Démarche VUML Statique et Dynamique

Application à une Etude de Cas

Y. Lakhrissi[1,2,3] — B. Coulette[1] — I. Ober[1] — M. Nassar[2] — A. Kriouile[2,3]

{lakhriss, coulette, ober}@univ-tlse2.fr
{nassar, kriouile}@ensias.ma

[1] MACAO-IRIT Université de Toulouse
[2] SI2M ENSIAS Rabat
[3] ACSYS-LIMIARF Faculté des sciences Rabat

RÉSUMÉ. *VUML (View based Unified Modeling Language) est un profile UML, qui permet d'analyser/concevoir un système logiciel par une approche combinant objets et points de vue. Parmi les bénéfices de l'approche VUML, on peut citer l'utilisation d'un modèle unifié avec points de vue multiples au lieu de plusieurs sous-modèles interdépendants et souvent incohérents, l'instanciation par point de vue, la gestion de la cohérence entre sous-modèles, etc. Cependant, à ce jour, les travaux réalisés sur VUML ne traitent pas les aspects dynamiques de la modélisation (la modélisation comportementale). Les travaux abordés dans ce document visent à combler ce manque. Dans ce document, nous proposons une méthode de description séparée et de coordination des machines à états des objets vues et base –objets composant un objet multivue en VUML–, reposant sur un nombre de concepts simples à utiliser. Le résultat essentiel est l'identification de deux machines à états particulières, capables de capturer le comportement dynamique d'un objet multivue. Les machines à états proposées suivent la spécification UML2.0, et sont attachées soit à une classe « base », soit à une classe « vue ». Une machine-vue représente le cycle de vie d'un objet-vue, tandis qu'une machine-base a pour but de créer la cohérence et la coordination entre les différentes machines-vue, et de spécifier le comportement commun aux acteurs. On appelle machine « multivue » l'ensemble formé d'une machine-base et des machines-vue dépendantes. Nous proposons aussi une démarche traitant à la fois l'aspect statique et l'aspect dynamique lors de l'analyse/conception un système par point de vue. Cette démarche étend la démarche VUML proposée par Nassar dans [5]. Cette dernière ne traite que la structuration statique des applications conçues en VUML. La démarche que nous proposons est constituée de trois phases qui sont : (i) Analyse globale, (ii) Analyse/conception par point de vue, et (iii) Fusion. Chacune de ces phases engendre deux volets : le premier guide les développeurs dans la structuration statique d'une application en VUML, le deuxième volet complète le premier en aidant à exprimer le comportement dynamique des objets multivues présents dans l'application. Nous avons validé notre travail à travers l'étude d'un système gérant une agence de réparation de voitures. L'objet Voiture a constitué la cible de notre développement multivue dans le traitement des aspects statiques (principalement à travers le diagramme de classes) et dynamiques (à travers les machines à états).*

MOTS-CLÉS : *UML, VUML, modélisation multivue, machines à états.*

Table des matières

1. Démarche VUML générique

1.1. Introduction

Lors du développement d'un système complexe, la construction d'un modèle global prenant en compte simultanément tous les besoins des acteurs est parfois impossible. Dans la réalité, soit plusieurs modèles partiels sont développés séparément et coexistent avec les risques d'incohérence associés, soit le modèle global doit être fréquemment remis en cause quand les besoins des utilisateurs évoluent. Pour répondre à cette difficulté, notre équipe travaille depuis plusieurs années sur l'intégration de la notion de point de vue dans l'analyse/conception de systèmes logiciels. C'est ainsi qu'un profil UML a été développé, appelé VUML (View based Unified Modeling Language), qui permet d'analyser/concevoir un système logiciel par une approche combinant objets et points de vue [3,5,6].

Parmi les bénéfices de l'approche VUML, on peut citer l'utilisation d'un modèle unifié avec points de vue multiples au lieu de plusieurs sous-modèles interdépendants et souvent incohérents, l'instanciation par point de vue, la gestion de la cohérence entre sous-modèles, etc. ([5,3]).

Le principal ajout à UML est celui du concept de classe multivue. Une classe multivue est une unité d'abstraction et d'encapsulation qui permet de stocker et restituer l'information en fonction du profil de l'utilisateur. Elle offre des mécanismes de gestion des droits d'accès aux informations et de gestion de la cohérence entre les vues dépendantes. De plus, VUML propose un modèle de composant multivues qui permet de représenter une classe multivues au niveau du diagramme de composants [1,2].

Cependant, à ce jour, les travaux réalisés sur VUML ne traitent pas les aspects dynamiques de la modélisation (la modélisation comportementale). Les travaux abordés dans ce document visent à combler ce manque.

La modélisation comportementale est en effet très importante dans la démarche de conception d'un système complexe, surtout dans le contexte de l'ingénierie dirigée par les modèles, où l'objectif est d'arriver à une automatisation des phases post-conception (codage, intégration, validation, etc.), automatisation qui doit être fondée sur un modèle de conception le plus complet possible. La modélisation comportementale en UML peut se faire à plusieurs niveaux d'abstraction, en partant des modèles d'ensemble comme les modèles d'interaction et d'activités qui représentent les interactions et l'enchaînement des activités entre les différents objets ou les composants du système, et en allant jusqu'à la description fine du comportement des objets ou des composants par des machines à états.

Les modèles d'ensemble, tels que le diagramme de séquence, permettent par définition la description d'un comportement selon un point de vue ou la combinaison de plusieurs points de vue. Dans ce travail nous nous concentrons sur la description du comportement individuel des objets multivues par des machines à états qui, elles, nécessitent des adaptations au niveau des concepts de modélisation UML.

Nous proposons une méthode de description séparée et de coordination des machines à états des objets vues et base –objets composant un objet multivue en VUML–, reposant sur un nombre de concepts simples à utiliser. Le résultat essentiel est l'identification de deux machines à états particulières, capables de capturer le comportement dynamique d'un objet multivue [9]. Les machines à états proposées suivent la spécification UML2.0 [7], et sont attachées soit à une classe « base », soit à une classe « vue ». Une machine-vue représente le cycle de vie d'un objet-vue, tandis qu'une machine-base a pour but de créer la cohérence et la coordination entre les différentes machines-vue, et de spécifier le comportement commun aux acteurs. On appelle machine « multivue » l'ensemble formé d'une machine-base et des machines-vue dépendantes.

Nous proposons aussi une démarche traitant à la fois l'aspect statique et l'aspect dynamique, dans l'optique de mener une analyse/conception par point de vue. Cette démarche étend la démarche VUML proposée par Nassar dans [5]. Cette dernière ne traite que la structuration statique des applications conçues en VUML. La démarche que nous proposons est constituée de trois phases (cf. figure 1) : la première est la phase d'analyse globale, la deuxième est la phase d'analyse/conception par point de vue, et la troisième est la phase de fusion. Chacune de ces phases est constituée de deux volets : le premier guide les développeurs dans la structuration statique d'une application en VUML, le deuxième complète le premier volet en aidant à exprimer le comportement dynamique des objets multivues présents dans l'application.

Nous avons validé notre travail à travers l'étude d'un système gérant une agence de réparation de voitures. L'objet voiture a constitué la cible de notre développement multivue dans le traitement des aspects statiques (principalement à travers le diagramme de classes) et dynamiques (à travers les machines à états).

Ce document est structuré de la manière suivante. Dans le reste de cette section, nous donnons un résumé de la démarche proposée. Puis, la section 2 traite la première phase de notre démarche en détaillant les étapes à suivre lors de l'analyse globale. La section 3 traite la deuxième phase de la démarche, en décrivant les étapes à suivre pour mener une analyse conception par point de vue sur les deux niveaux statique et dynamique. Ensuite, la section 4 traite la troisième phase de la démarche qui est la phase de fusion, durant laquelle les modèles statique/dynamique VUML sont produits par fusion des modèles statique/dynamique partiels par point de vue. En conclusion, nous synthétisons les résultats de notre étude et décrivons ses principales perspectives.

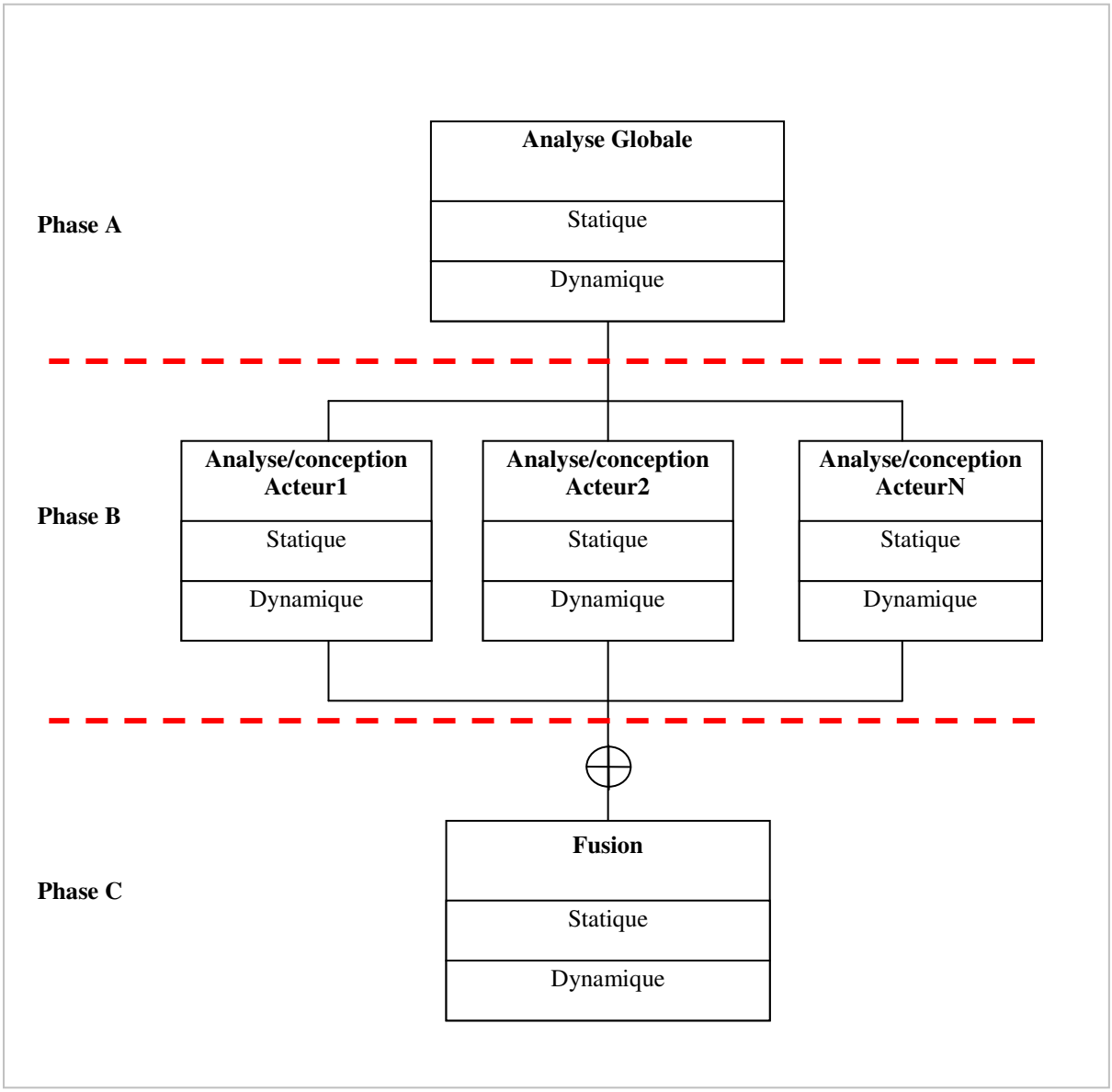

Figure 1. *Vue Générale de la démarche Statique/Dynamique VUML*

1.2. Résumé de la démarche

<table>
<tr><td rowspan="2">Phase A:

Analyse globale</td><td>Statique</td><td>à partir du cahier des charges :
1. Extraction des fonctionnalités principales du système.
2. Identification des acteurs (points de vue).
3. Détermination des besoins des acteurs à partir du cahier des charges.
4. Identification des cas d'utilisation des fonctionnalités principales en faisant apparaître les participations des acteurs.
5. Extraction des classes potentielles du système.
6. Réalisation d'un "modèle de base" donnant l'architecture du système.</td></tr>
<tr><td>Dynamique</td><td>1. Identification des classes réactives.
2. Identification des états potentiels pour chaque classe réactive.
3. Construction de la machine à états générale "machine-base" pour chaque classe réactive.</td></tr>
<tr><td rowspan="2">Phase B:

Analyse/ Conception par acteur</td><td>Statique</td><td>Pour chaque acteur :
1. Reprise des cas d'utilisation dont il est acteur principal.
2. Etablissement des diagrammes de séquence réalisant les cas d'utilisation.
3. Réalisation des diagrammes de classes préliminaires découlant des cas d'utilisation.
4. Elaboration du diagramme de classes détaillé pour le point de vue courant.</td></tr>
<tr><td>Dynamique</td><td>Pour chaque classe réactive identifiée dans la phase A :
1. Reprise de sa machine de base.
2. Etude de la signification de chaque état de la machine base pour l'acteur en question.
3. Création de la machine vue.
Exception: une classe réactive est identifiée lors du développement par point de vue :
1. Vérification de la présence de la machine-base associée dans le glossaire :
➢ Si elle existe : reprise et adaptation en cas de besoin.
➢ Sinon : développement et ajout de la machine-base dans le glossaire.
2. Développement de la machine vue.</td></tr>
<tr><td rowspan="2">Phase C:

Fusion</td><td>Statique</td><td>1. Harmonisation des modèles statiques élaborés pendant la phase B.
2. Détermination de la liste finale des classes multivues (celles identifiées dans le modèle de base et celles qui peuvent apparaître dans l'étape actuelle).
3. Fusion des modèles partiels par point de vue afin d’élaborer le modèle VUML global.</td></tr>
<tr><td>Dynamique</td><td>Pour chaque classe réactive multivue :
1. Harmonisation de la machine-base (si un acteur a rajouté un état supplémentaire, qui ne pourra pas être un raffinement des états existants)
2. Etablissement de la communication des machines vues par synchronisation.
Pour chaque classe réactive non multivue, la machine base devient inutile, on ne garde que la machine vue.</td></tr>
</table>

2. Phase A : Analyse Globale

L'analyse globale est la première phase de notre démarche. C'est une phase décentralisée de modélisation des exigences. L'objectif principal est d'arriver à identifier les besoins des différents intervenants du système, et les structurés en unités fonctionnelles sous forme de cas d'utilisations. Au cours de cette section, nous présentons les différentes étapes constituant cette phase, que se soit au niveau statique qu'au niveau dynamique. Nous illustrons toutes les étapes par des extraits de l'étude de cas "gestion d'une agence de réparation de voitures".

2.1. Analyse Globale Statique

2.1.1. Extraction des fonctionnalités principales du système

Le système d'information gérant cette agence de réparation doit assurer les fonctions suivantes :

- ***Gestion des voitures :*** le système doit être capable de gérer les voitures à l'intérieur de l'agence en assurant les tâches suivantes :
  - l'enregistrement des voitures et leurs propriétaires,
  - la gestion des opérations d'expertise et de réparation,
  - l'enregistrement des pannes détectées,
  - l'enregistrement des réparations effectuées,
  - la gestion des tests de vérification,
  - la sauvegarde de l'historique de chaque voiture une fois sortie du garage, inclus les historiques des pannes et les réparations apportées.

- ***Gestion du matériel :*** le système doit gérer les ressources matérielles :
  - la gestion des affectations des dépanneurs,
  - la gestion des affectations des outils,
  - la gestion des affectations des pistes (pour réaliser les expertises, les réparations et les tests),
  - la gestion des places du parking,
  - la gestion des pièces de rechange : affecter les pièces, signaler l'épuisement du stock pour une pièce particulière, commander des pièces auprès du fournisseur, etc.

- ***Gestion du personnel :*** gérer les ressources humaines de l'agence qui comporte :
  - l'affectation des électriciens/mécaniciens aux voitures nécessitant une expertise,
  - l'affectation des électriciens/mécaniciens aux voitures nécessitant une réparation,
  - l'affectation des électriciens et les mécaniciens aux voitures nécessitant un test.

- ***Gestion financière :*** attribuée au chef de l'agence, elle inclut :
    - la gestion des contrats avec les clients,
    - la gestion des contrats avec les fournisseurs,
    - la gestion des dépenses internes de l'agence.

2.1.2. Identification des acteurs (points de vue)

- Le client propriétaire de la voiture
- Le chef d'agence
- Le mécanicien
- L'électricien
- Le responsable atelier

2.1.3. Détermination des besoins des acteurs

En plus des tâches mentionnées ci-dessus, le système doit répondre aux besoins de chaque acteur, et doit leur fournir les fonctionnalités suivantes :

- au client propriétaire de la voiture :
    - consulter les informations concernant sa voiture,
    - suivre son évolution dans la chaîne de réparation,
    - consulter les rapports d'expertise et de réparation,
    - effectuer le test de vérification avant la livraison de sa voiture,
    - négocier les contrats avec le responsable financier de l'agence.

- au chef d'agence :
    - établir les contrats avec les clients,
    - établir les contrats avec les fournisseurs,
    - émettre les ordres aux mainteniciens pour démarrer l'expertise et la réparation,
    - faire le contrôle financier.

- aux mécanicien/électricien :
    - consulter l'historique des pannes,
    - rédiger et enregistrer les rapports d'expertise et de réparation,
    - pour chaque panne, enregistrer les pièces échangées,
    - enregistrer les détails des réparations apportées.

- au responsable atelier :
    - affecter les mainteniciens (électriciens/mécaniciens),
    - affecter les ressources matérielles (dépanneurs, outils de travail),
    - affecter les pistes pour effectuer les expertises, les réparations et les tests,
    - gérer le parking,
    - gérer et affecter les pièces de rechange.

2.1.4. Identification des cas d'utilisation

La figure 2 donne un aperçu général sur des fonctionnalités que le système doit assurer avec les acteurs responsables. Ces fonctionnalités sont: (1) la gestion des voitures, (2) la gestion des mainteniciens, (3) la gestion du matériel, (4) la gestion financière et (5) la supervision de l'agence.

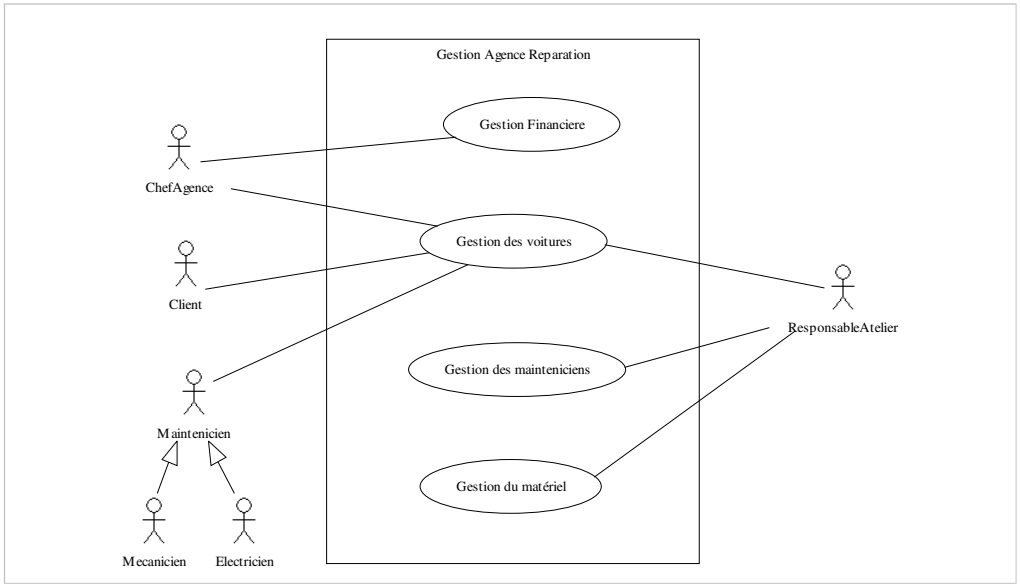

Figure 2. *Cas d'utilisation Général pour l'application "Gestion Agence Réparation"*

Les figures 3 et 4 détaillent les cas d'utilisation mentionnés dans la figure 2 avec les acteurs responsables.

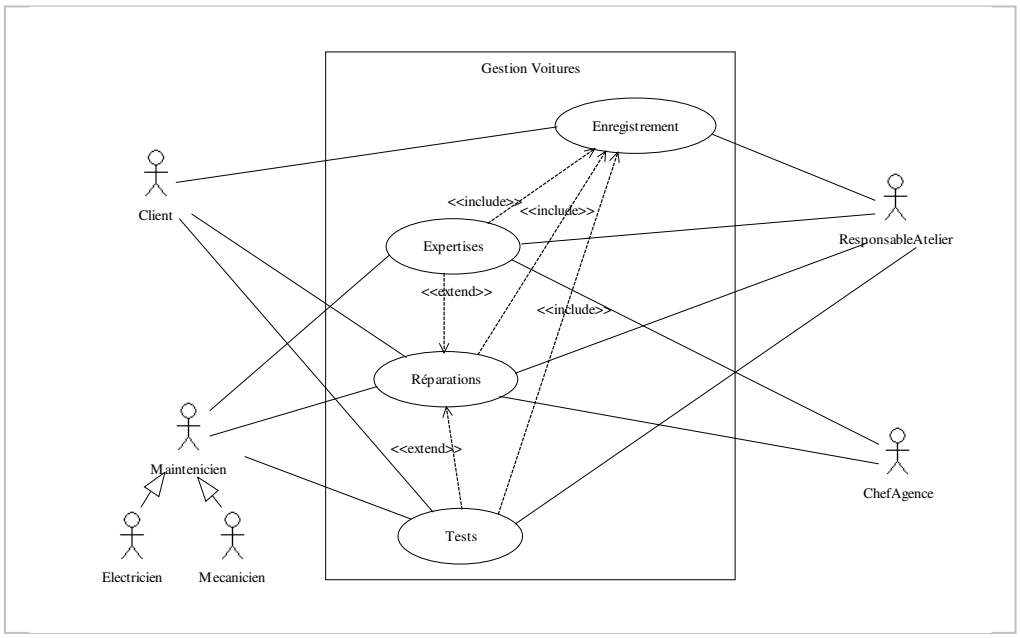

Figure 3. *Cas d'utilisation "Gestion de voitures"*

La figure 3 développe le cas d'utilisation "gestion des voitures". Gérer les voitures de l'agence revient à gérer leur enregistrement, les expertises et les réparations apportées à chacune d'elles, et les tests de vérification finals. Un client intervient dans le cas d'utilisation "enregistrement" par la communication des informations concernant sa voiture, il participe avec le chef d'agence à la réalisation des contrats d'expertise et de réparation et valide la réparation en effectuant le test final du

bon fonctionnement de sa voiture. Les mainteniciens interviennent dans les phases techniques telles que l'expertise, la réparation et les tests.

La figure 4 présente les cas d'utilisation "Gestion du matériel", "Gestion du personnel" et "Gestion financière".

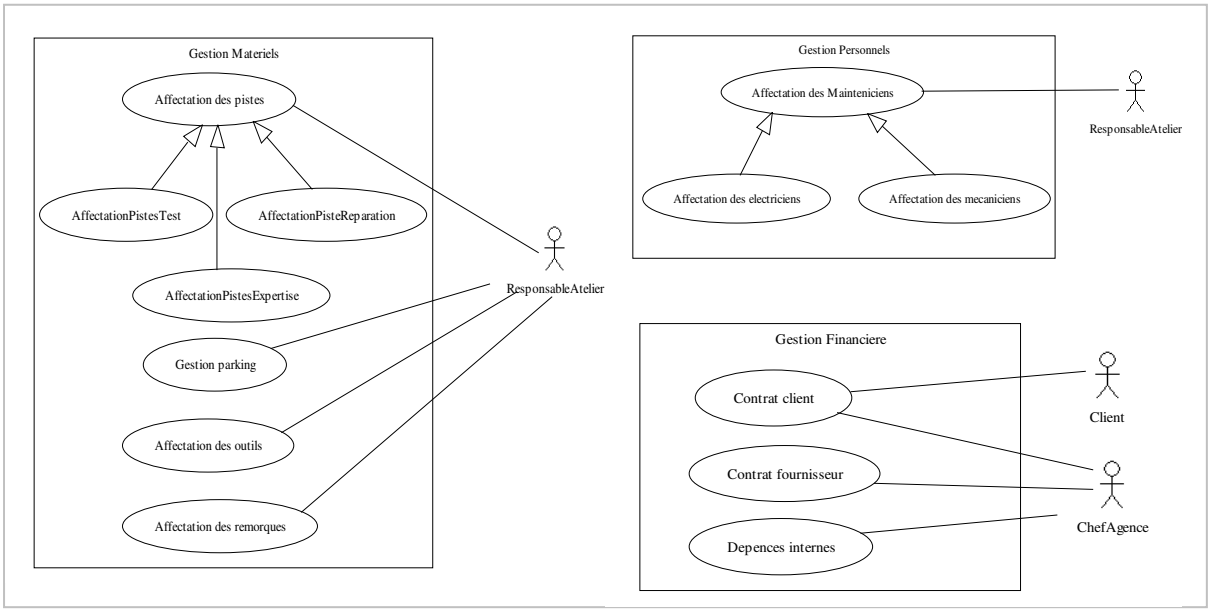

Figure 4. *Cas d'utilisation gestion matériel, gestion personnel et gestion financière*

2.1.5. Identification des classes principales du système

Suite à l'étape d'enregistrement de la voiture, le système crée automatiquement un objet "Voiture". Cet objet sera l'unique représentant de la voiture réelle à l'intérieur du système, il gère toute la chaîne de sa réparation depuis l'emplacement où elle est tombée en panne jusqu'à sa sortie de l'agence. Il réalise une double communication avec l'environnement : *interne* avec le système et *externe* avec les acteurs. L'objet "Voiture" est détruit par le système une fois la voiture sortie de l'agence. Avant sa destruction, le système doit sauvegarder l'historique de cette voiture avec la liste des pannes détectées et la liste des réparations apportées.

Les objets Expertise et Réparation seront des objets associés à l'objet Voiture. Ils vont servir à la représentation et la sauvegarde des détails des opérations de maintenance effectuées sur la voiture.

Une autre classe importante dans le système est la classe AgenceReparation, considérée comme le noyau gérant le système; elle crée la coordination entre les différents modules de l'application. Ces modules sont : (i) le module de gestion du matériel, (ii) le module de gestion du personnel, (iii) le module de gestion financière.

2.1.6. Construction du modèle de classes commun (partagé)

Le modèle de base donne l'architecture globale générique du système. Il résume d'une manière graphique la sous-section précédente sur l'identification des classes principales du système. Ce modèle sera par la suite –dans la phase décentralisée– communiqué aux équipes qui vont développer les modèles partiels par point de vue. Il est vu comme une "carte" qui dessine les frontières du système et qui fournit des consignes pour guider la conception par point de vue.

Le gain majeur tiré du développement d'un tel modèle est de faciliter la phase de fusion des modèles par point de vue pour produire le modèle VUML. En effet, l'établissement de ce modèle partagé limite les conflits de modélisation potentiels.

La figure ci-dessus représente le modèle de classes commun développé pour le système étudié.

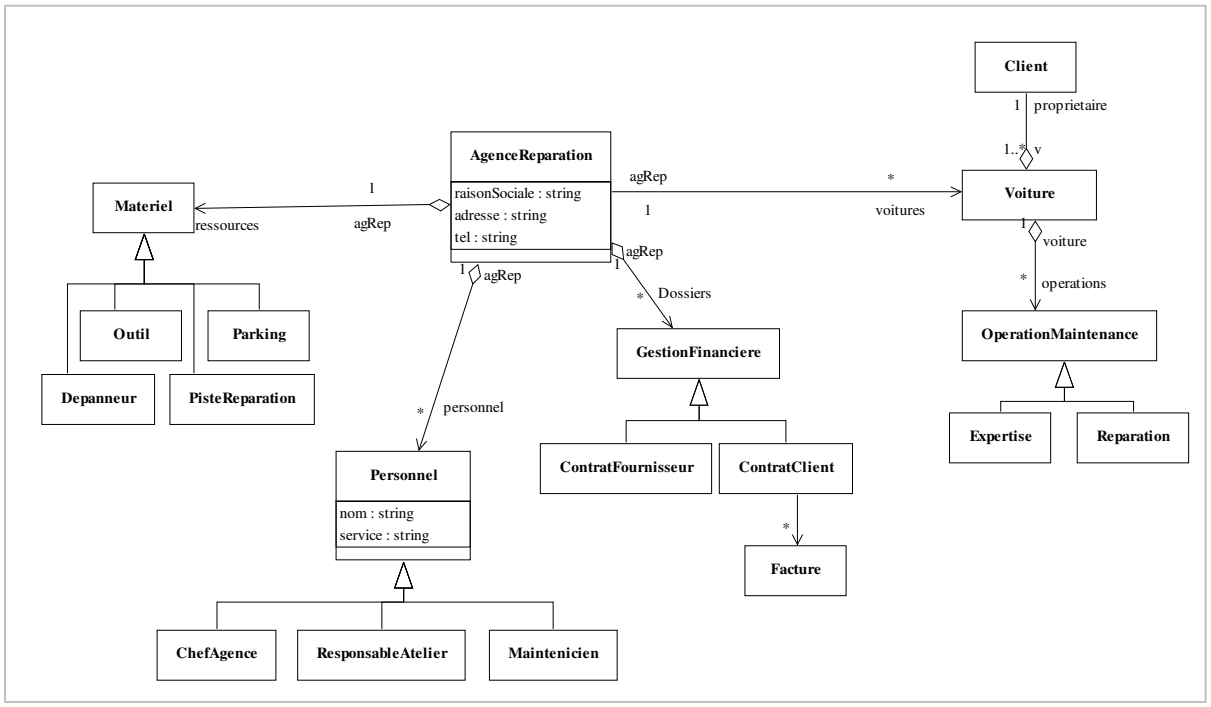

Figure 5. *Architecture du système « Gestion d'une agence de réparation de voitures »*

2.2. Analyse Globale Dynamique

Après l'achèvement de la phase d'analyse globale statique, on démarre la phase d'analyse globale dynamique. Durant cette phase de spécification dynamique, nous décrivons le comportement général du système en suivant la démarche suivante: d'abord, nous déterminons les classes réactives du système. Puis pour chaque classe réactive nous identifions les états potentiels qui couvrent son cycle de vie d'une manière générale sans entrer dans les détails. Enfin, nous déduisons ce que nous appelons une machine à états de base pour chaque classe réactive.

2.2.1. Identification des classes réactives

En partant des résultats de l'analyse globale statique, nous arrivons à déduire les classes réactives ainsi que celles potentiellement multivues.

Les classes que nous avons estimé être multivues sont : Voiture, Expertise, Panne, Réparation et Contrat. D'après l'analyse des cas d'utilisation et les besoins de chaque utilisateur, seule la classe Voiture possède un comportement réactif. Les autres classes sont considérées comme des classes statiques de données.

La liste des classes multivues identifiées dans cette étape, ainsi que le classement attribué à chacune d'elles en classes statiques (classes de données) ou en classes réactives, ne sont pas exhaustifs.

C'est tout à fait normal que d'autres classes multivues puissent apparaître dans la deuxième phase de la démarche lors de l'analyse détaillée par point de vue. Dans ce cas, nous procédons à un traitement d'exception comme étudié dans la sous-section 3.2.3.

2.2.2. Identification des états potentiels pour chaque classe réactive

Les états potentiels pour un objet sont choisis de manière à couvrir la totalité de son cycle de vie mais à un niveau d'abstraction élevé (c-à-d sans rentrer dans les détails). Pour notre exemple, le cycle de vie d'une instance de la classe Voiture est le suivant :

Une fois une voiture en panne, son propriétaire établit un contact préalable avec l'agence, et communique les informations nécessaires à l'enregistrement de sa voiture. La chaîne de réparation nominale suivie dans l'agence est la suivante: tout d'abord, amener la voiture vers le garage, ensuite procéder à l'expertise de la voiture sur les deux niveaux mécanique et électrique pour détecter ses pannes. Suite aux résultats de l'expertise, un contrat de réparation doit s'établir. Quand le contrat est établi, les mainteniciens procèdent à la réparation des pannes. La dernière étape est celle du test pour vérifier le bon fonctionnement de la voiture.

2.2.3. Construction de la "machine à états de base" des classes réactives

La machine-base définit le cycle de vie global d'un objet multivue, qui est partagée par toutes les vues. Elle est constituée d'états qui sont pertinents vis-à-vis de tous les acteurs, et de transitions entre ces états. De ce fait, la machine base est en général très abstraite. Chaque machine-vue développée dans la phase B, partage la même structure que la machine-base et la spécialise en rajoutant du comportement qui gère les requêtes provenant de l'acteur associé à la vue. L'ajout de comportement se fait notamment en raffinant les états (abstraits) de la machine-base en sous-machines (états composés UML).

Le cycle de vie de la voiture est constitué des états suivants :

- EnPanne : état initial de la voiture,
- EnEntreeGarage : état qui représente la démarche pour ramener la voiture de l'endroit où elle est tombée en panne vers le garage,
- EnProcedureExpertise : état représentant l'opération d'expertise sur la voiture pour détecter les pannes,
- EnProcedureReparation : après la détection des pannes, la voiture commence la procédure de réparation qui se compose de deux parties essentielles : (i) la négociation du contrat de réparation en se basant sur les résultats de l'expertise, et (ii) la réparation technique représentant l'action de réparation des pannes détectées dans la phase d'expertise,
- EnTest : étape finale avant la sortie du garage pour tester le bon fonctionnement de la voiture,
- Sortie : état représentant la fin du cycle de vie de la voiture dans le garage.

La figure 6 présente la machine à états de base pour l'objet Voiture. En fait, cette machine résume le cycle de vie de cet objet dans le système et donne l'ordonnancement temporel logique de ses

états. Cette machine est ensuite ajoutée dans le glossaire dans l'objectif qu'elle soit partagée et réutilisée -dans la phase B de la démarche- par les développeurs par point de vue.

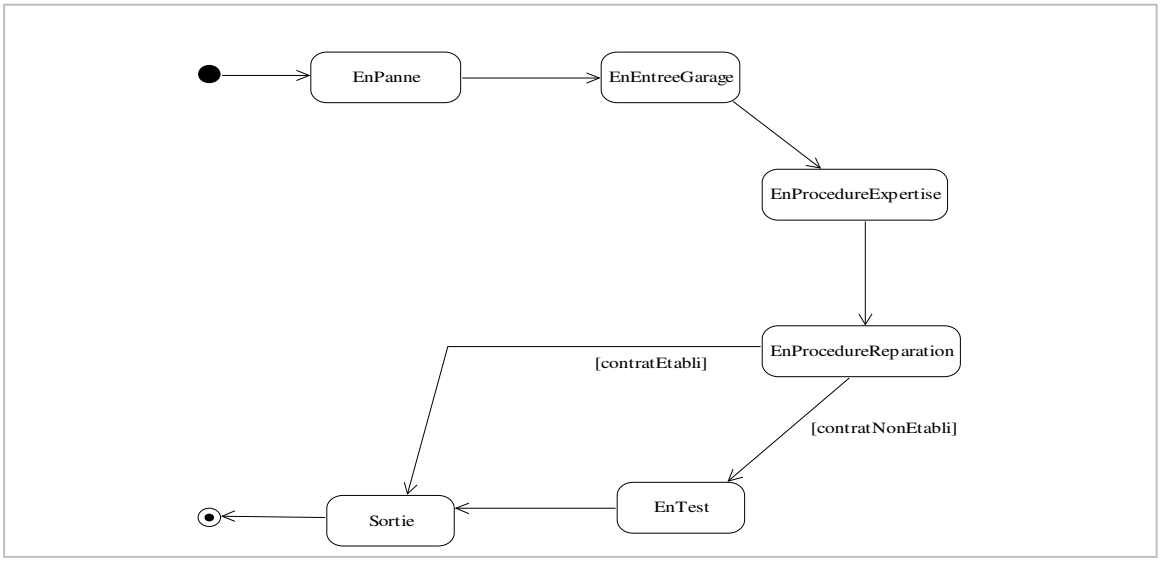

Figure 6. *Machine-base de l'objet multivue « voiture »*

3. Phase B: Analyse/Conception par Point de vue

3.1. Analyse/Conception Statique par Point de vue

C'est une phase de conception décentralisée, au cours de laquelle plusieurs équipes de concepteurs peuvent travailler séparément pour réaliser des modèles de conception par points de vue.

Pour chaque acteur, nous reprenons le modèle de base réalisé dans la phase A et nous développons un modèle partiel détaillé adapté aux besoins de cet acteur.

Il n'est pas obligatoire de développer toutes les classes du modèle de base, et aussi il n'est pas obligatoire de se restreindre aux éléments de ce modèle de base. Le concepteur peut :

- Omettre les éléments dont l'acteur n'a pas besoin,
- Supprimer les éléments auxquels l'acteur n'a pas accès,
- Rajouter de nouveaux éléments.

Le résultat de cette phase sera un ensemble de modèles UML, chacun représentant les spécificités d'un acteur, à un niveau de granularité fine.

Nous allons nous restreindre dans l'illustration des étapes de notre démarche au point de vue Client. Nous donnons à la fin de la section le résultat des autres points de vue en suivant les mêmes étapes.

3.1.1. Reprise des cas d'utilisation où l'acteur associé au point de vue traité est acteur principal

Nous résumons les besoins de l'acteur Client -propriétaire de la voiture- aux points suivants :

- consulter les informations concernant sa voiture,
- suivre son évolution dans la chaîne de réparation,
- consulter les rapports d'expertise et de réparation,
- effectuer le test de vérification avant la livraison de sa voiture,
- négocier les contrats avec le responsable financier de l'agence.

Les cas d'utilisation dont le client est acteur principal sont "Enregistrement", "Réparation", "Test" et "NégociationContrat". (Voir les figures 3 et 4).

3.1.2. Établissement des diagrammes de séquence réalisant les cas d'utilisation

3.1.2.1. Diagramme de séquence "Enregistrement" –Point de vue Client–

Nous résumons la procédure à suivre par un client pour que l'agence prenne en charge sa voiture par les étapes suivantes: le client (1) établit un premier contact avec l'agence de réparation, (2) communique ses informations personnelles, telles que son nom, son téléphone, etc… (3) communique les informations concernant sa voiture en panne, notamment son matricule, l'adresse où elle est en panne.

Après l'accomplissement de ces étapes la voiture sera enregistrée dans le système d'information de l'agence. Ces étapes sont illustrées dans le diagramme de séquence suivant :

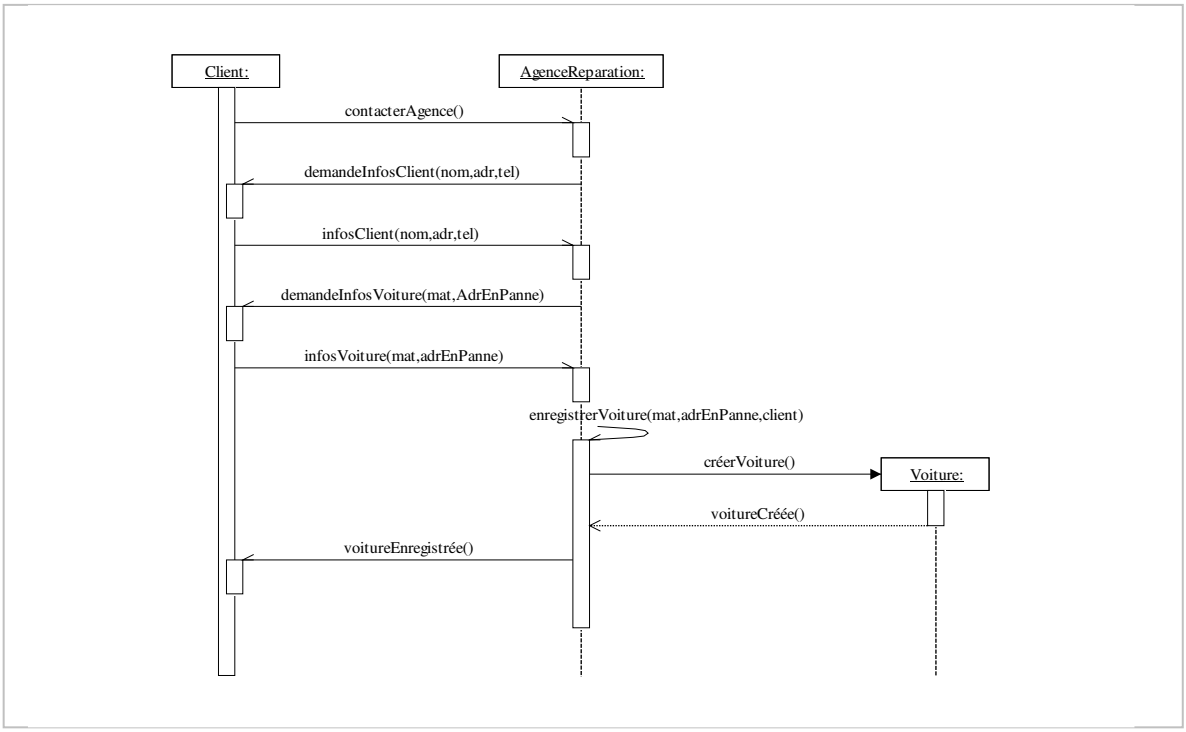

Figure 7. *Diagramme de séquence pour le cas d'utilisation "Enregistrement" –Point de vue Client–*

A partir de ce diagramme de séquence, on peut tirer un ensemble d'informations qui vont servir à l'enrichissement du diagramme de classe –Point de vue Client– :

- pour la classe Voiture, on déduit les attributs : matricule et adresseEnPanne,
- pour la classe AgenceReparation, on déduit la méthode publique contacterAgence() et les attributs adresse et tel,
- pour la classe Client on déduit les attributs : nom, adresse et tel.

3.1.2.2. Diagramme de séquence "Réparation" –Point de vue Client–

Les étapes à suivre pour établir la réparation de la voiture selon le point de vue Client sont les suivantes :

- consulter le rapport d'expertise,
- demander la réparation,
- remplir un formulaire pour décider les pannes à réparer,
- négocier le contrat.

L'ordonnancement de ces étapes est donné par le diagramme de séquence suivant :

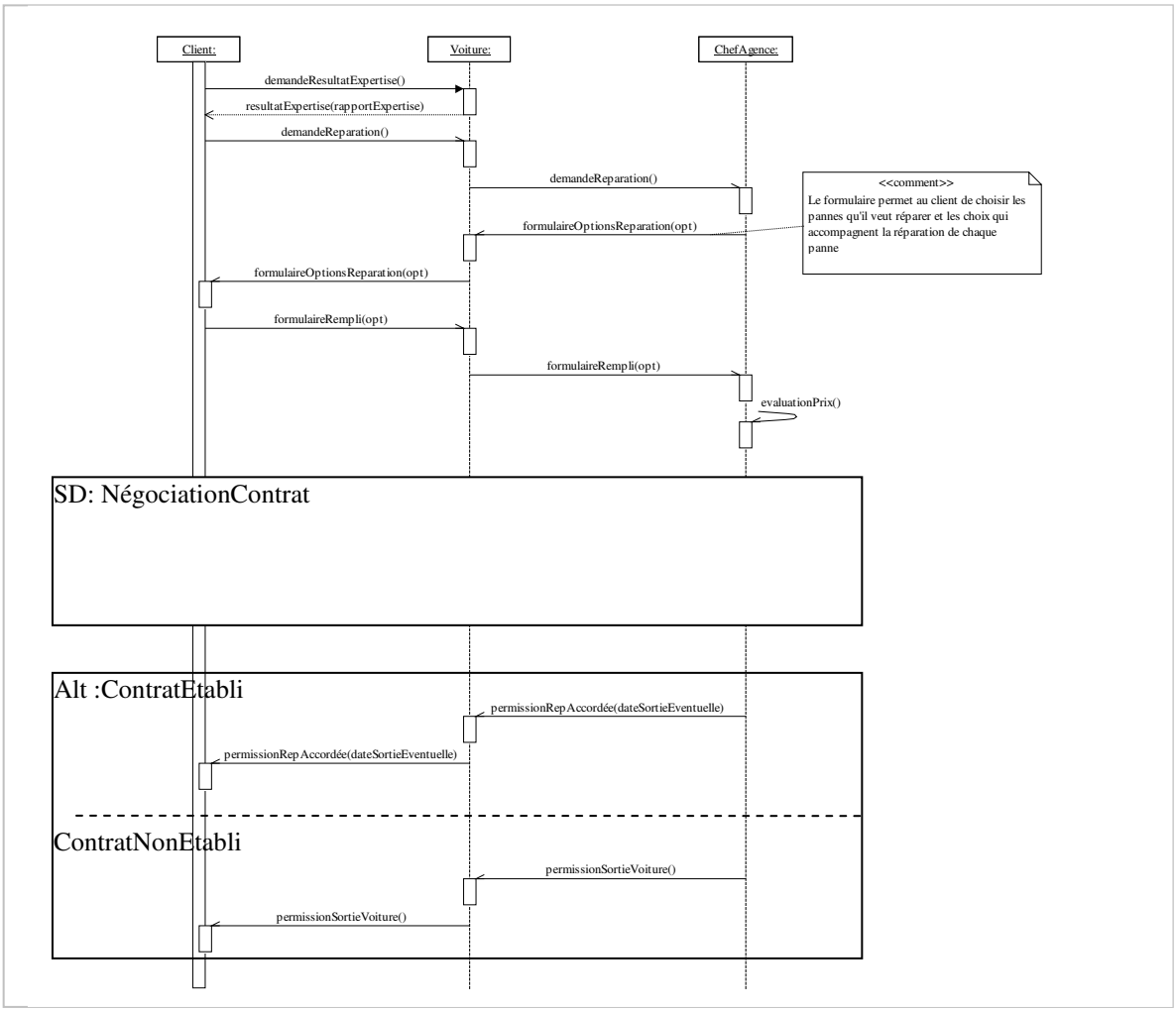

Figure 8. *Diagramme de séquence pour le cas d'utilisation "Réparation" –Point de vue Client–*

A partir de ce diagramme de séquence, on peut tirer des informations qui vont enrichir le diagramme de classe –Point de vue Client– :

Pour la classe Voiture : la méthode resultatExpertise() et les attributs : formulaireReparation, dateEventuelleSortie et le booléen contratEtabli.

3.1.2.3. Diagramme de séquence "Test" –Point de vue Client–

Une fois la procédure de réparation achevée, le client a le droit d'effectuer un test sur sa voiture pour vérifier son bon fonctionnement. Cette tâche de test est réalisée en suivant le scénario suivant :

- la demande du test par le client,
- l'affectation d'une piste de test pour effectuer le test,
- le test.

Ce scénario est traduit par le diagramme de séquence suivant :

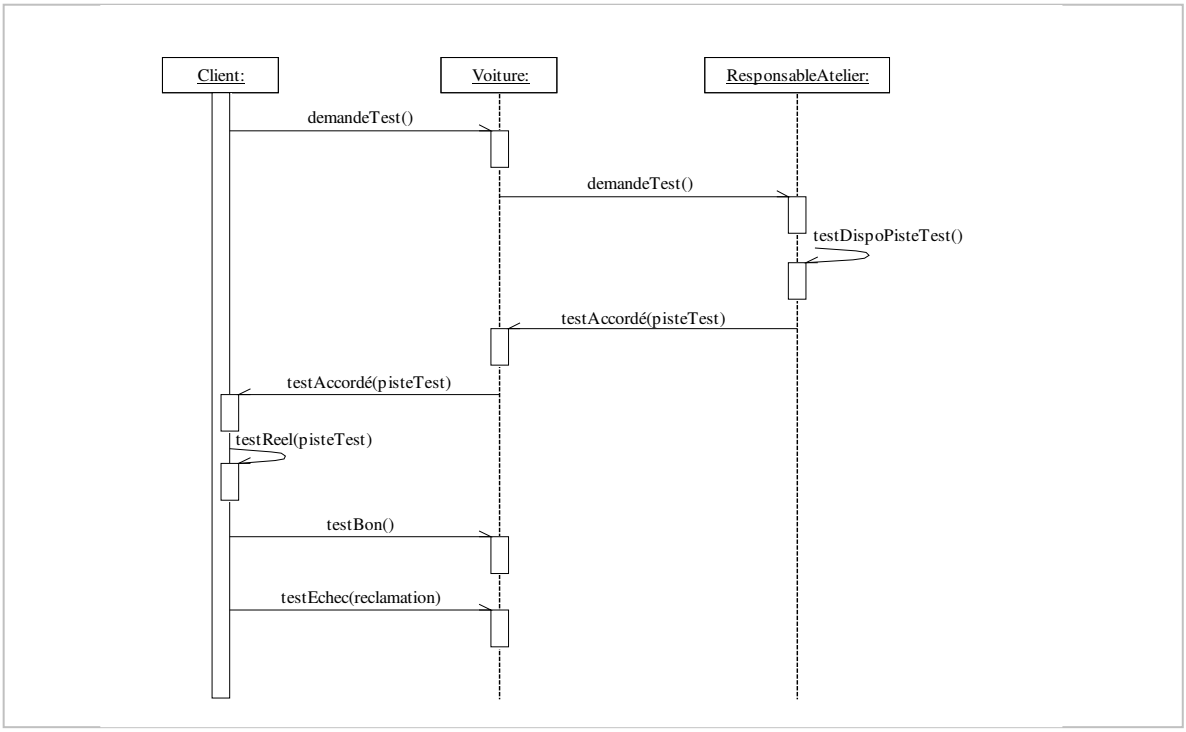

Figure 9. *Diagramme de séquence "Test" –Point de vue Client–*

3.1.2.4. Diagramme de séquence " Négociation Contrat " –Point de vue Client–

Le scénario de la négociation du contrat de réparation suit les étapes suivantes :

- le chef d'agence -en se basant sur les options de réparation choisies par le client- évalue les frais,
- le client a le droit de répondre et fait sa propre proposition,
- le chef d'agence donne sa réponse finale,
- le client donne sa décision.

Ce scénario est traduit par le diagramme de séquence dans la figure 10.

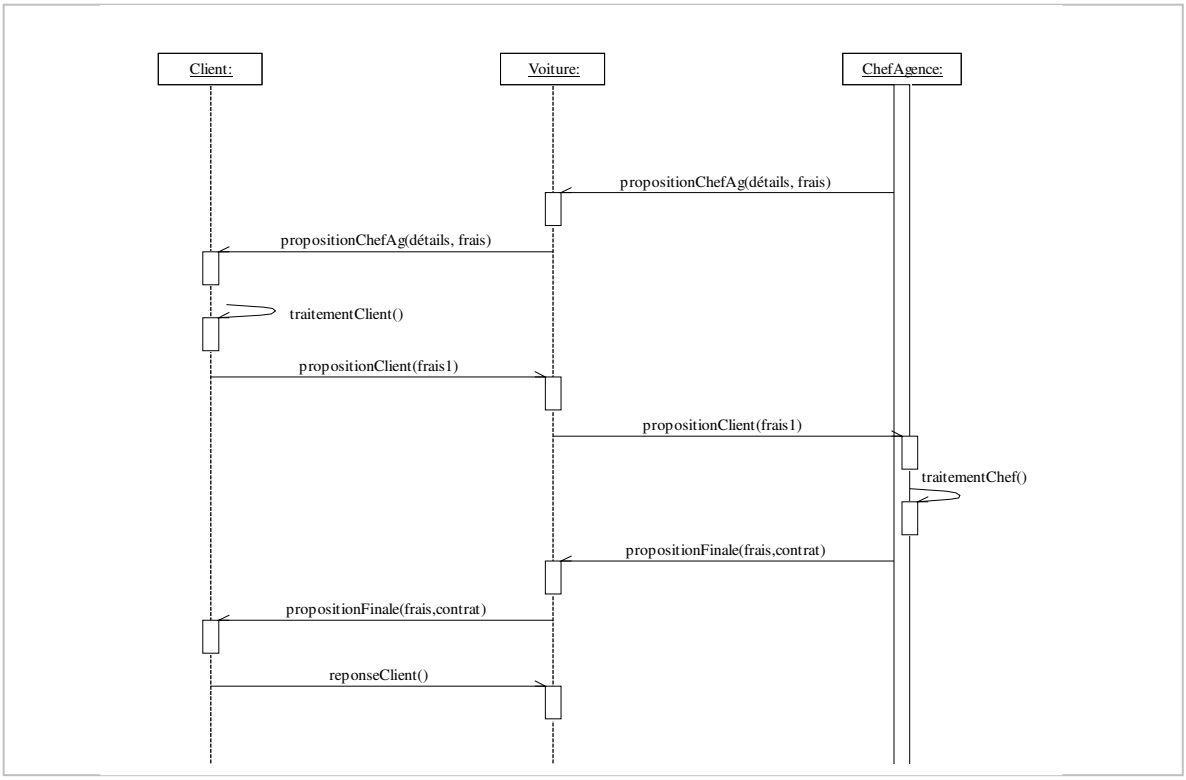

Figure 10. *Diagramme de séquence "NégociationContrat" –Point de vue Client–*

3.1.3. Développement du diagramme de classe détaillé pour le point de vue courant

Nous obtenons le diagramme de classe détaillé associé à un point de vue donné en rassemblant l'ensemble des informations déduites des diagrammes de séquence.

Pour le point de vue Client, son diagramme de classe détaillé est donné par la figure suivante :

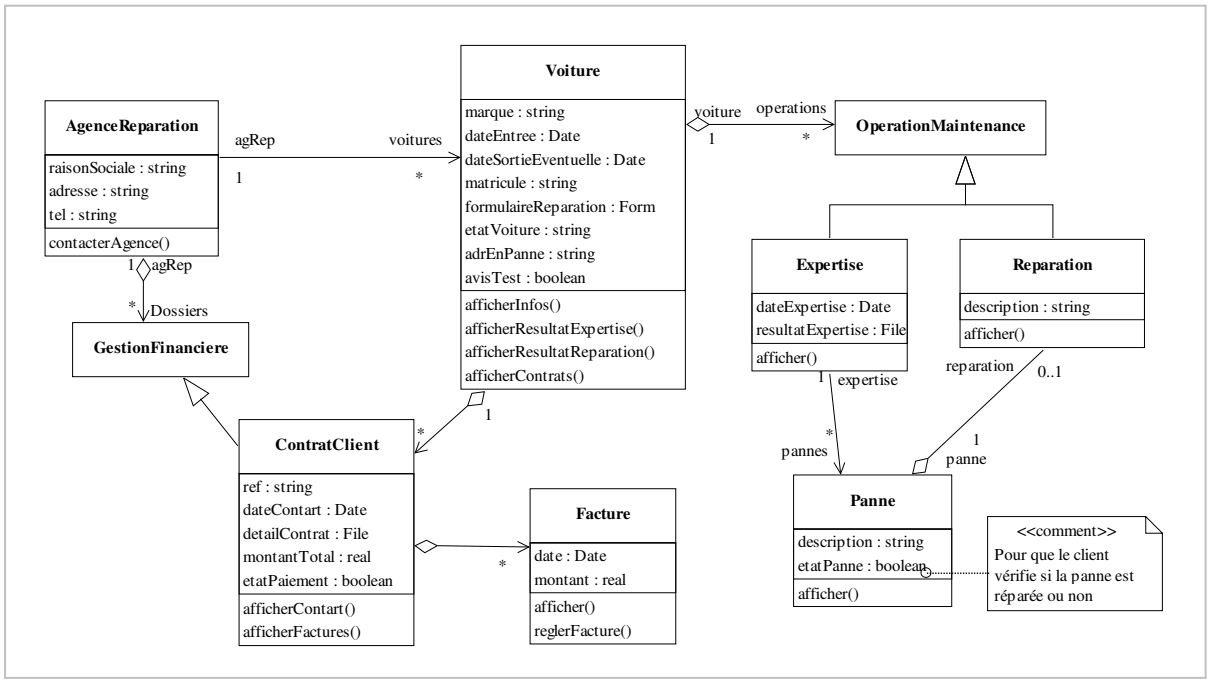

Figure 11. *Diagramme de classe UML –Point de vue Client–*

Nous signalons qu'il y a des fragments du diagramme de classe commun partagé qui ont été ignorés, tels que ceux correspondant à la gestion du personnel et à la gestion du matériel. C'est tout à fait normal du fait que le point de vue de l'acteur Client se focalise plutôt sur la partie de gestion de sa voiture, notamment les opérations de maintenances réalisées, l'établissement du contrat de réparation, etc...

Par la même démarche que nous avons suivie pour développer le diagramme de classes -point de vue Client-, nous élaborons les diagrammes de classes pour les autres points de vue (cf. figures 12, 13 et 14).

Prenons par exemple le diagramme de classes associé au point de vue de l'acteur Chef d'agence (figure 12), au contraire des autres points de vue, nous remarquons que les fragments liés à la supervision de l'agence et la gestion financière sont beaucoup plus développés. En effet ces deux fragments expriment des fonctionnalités représentant des préoccupations potentielles pour le chef de l'agence.

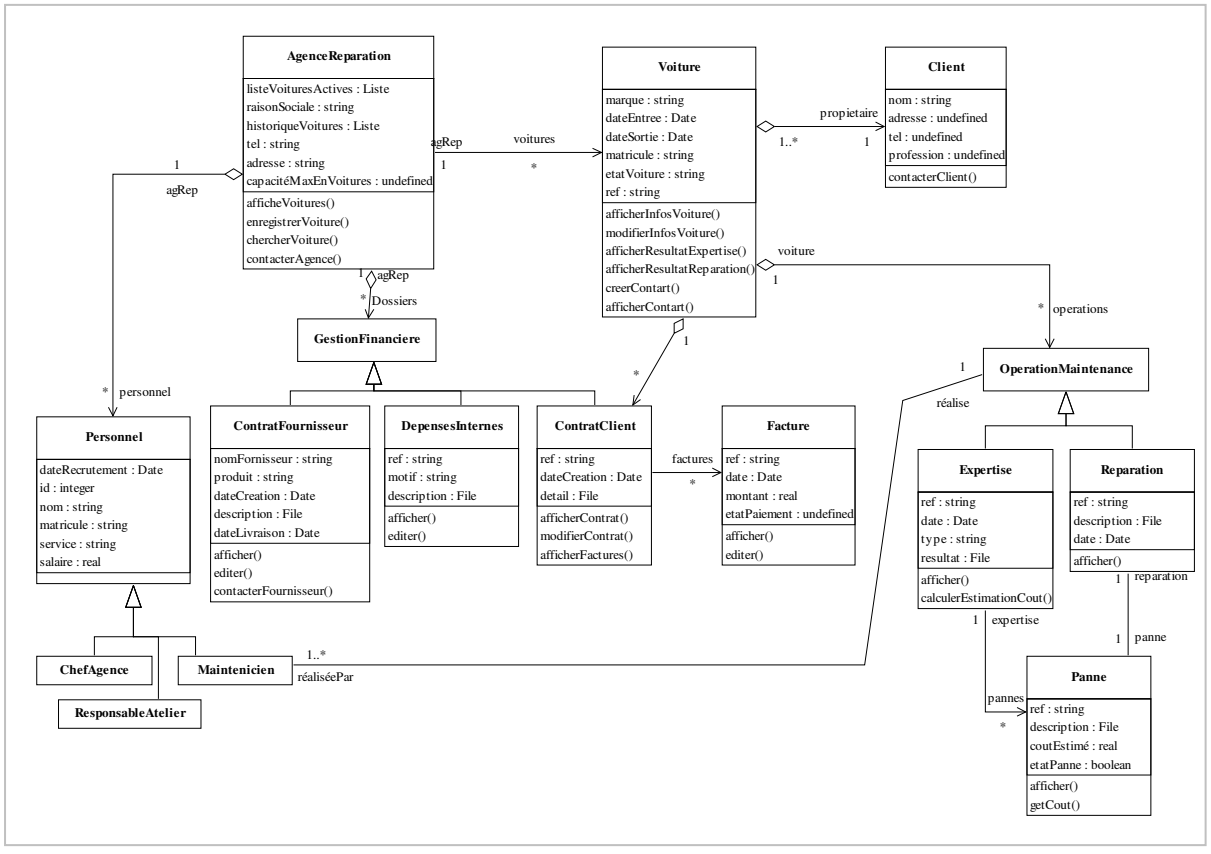

Figure 12. *Diagramme de classe UML – point de vue Chef de l'agence –*

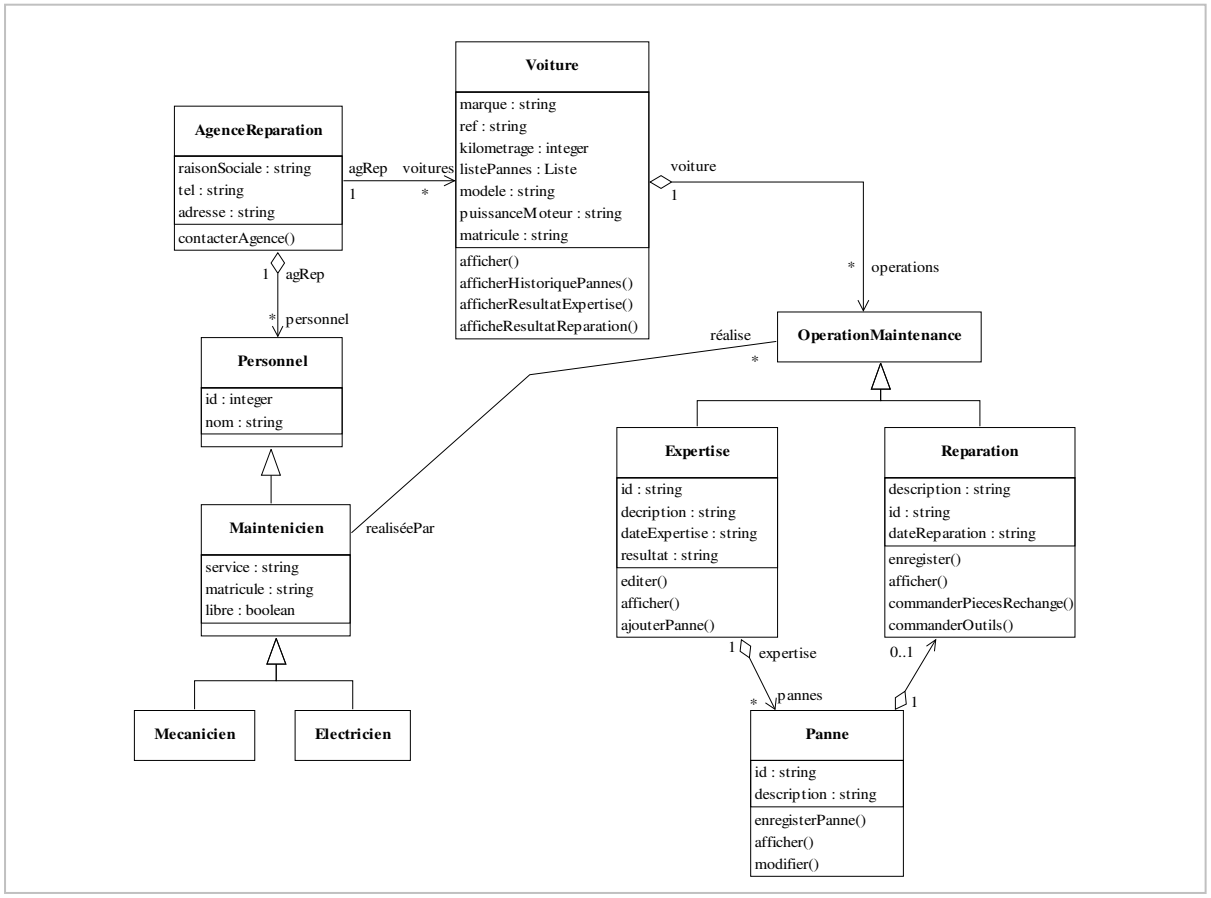

Figure 13. *Diagramme de classe UML - point de vue Maintenicien-*

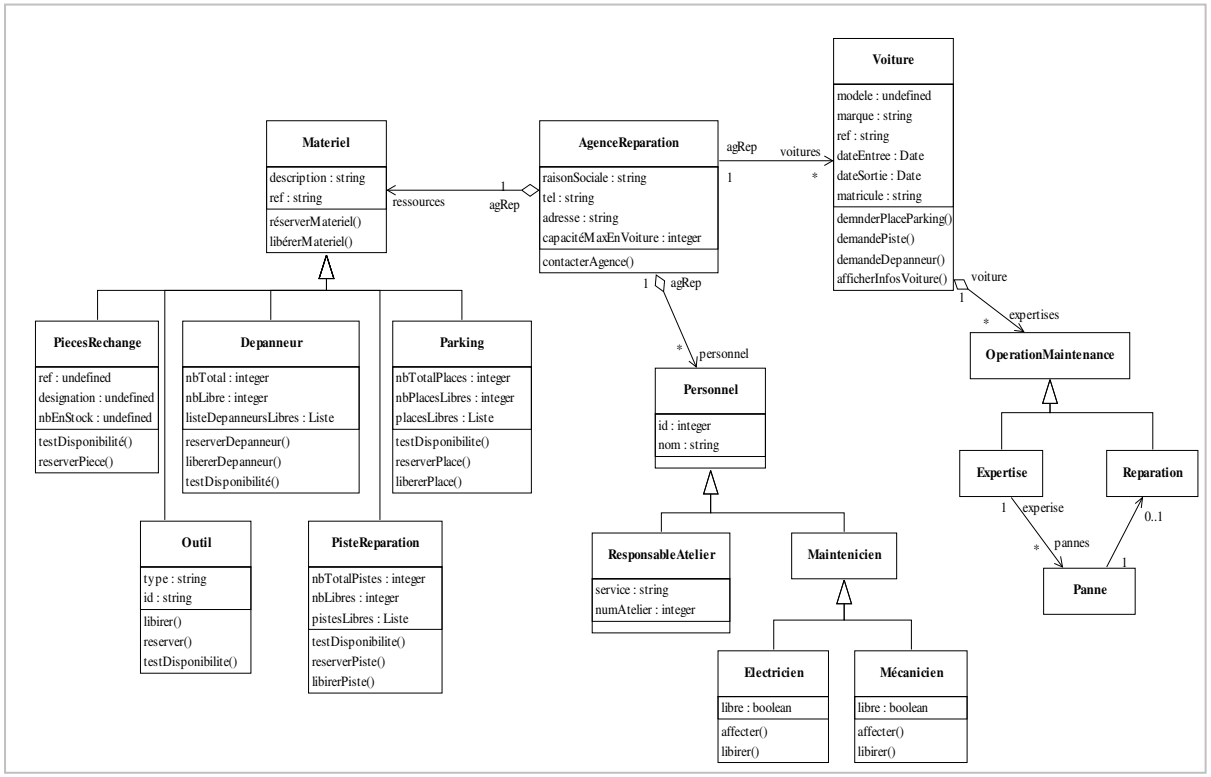

Figure 14. *Diagramme de classe UML -point de vue Responsable Atelier-*

3.2. Analyse/Conception Dynamique par Point de vue

Après l'établissement du modèle statique selon un point de vue particulier, on démarre la phase de spécification dynamique.

La démarche que nous adoptons pour construire les machines à états par point de vue est la suivante :

- reprendre la machine-base issue de l'analyse globale dynamique,
- pour chaque état de la machine base :
  - étudier sa signification pour l'acteur en question,
  - déterminer les conditions exigées pour l'acquérir,
  - déterminer les actions à accomplir,
  - déterminer les conditions pour le perdre,
- si cet état englobe une ou plusieurs activités de l'acteur considéré, procéder à son raffinement, sinon passer à l'état suivant.
- créer de la machine vue.

Pour mieux illustrer notre démarche, nous proposons de restreindre l'étude dynamique par point de vue à une partie de la SM base. L'état que nous allons détailler est l'état "EnProcedureReparation" durant lequel la voiture subit une série d'actions afin qu'elle retrouve son état normal de fonctionnement.

3.2.1. Étude de la signification des états de la machine base pour chaque point de vue

Rappel :

Un **état multivue** représente l'état d'un objet multivue, c'est un état à plusieurs interprétations selon l'acteur interagissant avec le système, il est défini par les sous-états suivants :

- Un **état-base** : état abstrait représente une situation durant la vie d'un objet multivue,
- Un ensemble **d'états-vue** : des états résultants du raffinement de l'état base en prenant en compte les points de vues des acteurs du système.

La figure 15 illustre l'état multivue "EnProcedureReparation" et comment il est interprété par chaque point de vue.

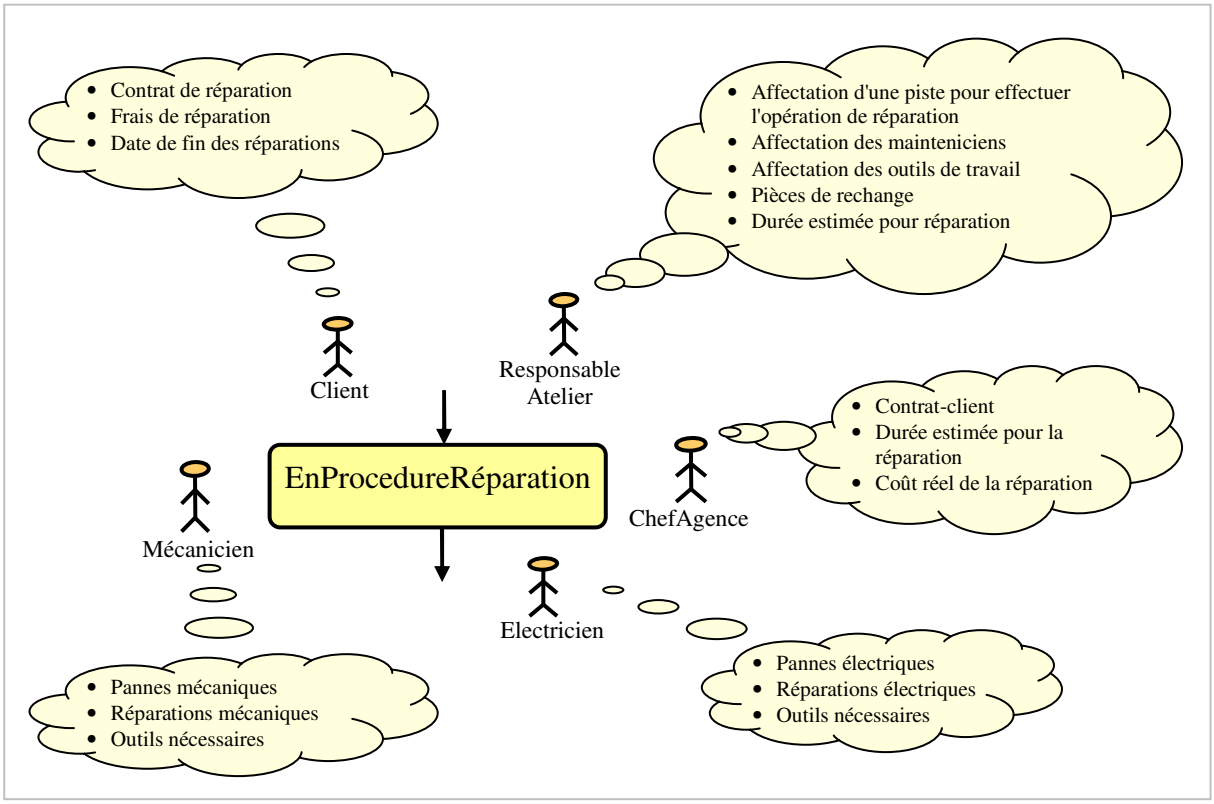

Figure 15. *Illustration de l'état multivue "EnProcedureReparation"*

En effet, l'état "EnProcedureReparation" est un état multivue. Chaque acteur interprète cet état selon son point de vue :

- Le maintenicien (électricien/mécanicien) s'intéresse aux points suivants :
  - le traitement des rapports des pannes détectées dans la phase d'expertise,
  - les réparations qu'il doit apporter,
  - les outils nécessaires pour effectuer la réparation,
  - les pièces de rechange.

- Le responsable atelier voit la réparation du coté logistique, il s'intéresse aux points suivants :
  - l'affectation d'une piste pour effectuer l'opération de réparation,
  - l'affectation des mainteniciens,
  - l'affectation des outils de travail,
  - les pièces de rechange,
  - la durée estimée pour la réparation.

- Le client s'intéresse aux points suivants :
  - le contrat de réparation,
  - les frais de réparation,
  - la date de fin des réparations.

- Le chef d'agence s'intéresse aux points suivants :
  - le contrat de réparation,
  - le coût réel des réparation et les bénéfices,
  - la durée estimée pour réparation.

3.2.2. Raffinement des états de la machine base pour chaque point de vue

Dans cette sous-section nous allons procéder au raffinement de l'état "EnProcedureReparation" selon chaque point de vue.

Comme on vient de voir dans la sous-section précédente, cet état est interprété d'une manière différente d'un point de vue à l'autre. Mais en général, on peut le découper en deux phases : la première est consacrée à l'établissement du contrat de réparation, la deuxième est consacrée à la réparation technique.

3.2.2.1. Développement de la machine-vue de l'objet Voiture pour le point de vue Client

En nous basant sur les diagrammes de séquence développés pour le point de vue Client, notamment ceux qui traitent la procédure de réparation et de négociation du contrat, nous aboutissons à la partie de la machine à états concernant l'état "EnProcedureReparation" (figure 16).

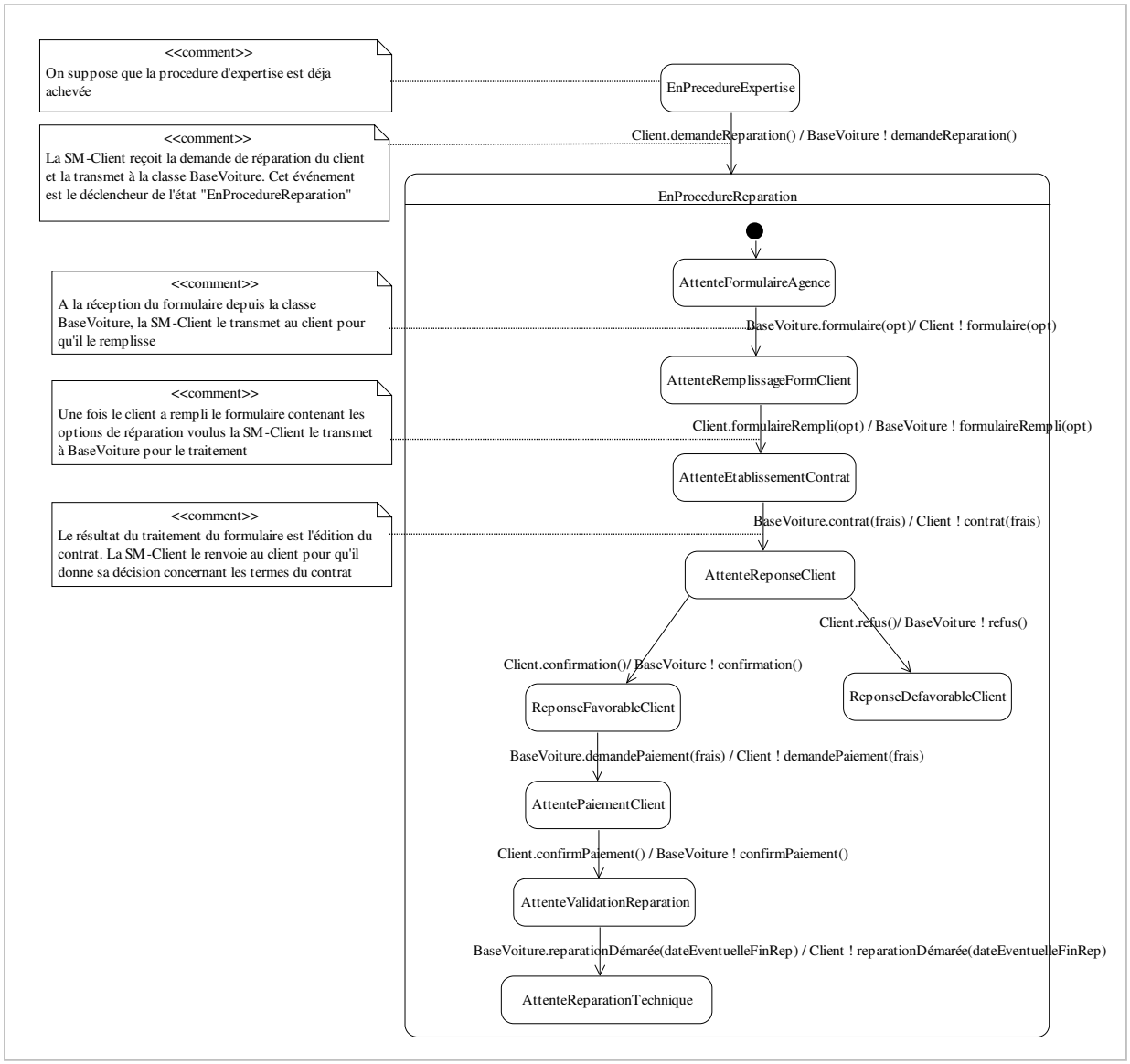

Figure 16. *Développement de l'état "EnProcedureRéparation" pour le point de vue Client*

La transition dans la machine-vue-voiture-client (SM-V-Client) depuis l'état "EnProcedureExpretise" vers l'état "EnProcedureReparation" est déclenchée par le signal *demandeReparation()* provenant de l'acteur Client. Or dans cette étape de la démarche (phase B), l'analyse/conception par point de vue est faite d'une manière décentralisée. De ce fait, la SM-V-Client retransmet ce signal vers la machine-base sans se soucier vers quel ou quels objets ce signal sera destiné.

Le même principe est appliqué dans le cas où un acteur attend un signal provenant d'une autre entité. Si c'est le cas, le développeur de ce point de vue suppose que le signal provient de la machine-base. C'est l'exemple du signal *formulaire(opt)* qui déclenche la transition entre l'état "AttenteFormulaireAgence" et l'état "AttenteRemplissageFormClient" sur la figure 16.

Le choix de centraliser la communication entre les différentes vues dans la machine-base est justifié par les points suivants :

- faciliter le développement des machines-vue : une vue ne communique qu'avec la classe base et/ou son acteur associé,
- faciliter la coordination après le développement des machines-vues,
- faciliter la maintenance et les éventuelles modifications des machines-vues en diminuant au maximum leurs dépendances.

3.2.2.2. Développement de la machine-vue de l'objet Voiture pour le point de vue ChefAgence

Pour le point de vue ChefAgence, le raffinement de l'état "EnProcedureReparation" est représenté dans la figure 17.

Le même principe sur la centralisation des communications dans la machine-base est appliqué. La SM-V-ChefAg quand elle reçoit le signal *demandeReparation()* provenant de la machine-base, le renvoie vers l'acteur ChefAgence. Suit à cela, le chef d'agence prépare un formulaire contenant la liste des pannes à réparer, ainsi que les choix qui accompagnent la réparation de chaque panne. Une fois le formulaire prêt, le chef d'agence l'envoie à la SM-V-ChefAg (c'est le signal ChefAg.formulaire(opt) déclanchant la transition vers l'état "AttenteRetourFormRempli").

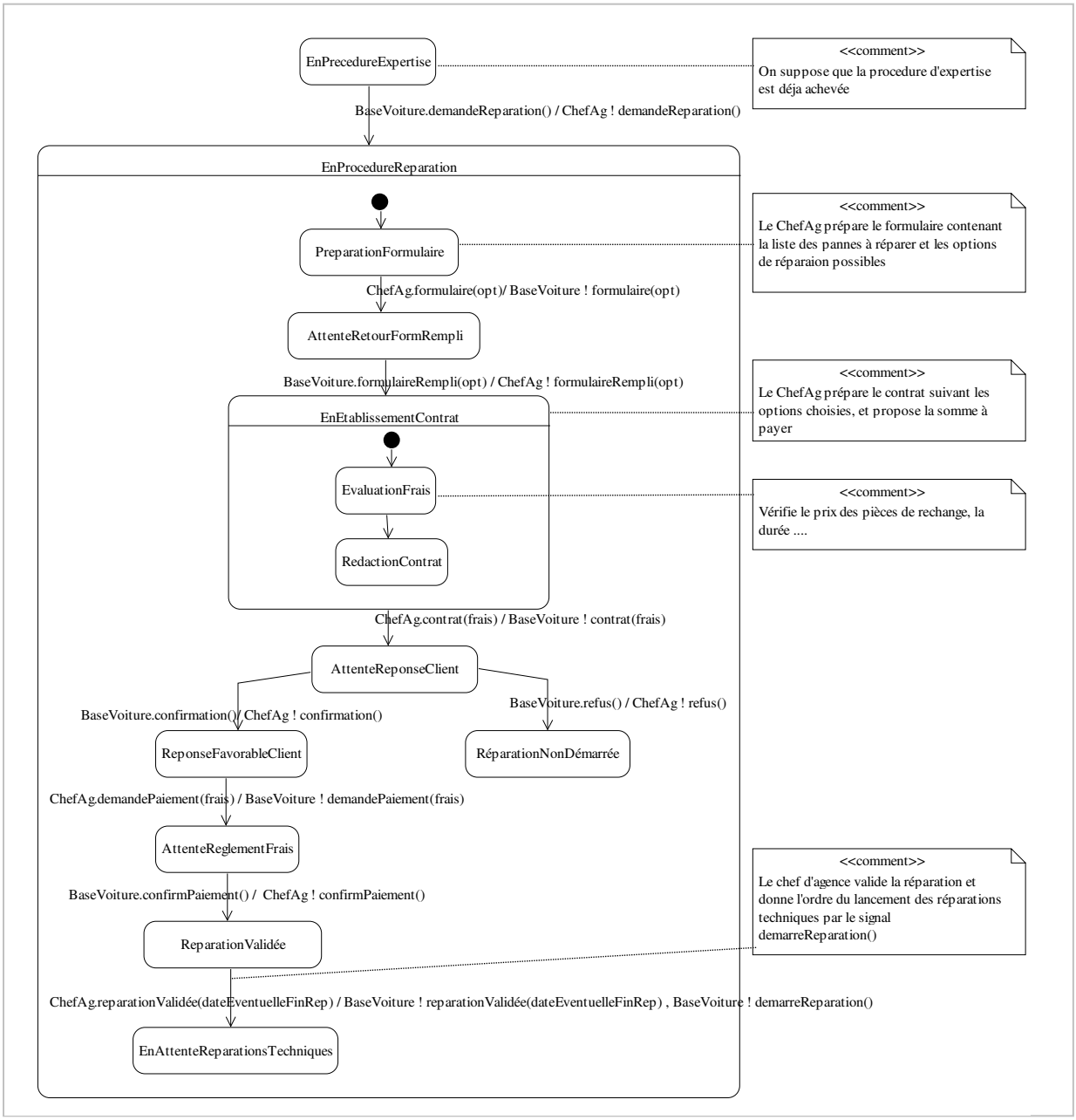

Figure 17. *Développement de l'état "EnProcedureRéparation" pour le point de vue ChefAgence*

3.2.3. Traitement de l'exception "une classe réactive est identifiée lors du développement par point de vue"

En développant le point de vue "Responsable Atelier", nous avons constaté que la classe "Materiel" est une classe réactive. Son rôle est d'effectuer la réservation du matériel nécessaire pour la réalisation des opérations de maintenance. Pour arriver à ses fins, elle communique avec le responsable atelier à travers un échange de signaux.

La classe "Matériel" n'a pas été identifiée comme multivue lors de l'analyse globale. Et donc pour gérer cette exception, nous proposons de procéder comme suit :

- Vérifier l'existence de la machine base de la classe Matériel dans le glossaire. Si elle existe, cela veut dire qu'un autre développeur l'a déjà rajoutée, dans ce cas, reprendre cette machine-base, et l'adapter en cas de besoin. Sinon développer la machine-base, et l'ajouter dans le glossaire,

- Développer la machine-vue pour le point de vue courant.

Pour notre cas, on suppose que la machine-base de la classe "Matériel" n'existe pas dans le glossaire. Donc nous procédons à la création de la machine-base (voir figure 18) et nous l'ajoutons dans le glossaire de l'application pour qu'elle soit réutilisée par un autre développeur en cas de besoin.

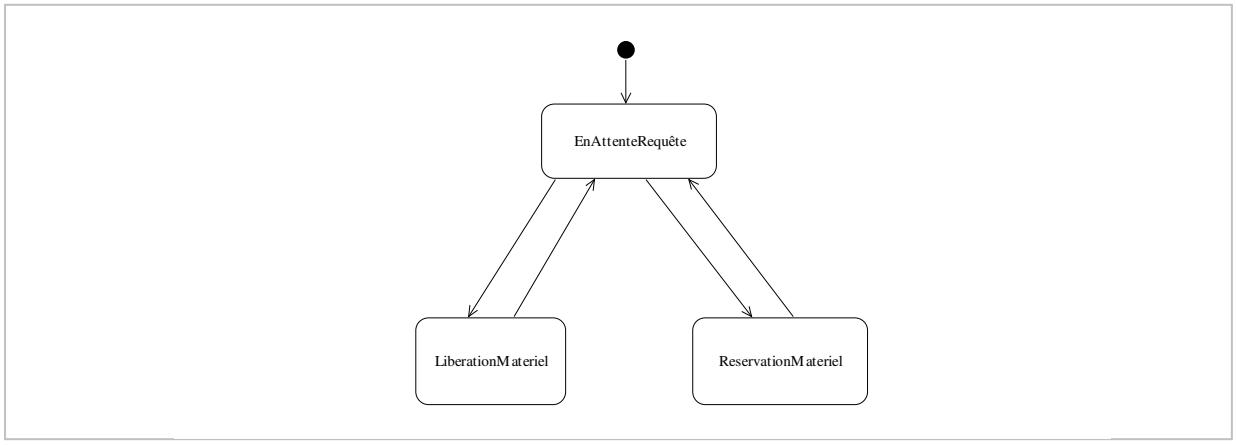

Figure 18. *SM-base pour la classe Matériel (identifiée dans le point de vue -Responsable Atelier-)*

Les états potentiels que nous avons identifiés pour un objet de la classe "Matériel" sont les suivants :

- EnAttenteRequête : état où l'objet "Matériel" est libre et prêt à répondre aux requêtes de réservation ou de libération d'un matériel particulier,
- ReservationMateriel : état pendant lequel cet objet procède à la réservation du matériel demandé,
- LiberationMateriel : état pendant lequel l'objet "Matériel" procède à la libération d'un matériel particulier.

Après la création de la machine-base de la classe Matériel et après qu'elle soit ajoutée dans le glossaire, nous continuons le développement dynamique pour le point de vue ResponsableAtelier par la création de la machine-vue correspondante (voir figure 19).

Initialement, l'objet Matériel est dans l'état "EnAttenteRequête". Suite à la demande par le ResponsableAtelier de réservation ou l'annonce de libération d'un matériel particulier, il change d'état vers l'un des deux états suivants :

- ReservationMatériel, par la réception du signal demandeMatériel(mat). Dans ce cas l'objet Matériel réalise un test de disponibilité pour le matériel demandé *mat*. Si le matériel est disponible, il renvoie au ResponsableAtelier la référence du matériel réservé. Sinon, il avertit le ResponsableAtelier de la non-disponibilité de ce matériel, puis il l'ajoute à la liste d'attente,

- LibérationMatériel, par la réception du signal demandeLibérationMatériel(mat). l'objet Matériel teste si le matériel *mat* est attendu par une requête de la liste d'attente. Si oui, il est affecté au premier qui a fait la demande de réservation de *mat*. Sinon le matériel *mat* est libéré par la fonction libérerMatériel(mat).

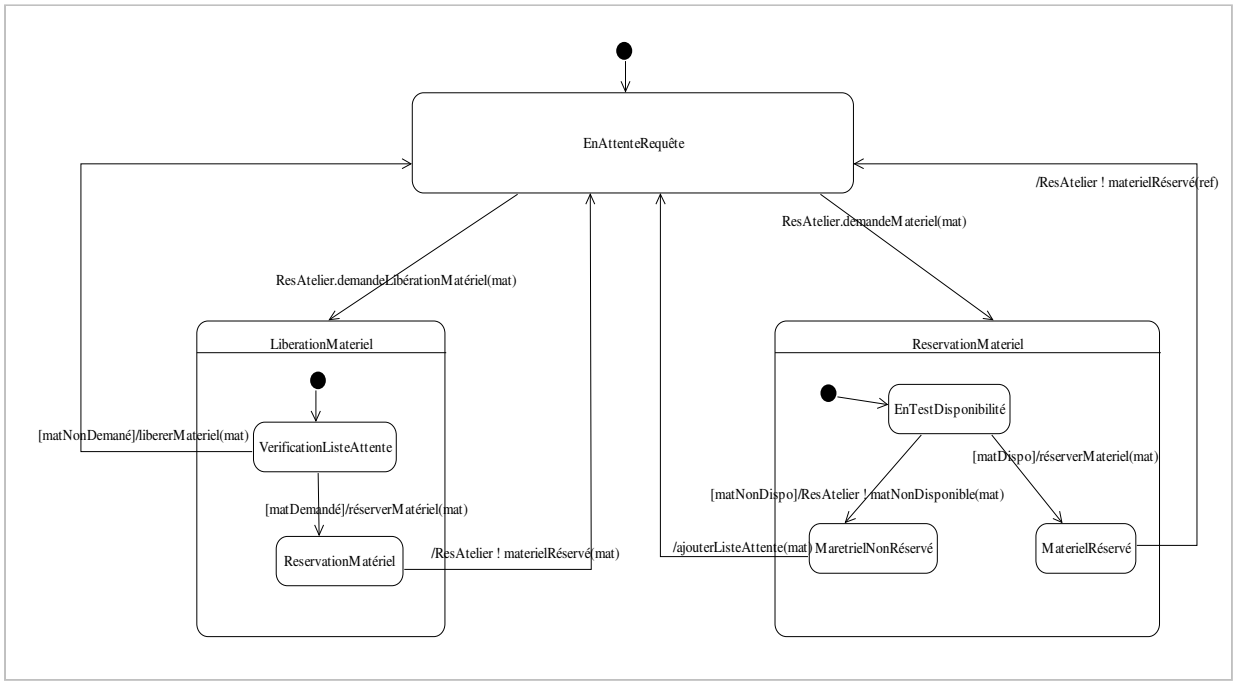

Figure 19. *SM-Vue pour la classe Matériel, point de vue -Responsable Atelier-*

4. Phase C : Fusion

L'objectif de cette étape est de créer la cohérence entre les différents fragments d'un objet multivue dans les deux niveaux statique et dynamique.

La figure 20 ci-dessous présente la structure statique/dynamique d'un objet multivue. La structure statique est représentée par les classes de données stéréotypées par <<base>> et <<view>>. La structure dynamique est représentée par les machines à états SM-base et SM-vues affectées aux classes statiques.

Un objet multivue est considéré comme un composant communiquant avec son environnement à travers un ensemble de ports :

- Un port pour chaque point de vue à travers lequel l'acteur considéré interroge sa vue.
- Un port destiné à la communication avec les autres objets de l'application.

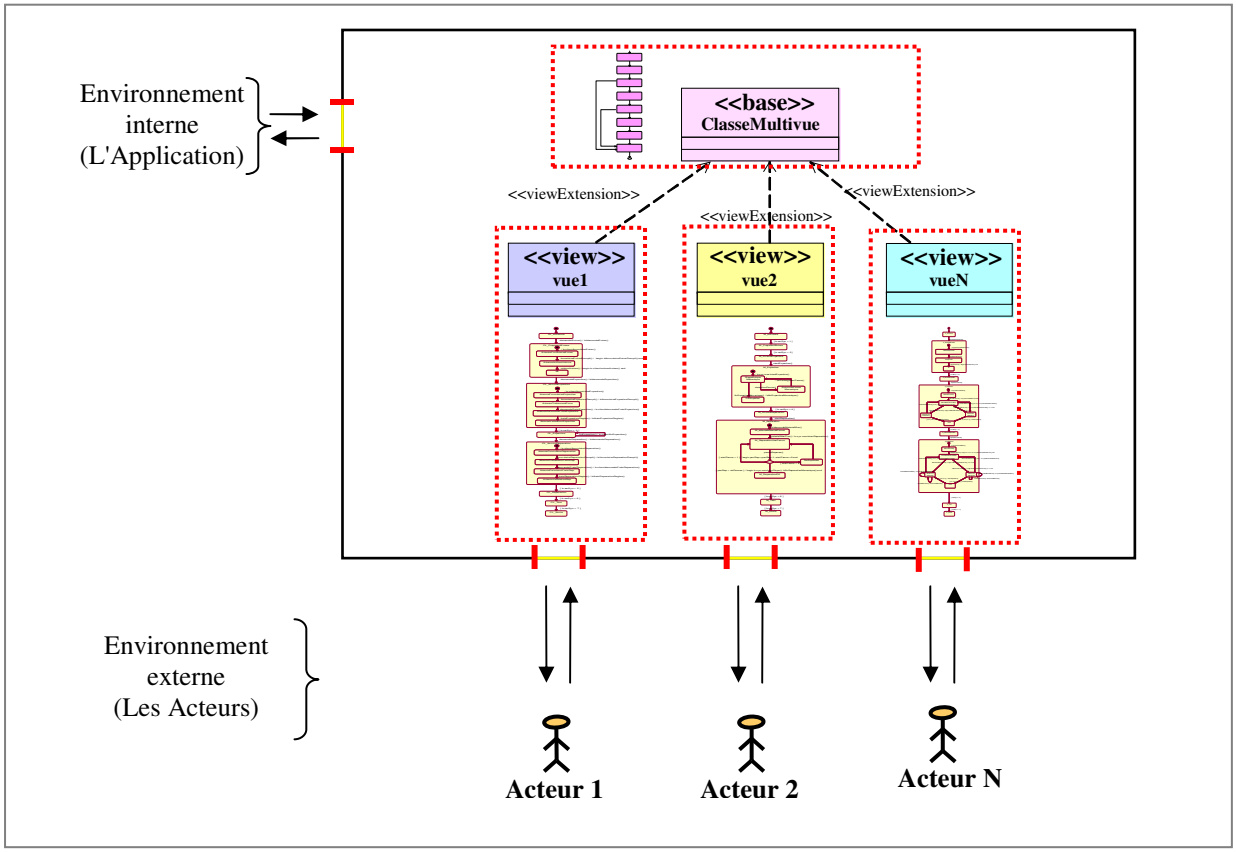

Figure 20. *Représentation statique/dynamique d'un objet multivue*

4.1. Fusion Statique

Le concept de la fusion peut être défini comme une opération de transformation de deux éléments ou plus en entrée pour donner un élément en sortie sans qu'il y ait de redondance de l'information dans l'élément en sortie. Cette définition est à la base de nombreuses techniques et approches mettant en œuvre des constats théoriques pour définir formellement l'opération de fusion. Dans cette section, nous n'allons pas traiter cette problématique, mais nous donnons juste les résultats obtenus.

4.1.1. Harmonisation des modèles statiques par point de vue

La première étape de la fusion est la comparaison des diagrammes de classes par points de vue qui peut révéler des ambiguïtés ou incohérences sur les classes (nommage, attributs, méthodes). Les diagrammes de classes élaborés précédemment peuvent être regroupés ensuite en un seul diagramme de classes VUML sans perdre les spécificités de chaque acteur.

4.1.2. Détermination de la liste finale des classes multivues

Les classes multivues sont des classes apparaissant dans au moins deux diagrammes de classes. Pour chaque classe multivue, il faut :

- identifier la base (attributs, méthodes, relations),
- ajouter les extensions correspondant aux vues.

4.1.3. Élaboration du modèle VUML global

Le résultat final de la fusion statique est représenté par le diagramme VUML de la figure 21. Nous signalons que nous pouvons -selon le nombre de classes et de vues- présenter les classes multivues soit sous forme éclatée en faisant apparaître toutes les vues, soit sous forme "iconifiée" en spécifiant le stéréotype <<*multiViewClass*>>.

Les classes multivues de cette application sont : Voiture, Expertise, Réparation, Panne, ContratClient, Personnel et AgenceReparation. Chacune de ces classes offre des vues spécifiques adaptées aux besoins des utilisateurs du système. Nous constatons l'existence de deux classes multivues potentielles, qui sont : la classe Voiture et la classe AgenceReparation :

- La classe Voiture représente une voiture réelle dans le système de gestion de l'agence, elle gère toute la chaîne de réparation depuis l'emplacement où la voiture réelle est tombée en panne jusqu'à sa sortie de l'agence. Et donc, elle connaît de nombreuses interactions et échange de données avec, d'une part les objets du système, et d'autre part les acteurs du système. La partie partagée d'une instance de Voiture – stéréotype <<base>> – regroupe les attributs/méthodes accessibles par tous les acteurs. Par exemple : son matricule, sa date d'entrée et la méthode afficherInfosVoiture(). La partie spécifique à chaque acteur – stéréotype <<view>> – représente les attributs/méthodes accessibles que par l'acteur associé. Par exemple, la date de sortie de la voiture et modifierInfosVoiture() ne sont accessible que par l'acteur ChefAgence.

- La classe AgenceReparation est considérée comme le noyau gérant le système, elle crée la coordination entre les différents modules de l'application. Ces modules sont : (i) le module de gestion du matériel, représenté par la classe Matériel; (ii) le module de gestion du personnel représenté par la classe Personnel; (iii) le module de gestion financière représenté par la classe GestionFinancière. Au niveau de cette classe des droits d'accès ont été attribués aux différents acteurs. Par exemple, le client n'a pas le droit de consulter les classes correspondantes à la gestion du personnel ou bien à la gestion du matériel. Par contre la classe Matériel n'est accessible que par les responsables des ateliers, et la classe GestionFinancière n'est accessible que par le chef d'agence.

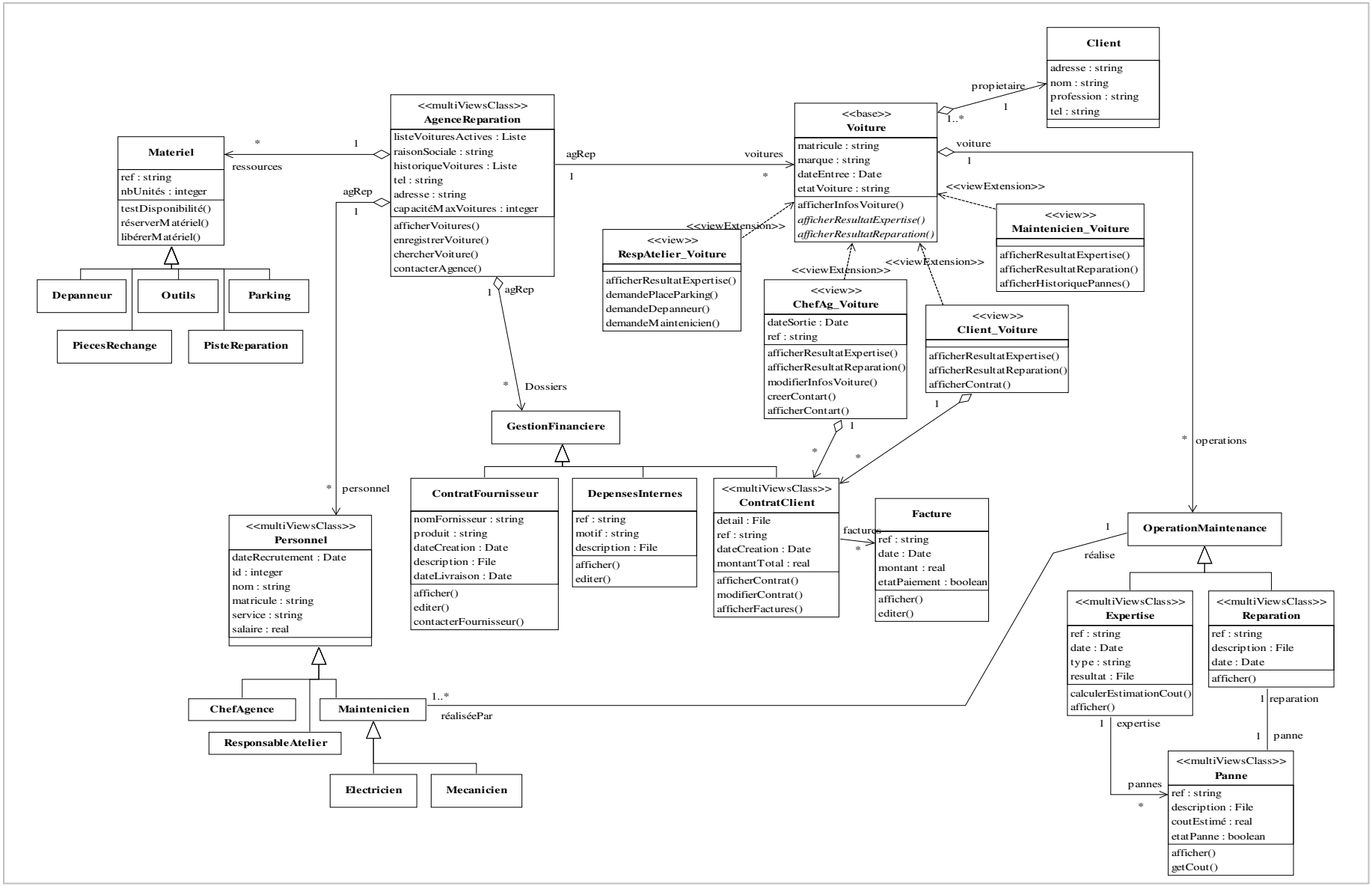

Figure 21. *Diagramme VUML obtenu par fusion des diagrammes de classes partiels des différents points de vue*

4.2. Fusion Dynamique

Après la fusion statique et la construction des classes multivues, on démarre l'étape de la fusion dynamique. L'objectif de cette étape est d'assurer la cohérence dans le comportement des objets multivues.

4.2.1. Synchronisation des machines-vues

Le principe de la synchronisation consiste à assurer la communication entre les différentes machines-vues à travers la machine-base. En effet, la synchronisation revient à alimenter la machine-base (voir la figure 6) par des messages dans l'objectif de créer la correspondance entre les machines vues.

Par exemple, dans la machine-vue du propriétaire de la voiture (voir figure 16), le signal *demandeReparation()* a été envoyé à la machine-base (le signal a été envoyé à la machine base car le développeur du point de vue client ignore qui va traiter ce signal). La machine-vue du chef d'agence (voir figure 17) attend ce signal pour franchir la transition depuis l'état EnProcedureExpertise vers l'état EnProcedureReparation. Le premier message de synchronisation montré sur la figure suivante vise la réalisation de cette correspondance. Le travail de synchronisation continue en suivant la même démarche jusqu'à produire la machine-base finale.

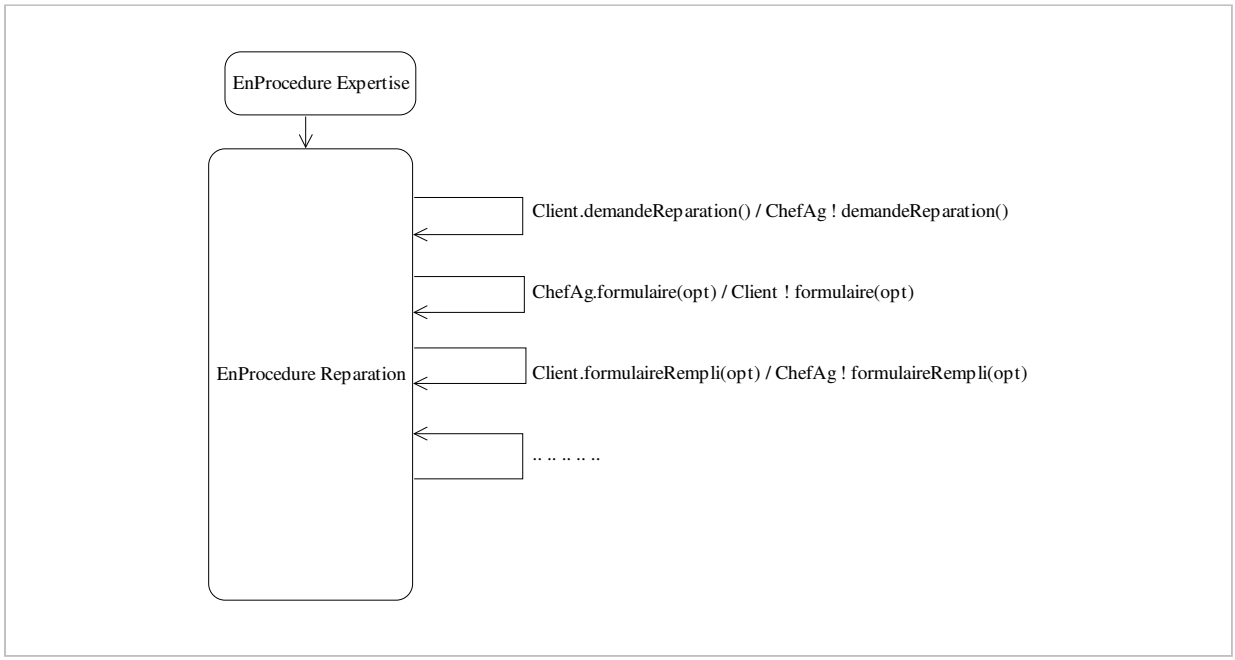

Figure 22. *Ajout de messages de synchronisation dans la machine-base pour l'état "EnProcedureReparation" (Extrait)*

4.2.2. Traitement de l'exception "une classe réactive est identifiée lors du développement par point de vue"

Du fait que la classe "Matériel" n'est pas une classe multivue, nous éliminons la machine-base, et donc la machine-vue développée pour le point de vue ResponsableAtelier (voir la figure 19) reste la seule machine représentant la classe "Matériel".

5. Conclusion

Dans le but de proposer une démarche globale de développement orienté objet par points de vue, nous avons étendu la démarche associée à VUML [5] qui ne traite que l'aspect statique d'une modélisation par point de vue. La démarche que nous avons proposée traite à la fois les aspects statiques et les aspects dynamiques dans une modélisation par point de vue. Ce travail s'appuie sur le profil VUML développé dans notre équipe [1,5,3].

Nous avons tout d'abord donné une vue générale sur les phases potentielles de la démarche, puis à travers l'étude de cas "Gestion d'une agence de réparation de voiture", nous avons détaillé -au fil des sections- les différentes étapes constituant les trois phases de notre démarche.

L'objectif principal de la démarche est d'arriver à produire le diagramme de classes VUML représentant la structure statique de l'application, complété par des machines à états multivues exprimant le comportement des objets multivues présents dans l'application.

Les machines à états proposées suivent la spécification UML2.0, et sont attachées soit à une classe « base », soit à une classe « vue ». Une machine-vue représente le cycle de vie d'un objet-vue, tandis qu'une machine-base a pour but de créer la cohérence et la coordination entre les différentes machines-vue, et de spécifier le comportement commun aux acteurs. On appelle machine « multivue » l'ensemble formé d'une machine-base et des machines-vue dépendantes.

Nous travaillons actuellement sur l'automatisation de la troisième phase de la démarche –phase de fusion– qui est jusqu'à maintenant manuelle. Une telle automatisation diminuera la complexité de développement par VUML et facilitera la transformation d'un modèle VUML en un modèle exécutable tel que le profil OmegaUML [4,8]. Comme VUML, OmegaUML est aussi un profil UML à base d'automates communicants, à la différence qu'il dispose d'une sémantique formelle complète et exécutable, et d'outils de simulation et de vérification de modèles, d'où l'intérêt de relier VUML à ce profil.

Références

[1] B. El Asri "Un modèle de composants multivue pour VUML" thèse ENSIAS, Université Mohamed V- Souissi Rabat, octobre 2005.

[2] B. El Asri, M. Nassar, A. Kriouile, B. Coulette, "Assemblage de composants multivues par contrats", Inforsid'2005, Grenoble, France, 24-27 mai, 2005.

[3] Coulette B., Kriouile A., Marcaillou S., "L'approche par points de vue dans le développement orienté objet des systèmes complexes", Revue l'Objet, vol. 2, n°4, février 1996, pp. 13-20.

[4] I. Ober, S. Graf and I. Ober. Validating timed UML models by simulation and verification. International Journal of Software Tools for Technology Transfer (STTT), Volume 8, Number 2, pages 128-145, April, 2006. Springer Verlag.

[5] M. Nassar “Analyse/conception par points de vue : le profil VUML”, thèse INPT, Toulouse, 28 septembre 2005.

[6] M. Nassar, B. Coulette, A. Kriouile, "Génération de code dans VUML". Journal Marocain d'Automatique, d'Informatique et de Traitement du Signal 2004.

[7] OMG. UML 2.0 superstructure specification. formal/05-07-04, Object Management Group, August 2005.

[8] W. Damm, B Josko, A Pnueli, A Votintseva A discrete-time UML semantics for concurrency and communication in safety-critical applications. In /Science of Computer Programming/ 2005. <http://www-omega.imag.fr/biblio/biblio.php?view=item&biblio=publications&key=Damm*05>

[9] Y. Lakhrissi, I. Ober, B. Coulette, M. Nassar, A. Kriouile. « Vers la notion de machine à états multivue dans le profil VUML ». WOTIC07, 5-6 Juillet 2007, Ensias, Rabat, Maroc.