un CD qui contient tous les projets Modelio réalisés au cours des séries de TP à savoir :

- Le projet Modélio appelé 'TP_SGDC' qui est l'étude du 'Système de Gestion de Demande de Congé' qui sera réalisée, pas à pas et au fur et à mesure, à travers toutes les séries, dans la partie 'Pratiquer : tutoriel à suivre'
- Le projet Modélio appelé 'Exercices d'applications' qui contient les solutions de chaque exercice donné à la fin de chaque série.

Le CD contient aussi la dernière version installable du logiciel Modelio.

# TP1 : Prise en main de l'outil Modelio

## 1.1. Objectif et rappels

L'objectif de cette série est de créer un projet et le structurer pour accueillir plusieurs diagrammes selon plusieurs vues du système à modéliser.

La modélisation permet de mieux comprendre le fonctionnement d'un système. C'est un moyen qui permet de maîtriser la complexité et d'assurer la cohérence d'un système. Dans un développement logiciel, un modèle est une représentation, à différents niveaux d'abstraction et selon plusieurs vues, de l'information nécessaire à la production et à l'évolution de l'application. Un modèle est un langage précis, commun, qui est connu par tous les membres de l'équipe de développement, il représente un moyen primordial pour communiquer.

UML offre un ensemble de diagrammes permettant de modéliser le logiciel selon différentes vues (Exemple : vue dynamique ou statique) et différents niveaux d'abstraction (analyse, conception) [9]. C'est un langage de modélisation orientée objet.

### *1.1.1. Modèle vs. Vue vs Diagramme*

- **Un modèle UML :** représente l'ensemble de tous les éléments modélisés. La présentation d'un modèle UML se compose de plusieurs documents de deux sortes : i- documents écrits en langage courant ; ii- documents formalisés (Les diagrammes)

- **Vue** : aborde un modèle (sous forme d'un diagramme) suivant une perspective qui se focalise que sur les éléments pertinents pour cette perspective. ex. : vue statique, vu fonctionnelle…,etc.

- **Un diagramme** est un dessin (représentation graphique) montrant une partie du modèle (graphique).

### *1.1.2. Diagrammes UML*

UML contient dans la version actuelle (2.5) 14 diagrammes. La figure 1-1 montre ceux qui appartiennent à la version 1.X et ceux qui ont été ajoutés à partir de la version 2

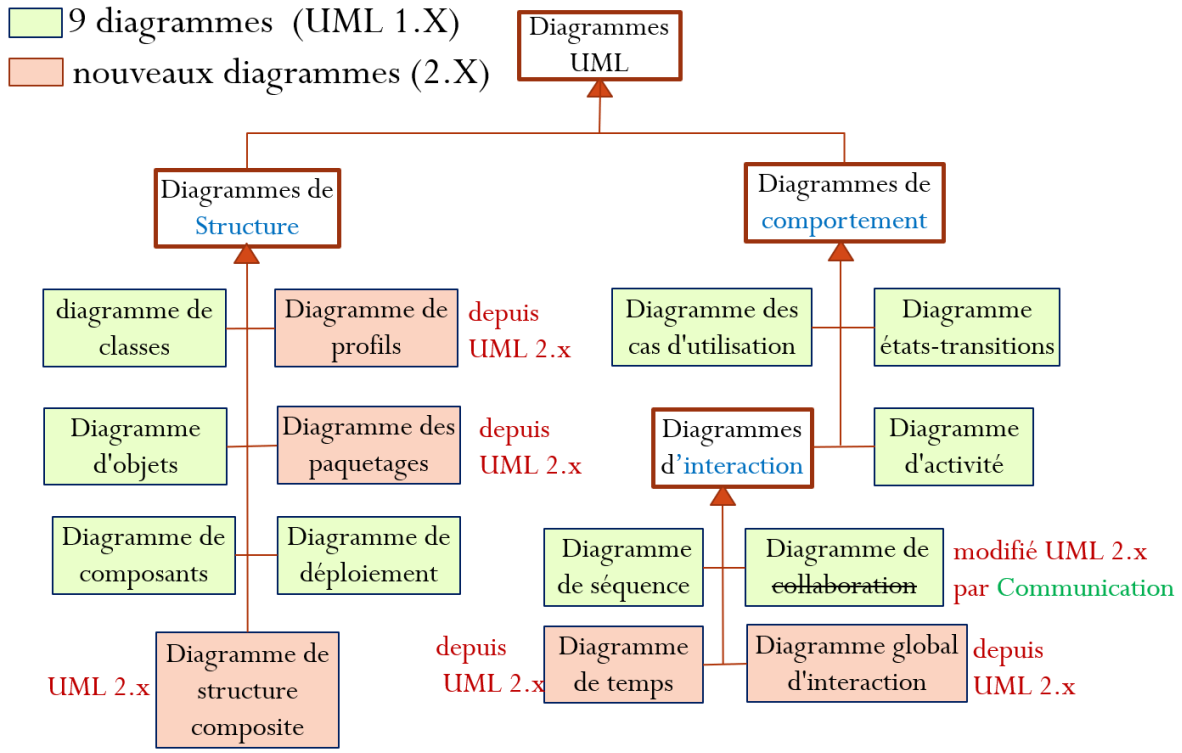



*Figure 1-1 : Diagrammes UML 1.X et UML 2.X*

Ces diagrammes peuvent exprimer l'aspect fonctionnel, dynamique ou statique d'un système (Figure 1-2)

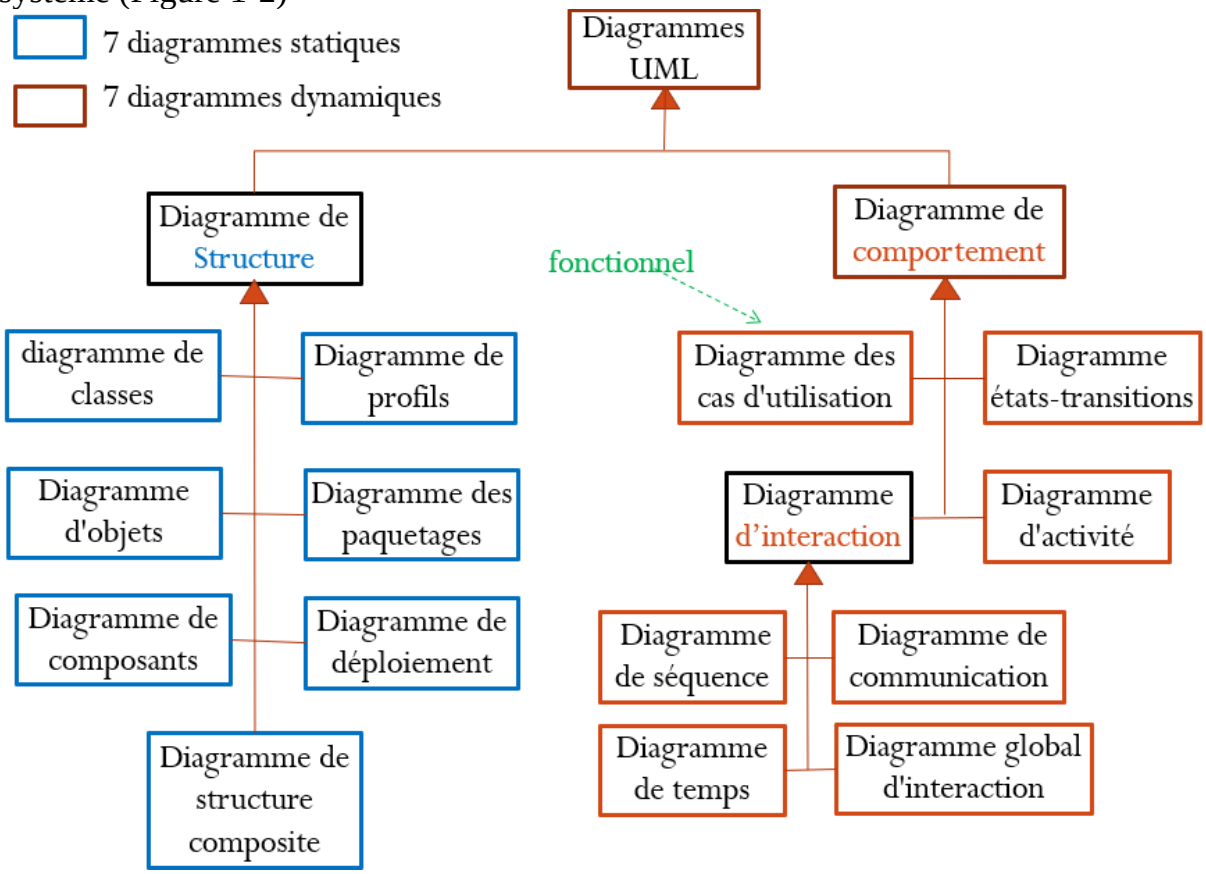



*Figure 1-2 : Diagrammes selon l'axe fonctionnel, statique ou dynamique*

### *1.1.3. Diagrammes et le modèle (Vues 4+1)*

La conception d'un logiciel est organisée autour des aspects fonctionnels et d'architecture qui peuvent être représentés selon plusieurs vues (logique, processus, développement et physique ) le tout guidé par la vue des cas d'utilisation (Vues 4+1). Le modèle (Vues 4+1) est connu aussi sous le nom du modèle de Kruchten [7]. Nous avons classé (figure 1-3) les diagrammes selon ce modèle tout en relevant l'aspect dynamique/ statique et aussi le niveau d'abstraction (analyse, conception).

Une première décomposition du système peut donc être faite à l'aide de 4+1 vues. Le schéma ci-dessous montre les différentes vues permettant de répondre, au mieux, aux besoins des utilisateurs.

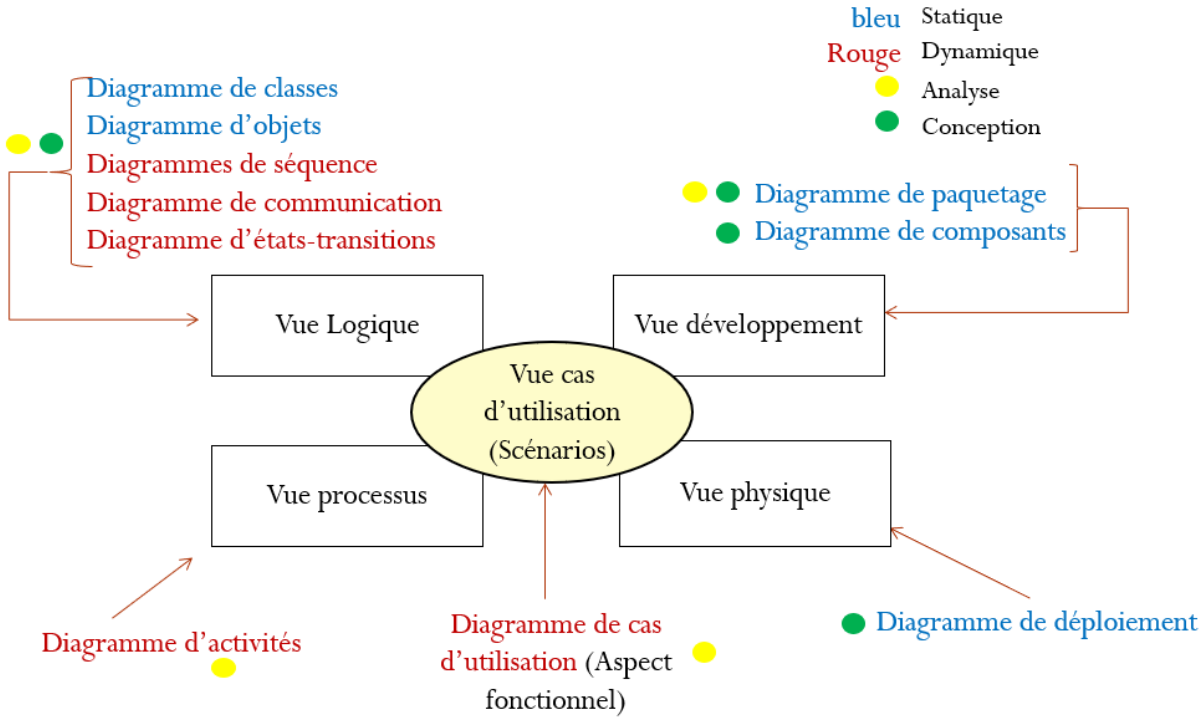



*Figure 1-3: Diagrammes selon la vues 4+1 avec l'aspect dynamique/statique et analyse/conception*

Plusieurs outils de modélisation UML existent. Le but de cette première série est de se familiariser avec l'outil Modelio. Le choix de Modelio en particulier est discuté dans la section qui suit.

## 1.2. Choix de l'outil de modélisation UML

UML est un langage de modélisation graphique à base de pictogrammes. Vu sa grande popularité et utilisation, un très grand nombre d'outils destinés à la

modélisation graphique de ce langage existe. Une comparaison de ces différents logiciels [1] permet d'orienter et de justifier le choix d'un outil par rapport à un autre.
Ce choix dépend de l'usage et du besoin de l'utilisateur. Les critères les plus importants auxquels nous avons prêté attention pour effectuer notre choix sont, entre autres, le support de la norme UML 2 et le nombre de digrammes offerts. Les fonctionnalités additionnelles qui font que l'outil est considéré comme un AGL (Atelier Génie Logiciel) tel que le reverse engineering, la gestion et l'exportation du code, …etc. ne sont pas une finalité pour notre TP mais un plus à considérer dans un projet entreprise. En effet, pour un usage académique, un bon logiciel de modélisation UML est avant tout un produit qui permet de :

- Modéliser efficacement les diagrammes UML, avec tous les composants (respect de la norme UML 2)
- Naviguer facilement et naturellement entre les diagrammes (organisation arborescente en paquetages)
- Exporter les diagrammes pour les intégrer dans les documents de conception.

Nous signalons que nous nous intéressons qu'aux logiciels libres ou open source. Dans ce sens, l'outil Modelio remplit bien plus que tous les critères énoncés ci-dessus car c'est un AGL qui supporte les standards UML2, BPMN2, MDA, SOA.

### 1.3. Présentation et téléchargement de l'outil Modelio

Modelio est un outil de modélisation UML disponible en version entreprise et community. Toutes les versions open source, sont téléchargeables à partir du site de référence [2]. Modelio supporte la plupart des diagrammes spécifiés par UML 2.0. Des générateurs de code (Java, C++, C#) y sont, par exemple, disponibles et d'autres modules peuvent êtres par la suite rajoutés. Vous pouvez retrouver la liste des fonctionnalités de Modelio Open Source dans [3]

Pour une prise en main rapide de Modélio rien de mieux que de le voir en interaction.

Pour cela vous pouvez visionner les vidéos [4] disponibles sur le site officiel ou la chaine youtube [5].

La version de Modelio utilisée pour la réalisation des exercices des séries de TP est la dernière version actuelle (au moment de la rédaction de ce polycopié), à savoir, 4.1. Par ailleurs, vous pouvez utiliser des versions antérieures comme la version 4.0 ou

encore la version 3.8. La dernière version est téléchargeable à partir du site officiel de Modelio. Télécharger [6] la version correspondante à votre système d'exploitation

## 1.4. Installation

A partir de la version 3.6, Modelio est devenu une application java qui est délivré avec un installable pour (Windows et linux). Cependant, pour MacOS X, il suffit de décompresser le dossier téléchargé et de lancer l'application.

Par ailleurs, l'application nécessite la dernière version de Java JDK 8.66.07 En cas de disfonctionnement ou de problèmes pour faire fonctionner Modelio, pensez à mettre à jour votre JDK.

## 1.5. Préparation du workspace

Le workspace est le répertoire de travail dans lequel seront stockés tous vos projets.

Vous pouvez définir dans votre workspace un nouveau projet par TP ou par étude de cas. Par défaut, Modelio utilise un workspace dans le dossier ~/modelio

1. Pour changer de workspace, *Fichier → changer d'espace de travail* par le menu de Modelio (voir figure 1-4).

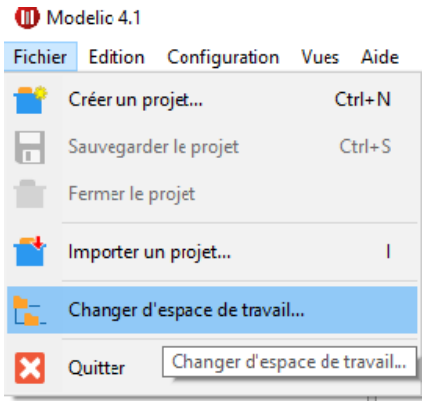



*Figure 1-4 : Menu Modelio : changer d'espace de travail*

2. Créer, dans un premier temps, un dossier (pour votre workspace) nommé 'TPs_Exercices_application' pour abriter tous vos TP (Exercices) et étude de cas du semestre avec Modelio. Pour cela, sélectionnez ~/uml/Modelio dans la boîte de dialogue, et créer le dossier en question.

## 1.6. Création d'un projet

Pour créer un projet, *Fichier→ créer un projet* dans le menu. Pour ce TP, nommez votre projet 'TPPrise_en_main'. Un paquetage nommé 'TPPrise_en_main' (voir figure 1-6) est automatiquement créé.

Pour créer un nouveau projet, il faut fermer le projet ouvert avec : *Fichier →Fermer le project.*

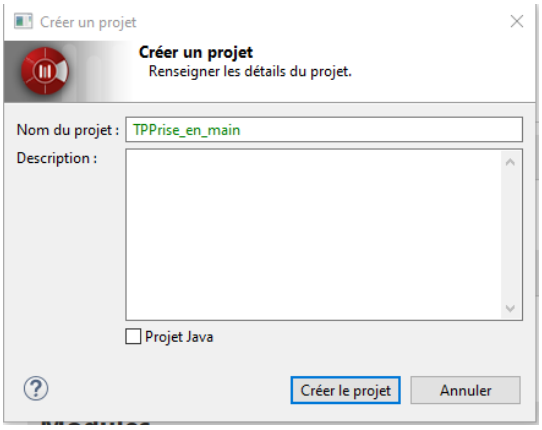



*Figure 1-5 : Créer et nommer un projet*

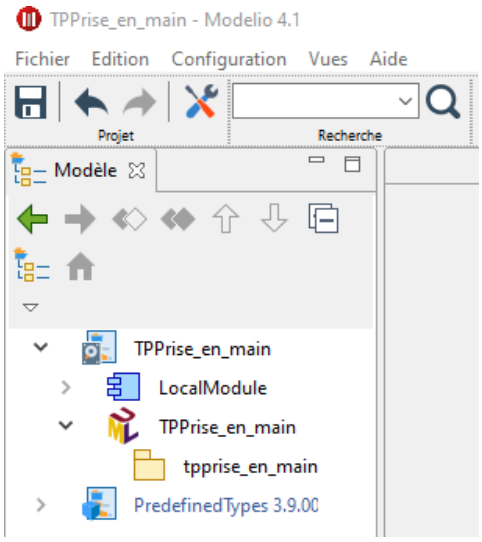



*Figure 1-6 : Paquetage du projet 'TPPrise_en_main'*

## 1.7. Organiser la structure d'un projet

Une modélisation d'un système peut être faite selon différentes vues et niveaux d'abstractions (Voir section rappel de cours).

Afin d'organiser vos diagrammes, créer une arborescence selon une vue fonctionnelle, statique et dynamique

Dans l'onglet modèle (Figure 1-7), créer des paquetages (bouton droit sur le paquetage *tp_prise_en main → Créer un élément → package*) créer dans chaque vue les diagrammes correspondants (*bouton droit sur le paquetage → créer un diagramme*

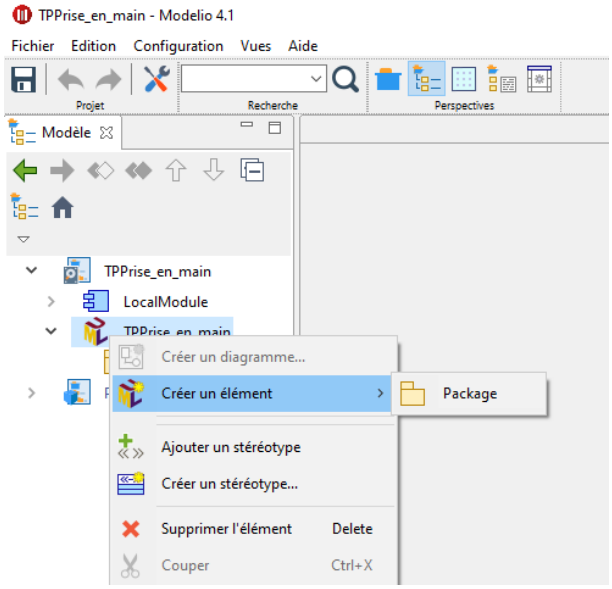



*Figure 1-7 : Créer un package*

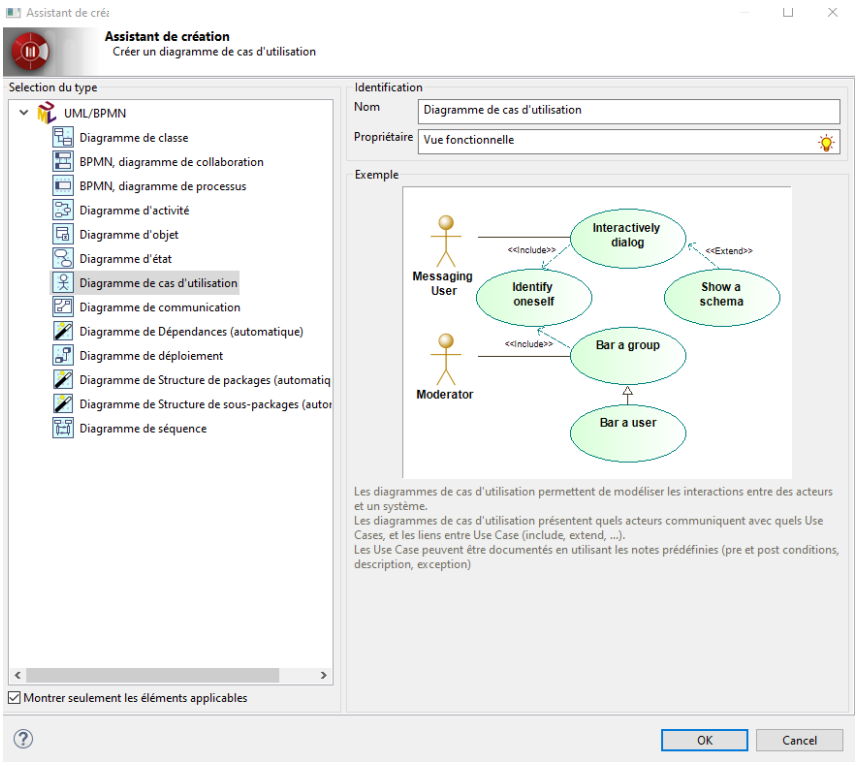



*Figure 1-8 : Ajout d'un diagramme UML*

- Pour chaque diagramme : explorer, déposer et voir les éléments constituants

La figure 1-9 montre l'explorateur du projet après l'ajout de quelques diagrammes à chaque vue.

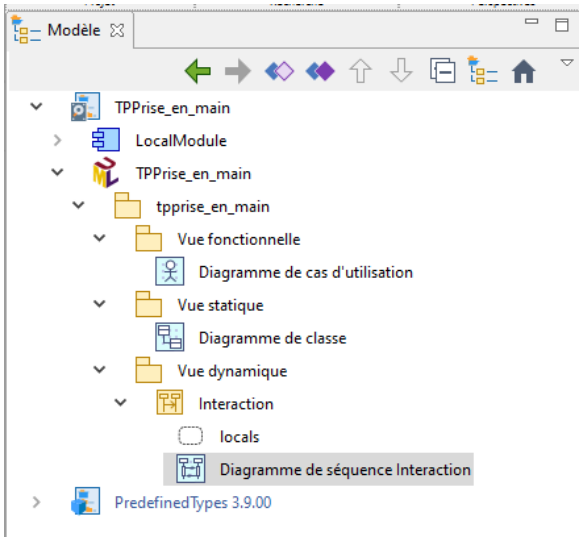



*Figure 1-9 : Explorateur du Projet TPPrise_en_main avec les 3 vues et quelques diagrammes correspondants*

## 1.8. Générer la documentation de vos modèles

Rajouter à votre projet le module WebModelPublisher. Pour cela, dans le menu : *configuration→ module→ ajouter* . Cliquer sur *'modeler module' → ajouter (Figure 1-10), u*ne fenêtre apparaitra (Figure 1-11). Pour déployer l'un des modules proposés, choisir le module en question est cliquer sur *Déployer dans le projet*

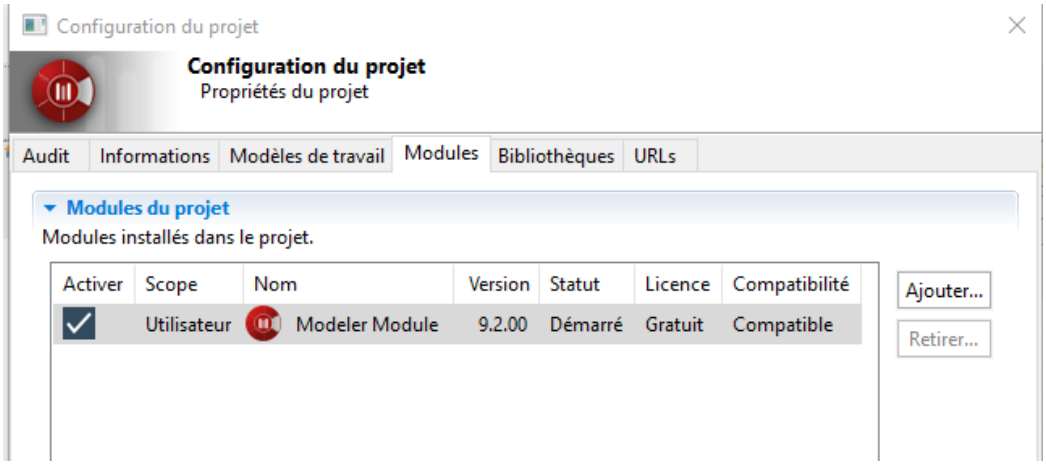



*Figure 1-10 : Configuration du projet pour ajouter des modules*

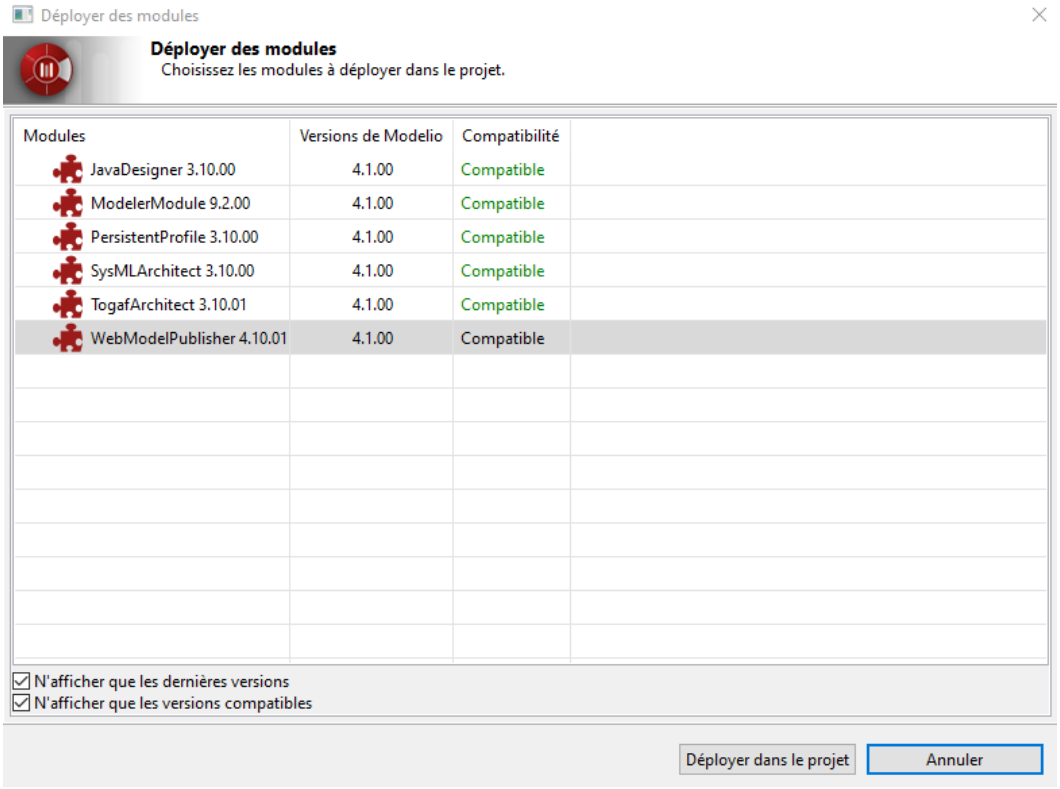



*Figure 1-11 : Déployer un module dans votre projet*

Cliquer avec le bouton droit sur le paquetage *tp_prise_en main→web Model Publisher → Générer* (voir figure 1-12). Toute la structure de votre projet (avec les diagrammes et éléments) est organisée dans le site généré (voir figure 1-13).

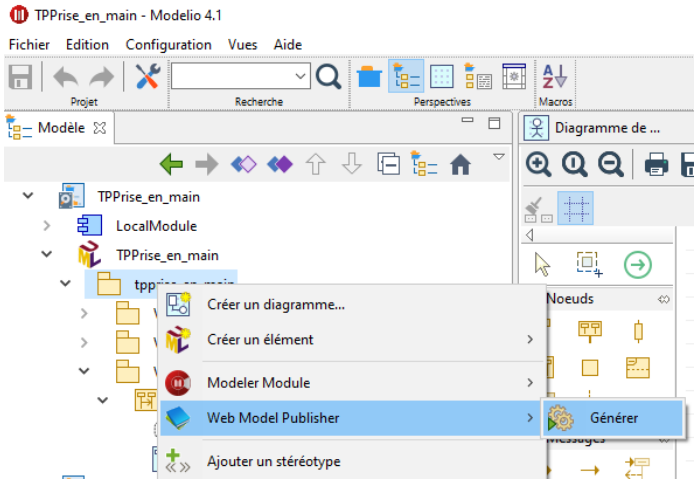



*Figure 1-12 : Générer la publication de la documentation du projet TPPrise_en_main*

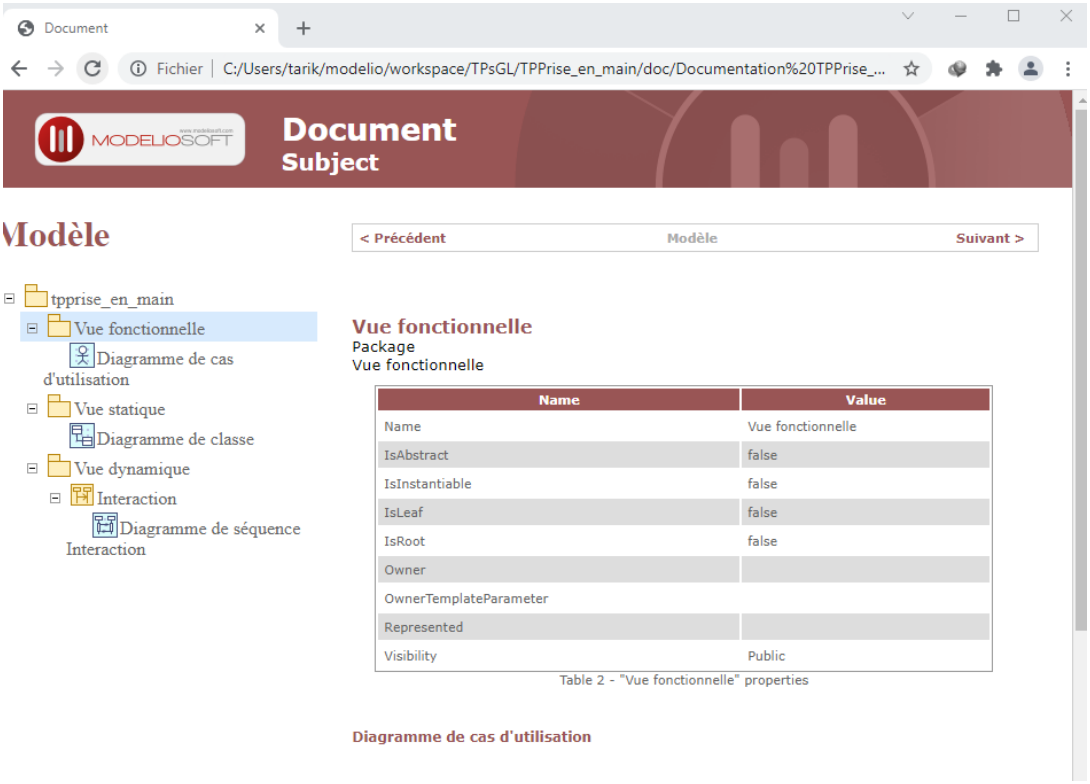



*Figure 1-13: La documentation de la structure du projet organisé sous forme de pages web*

## 1.9. Sauvegarder et ouvrir un projet Modelio

1- Pour sauvegarder notre projet, il suffit de le faire tout simplement à partir du menu et notre projet est automatiquement sauvegardé. Notons que le dossier du projet se trouve dans le worspace actif qui a été défini au moment de la création du projet.

2- Quand on démarre Modelio tous les fichiers projets se trouvant dans le workspace actif sont automatiquement affichés dans l'onglet '*espace de travail'*. Si on veut ouvrir un projet qui ne se trouve pas dans le workspace actif. On peut :

a- Soit changer le worspace à partir de modélio en donnant le chemin du dossier qui abrite le projet en question ;

b- Soit copier le fichier manuellement dans le worspace actif.

c- Soit on peut l'importer à partir de modélio vers notre workspace actif. Dans ce cas, il faudra que le dossier du projet soit sous forme d'archive zippée. Il suffit, à partir du menu, cliquer sur '*importer le projet'* en sélectionnant l'archive du projet.

## 1.10. Exercice d'application

### *1.10.1. Enoncé*

Créer un autre projet nommé 'PRJ selon Vues 4+1' et présenter une arborescence pour modéliser le système selon le modèle **vues 4+1** en associant les diagrammes correspondants à chacune des vues.

### *1.10.2. Corrigé*

La figure 1-16 donne un aperçu des 4 vues +1 avec quelques diagrammes

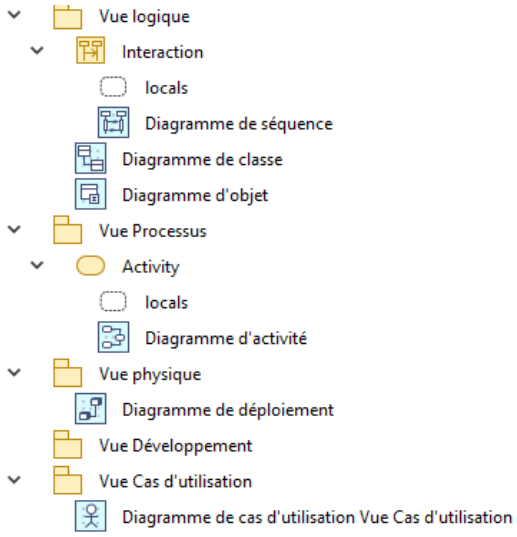



*Figure 1-16: Vue 4+1 avec quelques diagrammes*

# TP 2 : Diagramme de cas d'utilisation et la description textuelle

## 2.1. Objectif et rappels

L'objectif de ce TP est de créer un diagramme de cas d'utilisation à partir de l'analyse des besoins.

L'analyse des besoins est la première étape dans le cycle d'un développement logiciel, elle est incontournable car elle va préciser les objectifs, les services (fonctions) et les principales contraintes sur le système étudié, quel que soit le type d'application informatique (système d'information (de gestion), système temps réel (automatismes, contrôle de processus) ou calcul scientifique). La première étape est celle qui va permettre de comprendre ce que veut le client.

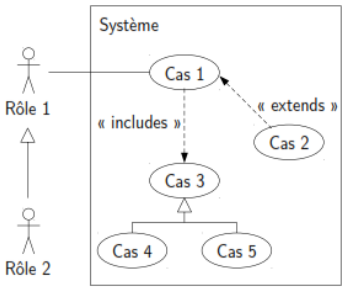
La capture des besoins du client établit les besoins fonctionnels et non fonctionnels du système à modéliser et la compréhension de son contexte.

Les besoins fonctionnels sont les services attendus par le demandeur.

Les besoins non fonctionnels regroupent les contraintes (systèmes existants, plateformes, standards, distribution..) et des critères du résultat attendu (performance, coût, ..) .

L'activité d'analyse des besoins produit les diagrammes de cas d'utilisation, qui sont illustrés par des scénarios (diagrammes de séquences réduits au système et aux acteurs externes).

Dans cette série, nous allons réaliser des diagrammes de cas d'utilisation et leurs descriptions textuelles. Tous les éléments avec leurs notations relatives à ce diagramme sont regroupés dans le mémento au tableau1.

| DIAGRAMME DE CAS D'UTILISATION |
|---|
| **Objectif** : représenter les fonctionnalités du système d'un point de vue utilisateur. |
| **Formalisme** :  |



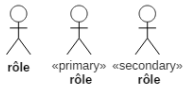
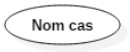
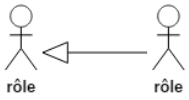
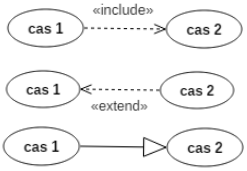
| Eléments | | Notation |
|---|---|---|
| Acteur | **Rôle** joué par une **entité externe** au système peut être **primaire** ou **secondaire** |  |
| Cas d'utilisation | **Fonction** du système : représenté par un verbe à l'infinitif qui décrit une action. |  |
| Relations entre :  | Une seule relation entre acteur et utilisateur **'relation d'association'** |  |
|  | Une seule Relation entre acteurs **''généralisation spécialisation** » |  |
|  | Trois relations entre les cas : **inclusion, extension et généralisation-spécialisation** |  |



*Tableau 1 : Aide-mémoire de la notation utilisée dans le diagramme de cas d'utilisation*

## 2.2. Pratiquer : Tutoriel à suivre

Dans cette partie, nous allons réaliser, sous Modélio pas à pas, le diagramme de cas d'utilisation (le diagramme à réaliser (solution) est donné dans la figure 2-1). Il est relatif au Système de Gestion de Demande de Congé SGDC dont la description est présentée dans la section suivante. Ce système sera repris dans les prochaines séries de TP pour réaliser les autres diagrammes de modélisation.

### *2.2.1. Description du système 'Gestion de Demande De Congés' (SGDC)*

L'entreprise 'SarlMed' souhaite automatiser la gestion des demandes de congés de ses employés. Pour cela, SarlMed a fait appel à une société de développement pour réaliser le futur logiciel nommé 'PlanCongé'.

Les employés peuvent à travers ce logiciel émettre une demande de congé. Pour cela, l'employé se connecte et s'identifie au système via son compte (login et mot de passe). À noter que pour l'identification, le système permet à l'employé de saisir son mot de passe s'il est erroné trois fois, sinon son code est invalide et la connexion est rompue. Si l'identification est faite, le client choisit la fonction 'émettre une nouvelle demande de congé'. Le système lui retourne une fiche à remplir. L'employé renseigne les informations (dates de sortie et retour) et choisit le type de congé (annuel ou autre).

Dans le cas d'un congé annuel, le système vérifie, en consultant le logiciel du service personnel, si l'employé a le droit de prendre son congé, par rapport aux dates du congé de l'année dernière.

Dans le cas d'une demande de congé non annuel, l'employé saisit le motif (maladie, accouchement, raisons familiales, …,etc.) du congé et peut accompagner la demande d'un justificatif (exemple : ordonnance médicale, raisons familiales (décès, ..., etc.). Le système enregistre sa demande et affiche un message de confirmation.

Le chef de chaque service de l'entreprise utilise le système pour traiter les demandes de congés en attente de traitement. Le traitement peut être soit un refus ou une acceptation.

En cas de refus, le supérieur hiérarchique doit justifier son refus. Le responsable des ressources humaines se charge de valider les demandes de congés acceptées. L'employé a la possibilité, à tout moment, d'annuler sa demande seulement si cette dernière n'a pas encore été traitée.

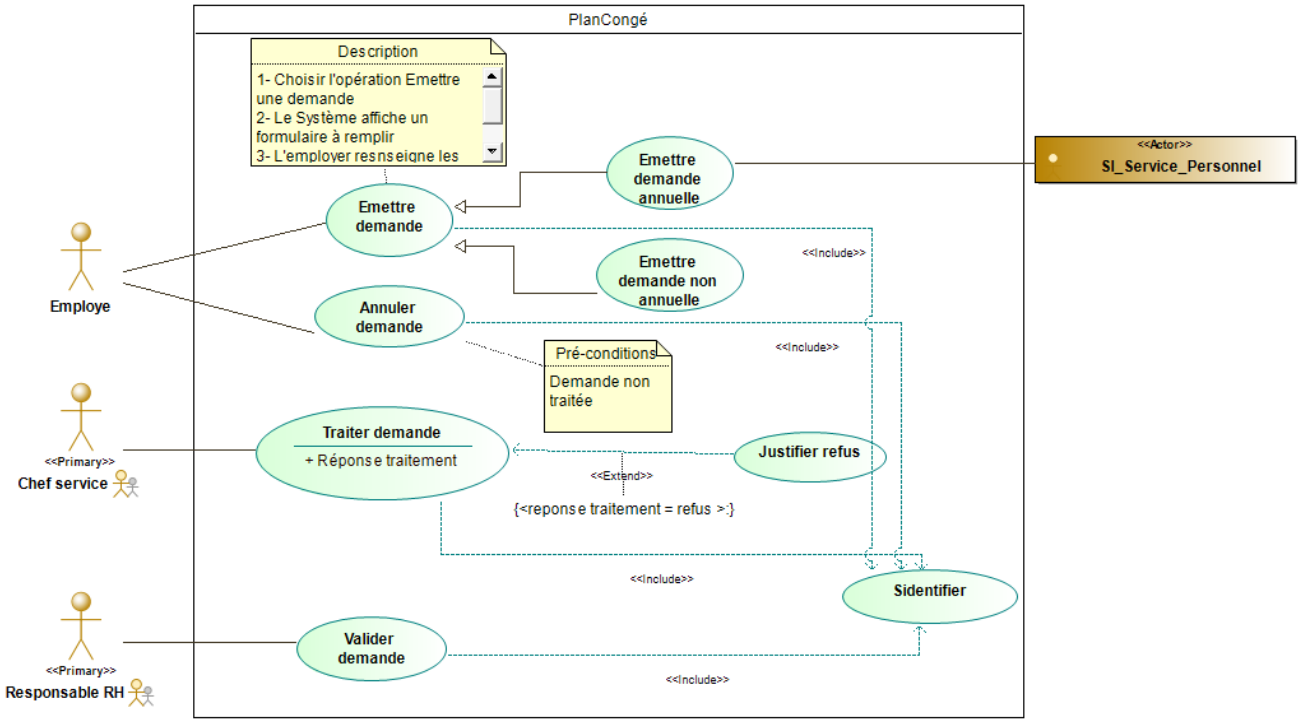



*Figure 2-1 : Diagramme de Cas d'utilisation du système SGDC*

### *2.2.2. Créer votre projet et diagramme de cas d'utilisation*

Pour pratiquer et se familiariser avec Modelio; il s'agit, dans un premier temps, de réaliser le diagramme de cas d'utilisation de la solution qui est donnée dans la figure 2-1 du SGDC en suivant les étapes suivantes :

- Pour ceux qui travaillent sur les Pc du labo : Créer un *workspace* portant votre nom, ce dernier contiendra tous vos TPs du semestre. N'oubliez pas de vous positionner dans ce dernier : *Fichier→changer d'espace de travail*.
- Créez un projet nommé 'TP_SGDC'
- Afin de mieux gérer et organiser votre travail, créer le package 'vue cas d'utilisation' sous lequel vous créez :
  - Le package 'Acteurs' : qui contiendra tous les acteurs du système.
  - Le package 'Cas_d'utilisations' : qui contiendra tous les cas.
  - Le diagramme : utiliser le menu contextuel → *créer un diagramme*

### *2.2.3. Ajouter les acteurs , cas d'utilisation et les liens*

Pour ajouter des acteurs, il suffit de les déposer à partir de l'onglet *Nœuds* ; puis il faudra les nommer avec le raccourci F2. On pourra changer les propriétés de l'acteur comme la couleur en faisant apparaitre l'onglet *symbol* (côté droit de la zone de dessin

du diagramme). À noter qu'on peut les ajouter aussi à partir de l'explorateur du projet. Pour cela, sélectionner le package 'acteurs' et à partir du menu contextuel choisir : créer un *Créer un élément → actor* . Le faire glisser, avec la souris, sur la zone du dessin.

Pour ajouter des cas d'utilisation on procède de la même manière que pour les acteurs.

Pour ajouter un lien, il faudra sélectionner le lien adéquat (héritage, extension inclusion, ..) à partir de l'onglet '*Liens'* et lier les deux objets en question (acteur, cas) ou (cas, cas).

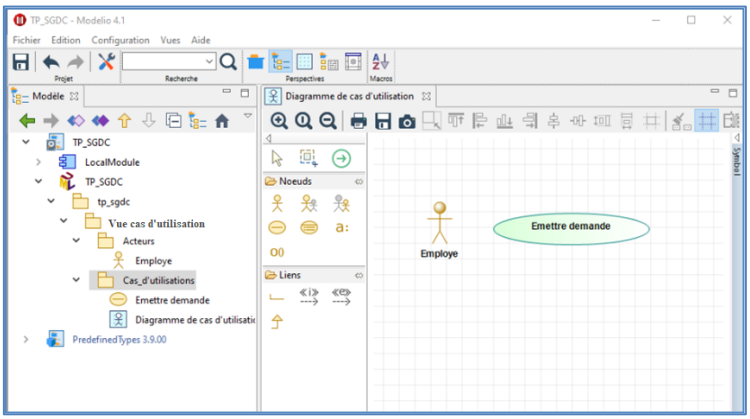



*Figure 2-2: Fenêtre Modelio de CU qui contient l'acteur client et le cas d'utilisation retirer de l'argent*

### *2.2.4. Faire apparaitre le cadre du système*

Sélectionner la zone de dessin du digramme et faire apparaitre l'onglet '*symbol'* (figure 2-3), et cochez la propriété '*Afficher le system boundary'*

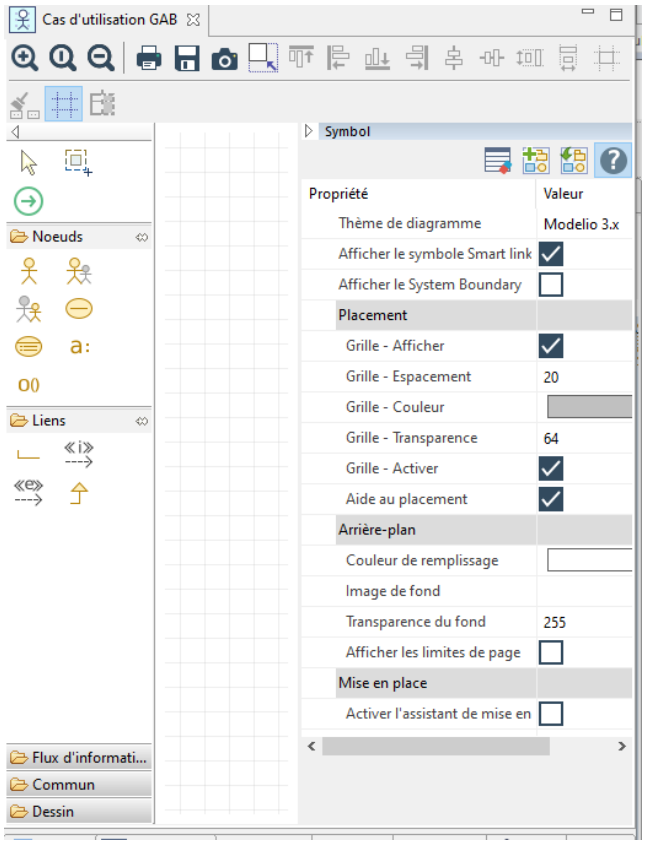



*Figure 2-3 : Onglet symbole de zone de dessin du diagramme de Cas d'utilisation*

### *2.2.5. Choisir un stéréotype pour un acteur non humain*

L'acteur '*SI_Service_Personnel'* et un système connecté et non pas un humain. Pour indiquer cela, on peut lui ajouter un stéréotype <<*System*>> pour ce faire : Sélectionnez l'acteur '*SI_Personnel'* et à l'aide d'un clic droit, ouvrez le menu contextuel '*ajouter un stereotype'*. Sélectionnez le stéréotype <<*System*>> dans la liste (figure 2-4) et validez en cliquant sur `OK``.

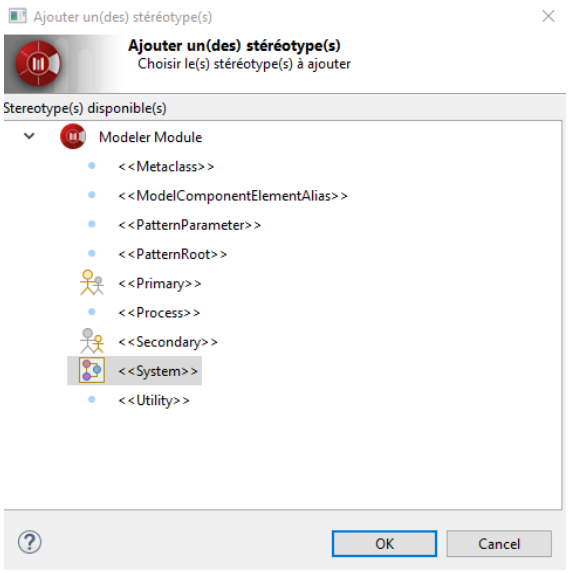
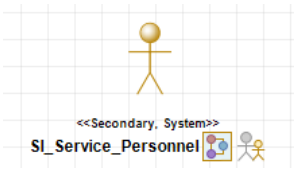



*Figure 2-4: Ajouter un stéréotype*

Le stéréotype <<*System*>> est alors ajouté sur le diagramme au-dessus du nom de l'acteur. Le nom de l'acteur dans l'arborescence de l'onglet '*Modèle'* change également en fonction du stéréotype choisi.




### *2.2.6. Changer l'apparence graphique d'un acteur non humain*

On peut aussi changer l'apparence graphique d'un acteur. Sélectionnez l'acteur '*SI_Service_Personnel'*. Ensuite, à partir de l'onglet '*symbol'*, il suffit de changer la propriété : '*mode de représentation'* **en** choisissant '*structuré'* .


On peut aussi mettre la propriété '*mode d'affichage du nom'* à '*Hidden'*. On peut aussi supprimer le stéréotype '*System'* et '*Secondary'* car la représentation par cadre représente d'emblée un acteur non humain. Pour cela, allez à l'onglet Propriété (figure 2-5) en bas de la fenêtre, sélectionnez le stéréotype et supprimez-le en cliquant sur la touche sup.

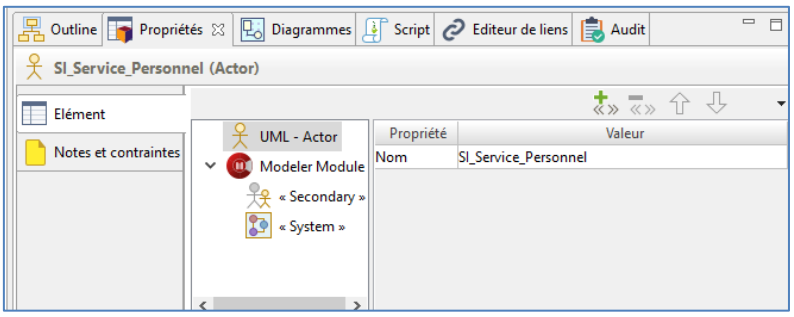



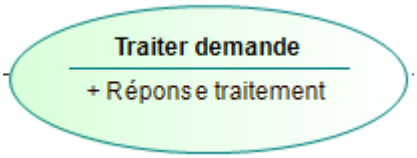

*Figure 2-5 : Onglet propriété de l'acteur SI_Service_Personnel*

### *2.2.7. Rajouter un point d'extension à un cas d'utilisation*

Dans le cas où il y'a une relation d'extension entre deux cas d'utilisation, On peut rajouter un point d'extension qui permet de montrer la condition d'extension du cas initial. C'est le cas entre les cas '*Justifier refus'* et '*traiter demande'*. En effet, le point d'extension qui permet de passer au cas *'Justifier refus'* est la réponse de traitement avec la condition : *réponse négative*.

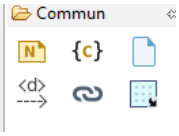
Pour représenter cela, il faudra :

1- Ajouter le point d'extension : choisir l'objet  à partir de l'onglet '*Nœud'* et le déposer sur le cas qui va être étendu. Dans notre cas, c'est le cas '*Traiter demande'*, le point d'extension est



alors ajouté au cas d'utilisation, il suffit de le nommer dans l'onglet '*propriété'*.

2- Ajouter la condition au lien d'extension : Choisir {c} à partir de l'onglet Commun et la déposer sur le lien. Il faudra donner la condition dans l'onglet '*propriété'*




La représentation du point d'extension est donnée dans la figure 2-6.

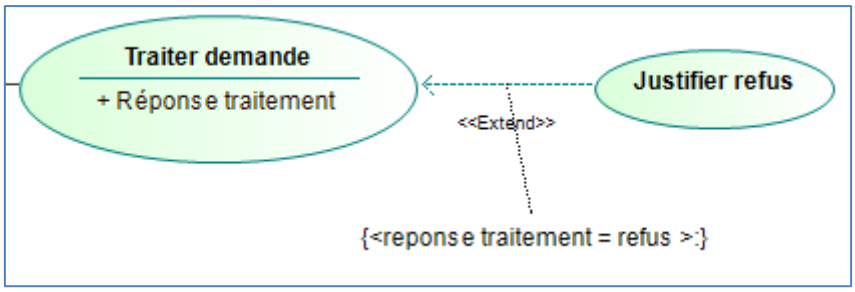



*Figure 2-6 : Le point d'extension entre les cas: 'traiter demande' et 'justifier refus'*

### *2.2.8. Rajouter la description textuelle d'un cas d'utilisation*

Cliquez sur le menu sur *Vues→ Note et contraintes* Un onglet est rajouté en bas de la fenêtre au même niveau que propriétés.

Sélectionner le cas d'utilisation '*Emettre une demande'*. Puis, dans l'onglet '*note et contraintes'*, sélectionner la note '*Description'* et insérer (dans la zone de texte à droite) le texte du scénario nominal et ses alternatifs (pris de la description du système) voir figure 2-7, faire la même chose pour les autres notes : exception, pré-condition….

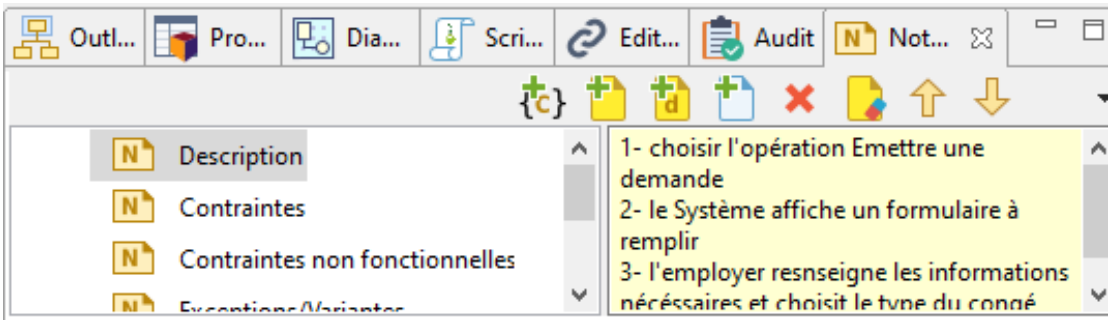



*Figure 2-7 : Onglet description textuelle: cas Retirer argent*

### *2.2.9. Rajouter une contrainte de pré-condition*

Dans le SGDC, le cas *'Annuler demande'* ne pourra être actionné que dans le cas où la demande n'est pas encore traitée. Pour mettre en évidence cette contrainte sur notre diagramme, on pourra utiliser une contrainte qui sera attribuée au cas *'Annuler*

*demande'*. Pour cela, comme pour le rajout de la description textuelle, il faudra à partir de l'onglet propriété, sélectionner '*Notes et contraintes'* et choisir '*Pré-condition'* et renseigner la contrainte (figure 2-8).

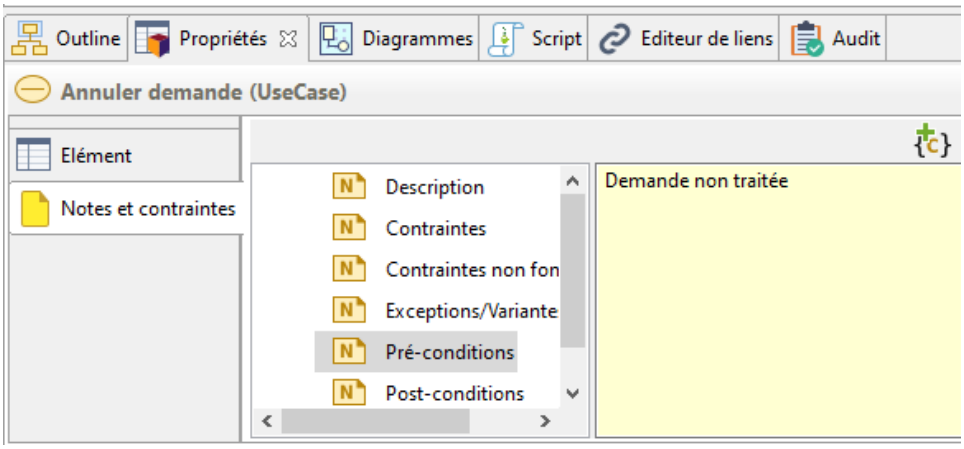



*Figure 2-8 : Renseigner la note de pré-condition du cas annuler demande*

Pour faire apparaitre la note '*pré -condition'* et '*description'* relatives aux cas d'utilisation dans le diagramme de cas d'utilisation (voir figure 2-9), il suffit de la sélectionner et de la faire glisser dans la zone de dessin de votre diagramme.

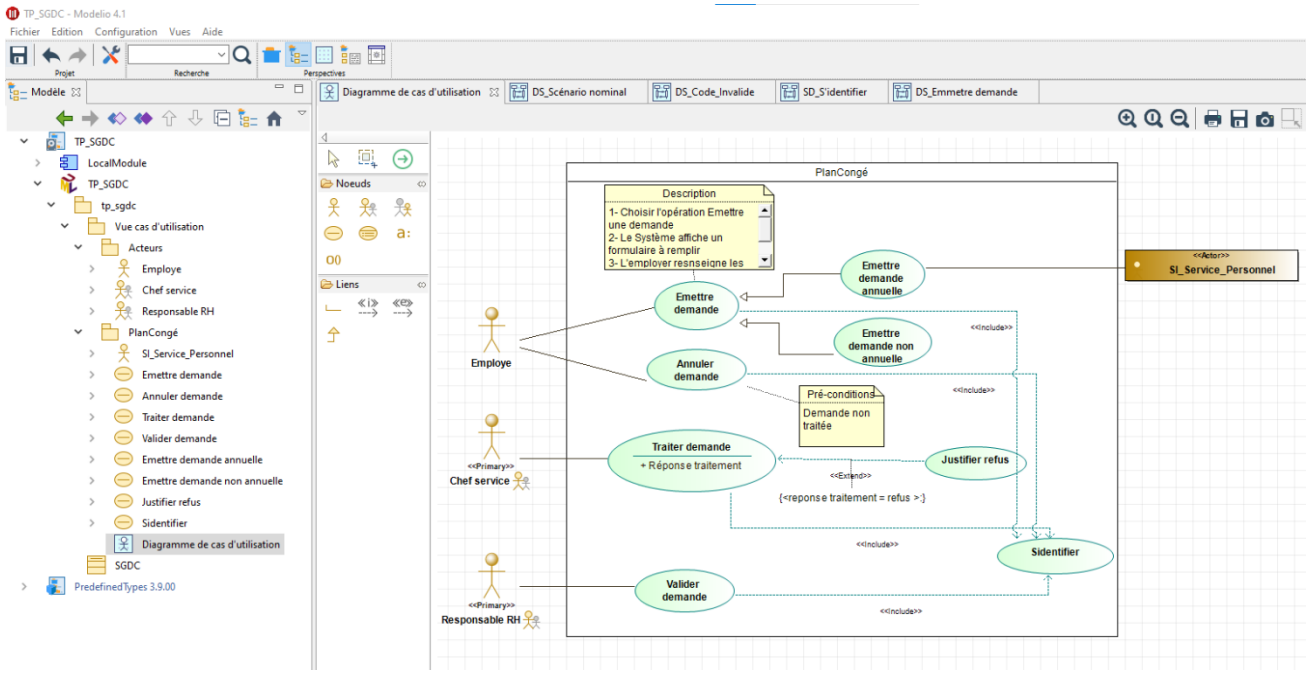



*Figure 2-9 : Diagramme de cas d'utilisation du système SGDC dans Modelio*

Une fois le diagramme terminé, on peut copier le diagramme en tant que graphique sous forme d'une image png avec l'une des deux icônes. 

## 2.3. Exercice d'application

### *2.3.1. Enoncé*

Nous souhaitons mettre en place un logiciel qui permettra de constituer le dossier médical d'un patient. Pour cela, nous avons les acteurs suivants : secrétaire médicale, médecin et BDD_médicaments.

À chaque admission d'un patient, la secrétaire crée un dossier médical et doit renseigner les antécédents médicaux du patient. Si le patient existe déjà, elle met à jour son dossier (changement d'adresse, numéro de téléphone …).

À chaque consultation, le médecin rédige une ordonnance en utilisant la BDD médicaments. Dans certains cas, l'ordonnance est accompagnée par une prescription d'examens complémentaires qui peuvent être des analyses ou une radio ;

Dessiner le diagramme complet de cas d'utilisations.

### *2.3.2. Corrigé*

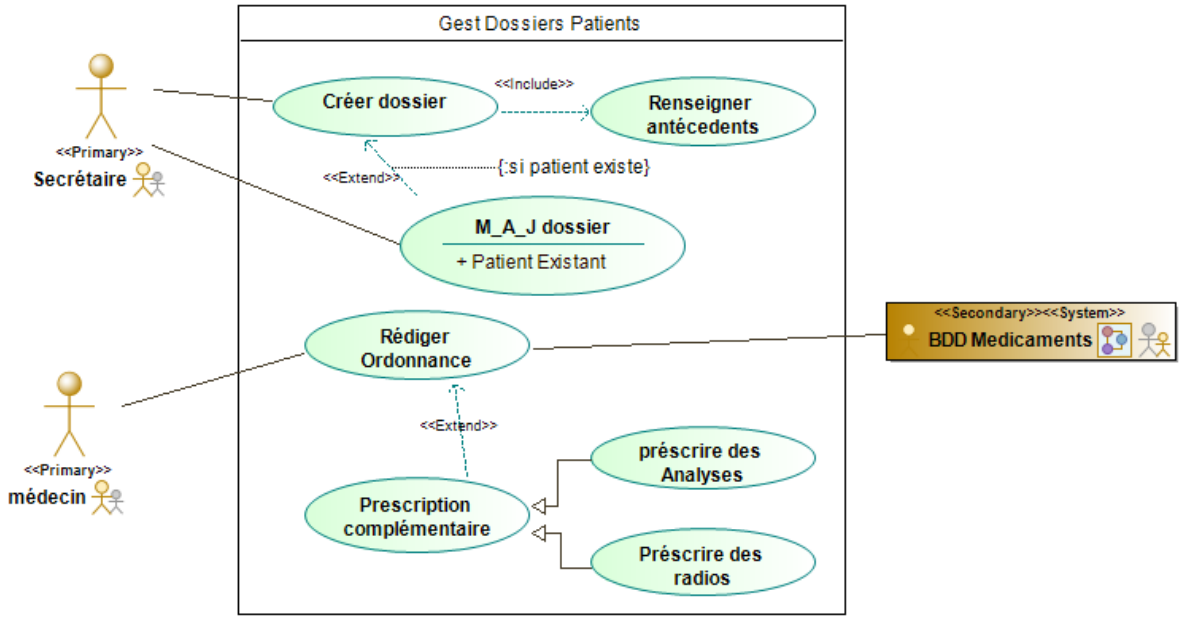



*Figure 2-10 : Diagramme de Cas d'utilisation: Gestion de dossiers de Patients*

# TP 3 : Description des cas d'utilisation : Diagramme de Séquence Système

## 3.1. Objectif et rappels

L'objectif de ce TP est de décrire la dynamique d'un cas d'utilisation. Nous rappelons qu'un cas d'utilisation peut être décrit par un texte dans des scénarios (Scénarii ) non représentés dans le diagramme de cas d'utilisation. Un cas d'utilisation est égal à un ensemble de scénarios. Un scénario décrit la suite logique des interactions entre l'acteur et le système.

La description des scénarios est faite de manière textuelle et/ou accompagnée par un diagramme de séquence système. La figure 3-1 représente la dynamique de description des cas d'utilisation.

Nous traitons dans ce TP la description dynamique d'un cas d'utilisation par la description textuelle et le diagramme de séquence système

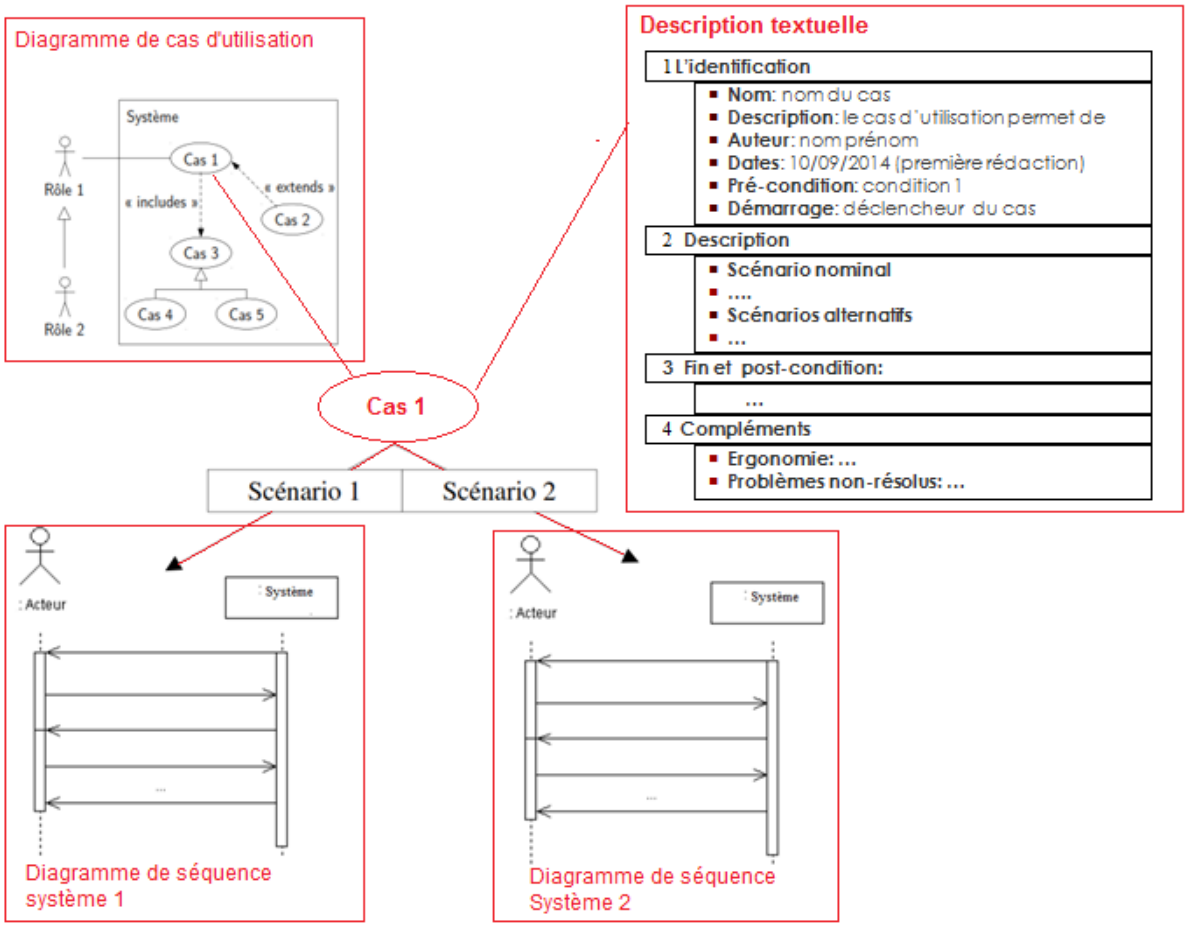


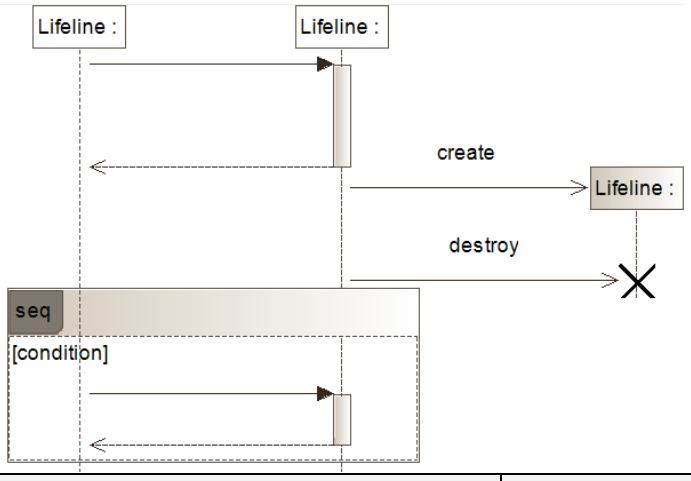

*Figure 3-1 : Description de la dynamique d'un cas d'utilisation*

Le tableau 2 résume la notation des éléments les plus utilisés dans la réalisation du diagramme de séquence

**DIAGRAMME DE SEQUENCE**

**Objectif** : Montre l'ordre des échanges de messages entre objets et le passage du temps. C'est un diagramme temporel

**Formalisme** :



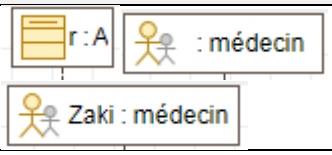
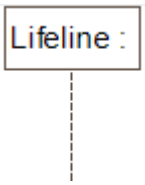
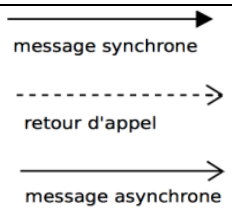

| Eléments | | Notation |
|---|---|---|
| Objet | Les participants : ce sont les instances des acteurs ou de classes. Ça peut être un objet typé, identifié ou anonyme |   |
| Ligne de vie | C'est l'objet avec la ligne en pointillé. Elle permettra de déposer les messages de manière chronologique de haut vers le bas respectivement de plus anciens au plus récent |   |
| Barre d'activation | Montre quand l'objet est actif, à la suite d'une invocation de message (Interaction entre deux objets). La longueur du rectangle indique la durée pendant laquelle les objets restent actifs |  |
| Messages | C'est l'interaction entre objet. Ça peut correspondre à l'appel d'une méthode ou à un signal. |   |
| Structure de contrôle (Fraguements) | Les fragments d'interaction (UML2) permettent d'indiquer qu'un groupe de messages est optionnel (**opt**), répété (**loop**) ou d'indiquer une référence (**ref**),.. Il existe 12 opérations de fragments |   |

*Tableau 2 : Aide-mémoire de la notation utilisée dans le diagramme de séquence*

## 3.2. Pratiquer : Tutoriel à suivre

Pour pratiquer et se familiariser avec les diagrammes de séquence ; nous allons reprendre et continuer à modéliser le système SGDC énoncé dans la série TP 2. Nous allons décrire les cas d'utilisations avec les diagrammes de séquences système.

Nous allons présenter, dans ce qui suit, comment réaliser pas à pas, le diagramme de séquence système DSS du cas d'utilisation '*s'identifier'* avec les trois scénarios : nominal (Solution donnée dans la figure 3-2), alternatif et d'exception (solution donnée dans la figure 3-6).

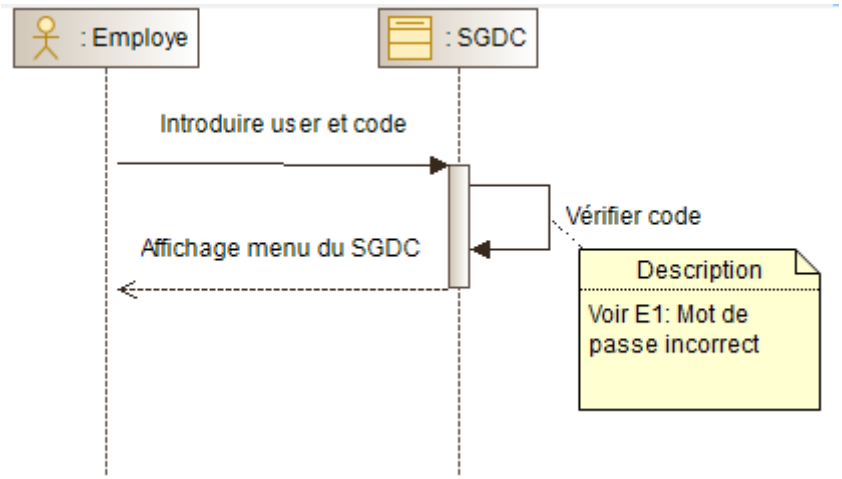
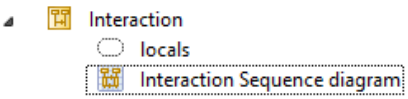



*Figure 3-2: Diagramme de séquence système : scénario nominal s'identifier*

Ce TP est une continuité du TP précédent ; donc, sélectionner le projet du Tp précédent qui a été nommé '*TP_SGDC'*.

### *3.2.1. Créer le diagramme de séquence 'S'identifier' avec le stéréotype scénario*

Sélectionner le cas d'utilisation *'s'identifier'* à partir de l'explorateur de la perspective modèle, bouton droit de la souris→*créer un diagramme*→*Diagramme de séquence*




Vous remarquerez qu'un élément *'Interaction'* est créé. Ce dernier contient le *diagramme de séquence* et l'élément *'Locals'* qui va contenir les objets qui seront déposés sur le diagramme.

Pour montrer que c'est un diagramme de séquence système (scénario), on va rajouter un stéréotype scénario. Pour cela, sélectionner *'Interaction'* et avec le bouton droit ; '*ajouter un stereotype'* et choisir <<*scenario*>> à partir de la fenêtre qui apparait. Renommer l'interaction '*Scénario nominal'* et renommer le diagramme de séquence *'SD_Scénario nominal'*

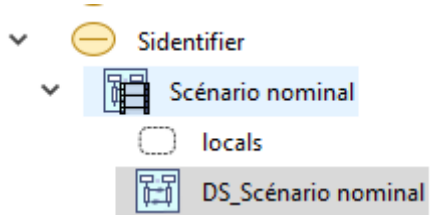



*Figure 3-3 : Aperçu du cas s'identifier avec le DS avec le stéréotype scénario*

### *3.2.2. Déposer les objets*

Le diagramme de séquence représente les interactions entre objets (instances de classes), dans le cas de diagramme de séquence système, les interactions sont entre les objets acteurs et l'objet système (considéré comme boite noire). Les acteurs sont déjà créés dans le diagramme de cas d'utilisation. Cependant, il faudra créer la classe système pour pouvoir l'instancier. Dans ce cas, avant toute chose, il faudra commencer par créer une classe qui représente le système.

Positionner vous sur le package qui représente votre système '*TP_SGDC'*, bouton droit de la souris : *Créer un élement→ class.* Nommer cette classe 'SGDC'. Pour l'instant, cette classe ne contient ni attributs, ni opérations. Cette dernière, devra à la fin de la modélisation de tous les cas d'utilisation, contenir toutes les opérations (interactions) de tous les cas d'utilisation du système.

Choisir la classe 'SGDC' initialement créée et faites-la glisser sur votre zone de dessin du diagramme de séquence. Un objet avec sa ligne de vie est créé dans votre zone de dessin et un objet est rajouté dans l'élément *'Locals'*

L'objet créé par défaut est un objet identifié typé, c-à-d l'objet est par défaut nommé 'r' et il est instancié de la classe 'SGDC'. Vu que nous n'allons pas représenter un scénario concret ; nous allons utiliser des objets anonymes typés. Sélectionner l'objet 'r :SGDF' à partir de l'élément *'locals'* et à partir de l'onglet *'élément'* effacer la valeur 'r' de la propriété *'Nom.* Vous pouvez faire aussi la même opération dans l'onglet qui apparait en double cliquant sur l'objet.

Faites la même chose pour les acteurs.

**Remarque :** Les objets peuvent être connectés à des classes existantes ou bien être créés indépendamment de toute classe. Dans le deuxième cas, les objets sont non typés

et la ligne de vie est créée à partir de la palette du diagramme '*Nœuds'* avec l'icône 
Une ligne de vie est créée, il suffit alors de la nommer.

### *3.2.3. Créer les messages*

Vu que nous sommes encore au stade de l'analyse des besoins (diagrammes de séquences système). Nous n'allons pas exploiter tous les types de messages. Nous n'aurons besoin que de messages synchrones → et d'un retour de messages ⇠ . Les autres seront détaillés dans le diagramme de séquence de conception. A cet effet, choisir un message synchrone de la palette et le déposer sur la ligne de vie d'un objet, ensuite le relier à un autre objet. Sélectionner la flèche du message et nommer l'opération dans la propriété '*Nom'* de l'onglet '*propréiétés'*

### *3.2.4. Créer le scénario nominal 'S'identifier' avec exceptions et alternatifs (utilisation des fragments)*

Les scénarios 'exceptions' et 'alternatifs' peuvent être signalés dans le scénario nominal avec des notes de description qui font références aux descriptions textuelles des scénarios alternatifs (inclusion et extension) et/ou d'exception (voir figure 3-2). Nous pourrons rajouter ces alternatifs et exceptions dans le scénario nominal en utilisant les fragments. Nous allons maintenant représenter le scénario d'exception (figure 3-4) qui va être référencé, plus tard, dans le scénario nominal avec alternatifs (figure 3-6)

#### *3.2.4.1. Création du scénario 'Code Invalide'*

Procéder de la même manière que la création du scénario nominal et créer le digramme de séquence système que vous nommez '*DS_code invalide'*, ce dernier sera rattaché toujours au cas d'utilisation '*s'identifier'*. Nommez votre Interaction '*Scénario alternatif*' avec le stéréotype scénario. Déposer les objets et créer les messages (figure 3-4)

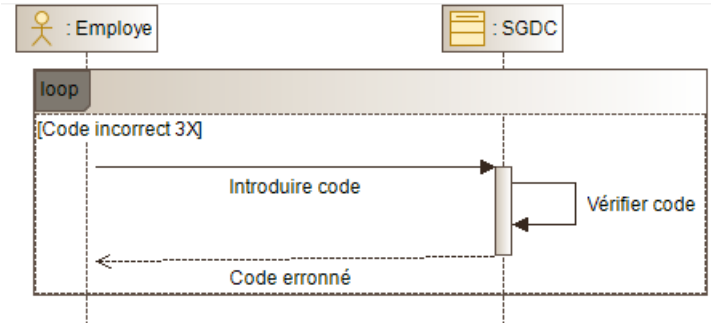



*Figure 3-4 : Scénario code incorrect*

- **Créer le fragment loop**

Dans la palette '*Nœuds'* choisir l'élément *'combined fragment'*, icône ; et cliquer sur la zone de dessin, faites glisser la souris pour prendre en considération les deux lignes de vie ensuite relâcher la souris. Le fragment se compose de deux zones : l'opérateur (seq, loop, ect) et la condition (appelée *'guarde'*) deuxième cadre. Choisir le fragment complet et dans les propriétés de l'onglet '*élément'*, changer l'opérateur en choisissant Loop (figure 3-5). Choisir le deuxième cadre et saisir la condition dans la propriété '*Guarde'*

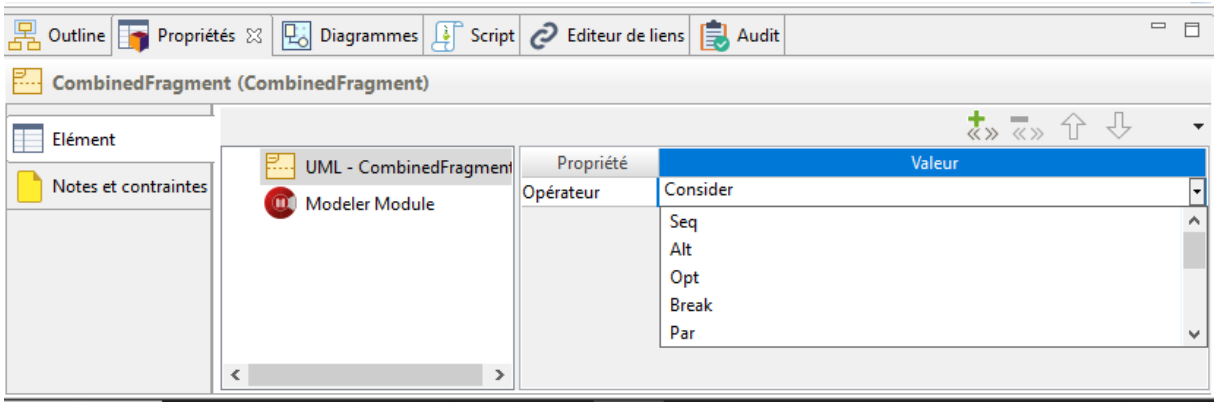



*Figure 3-5 : propriété de l'opérateur du fragment*

### *3.2.4.2. Création du scénario nominal 'S'identifier' avec alternatifs*

Nous allons à présent, créer le scénario qui décrit le scénario nominal avec les exceptions et alternatifs du cas '*s'identifier'*. Pour cela, rajouter un autre diagramme de séquence, nommez l'interaction tout simplement *'S'identifier'*.

Nommez votre diagramme de séquence *'SD_s'identifier'*. Représenter tous vos objets et messages (voir figure 3-6)

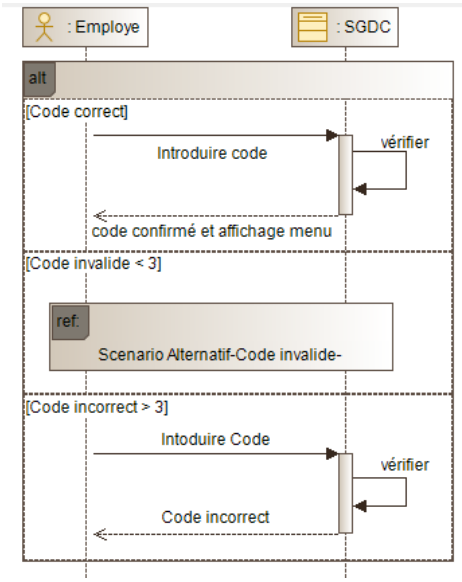



*Figure 3-6 : Scénario du cas d'utilisation 's'identifier' avec alternatifs et exceptions*

a) Créer le fragment 'alt'

Rajouter le fragment 'alt' de la même manière que celui de 'loop'. La particularité de ce fragment est le fait qu'il contient deux alternatifs (conditions), donc 3 '*Guarde'*. Pour rajouter la deuxième et troisième alternative, sélectionner dans la palette l'icône '*interaction operand*'  et déposer l'élément à l'intérieur du compartiment du premier '*Guarde'*, une deuxième alternative est créée. La même chose pour le troisième compartiment.

b) Créer le fragment ref

Nous allons à présent rajouter le fragment qui fait référence au scénario *'code invalide*'. Pour cela, déposer un élément interaction '*use'* représenté par l'icône  . Dans l'onglet '*Elément'*, allez dans la propriété '*Intéraction utilisée'* et cliquez sur la zone où est inscrit <*null*>, faites la combinaison des touches Ctrl+espace. Il ne vous reste qu'à choisir le scénario alternatif *'Code invalide'* parmi la liste des scénarios proposés (figure 3-7).

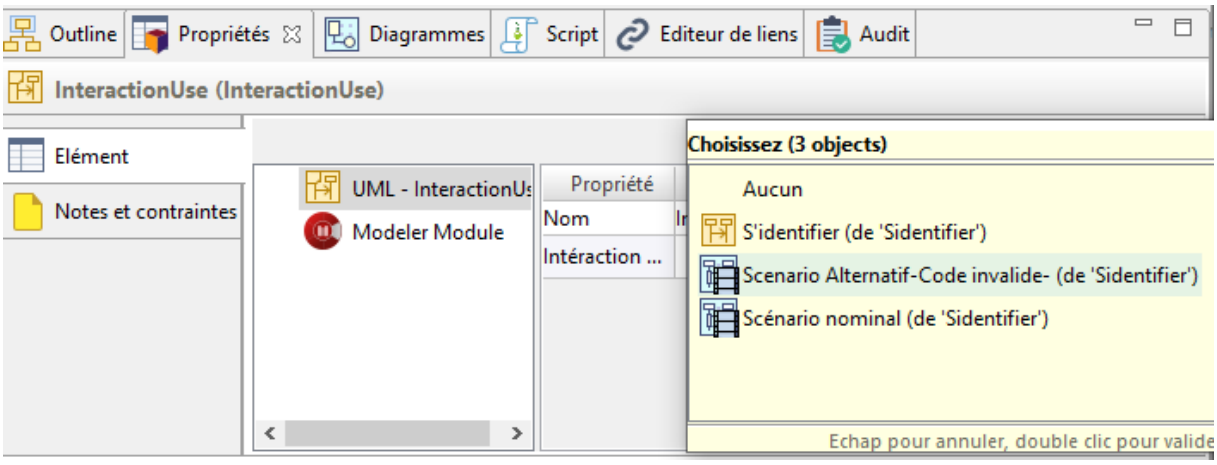



*Figure 3-7: Choisir le diagramme à référencer*

## 3.3. Exercice d'application

### *3.3.1. Enoncé*

Créer le scénario nominal du cas d'utilisation '*émettre une demande de congé'* . La description est donnée dans la série de Tp 2 (section 2.2.1)

### *3.3.2. Corrigé*

La figure 3-9 donne le diagramme de cas d'utilisation du cas *'émettre une demande' de congé* réalisé par Modélio. Le diagramme de séquence a été ajouté dans le cas d'utilisation *'Emettre demande'* qui se trouve sous le package '*Cas PlanCongé'* du projet '*TP_SGDC'* (voir figure 3-8)

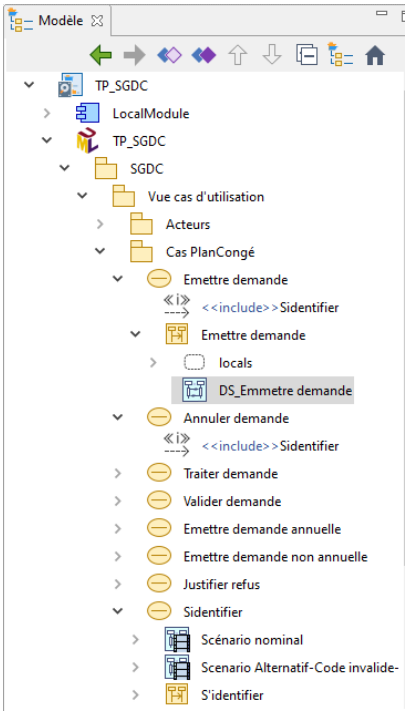



*Figure 3-8 : Explorateur de modèle du système SGDC (DS_Emettre Demande)*

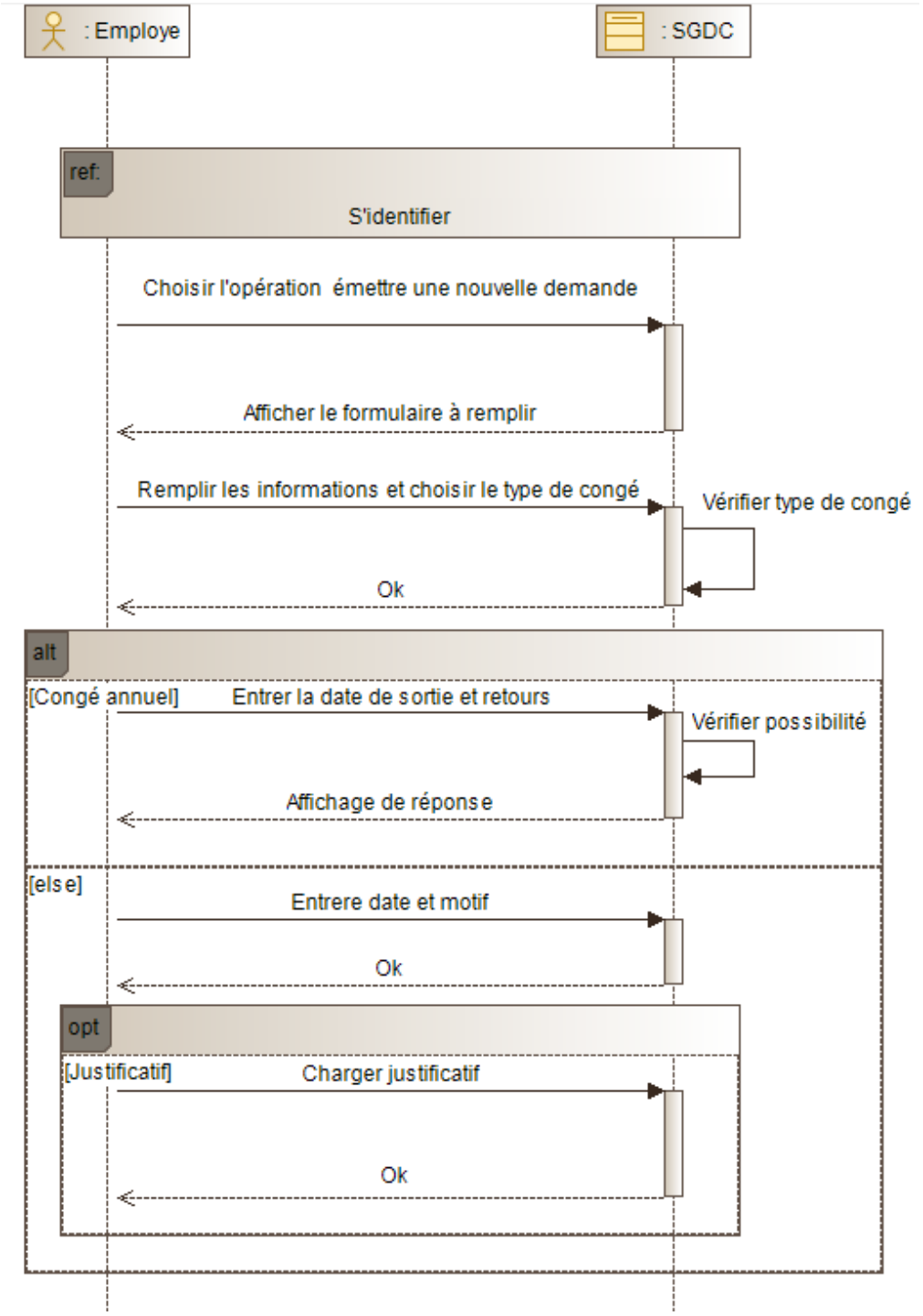



*Figure 3-9 : Diagramme de séquence (scénario nominal) du cas 'Emettre demande'*

# TP 4 : Description de l'aspect statique : Diagramme de classes

## 4.1. Objectif et rappels

Nous avons vu dans les Tps précédents comment modéliser les besoins des utilisateurs en définissant les fonctionnalités du système. Cette modélisation est faite à travers le diagramme de cas d'utilisation. La dynamique (les interactions) de chaque cas d'utilisation est ensuite définie de manière textuelle et peut être réalisée par un ou plusieurs diagrammes de séquence système pour représenter les différents scénarios. Nous verrons dans ce TP comment modéliser la structure logique du système, c'est-à-dire les aspects statiques du système à travers le diagramme de classes.

Le diagramme de classes est l'un des diagrammes le plus important dans une modélisation objet. Il est utilisé dans la phase d'analyse et conception :

- En analyse, il a pour objectif de décrire la structure des entités manipulées par les utilisateurs.
- En conception, le diagramme de classes représente un niveau de détail plus important comme la spécification du sens de la navigabilité, la visibilité et types des attributs, …,etc. En effet, Spécifier le sens de navigation n'est utile que dans la conception dans la phase du passage au développement, le sens de la navigation permet à la classe de prendre connaissance de l'objet de l'autre classe selon les cardinalités :

1 : un attribut de type objet de la classe est rajouté,

* : une liste d'objets est rajoutée.

Les détails ajoutés au diagramme de classe permettent la génération, par la suite, de la structure d'un code orienté objet (java, c,..) ou schéma physique (sql) d'une base de données.

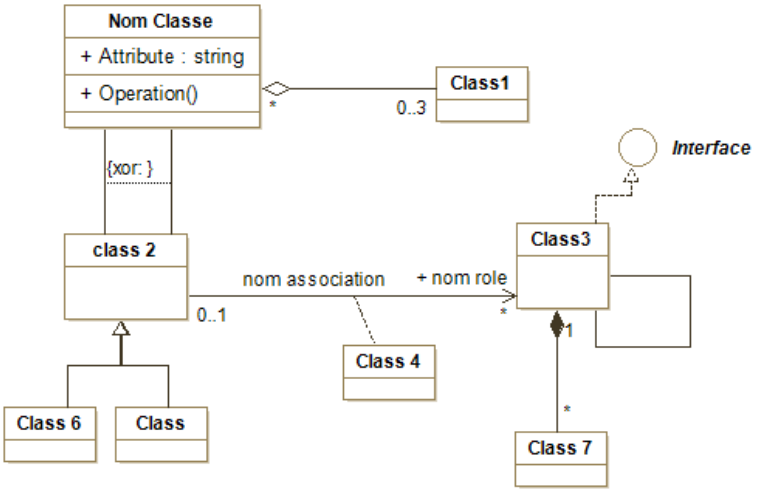
### *4.1.1. Les éléments du diagramme de classes*

Le tableau 2 présente un résumé (notation et descriptif) des éléments les plus importants du diagramme de classe.

**DIAGRAMME DE CLASSE**

**Objectif** : représenter les informations du domaine (relatives à l'aspect statique du système)

**Formalisme** :



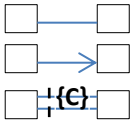
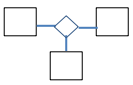
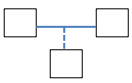
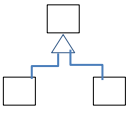
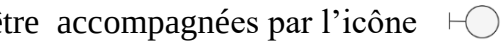

| Eléments | | Notation |
|---|---|---|
| **Classe** | Ensemble **d'objets** sur lesquels on peut reconnaître des similitudes. |  |
| **Attribut** | Information ***élémentaire*** composant une classe. | |
| **Opération** | Fonctionnalité assurée par la classe | |
| **Association** | Association<br>Navigation<br>Association avec contrainte |  |
| **Association réflexive** | Association mettant en relation une classe avec elle-même. |  |
| **Association n-aire** | Association mettant en relation plusieurs classes. |  |
| **Classe association** | Association porteuse d'attributs. |  |
| **Multiplicité** | Nombre d'instances impliquées dans l'association. | |
| **Agrégation faible** | Notion de contenance |  |
| **Composition** | Agrégation forte |  |
| **Généralisation spécialisation** | Permet d'identifier parmi les objets d'une classe (générique (super-classe)) des sous-ensembles d'objets (des classes spécialisées (sous-classes)) ayant des définitions spécifique |  |

*Tableau 3 : Aide-mémoire de la notation du diagramme de classe*

### *4.1.2. Types de classes et les stéréotypes associés :*

Le diagramme de classe appartient à la vue logique d'un système, il décrit son aspect statique. Il contient différents types de classes.

1- Classes entité : qui représentent la logique **métier**. Elles sont représentées par le stéréotype <<entity>>. Elles sont relevées dans la phase d'analyse. Certains outils permettent de les représenter par un cercle 

2- Classes d'IHM : qui décrivent la **présentation**. Elles sont représentées par le stéréotype <<boundary>> et peuvent être accompagnées par l'icône 

3- Classes de contrôle : qui représentent la logique **applicative.** Elles sont représentées par le stéréotype <<control>> et l'icône cercle avec flèche 

Les deux derniers types de classe sont rajoutés et détaillés dans la phase de conception. Ces stéréotypes sont dits de JACOBSON [8]. Ils ne sont pas obligatoires mais nécessaires pour représenter le système avec une architecture MCV (Model Contrôleur Vue) pour plus de détails voir le livre de UML 2 par pratique.

## 4.2. Pratiquez : tutoriel à suivre

Pour pratiquer et se familiariser avec les diagrammes de classes, nous allons reprendre la description du système SGDC décrit dans le TP 2. Cette description est complétée par les règles de gestion suivantes :

- L'entreprise contient plusieurs filiales et chaque filiale contient plusieurs services, certains services peuvent être communs à plusieurs filiales.
- Un employé est identifié par son numéro de carte professionnelle. Il est affecté à un service et peut émettre plusieurs demandes de congé.
- Un service a un responsable (Chef de service). Le responsable est un employé de l'entreprise.
- Un employé a un supérieur hiérarchique. Ce dernier est le supérieur de plusieurs employés.
- La demande de congé est traitée par le Chef de service (responsable) du demandeur, on enregistre la date du traitement de la demande et la réponse de la demande (acceptée ou rejetée). On souhaite aussi avoir l'information sur le nombre d'employés de l'entreprise.

Nous commencerons dans ce TP par relever les classes entités du système. Pour cela, ouvrer le projet SGDC et créer, au même niveau que le package 'Vue cas d'utilisation', un premier package nommé *'Vue Logique'*, ce dernier contiendra le package *'Logique métier'* qui va contenir les classes entités persistantes de notre système (figure 4-1).

Créer ensuite le diagramme de classe : *créer un diagramme→ diagramme de classe*)

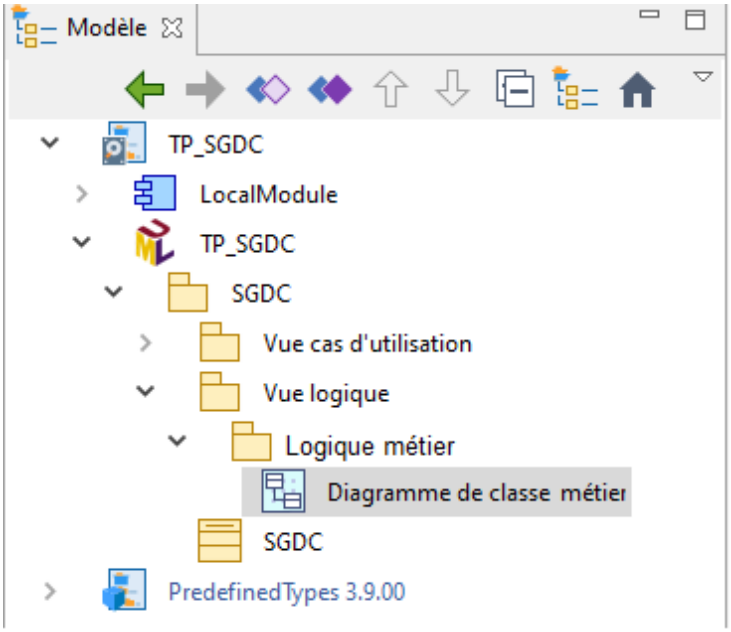



*Figure 4-1 : Organisation des packages du SGDC*

### *4.2.1. Créations d'une classe*

- Pour ajouter une nouvelle classe, sélectionner '*Class'*, représentée avec l'icône  du groupe '*Modèle de classes'* de la palette graphique, puis cliquez sur votre diagramme de classes à l'endroit souhaité pour la faire apparaître. Renommez la classe '*Employé*' (en utilisant F2 par exemple)
- Les stéréotypes de Jacobson ne sont pas prédéfinis dans Modelio, contrairement à d'autres outils, mais rien de plus simple pour les ajouter. Pour montrer que c'est une classe entité, on peut créer le stéréotype <<*Entity*>>. Pour cela, à partir de l'explorateur du modèle en se plaçant sur la classe concernée, puis en ouvrant d'un clic droit le menu contextuel pour choisir '*Créer un stéréotype'* donner le nom *'Entity'*. Il suffit de l'ajouter par la

suite a toutes les classes du même type *'ajouter un stéréotype'* et faire apparaitre le texte du stéréotype comme on a l'habitude de le faire dans l'onglet *Symbol → mode d'affichage des stéréotypes* : text

### *4.2.2. Ajouter des attributs*

Pour créer un nouvel attribut, il est possible d'utiliser la palette graphique et de faire un glisser-déposer de l'icône *'Attribute'* sur la classe concernée. Ou soit directement sur le diagramme à partir du raccourci de création 'a', une fois la classe sélectionnée. L'attribut apparaît alors dans la classe et il suffit de le sélectionner et de le renommer (F2) ou modifier la propriété *'Nom'* dans l'onglet *Propriété→élément* .Rajoutez l'attribut *'titre'*, *'durée'* et *'prix'* à la classe formation

**Remarque:** par défaut le type de l'attribut est string, nous verrons comment paramétrer les propriétés de l'attribut dans la section qui suit.

### *4.2.3. Propriétés d'un attribut*

Comme tout autre élément, un attribut possède des propriétés dans l'onglet *'Elément'* et *'Symbole'*. Les propriétés d'un attribut vont pouvoir être modifiées à travers l'onglet *'Elément'* (Figure 4-2).

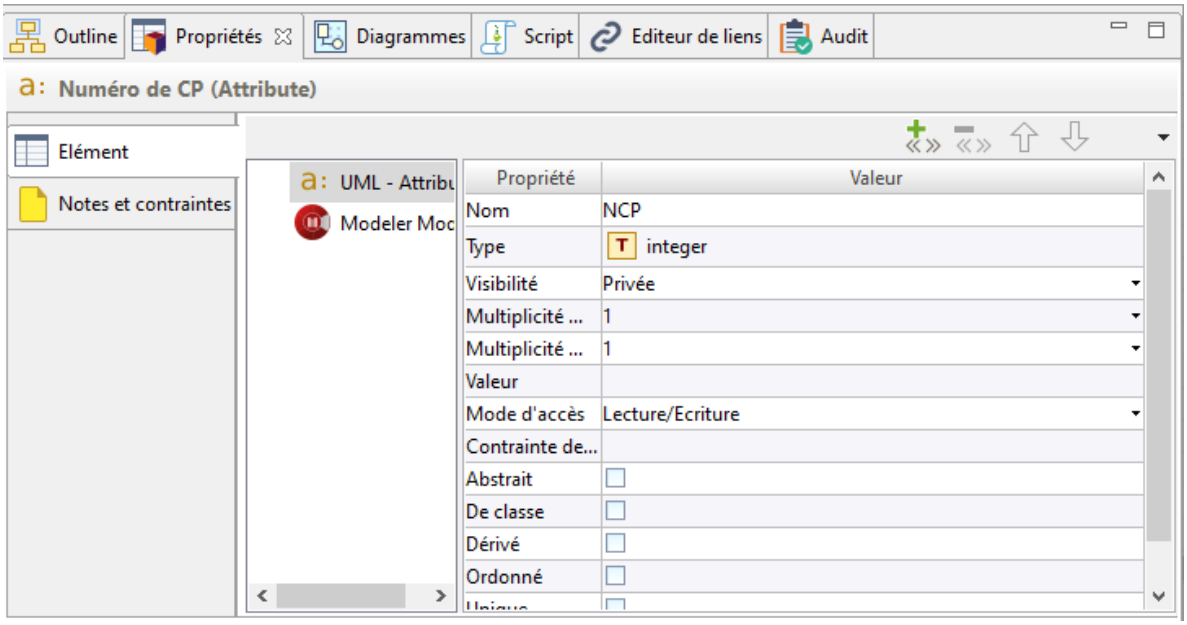



*Figure 4-2 : Propriétés des attributs*

#### *4.2.3.1. Visibilité*

Par défaut, le niveau de visibilité est public '**+**'. Nous allons modifier le niveau de visibilité de tous les attributs afin de les rendre tous privés '**-**'

Placez-vous sur le premier attribut numéro de carte professionnelle '*NCP'*. Rendez-vous dans son onglet '*Elément'*. Changez la valeur de la propriété '*Visibilité'* de '*Publique'* vers '*Privée'*. Faites-en de même avec les autres attributs.

#### *4.2.3.2. Type prédéfini*

Par défaut, la proprité '*type'* est '*string'*. Placez-vous sur l'attribut '*NCP'*. Ensuite, sur la valeur de la propriété '*Type'*. Ecrivez '*integer'* à la place de '*string'*. Vous pouvez alors utiliser l'auto-completion c-à-d. faire un CTRL+ESPACE. Modelio vous propose de choisir le type '*integer'* qu'il connait '*integer (from 'UML Type')'*, double cliquez dessus pour l'affecter comme nouvelle valeur de la propriété '*Type'*. Modifiez, de même, la propriété '*Date_Prise_Fonction'* en transformant le type '*string'* en '*date'*.

Notez qu'on peut aussi cocher les options : '*dérivé* ', '*de classe'* pour montrer qu'un attribut est dérivé ou statique (attribut classe). D'autres options peuvent aussi être sélectionnées comme '*abstract', ...* , etc.

#### *4.2.3.3. Type énuméré*

Vous remarquerez que le type de l'attribut '*Motif Congé'* de la classe 'Demande' (voir diagramme figure 4-3) est de '*type Motif'*. Ce dernier, n'est pas un type prédéfini, ni une classe (on est au niveau analyse, les attributs de type classe sont rajoutés au niveau conception détaillée au moment de la génération de code). Ce type est un '*type énuméré'* pour le définir, il faudra rajouter à partir de la palette une énumération (icône 12... ) que vous nommerez '*Type Motif*. Pour chaque motif de congé, il faudra rajouter un *Enumeration literal* (icône ) ) comme pour les attributs.

#### *4.2.3.4. Attribut de classe*

L'attribut '*NBEmployé'* de la classe '*Entreprise'* (voir Figure 4-3) est un attribut de classe. Il est représenté par un soulignement de l'attribut dès que la propriété est cochée dans l'onglet '*élément'*.

#### *4.2.3.5. Attribut dérivé*

L'attribut dérivé est un attribut dépendant d'autres attributs comme c'est le cas de l'attribut '*durée'* de la classe '*Demande'*. Il est représenté par le caractère '/' dès que la propriété '*Dérivé'* est cochée

- Rajoutez les autres classes ( avec attributs) recensées dans la figure 4-3.

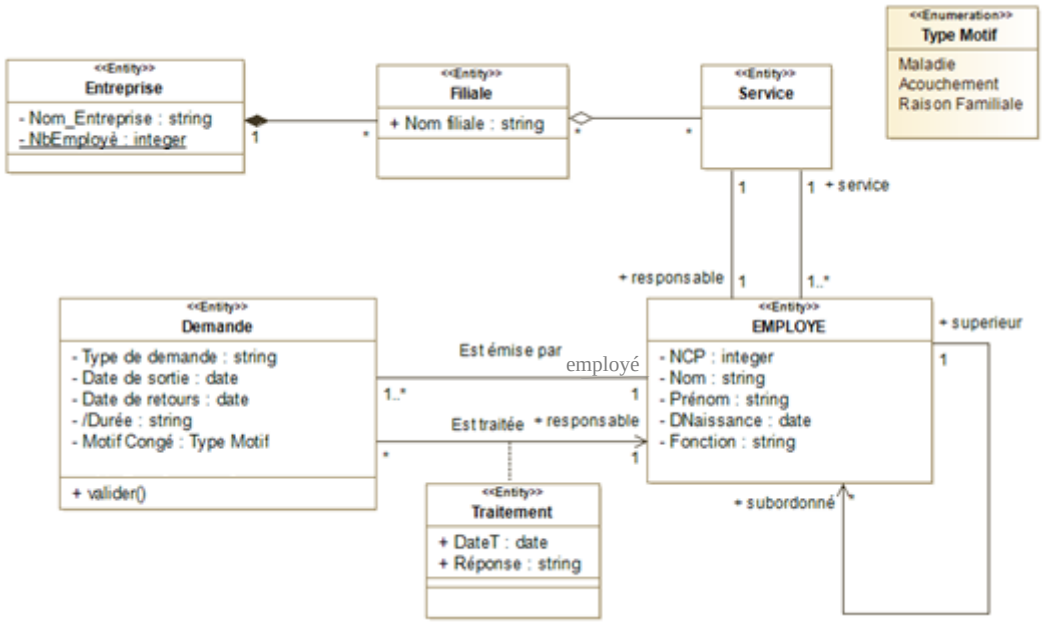



*Figure 4-3 : Diagramme de classe modélisant une partie du SGDC*

### *4.2.4. Ajouter une opération à une classe*

À cette étape de l'analyse, il s'agit seulement d'identifier les classes avec attributs. Les opérations sont ajoutées dans des itérations ultérieures du modèle; ceci, après avoir identifié les opérations à partir des diagrammes d'interactions (communication et séquence) qui seront vus dans les prochains TP. Mais à titre d'exemple, on peut rajouter l'opération '*valider()'* à la classe '*Demande'*. Pour créer une nouvelle opération dans une classe, on procède de la même manière que pour créer un nouvel attribut :

- Soit à partir de la palette graphique et d'un glisser-déposer de l'icône '*Operation'* dans le diagramme sur la classe concernée.
- Soit directement sur le diagramme à partir du raccourci de création 'o', une fois la classe sélectionnée.

- Soit à partir de l'explorateur du modèle en se plaçant sur la classe concernée, puis en ouvrant d'un clic droit le menu contextuel pour choisir (*Créer un élément element → Operation*)

Pour le moment, nous ne souhaitons pas créer d'opérations dans nos classes. Cette étape sera détaillée dans le prochain TP après avoir défini les diagrammes d'interactions pour relever les opérations des classes.

### *4.2.5. Créer une association*

Nous allons à présent créer l'association entre la classe '*Service'* et '*Employé'*. Pour créer une nouvelle association, il est possible d'utiliser la palette graphique. Sélectionnez l'icône association ↦ dans le diagramme sur la classe concernée.

Cliquez sur la classe origine de l'association : '*Service'*. Cliquez sur la classe destination de l'association : '*Employé'.*

**Remarque**: Un raccourci clavier est également disponible pour créer plus rapidement une association. Il s'agit de la touche ESPACE.

Une association navigable est alors créée entre les deux classes. Le rôle de la classe destination apparaît et possède le même nom que la classe Employé. Une multiplicité de 0..1 est attribuée par défaut à chaque classe. Il est donc maintenant nécessaire de paramétrer cette association en fonction de nos besoins dans l'onglet *'Elément'* qui va nous permettre de retrouver toutes les propriétés liées à l'Association (multiplicités, nom de l'association et rôles, navigabilité, etc)

Rajouter les autres associations selon leur type à partir de la palette et compléter le diagramme.

### *4.2.6. Le module 'profil de persistance'*

Notez bien que nous aurions pu utiliser le module '*Profil de persistance'* pour créer le diagramme de classes entités. Néanmoins, nous pouvons encore, à ce stade, transformer notre diagramme de classes en un diagramme logique de données relationnelles qui sera, par la suite, automatiquement transformé en un script SQL pour la création de la base de données.

Pour cela, rajouter le module *'Profil de persistance'* à votre projet *configuration→modules→ajouter→PersistentProfile3.10.00.* (figure 4-4)

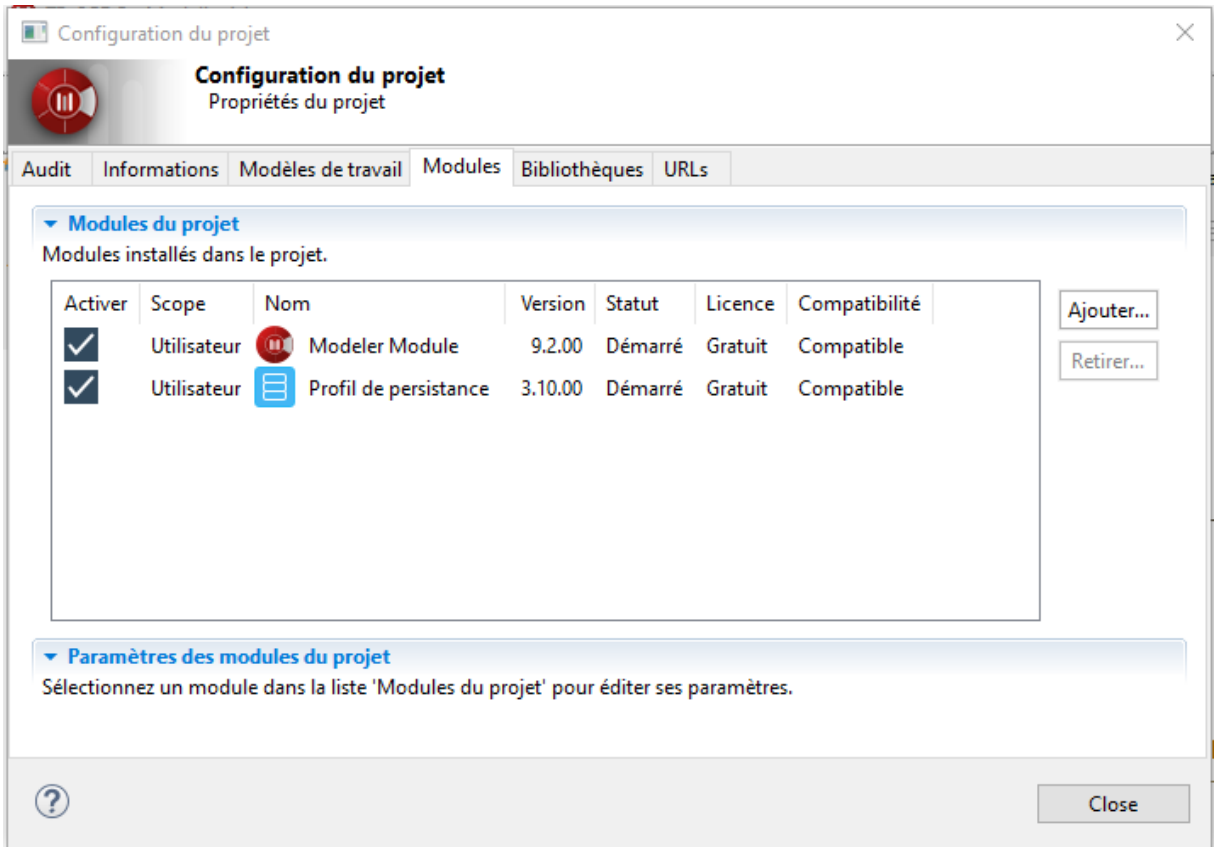



*Figure 4-4 : Le module profil de persistance*

Positionnez-vous sur votre package *'Logique métier'*, bouton droit de la souris et choisir : *Profil de persistance → transformer le modèle UML en modèle de données*.

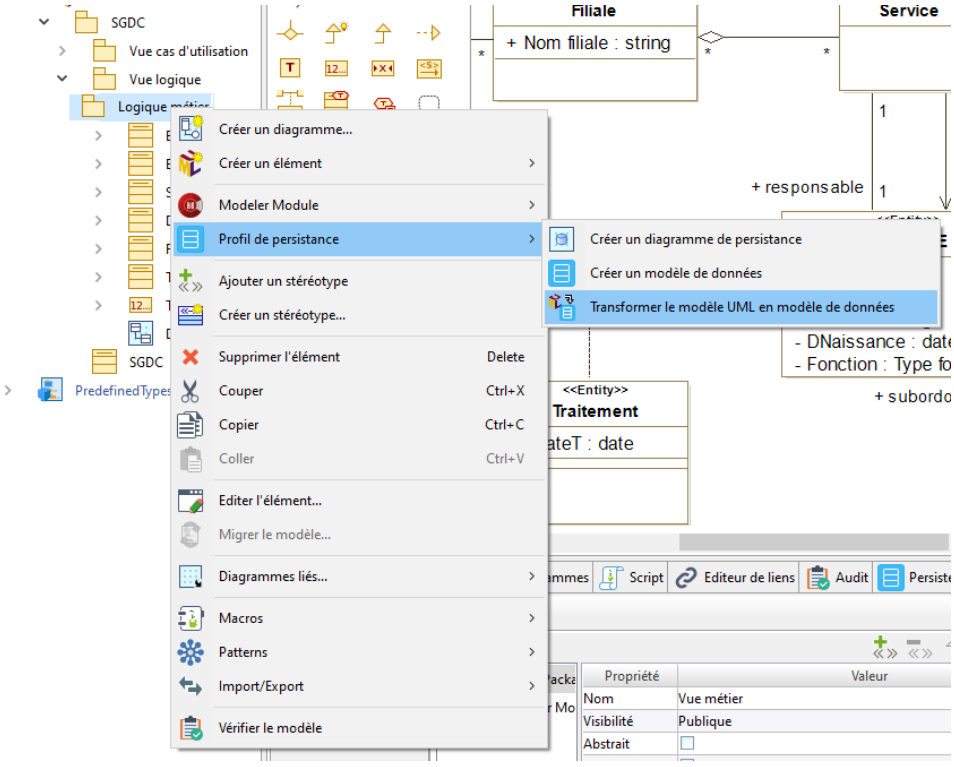



*Figure 4-5 : Transformer le modèle UML en modèle de donnée*

Votre diagramme est transformé en un modèle relationnel de données. Un nouvel onglet '*Persistence'* a été rajouté avec les autres onglets (propriétés, Java..etc.). Sélectionner un attribut, une association ou une classe et cliquer sur l'onglet '*Persistence'* pour pouvoir rajouter un identifiant, déterminer la foreigne key et les autres propriétés et contraintes bases de données.

Noter qu'on peut directement créer un module de données, un diagramme de données avec le bouton droit de la souris positionné sur le package : *profil de persistance→créer un modèle de données*, ou encre *créer un diagramme de persistance*.

Il ne vous reste plus, par la suite, qu'à générer votre code SQL (avec une version d'essai de Modelio SqlDesigner) ou un autre outil gratuit.

### *4.2.7. Exemple de génération du code java :*

La génération du code ne vient qu'en dernier lieu après avoir complété et renseigné tout le diagramme de classes dans la phase de conception détaillée, mais à titre d'exemple nous allons générer le code java de la classe '*Employé*'.

Pour cela il faudra d'abord rajouter le module *'java designer'*. Pour cela, allez dans le menu *configuration→ modules → ajouter* et décocher *'n'afficher que les dernières version's* et choisir *java designer 3.10.00* (pour Modelio V4.1.00) sinon la version adéquate selon votre version de Modelio. L'onglet java est rajouté avec les autres en bas de l'écran.

Sélectionnez la classe '*Employé'* et dans l'onglet Java sélectionnez *'Elément Java'*. Ceci indique que la classe fera l'objet d'une génération de code.

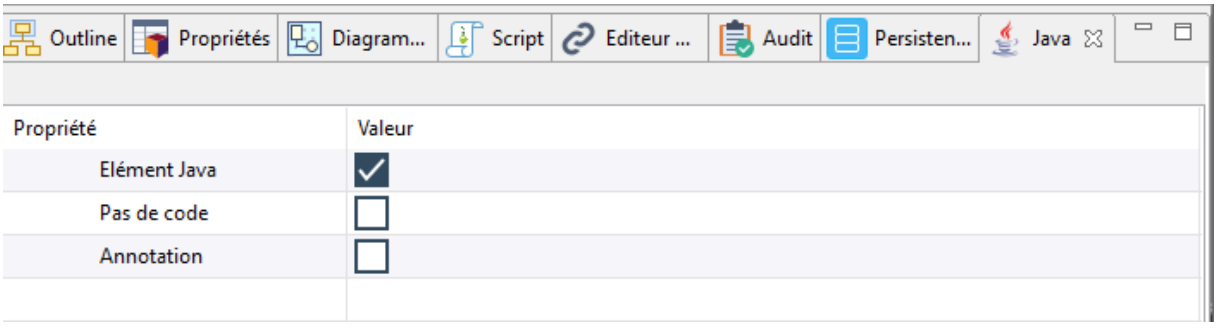



*Figure 4-6 : L'onglet java*

Sélectionnez l'attribut '*NCP'* et indiquez dans l'onglet '*Java'* qu'à cet attribut devront correspondre des accesseurs en Java (c.à.d. cocher les cases *'setter'* et *'getter'*).

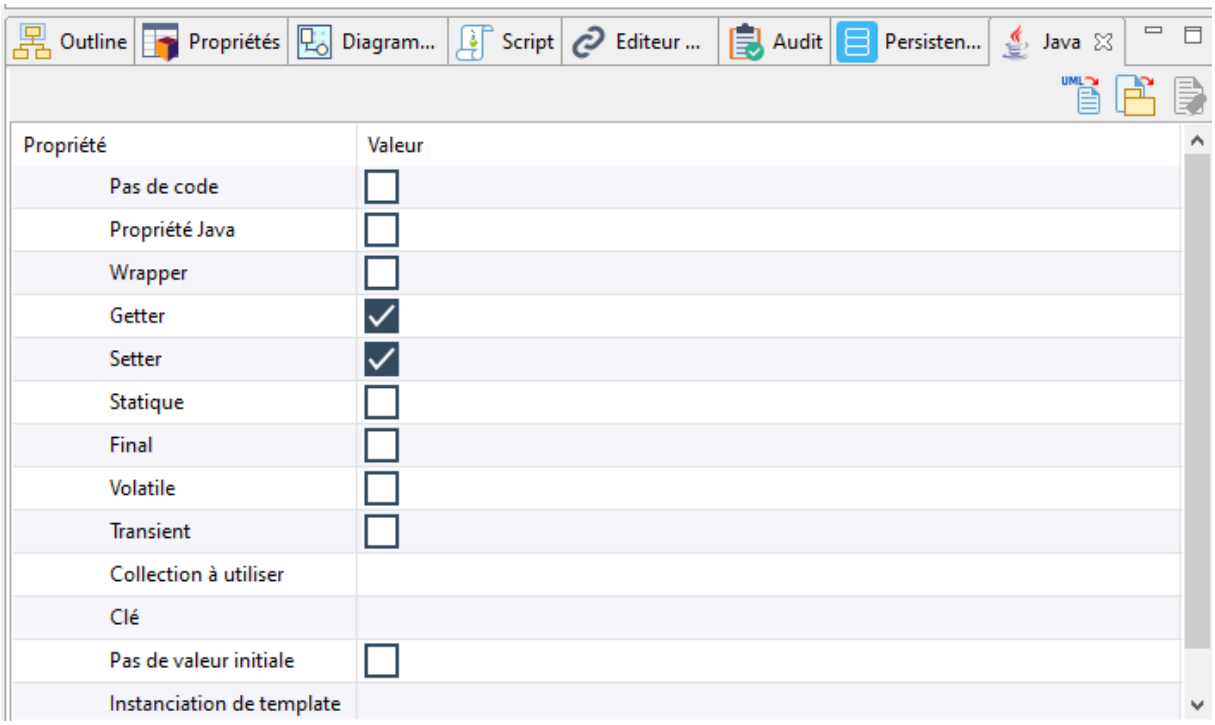



*Figure 4-7 : Propriétés de l'attribut dans l'onglet java*

Les méthodes *'setTitre()'* et *'getTitre()'* apparaissent dans l'arborescence de la classe *'Employé'* (figure 4-8), mais pas forcément dans le diagramme. En effet, plusieurs diagrammes de classes (analyse, conception détaillée) peuvent être définis pour un seul projet, chacun représentant une partie ou un niveau de détail différent du système. Il est donc nécessaire de faire apparaître explicitement ce que l'on veut dans chaque diagramme de classes.

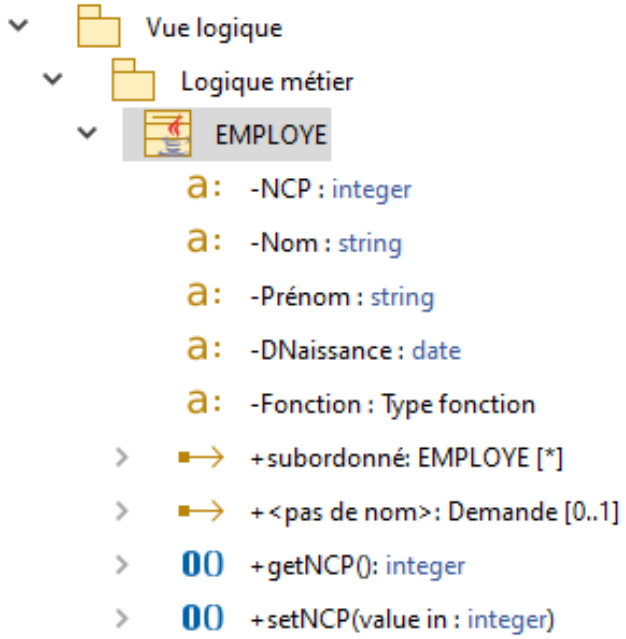



*Figure 4-8 : Setter et getter ajoutés à la classe 'employé' dans l'arborescence du Modèle*

- Pour faire apparaître les accesseurs (s'ils n'apparaissent pas dans le diagramme), glissez-les de l'arborescence vers la classe '*Employé'*.

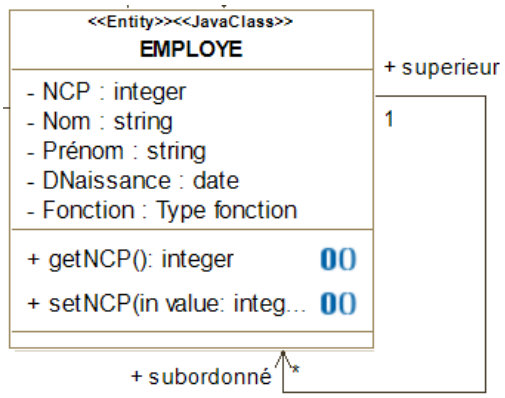



*Figure 4-9 : Classe Employé avec les getters et setters et le stéréotype JavaClass*

**Remarque** : avant de générer le code java, il faudra orienter la navigation de toutes les associations en relation avec la classe '*Employé'*. Faute de quoi, la variable de références '*service'* (exprimée grâce aux rôles) ne sera pas rajoutée à la classe '*Employé'*. La même chose pour les autres classes, Par exemple la génération de la classe '*Demande'* rajoutera deux variables de la classe 'E*mployé'* (employé et responsable)

Sélectionnez la classe Formation dans le diagramme et générer le code au moyen du menu contextuel *java Designer → Générer* . On peut faire de même à partir de l'élément correspondant dans l'arborescence du projet.

- Pour afficher le code généré, il suffit de sélectionner *java Designer →edit.* Le code source généré par Modélio apparaît dans une fenêtre.

*EMPLOYE | DS_Scénario nominal | DS_Code_Invalide | SD_S'identifier

```
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import com.modeliosoft.modelio.javadesigner.annotations.objid;

@objid ("623808d5-f4df-4e31-bc18-da34ffac1b1a")
public class EMPLOYE {
   @objid ("d65f6715-f6c5-4abb-91d8-448e126369dc")
   private int NCP;

   @objid ("4593349c-0812-487e-ada9-158acff7e166")
   private String Nom;

   @objid ("f6bf67e6-00af-47d1-b951-1af13671acca")
   private String Prénom;

   @objid ("536c2b8d-ab3a-4720-8140-73825e082bef")
   private Date DNaissance;

   @objid ("96e55060-891c-40d9-a09e-01990056bf52")
   private String Fonction;

   @objid ("ea1a78d6-2fce-413a-bb16-113c3a02151f")

   @objid ("7859ba01-b75f-435f-bb65-20e97ecbea53")
   public Service service;

   @objid ("f816657a-eaf7-4e63-b462-d8c89a1b453e")
   public Service ;

      public int getNCP() {
      // Automatically generated method. Please delete this comment before entering specific code.
      return this.NCP;
   }

   @objid ("5673b9a6-377c-4aaf-89a2-ce01dc33bade")
   public void setNCP(final int value) {
      // Automatically generated method. Please delete this comment before entering specific code.
```

*Figure 4-10 : Fenêtre du code java généré automatiquement par Java Designer*

## 4.3. Exercice d'application

### *4.3.1. Enoncé*

Une société de formations souhaite gérer les cours dispensés dans ses écoles privées. Pour cela, on dispose des renseignements suivants :

- Chaque école possède un site web et est structurée en départements.
- Un département regroupe des enseignants. Parmi eux, l'un en est responsable.
- L'enseignant et l'étudiant sont définis par un numéro, le nom, prénom, tél et mail. L'enseignant est défini aussi par la date de prise de fonction et son domaine. L'étudiant, quant à lui, est défini aussi par son année d'entrée à l'école.
- Chaque enseignant ne dispense qu'une seule matière au sein de l'école. En revanche, une matière peut être enseignée par plusieurs enseignants mais a toujours lieu dans la même salle de cours qui a un nombre de places déterminé.
- Les étudiants suivent plusieurs matières et reçoivent une note pour chacune d'elles.

- Nous souhaitons pouvoir avoir des fonctions qui permettent de calculer la moyenne générale d'un étudiant.

Etablir le diagramme de classes correspondant sachant qu'on se limite à une seule année d'étude, car elle sera configurée au début de chaque année dans les paramètres du logiciel

### *4.3.2. Corrigé*

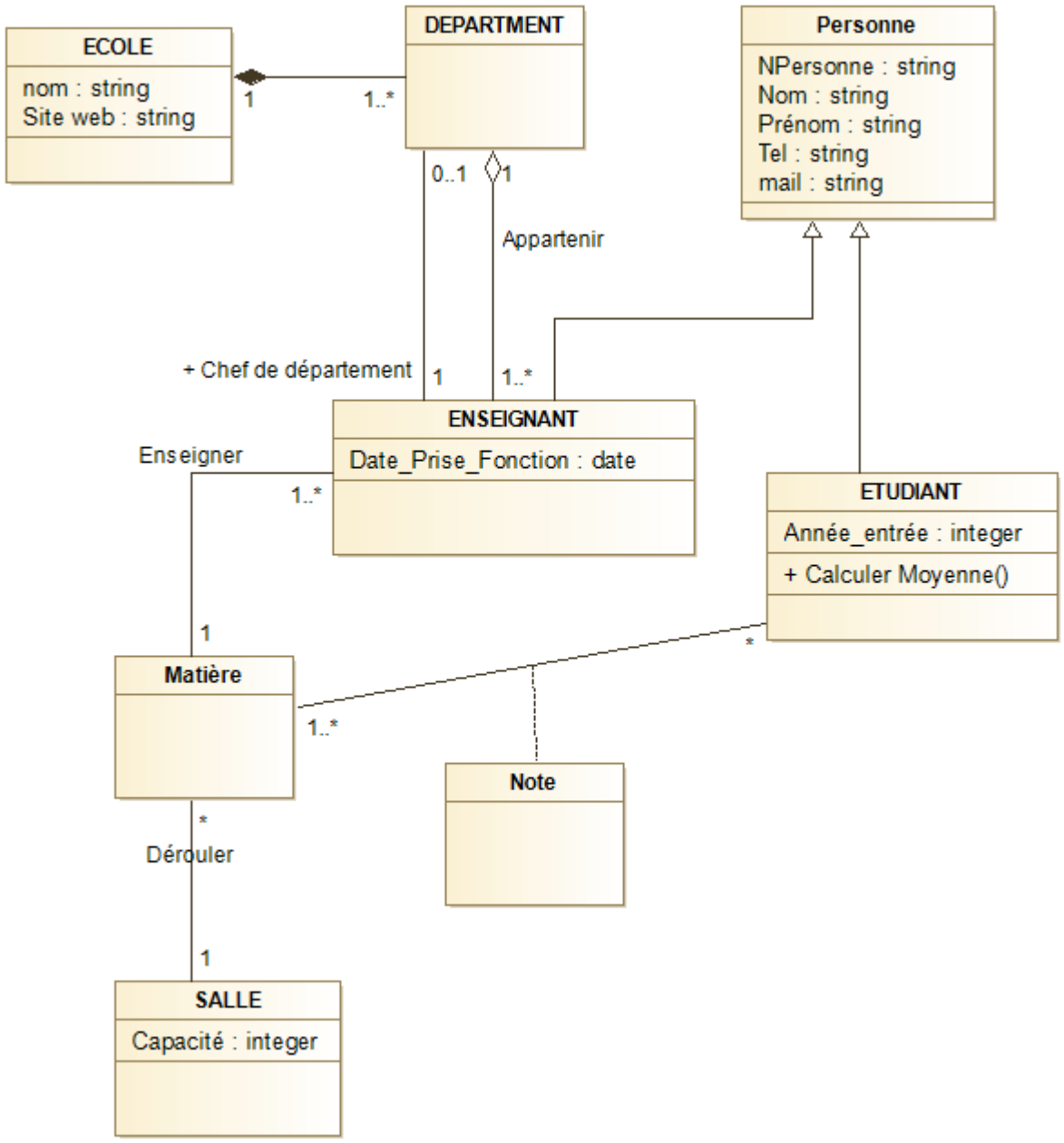



*Figure 4-11 : Diagramme de classe de l'exercice d'application*

# TP 5 : Diagramme de séquence (boite blanche) et Diagramme de classe

## 5.1. Objectif et rappels

Il y'a deux types de diagrammes de séquence : le diagramme de Séquence Système DSS (boite noire) et le diagramme de Séquence de conception appelé aussi objet (boite blanche) où les objets du système sont représentés. Pour chaque cas d'utilisation, un DSS (boite noire) est créé. De même, pour chaque DSS, un diagramme de séquence de conception (boite blanche) et un digramme de classes participantes [9] seront créés (voir figure 5-1). Le diagramme de classes participantes est présenté dans la méthode proposée par Roque [9]. C'est un diagramme préliminaire qui permet de délimiter les classes par cas d'utilisation. La réunion de tous les diagrammes doit aboutir au modèle statique d'analyse. Dans ce TP, nous allons voir comment créer un diagramme de classes participantes et de séquence de conception correspondants à un DSS

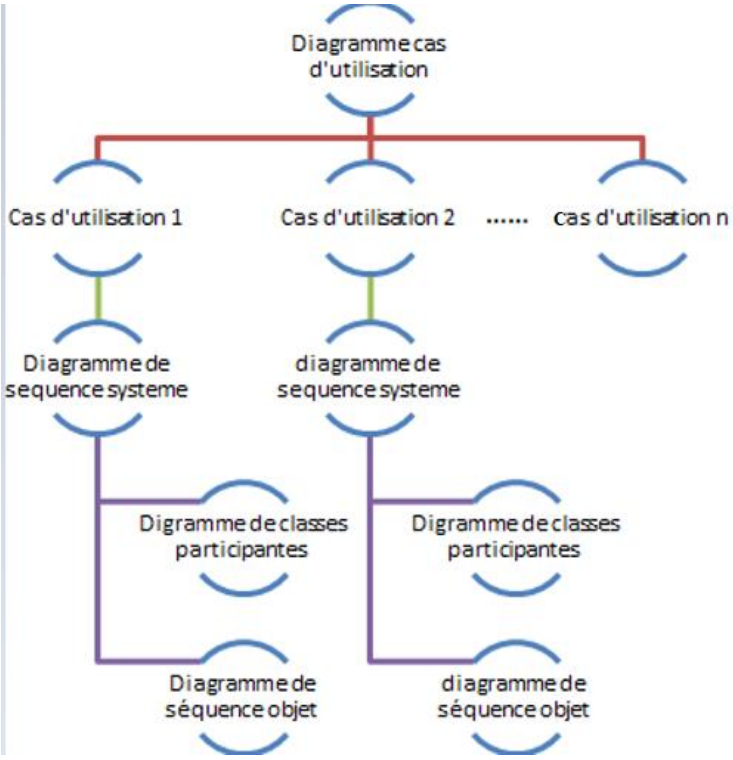



*Figure 5-1 : Relations entre diagramme de cas d'utilisation, séquence système, classes participante et séquences objets*

## 5.2. Pratiquez : Tutoriel à suivre

Ce TP s'inscrit dans la continuité des Tps précédents. Nous avons déjà établi, dans le TP passé (TP 4) le diagramme de classes entités du domaine stéréotypées <<*Entity*>> (modèle persistant qui constituera la base de données de notre application) du système SGDC.

Nous allons dans ce TP, rajouter certaines classes IHM stéréotypées *<<boundary>>* ou tout simplement *<<IHM>>* et enfin d'autres classes de contrôle *<<Control>>*. Pour cela, ouvrez votre projet *'TP_SGDC'* et rajoutez, dans la vue logique, deux autres packages :

a) *'Logique présentation'* pour les classes IHM ;

b) *'Logique applicative'* pour les classes de contrôle.

Nous allons, dans ce qui suit, créer le diagramme de classes participantes et le diagramme de séquence de conception pour le cas d'utilisation *'Emettre (ou Ajouter) demande'*. Ce cas est déclenché par l'acteur *'employé'*.

### *5.2.1.* Création du diagramme de classes participantes 'DCP_Ajouter_Demande'

Ajouter la classe *'Ecran_Demande'* dans le package *'Logique présentation'* et la classe *'Control_Demande'* dans le package 'logique applicative', créer et rajouter aussi les stéréotypes adéquats *(<<boundary>>* ou *<<IHM>>* ou *<<Dialogue>>* pour les classes présentation et *<<Control>>* pour les classes du package 'logique applicative'). Créer un digramme de classe dans le package 'vue logique'. Nommez le *'DCP_Ajouter_Demande'*. Ce diagramme permet de relever les classes relatives au cas *'Ajouter une demande de congé'* d'un employé.

Reproduire le diagramme suivant (Figure 5-2)

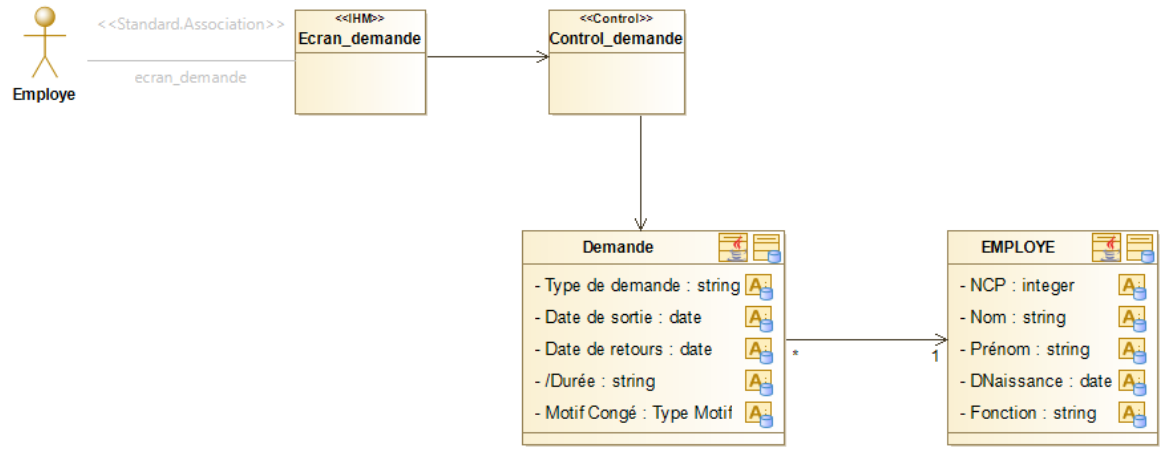



*Figure 5-2 : Diagramme de classes participantes préliminaire du cas 'Ajouter demande'*

### *5.2.2. Création du diagramme de séquence 'DS_Ajouter_Demande'*

Dans le package *'vue logique'* ajouter un nouveau package *'vue dynamique'* qui va contenir tous les diagrammes de séquences de votre application. Dans ce dernier, créer le

diagramme de séquence, nommer l'interaction : *'Ajouter_Demande'* et le diagramme de séquence *'DS_ Ajouter_Demande'*

#### 5.2.2.1. Rajouter des objets au 'DS_ Ajouter_demande' :

Déposer les objets à partir de : l'acteur *'Employé'*, la classe '*Ecran formation'*, la classe '*Controleur formation'*, la classe '*Demande'* (nommer l'objet *'D'*). Concernant le dernier objet *'Demandes'*, c'est une instance de la classe qui vient de la bibliothèque JDK. C'est la classe 'ArrayList' pour représenter une liste : 'multi Objet.' Vous ne pouvez ajouter cette bibliothèque à Modelio que si vous êtes connectés. Pour cela, allez dans *configuration→bibliothèques→Ajouter depuis le site de mise à jour* et cochez JDK. La librairie jdk est ajoutée (figure 5-3) , il faudra choisir la classe dans : *java→util→arraylist*


*Figure 5-3 : La librairie JDK ajoutée*

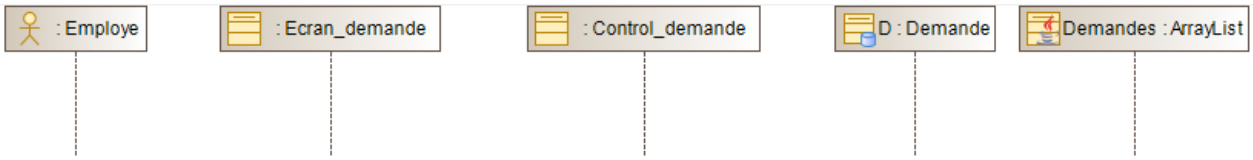



*Figure 5-4: Objest du DS_Ajouter_Demande*

#### 5.2.2.2. Rajouter des messages (opération) entre objets :

Ajouter un message synchrone entre les objets *': Employé'* et *':Ecran_Demande'*. Renseigner les propriétés du message avec '*Nom* : *Entrer informations'* et *'Argument* : *Type, Date de sortie, date de retours, motif et employé'*

Rajouter aussi les messages suivants (voir figure 5-5) :

- *'Initialiser Demande'* avec les arguments : *Type, Date de sortie, date de retours, motif, employé.*
- *'Create'* : c'est un message de type *'creation message'* de la palette. Mettre aussi les mêmes arguments *Type, Date de sortie, date de retours, motif, employé*
- *'Ajouter à la liste des demandes'* : c'est un message pour ajouter la demande qui vient d'être créée à la liste des demandes, son paramètre est '*D'*.

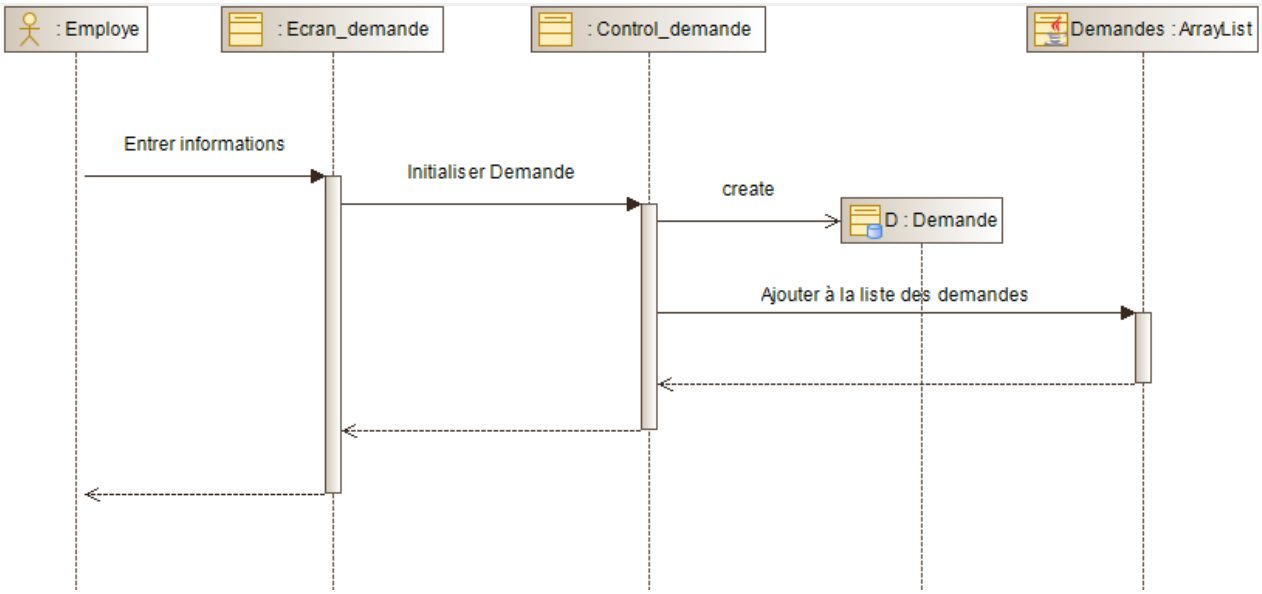



*Figure 5-5 : Objets et messages du DS_Ajouter_Demande*

#### *5.2.2.3. Transformer les messages en opérations :*

A présent, nous allons transformer ces messages pour les rajouter aux différentes classes concernées.

Pour cela, sélectionner le message en question exemple : '*Entrer informations'* et avec le bouton droit de la souris, sélectionner : *Modeler Module → créer une opération depuis ce message*. Vous pourrez modifier le nom de l'opération ou le laisser comme celui du message en question. Changer le nom de cette opération par *'initialiser Informations'*. L'opération a été ajoutée à la classe en question et la valeur de la propriété '*invoked'* a été mise à jour avec le nom de l'opération en question (Figure 5-6).

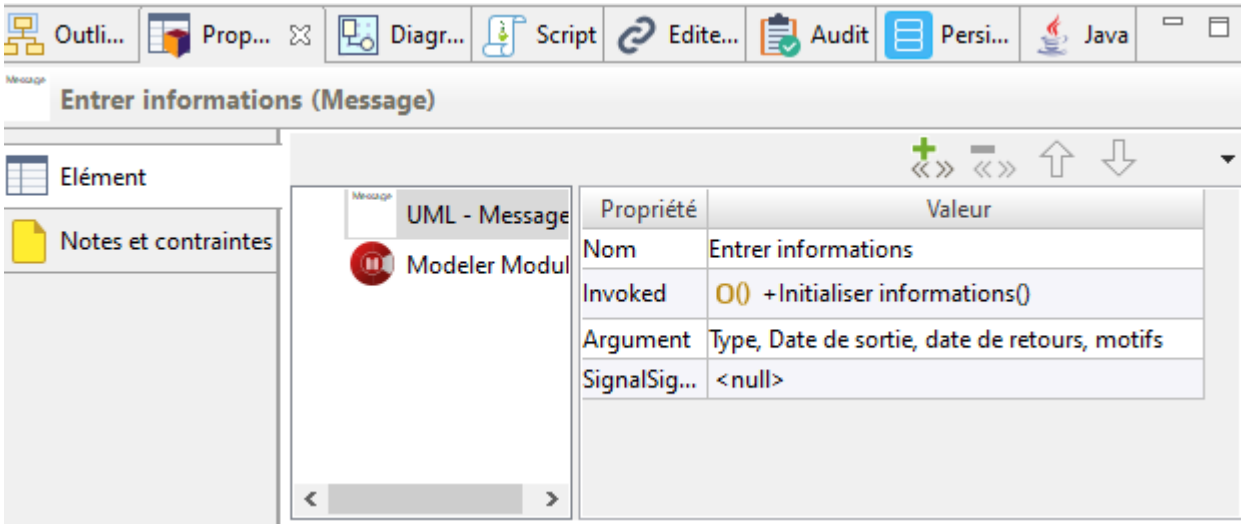



*Figure 5-6 : Propriété invoked du message Entrer informations*

Faites la même chose pour les autres messages sauf pour celui de 'ajouter' entre le contrôleur et l'objet *'Demandes'*. En effet, il ne s'agit pas d'ajouter une opération à la classe '*ArrayList'*, car elle est prédéfinie dans Java. De ce fait, il suffit d'invoquer la méthode '*add()'* de '*Arraylist'*. Donc, dans les propriétés du message en question, choisir la méthode pour la propriété 'invoked'.

Note : pour le message '*create'*, on peut d'abord créer un élément dans l'explorateur en se positionnant dans la classe formation et bouton droit, *Créer un élémnent* → *constructor* et on l'associe par la suite dans la propriété du message dans *invoked*

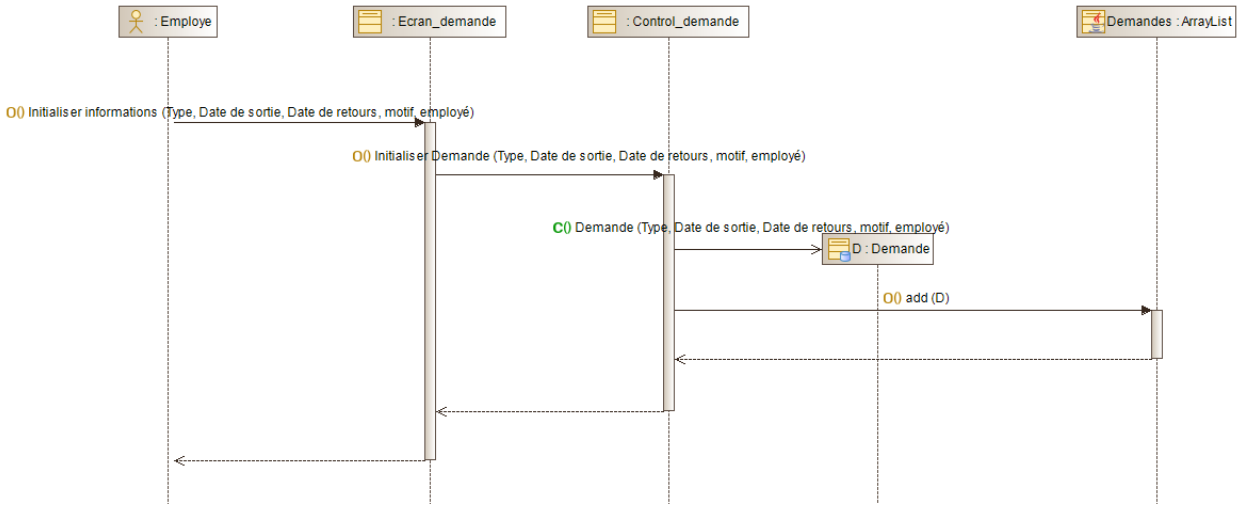



*Figure 5-7 : DS_ajouter_Demande avec signature des opérations*

#### *5.2.2.4. Propriétés des opérations:*

Sélectionner dans l'explorateur une des opérations qui vient d'être rajoutée et avec le bouton droit de la souris, sélectionner '*Editer élément'*. Rajouter les paramètres de chaque opération, pour désigner leur types, exemple Type : string , Date de sortie : date.

### *5.2.3.* Diagramme de classes participantes 'DS_Ajouter_formation' (2ème itération)

Revenez à votre diagramme de classes participantes et faire apparaitre les opérations de vos classes (voir figure 5-8), la nature des liens et le sens de navigation des liens. Noter qu'il faut créer un lien de dépendance entre la classe '*Control_demande'* et '*Employé'* car l'objet *Employé* sera utilisé juste comme paramètre dans une opération de la classe '*Control_demande'*

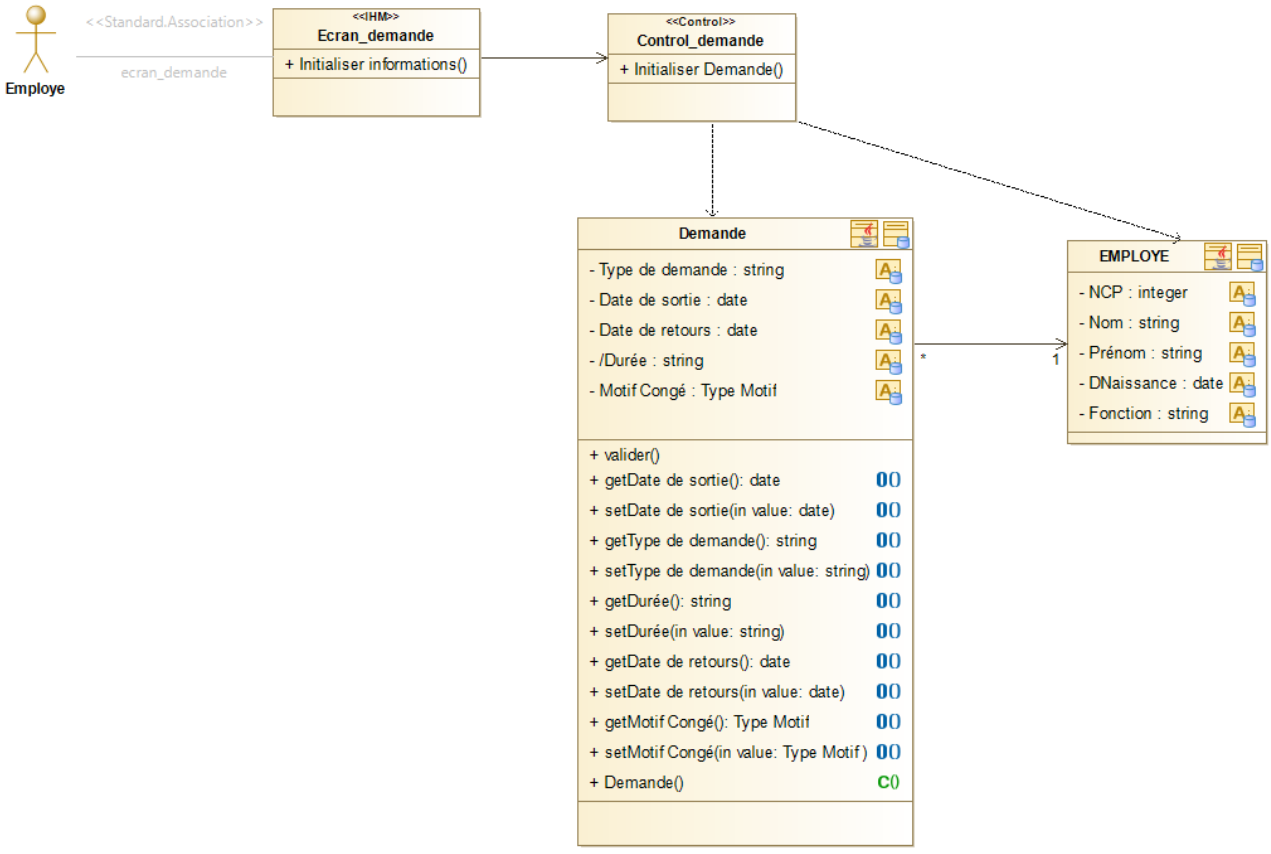



*Figure 5-8 : DCP_Ajouter_Demande (deuxième itération)*

## 5.3. Exercice d'application

### *5.3.1. Enoncé*

Soit une application de gestion d'emprunts de livres d'une bibliothèque universitaire.

Créer le diagramme de séquence qui permet à la bibliothécaire d'emprunter un exemplaire d'un livre de la bibliothèque (Cas d'utilisation 'Emprunter un exemplaire'). Sachant que l'étudiant et le livre demandé ont déjà été trouvés par le cas 'rechercher étudiant' et 'rechercher livre' et seront identifiés dans le diagramme par '*LivreTrouvé'* et *'EtudiantTrouvé'*. L'emprunt se fait à travers l'écran Emprunt

L'emprunt se fait de la façon suivante : l'opération *emprunter (EtudiantTrouvé)* de la classe LIVRE est invoquée. Si le nombre d'exemplaires du livre demandé est supérieur à 0 dans la bibliothèque ; alors un des exemplaires disponibles est extrait via la méthode *extraireExemplaire(),* une instance de la classe PRET est créée, puis l'exemplaire extrait de la bibliothèque est attribué à l'étudiant grâce à l'invocation de l'opération *attribuer(ExemplaireTrouvé).*

### *5.3.2. Corrigé*

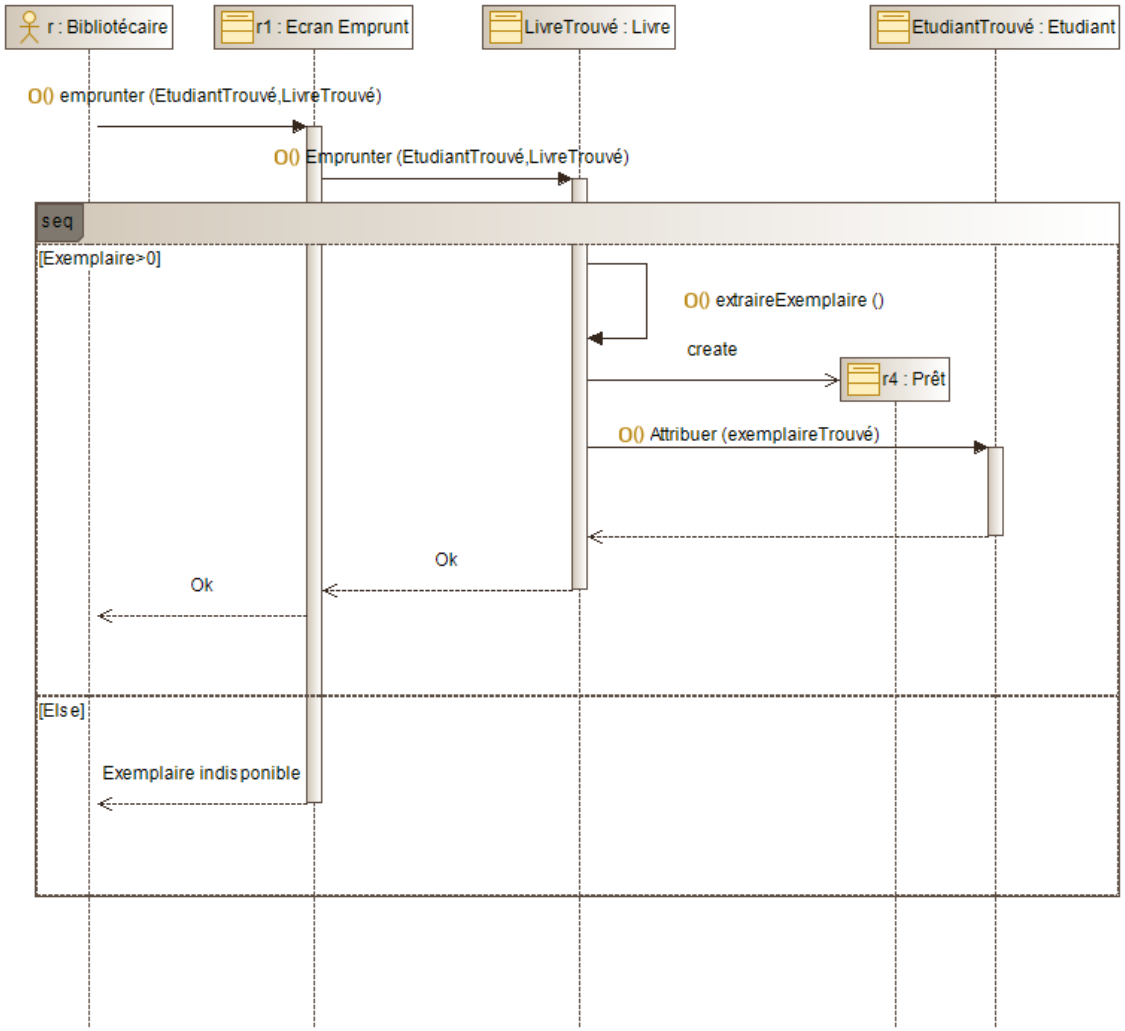



*Figure 5-9 : Diagramme de séquence d'emprunter exemplaire livre*

# TP 6 : Diagramme d’activité

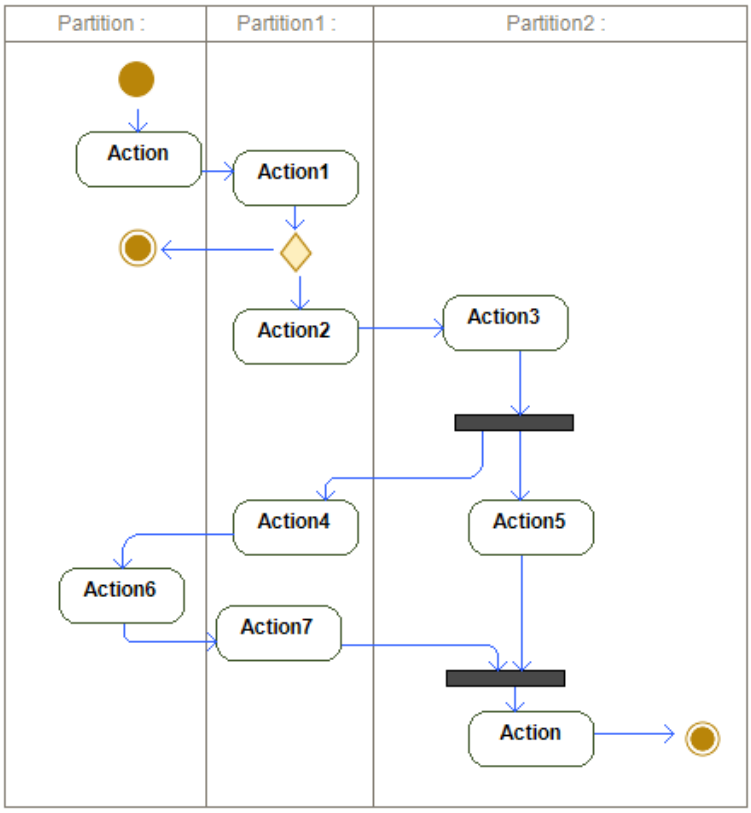
## 6.1. Objectif et rappels

Il s'agit dans ce TP de créer un diagramme d'activité pour représenter le processus *'Traiter une demande de congé'* du système 'SGDC' étudié dans les précédents Tps. Le diagramme d'activité peut être utilisé pour décrire une méthode, présenter un processus métier, ou encore le déroulement d'un cas d'utilisation. La définition et la notation des éléments les plus importants du diagramme d'activité sont résumées dans le tableau 4

**DIAGRAMME D'ACTIVITE**

Est une variante (simplification) du diagramme d'état -transition réduit à état- action
Il représente graphiquement des traitements comme le processus métier du système à étudier, le déroulement d'un cas d'utilisation ou encore le comportement d'une méthode.

**Formalisme** :




| Eléments | | Notation |
|---|---|---|
| Point de démarrage | Début du processus | |
| Point d'arrêt | Fin du processus | |
| Action | C'est un comportement décrit par un séquencement organisé d'unités. Les unités les plus simples sont des actions . | Action3 |
| Transition | Passage séquentiel d'une action à une autre | |
| | Passage avec prise de décision d'une action à d'autres | |

*Tableau 4 : Aide-mémoire de la notation utilisée dans le diagramme d'activité*

**Présentation du processus:**

1. Le processus de traitement de demande de congé est initialisé lorsque le responsable de service reçoit une demande de formation de la part d'un employé. Cette demande est examinée par le chef de service et transmet son accord ou son désaccord à l'intéressé.
2. En cas d'accord, le responsable RH étudie encore la demande pour voir s'il peut lui accorder le congé.
3. En cas de refus, le responsable RH doit justifier le refus à l'employé

Le processus comporte un ensemble d'actions ordonnées dans le temps et affectées à un des acteurs suivants : employé, chef de service et responsable RH

## 6.2. Pratiquez : Tutoriel à suivre

Pour réaliser le Tp, il faudra ajouter d'abord la **vue processus** à votre projet au même niveau que la vue logique du TP 4 et 5. Créer ensuite dans la vue processus un diagramme d'activité que vous nommerez 'Processus Traitement Demande'

### *6.2.1. Création des partitions (couloir)*

Pour créer des partitions, cliquez-déposez une icône *Vertical Partitions*  (que vous trouverez dans la catégorie Partitions) dans la fenêtre du diagramme. Par défaut, cela crée deux partitions. Si vous avez besoin d'une partition supplémentaire, cliquez sur l'icône *Sibling Partition*  et déposez à l'endroit souhaité à côté des partitions existantes. Chaque partition représente un acteur qui réalise certaines actions de l'activité. Un acteur a un nom et un type. Pour spécifier cela dans le diagramme, cliquez sur la partition concernée puis allez à l'onglet '*Elément'* où vous pouvez modifier la propriété '*nom'* et précisez le type du participant (dans la propriété '*Représenté par'*). Notez que le type (acteur en question) doit déjà être crée dans le projet. Pour rappel, les acteurs sont déjà créés dans la vue *cas d'utilisation* sous le package *Acteurs*.

### *6.2.2. Ajout des nœuds : initial, final*

Pour créer un nœud  initial , ou final  , cliquez-déposez une icône '*Initial Node'*, ou '*Activity Final Node'*, (que vous trouvez dans la catégorie '*Nœuds de contrôle'*) dans la fenêtre du diagramme. Dans le cas où vous avez des partitions, vous pouvez associer un nœud à une partition en le déposant dessus.

### *6.2.3. Ajout une action*

Pour créer une action, cliquez-déposez une icône '*Action'*  (que vous trouvez dans la catégorie '*Nœuds de contrôle'*) dans la fenêtre du diagramme et sur la partition correspondante.

Vous pouvez renommer l'action en cliquant dessus, puis en allant sur '*Elément'* de l'onglet '*Propriétés'*.

### *6.2.4. Ajouter une transition*

Pour créer une transition entre deux actions, cliquez sur l'icône *Flow (auto)*  (que vous trouvez dans la catégorie '*Flux'*) ensuite cliquez sur l'action de départ puis sur l'action d'arrivée.

### *6.2.5. Ajouter un nœud de décision*

Pour créer un nœud de décision ou de fusion, cliquez-déposez une icône '*Decision-Merge'*  (que vous trouvez dans la catégorie '*Nœuds de contrôle'*). Une transition sortante d'un nœud de décision a une condition de garde que vous pouvez préciser dans '*Elément*' de l'onglet '*Propriétés'*.

Reproduire le diagramme final donné dans la figure 6-1

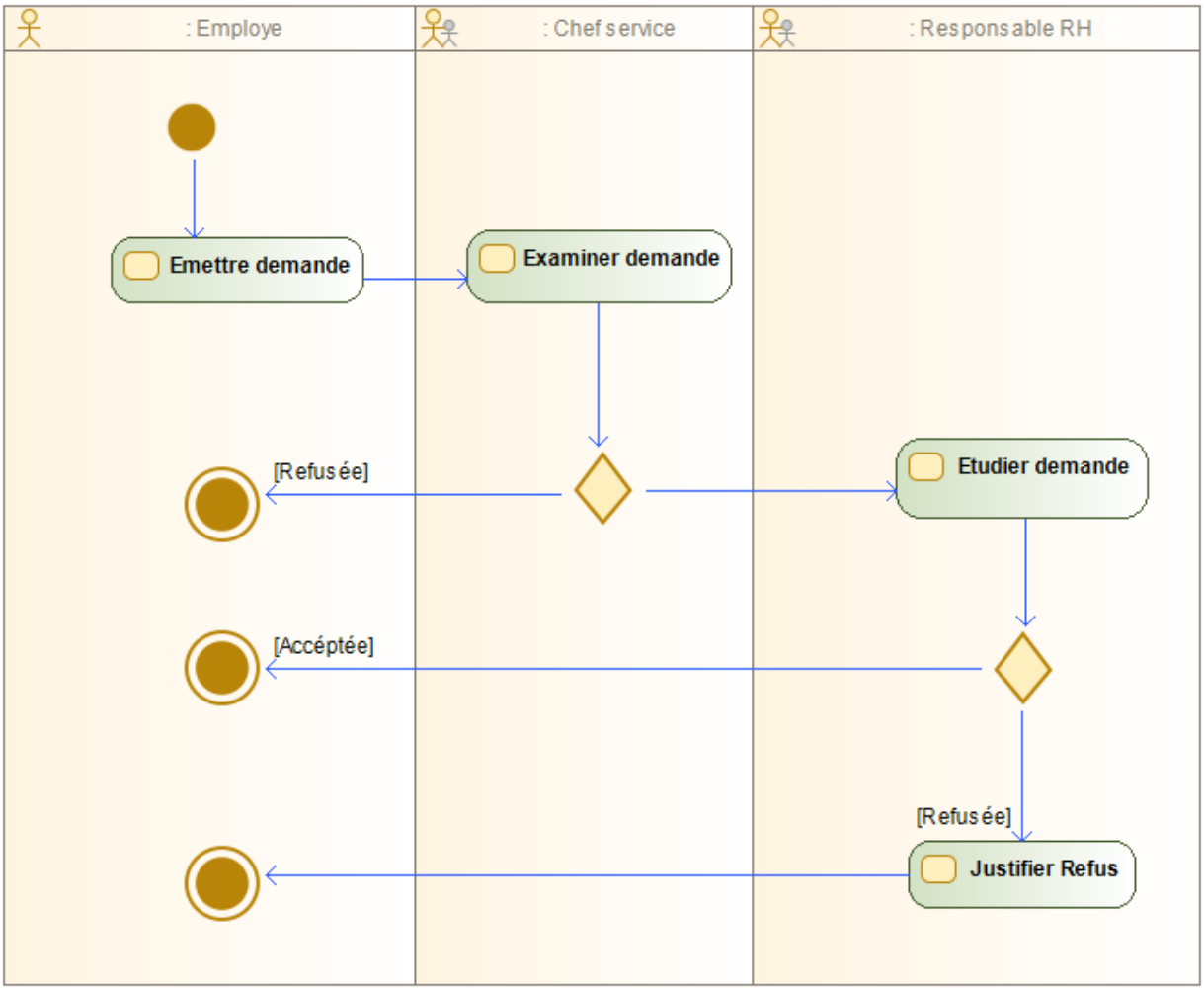



*Figure 6-1 : Diagramme d'activité du processus de demande de congé*

## 6.3. Exercice d'application

### *6.3.1. Enoncé*

Une entreprise permet à ses employés de bénéficier d'un ensemble de formations proposées par des organismes de formations. Pour cela, les employés peuvent émettre une demande de formation au responsable de formation au sein de l'entreprise. Le processus de formation est initialisé lorsque le responsable de formation reçoit une demande de formation de la part d'un employé.

Cette demande est instruite par le responsable qui la qualifie et transmet son accord ou son désaccord à l'intéressé.

En cas d'accord, le responsable recherche dans le catalogue des formations agrées un stage qui correspond à la demande de l'employé et lui propose une liste des prochaines sessions.

Lorsque l'employé fait son choix, le responsable de formation l'inscrit à la session auprès de l'organisme de formation concerné.

A la fin de sa formation, l'organisme de formation émet une facture qui sera contrôlée par le responsable de formation puis payée par le comptable.

Réaliser diagramme d'activité qui décrit le processus de traitement de formation.

### *6.3.2. Corrigé*

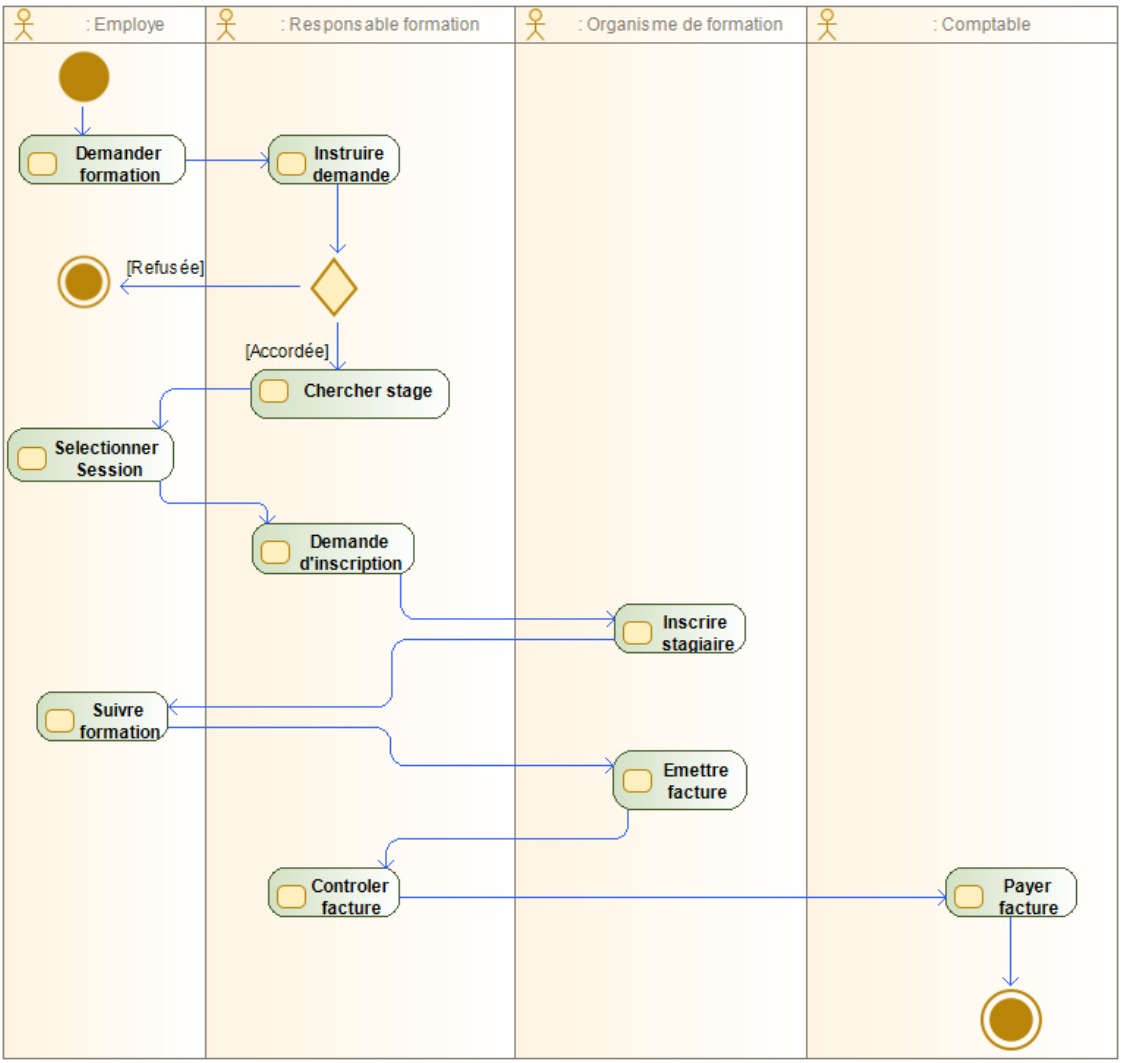



*Figure 6-2 : Diagramme d'activité du processus de demande de formation*

## Références webographiques

Le dernier accès à tous les sites web de ces références a été fait le 28/11/2021.

[1] https://manurnx.wp.imt.fr/2017/01/23/choix-d-un-outil-de-modelisation-uml/
[2] https://www.modelio.org/
[3] https://www.modelio.org/about-modelio/features.html.
[4] https://www.modelio.org/resources-menu/videos.html
[5] https://www.youtube.com/user/ModelioCommunity
[6] https://www.modelio.org/downloads/download-modelio.html.

## Références bibliographiques

[7] Guide pratique du RUP, P. Kruchten, P. Kroll, 2003,Campus Press
[8] Le processus unifié de développement logiciel, I. Jacobson, G. Booch, J. Rumbaugh, 2000, Eyrolles
[9] UML 2 en action. De l'analyse des besoins à la conception J2EE, 3ème édition P.Roques, F. Vallée, 2004, Eyrolles.
[10] UML 2 pour les développeurs: Cours avec exercices corrigés. X. Blanc, I.Mounier. Eyrolles, 2011.

# FACULTÉ DES SCIENCES

# Faculty of Sciences

# كلية العلوم

Faculté des Sciences – Tidjani HADDAM

Tél.: 043 21 63 70 / Tél &Fax: 043 21 63 68 / 043 21 63 71

Site Web: www.fs.univ-tlemcen.dz

Email : vdrpg.facscience@gmail.com