ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN INGÉNIÉRIE
M.Ing.

ASSEMBLAGE DE MODÈLES DE COMPOSANTS MÉTIER

PAR
GILLES BAPTISTE AMBARA-MOUNGUET

MONTRÉAL, LE 11 JANVIER 2007

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Roger Champagne, directeur de mémoire
Département du génie logiciel et des TI à l'École de technologie supérieure

M. Éric Lefebvre, co-directeur
Département du génie logiciel et des TI à l'École de technologie supérieure

Mme Sylvie Ratté, présidente de jury
Département du génie logiciel et des TI à l'École de technologie supérieure

M. Michel Lavoie, membre du jury
Département du génie logiciel et des TI à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 11 DÉCEMBRE 2006

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ASSEMBLAGE DE MODÈLES COMPOSANTS MÉTIER

Gilles Baptiste Ambara-Mounguet

SOMMAIRE

La concurrence entre entreprises entraîne celles-ci vers des solutions sophistiquées en matière de systèmes logiciels. Cette sophistication implique la montée de la complexité de ces systèmes et pousse la recherche en génie logiciel vers des processus de développement par composants ou par réutilisation de composants.

Ce projet consiste à assembler des modèles de conception vus en tant que composants. Nous y recensons un ensemble de techniques ou procédés actuels permettant l'assemblage de composants en vue de l'obtention d'une application. Nous examinons les forces et les faiblesses de ces techniques et nous tirons les éléments essentiels susceptibles d'aider à déterminer une méthode d'assemblage de modèles de composants métier. Pour atteindre cet objectif, nous répertorions les éléments importants des méthodes d'assemblage afin de les utiliser dans ce travail de recherche.

L'utilisation d'un cas d'étude permettant d'expérimenter la méthode proposée a permis d'établir un assemblage de composants. Cet assemblage présente un modèle d'application fonctionnelle ayant fondamentalement les mêmes atouts que le modèle obtenu par une méthode d'ingénierie, notamment le respect des fonctionnalités et de l'enchaînement des processus.

Cependant, bien que nous démontrions la faisabilité d'un assemblage de modèles de composants métier, un travail important reste à faire notamment dans l'uniformisation des normes de développement de ces modèles. Un autre élément tout aussi important à examiner est l'automatisation de la méthode de la glu utilisée dans ce projet. Il présente des éléments encourageants, en l'occurrence les règles de correspondance, de combinaison et de remplacement, permettant de croire à une éventuelle automatisation.

BUSINESS MODEL COMPONENTS ASSEMBLING

Gilles Baptiste Ambara-Mounguet

ABSTRACT

The competition between organizations implies sophisticated software solutions for their management. These solutions increase the complexity of systems and forces research in software engineering to consider a component development process or component reuse process.

This project consists of composing models seen as components. We list and examine a set of composition processes used today to obtain a software application. We analyze the strengths and weaknesses of each component composition process that can help us to find a component composition method. In order to reach our goal, we gather important information from existing assembling methods in order to use them in our research work.

A case study allowed the experimentation of the assembly method to assess its usefulness. The assembly obtained is a normal component model having basically the same characteristics (functionalities and workflow) as a typical model obtained by another engineering method of development.

However, even if we show the feasibility of component model assembling, it still remains to standardize the development rules in this subject. Another important technique used in this project which requires more work is the automation of the assembling method named the "glue". It shows encouraging signs about the possibility to be automated.

REMERCIEMENTS

Je remercie très sincèrement Monsieur Roger Champagne, directeur de ce projet, et Monsieur Éric Lefebvre, co-directeur de ce projet. Je voudrais par ce mémoire vous exprimer ma reconnaissance pour les conseils ingénieux et pertinents que vous m'avez donnés durant ce projet ainsi que la confiance, la compréhension et la patience que vous m'avez témoignés. Je vous dédie ce travail. Merci infiniment.

Un grand merci aussi à ma femme, Catalina Ambara, qui a su faire preuve de patience et m'a soutenu moralement depuis le début de ma maîtrise.

TABLE DES MATIERES

LISTE DES TABLEAUX

LISTE DES FIGURES

ABREVIATIONS ET SIGLES

AOP	Aspect Oriented Programming
GRASP	General Responsibilities Assignment Software Patterns
ISC	Invasive Software Composition
CIM	Computation Independent Model
PIM	Platform Independent Model
PSM	Platform Specific Model
MDA	Model Driven Architecture
OOP	Object Oriented Programming
OMG	Object Management Group
SOP	Subject Oriented Programming
UML	Unified Modeling Language

INTRODUCTION

La mondialisation de l'économie rime avec la communication entre entreprises, que cela soit localement ou à distance. De grands groupes ne peuvent plus exister seuls et sont obligés de fusionner. Du point de vue du génie logiciel, cela entraîne une complexité dans la réalisation des systèmes et une flexibilité dans leur conception. La vitesse de l'évolution technologique, la montée des systèmes distribués, l'hétérogénéité des plates-formes, le mode de fonctionnement de plus en plus compliqué de systèmes, la demande sans cesse croissante de systèmes informatiques et par-dessus tout, des systèmes pouvant s'adapter à des environnements variés de fonctionnement, rendent difficile la conception de systèmes aujourd'hui. Différents concepts, tant du point de vue de la conception que de la réalisation, ont vu le jour afin de répondre à cette réalité parmi lesquels on peut citer le processus unifié, les différents patrons d'analyse et de conception, la programmation orientée objet et bien d'autres [1].

Le processus RUP [2] en particulier est un processus d'ingénierie qui permet de séparer le processus de développement de logiciels en plusieurs sous processus afin de minimiser les risques et les coûts pendant le développement.

Par ailleurs la programmation orientée objet, bien adaptée au concept RUP [1], introduit la notion d'objets. La granularité de ces objets est du niveau code et ne permet pas une réutilisation à haut niveau des modules développés. De plus, le problème du décalage entre le code et les modèles de conception reste entier [3].

Il faut donc penser à des méthodes de réutilisation ou d'automatisation de la génération de code. C'est dans ce sens que le concept MDA [4] a été promu par l'OMG [5]. Il introduit la génération du code à partir de modèles que nous verrons plus loin dans ce document.

Naturellement, dans le cadre de la génération de code, il faudrait monter de plusieurs niveaux d'abstraction dans le processus de réalisation d'un logiciel et travailler avec les modèles du logiciel, à partir desquels la génération automatique du code sera obtenue. En fait, toute la précision que l'on rencontre dans la phase d'implémentation se retrouverait ainsi dans la conception de modèles. Mais une bonne partie du problème du génie logiciel se trouverait ainsi réglée. Nous pourrons par exemple, à partir de modèles, comprendre l'implémentation de l'application. Il n'y aurait plus de décalage entre les modèles et le code.

L'autre partie du problème pourrait être résolue par la réutilisation de modèles. Un modèle fonctionnel d'une application X pourra servir dans une autre application Z sans grands changements. Nous pourrions ainsi nous constituer un catalogue de modèles de modules communs à plusieurs applications.

Cette étude se penche sur le problème en traitant l'assemblage de modèles d'une application. Pour ce faire, nous l'avons divisée en quatre chapitres.

Dans le premier chapitre, nous présentons les techniques actuelles d'assemblage, leur mode de fonctionnement ainsi que leurs caractéristiques. Dans le deuxième chapitre, nous expérimentons ces techniques via un cas d'étude. Dans le troisième chapitre, nous introduisons la méthode d'assemblage de composants proposée basée sur les expérimentations faites précédemment, ainsi que le concept des composants eux-mêmes. Dans le quatrième chapitre, nous faisons une expérimentation de la méthode proposée avec le même cas d'étude que celui utilisé dans le deuxième chapitre. Et enfin, nous tirons les conclusions sur le travail effectué et nous exposons les recommandations sur la composition des modèles pour les futurs projets liés à ce sujet.

OBJECTIFS DU PROJET

Dans la problématique d'un développement et d'une réalisation efficace et performante de logiciel, la réutilisation de composants est une option à considérer. En effet, obtenir une application à partir de composants déjà existants est un sujet qui a tout son intérêt dans le domaine du génie logiciel.

Ce projet a pour objectif, via les concepts de l'approche MDA (Model Driven Architecture), d'apporter une critique sur les méthodes d'assemblage qui existent déjà, en analysant leurs forces et leurs faiblesses, afin d'en tirer une conclusion sur ce qui pourrait être fait pour établir une méthode d'assemblage. Le but est d'obtenir un assemblage de composants à partir d'une technique basée sur ce qui existe déjà. L'utilisation de composants est justifiée par le fait qu'ils incitent généralement à la réutilisation ou à l'assemblage.

Dans bien des approches, le concept de réalisation d'applications par assemblage des composants est déjà établi [7]. Il ne s'agit donc pas de réinventer la roue mais de proposer une technique d'assemblage à partir des conclusions qui seront tirées des méthodes existantes étudiées.

Le modèle qui servira à représenter les composants est le modèle PIM (Platform Independent Model), tiré des concepts de l'approche MDA. Ce modèle a des atouts qui favorisent ce choix. Il a un niveau d'abstraction élevé, étant le modèle à partir duquel découlent les autres modèles [4]. Il est assez proche des modèles du domaine et du modèle de conception, modèles bien connus dans le processus de développement de logiciels [1].

CHAPITRE 1

ÉTAT DE L'ART

Le développement par composants à différents niveaux de granularité est une réalité dans le génie logiciel. En effet, bon nombre d'approches tournent autour de ce concept [7, 8]. En orienté objet, la composition existe déjà pour des composants à fine granularité. Nous pouvons parler de bouton, case à cocher ou bien d'autres éléments permettant l'assemblage de petits composants d'une application.

Des définitions ont été données et ont toutes, plus ou moins, les mêmes critères et un niveau de granularité semblable. Nous pouvons citer [8] :

> «A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition. »

Un composant logiciel est une unité de composition ayant des interfaces contractuelles spécifiques et des dépendances explicites à son contexte d'utilisation seulement. Un composant logiciel peut être déployé indépendamment et être sujet à une composition par une tierce partie.

Cette définition, bien que très générale, résume les caractéristiques essentielles d'un composant. Les notions d'indépendance et d'interfaces d'un composant sont des notions que nous retrouvons dans l'ensemble des définitions de composant. Dans cette étude, le composant sera considéré comme une unité indépendante ayant des interfaces servant à s'adapter à son environnement de fonctionnement.

Dans ce chapitre, les techniques ou les approches d'assemblage de composants seront présentées. Il va de soi que ces techniques ou ces approches se basent sur un type de

composant donné. Il n'est pas question ici de discuter le type de composant qu'il faudrait avoir pour obtenir un meilleur assemblage. Chaque auteur de technique propose lui-même un type de composant sur lequel il s'est basé pour obtenir un assemblage d'éléments. Mais le but de cette étude est de présenter, peu importe le type du composant, une technique ou une approche viable d'assemblage générique.

1.1 Model Driven Architecture (MDA)

Bon nombre de recherches dans le développement de logiciels ont pour objectif de faciliter le passage entre la théorie et la pratique dans la réalisation d'une application logicielle. Pendant des années et encore aujourd'hui, l'artefact qui dure le plus en terme de temps est le code source de l'application. L'arrivée du langage UML a permis de promouvoir la modélisation dans la conception de logiciels. L'approche MDA a pour défi de remplacer le code en tant qu'artefact le plus durable et le plus viable du point de vue de la représentation du fonctionnement de l'application, par des modèles à partir desquels le code serait généré [9].

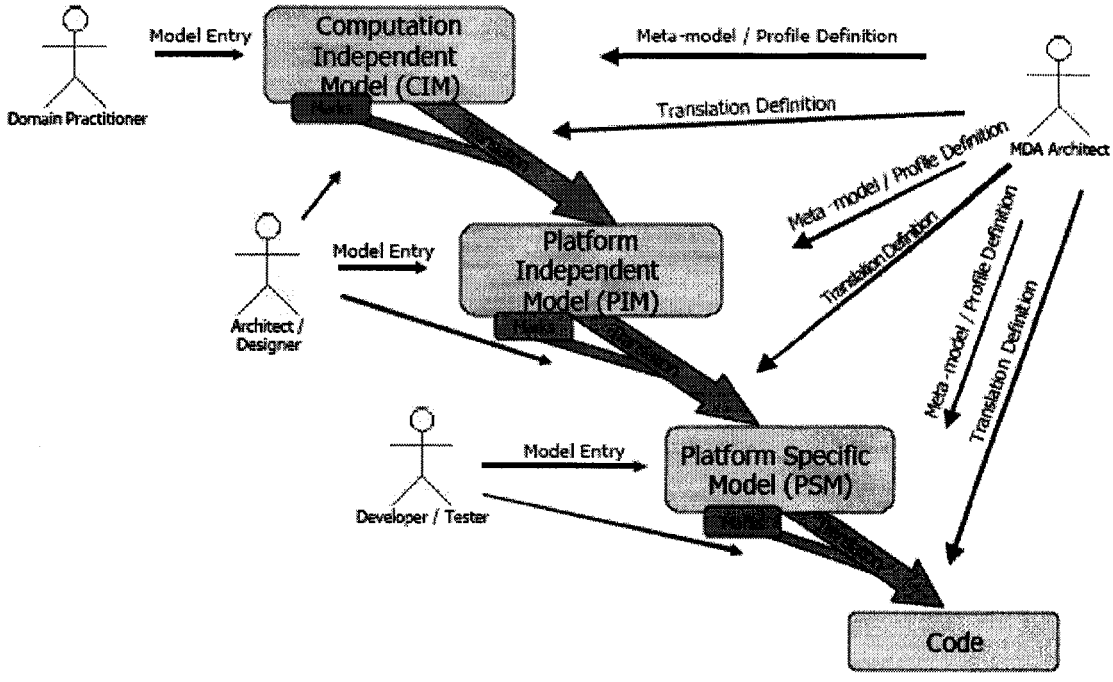


Figure 1.1 Modèles du concept MDA[1]

Pour ce faire, le concept se munit de quatre principaux artefacts (voir figure 1.1) qui sont le modèle CIM (Computation Independent Model), le modèle PIM (Platform Independent Model), le modèle PSM (Platform Specific Model) et le modèle du code.

Le premier modèle, CIM (Computation Independent Model), est le modèle qui spécifie le domaine d'application du système à développer. Les informations qui y sont contenues sont des informations fondamentales des exigences du système à développer. Il est par la suite transformé en modèle PIM. C'est un modèle qui n'est lié à aucune plateforme technologique.

Le deuxième modèle, PIM (Platform Independent Model), est le modèle qui exprime le fonctionnement de l'application indépendamment de toute plate-forme technologique.

[1] Source : http://www.omg.org/mda/mda_files/OCI2004.pdf

C'est ce modèle qui est utilisé, suivant des patrons de transformation, afin d'obtenir le modèle PSM. Un modèle PIM peut être transformé en plusieurs autres modèles PSM dépendamment des plates-formes sur lesquelles nous travaillons.

Dans le concept MDA, le modèle PIM est le modèle dont le niveau d'abstraction est le plus élevé après le modèle CIM, et dont le contenu est transformable pour l'obtention du modèle PSM. C'est le modèle qui résistera à tout changement technique ou technologique à cause du fait qu'il est indépendant de la plate-forme de l'application à développer.

Le troisième modèle, PSM (Platform Specific Model), est le modèle obtenu du modèle PIM par un mécanisme de transformation selon la plate-forme technologique pour laquelle une application serait implémentée. Il contient des propriétés propres à la plate-forme qui serait utilisée pour une application donnée. Cette plate-forme peut être J2EE, .NET ou autre. Par exemple la transformation d'un modèle PIM en PSM sous la plate-forme J2EE impliquerait l'apparition des "entity beans" et "session beans" dans le modèle PSM.

Le quatrième et dernier modèle, le code, est l'artefact qui est obtenu via transformation du modèle PSM. Le code est donc également spécifique à la plate-forme technologique que nous voulons utiliser pour notre application. Ainsi, le code ne représente plus le principal artefact durable de l'application à concevoir. C'est plutôt le modèle PIM qui joue ce rôle, les autres modèles découlant du modèle PIM via des transformations automatisées.

Bien que des applications aient été développées par l'approche MDA, il n'en reste pas moins que des études doivent être menées pour cerner l'ensemble des problèmes entravant encore la viabilité de cette approche, notamment l'interopérabilité des

modèles utilisés par les outils qui permettent de transformer des modèles ou de générer du code [10].
Dans ce projet, nous voulons assembler des modèles PIM qui sont issus de l'approche MDA. Le choix du modèle PIM est justifié par le fait que le modèle CIM est proche de l'aspect théorique du système à développer et le modèle PSM est dépendant d'une plateforme technologique.

L'idée d'obtenir du code à partir de la transformation des modèles PIM et PSM est celle qui guide cette démarche, d'où l'intérêt pour cette approche.

1.2 Concepts ou approches d'assemblage de modèles

Cette section passe en revue les concepts ou les approches qui permettent d'assembler des composants. Ces approches sont choisies en fonction de ce qu'elles apportent dans la démarche de proposition de la méthode d'assemblage de composants. C'est ainsi qu'il sera question de l'approche Model Driven Architecture décrite ci-dessus, qui ne propose pas de méthode d'assemblage, mais dont les concepts seront utilisés dans ce document y compris dans d'autres approches qui seront abordées.

Nous parlerons de l'approche proposée par Salim Bouzitouna [11] que nous appellerons la règle de la glu. Cette approche propose une règle d'assemblage de modèles PSM de deux ou plusieurs applications en se basant sur une expression de composition de leur modèles PIM. Cette approche sera présentée dans la section 1.2.1.

Une autre approche vue dans ce chapitre est celle de la théorie des couches spécifiques et non spécifiques proposée par Ivar Jacobson et Pan-Wei Ng [12]. Cette approche est basée sur l'assemblage des couches obtenues par superposition, une couche étant le complément d'une autre.

La dernière méthode étudiée est celle de la composition chirurgicale de composants proposée par Abmann Uwe [7]. La composition chirurgicale est obtenue en adaptant ou en modifiant les composants à assembler de manière à ce qu'ils puissent s'accommoder parfaitement à leur nouvel environnement de fonctionnement.

1.2.1 Règle de la glu

L'objectif de ce projet est d'obtenir une méthode d'assemblage de modèles PIM. En général, la réutilisation de composants s'apparente à l'assemblage de ceux-ci. Le remplacement d'un composant est également une forme d'assemblage. Dans cette étude, nous voulons un assemblage de modèles de composants. Mais cet assemblage doit nécessairement passer par la définition du composant et une démarche de développement conduisant à cet assemblage.

Si nous considérons l'ensemble des définitions faites sur le composant, l'élément clé qui revient dans presque toutes les définitions est l'indépendance de celui-ci. Nous pouvons voir que la notion d'indépendance est bien présente, mais la granularité de ce composant et son découpage restent vagues.

La méthode d'assemblage de la glu [11] permet l'assemblage de modèles PSM à partir d'une glu tirée de l'expression de composition obtenue par l'étude des modèles PIM correspondant à ces PSM. La génération de la glu est obtenue automatiquement via un générateur, afin de permettre de composer les modèles PSM (voir figure 1.2). Nous définirons l'expression de composition dans les prochains paragraphes.

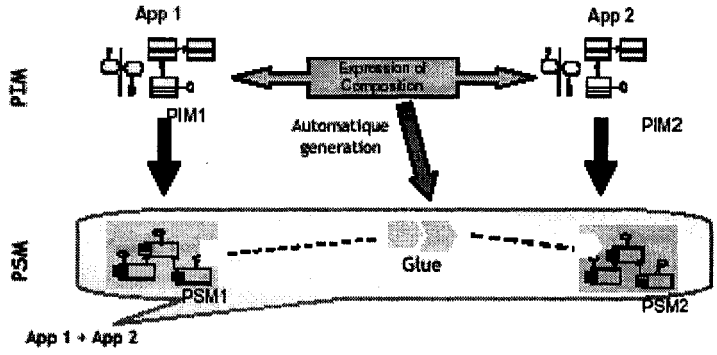


Figure 1.2 Approche d'assemblage par la glu[2]

Il est important de remarquer que cette méthode s'intéresse à l'assemblage des modèles PSM et non seulement aux modèles PIM. De ce fait, nous ne serons concernés que par la première idée de cette méthode qui est de déterminer l'expression de composition de modèles PIM.

L'expression de composition est un ensemble de règles qui permettent de relier les modèles PIM à travers leurs éléments. Ces règles recherchent la similarité de concepts au niveau des modèles afin de les composer. Trois types de règles permettent d'établir l'expression de composition. Ces règles sont :

a. les règles de correspondance;
b. les règles de combinaison;
c. les règles de remplacement.

Les règles de correspondance établissent les correspondances entre les différents éléments composant un modèle. Ces éléments doivent être de même type et spécifier le même concept.

---

[2] Source : Composition rules for PIM reuse, par Salim Bouzitouna et Marie-Pierre Gervais, Université de Paris X

Ces règles sont justement ce qui est intéressant car nous voulons regrouper chaque composant développé séparément. Ce regroupement se fait à travers les entités qui se retrouvent dans plusieurs modèles.

Les règles de combinaison définissent la manière dont l'intégration devra être faite au niveau des deux modèles suite à la correspondance de leurs éléments. La démarche consiste à retenir un seul élément parmi ceux qui sont mis en correspondance. Toutefois, dans le cadre des opérations, l'intégration est caractérisée par l'ordre de leur exécution.

Les règles de remplacement, quant à elles, permettent le remplacement des éléments d'un modèle PIM donné par des éléments de même genre d'un autre modèle PIM. Les remplacements effectués sur un modèle seront reportés dans un autre dit de modification, permettant le retraçage du modèle source.

Toutes les règles vues s'appliquent à tous les éléments d'un modèle d'application, c'est-à-dire un paquetage, une classe, une méthode ou un attribut.

1.2.2 Assemblage par les couches des cas d'utilisation

La deuxième approche fait appel à un concept sur le mode d'assemblage des modèles PIM. Celui-ci est entièrement lié aux cas d'utilisation rédigés selon les principes de la programmation orientée aspect [12]. Pour bien comprendre ce concept, il faut comprendre le concept de la programmation orientée aspect. Les paragraphes suivants en font une brève revue.

La programmation par aspects est un concept émergeant ayant les moyens techniques de combler les faiblesses de l'OOP (Object Oriented Programming) [7], notamment dans la modularité des éléments, en ce sens qu'il apporte la séparation de la logique d'une application d'avec les autres préoccupations telles que la performance, la sécurité

et bien d'autres [7, 13]. L'AOP (Aspect Oriented Programming) propose de séparer ces deux dimensions lors même de l'implémentation de l'application et de pouvoir les rassembler au moment voulu pour éviter l'enchevêtrement des modules ("crosscutting"). Cet enchevêtrement du corps d'application et des préoccupations réparties un peu partout dans le code entraîne la confusion ("tangling") dans celui-ci et diminue la modularité et la qualité du code, et rend difficile la maintenance.

L'enchevêtrement des préoccupations n'est pas propre au développement orienté objet car déjà dans les cas d'utilisation, qui sont considérés comme étant une étape importante vis-à-vis des procédés de développement, nous retrouvons le phénomène de "crosscutting". Le développement par composants est également un moyen de pallier à ce genre de phénomène car les composants peuvent-être considérés comme des préoccupations.

Le concept AOP propose dans son mécanisme d'opération, de considérer les préoccupations (sécurité, performance etc) comme des aspects, donc des objets en dehors du corps de l'application, et de les réinsérer pendant le fonctionnement de l'application par une opération d'assemblage ("weaving") à des points bien précis ("join points") dans le code. Cette opération d'assemblage permet de composer les préoccupations développées séparément et les objets de la logique de l'application.
Le déroulement de l'opération "weaving" se passe comme suit :

a. le corps de l'application est développé séparément des aspects de cette application;
b. les aspects sont traités à part du corps de l'application pour éviter l'enchevêtrement direct avec celui-ci;
c. des points de jonction sont aménagés afin de recevoir un aspect. Il peut s'agir d'un aspect concernant la sécurité ou encore la performance de l'application;

d. le corps de l'application peut être par la suite séparé de l'aspect dont il s'est servi pour des problèmes de sécurité ou de performance.

Par ces points, nous pouvons constater que les techniques d'assemblage sont déjà pratiquées par les machines et qu'il faudrait rehausser le niveau d'abstraction pour pouvoir assembler des modèles PIM.

Pour revenir à la théorie des couches, un cas d'utilisation permet de mettre en évidence l'architecture d'un logiciel ou d'une partie du logiciel à développer vu de l'extérieur. Pour obtenir tout le fonctionnement de l'application, il faut faire appel à l'ensemble des cas d'utilisation développés. L'application finale est obtenue par la superposition de tous les cas d'utilisation. Nous parlons ainsi de couches et, dans chaque couche, nous retrouvons des éléments qui sont spécifiques au cas d'utilisation et d'autres non [12].

Par définition, une couche est une entité ou une classe qui peut se retrouver dans plusieurs cas d'utilisation, c'est-à-dire par exemple que les cas d'utilisation A et B ont une même entité, la classe X. Dans ce cas-ci, nous parlerons de couches non spécifiques Xa et Xb car, une fois réunies, elles forment la classe X dans son entier. En d'autres termes, la combinaison ou la superposition de Xa et Xb donne X, d'où la notion de couche.

Une couche spécifique au cas d'utilisation est donc une entité ou une classe propre à un cas d'utilisation et dont l'usage est limité à ce seul cas d'utilisation. Par exemple, dans une application de location de voitures, la classe "Réservation" est une couche spécifique au cas d'utilisation "Réserver" et n'est utilisée dans aucun autre cas d'utilisation (sauf comme intrant). Une couche spécifique au cas d'utilisation porte l'information la concernant. Cette information permet d'avoir un aperçu de l'entité.

Une couche non spécifique au cas d'utilisation est une entité ou une classe qui se retrouve dans plusieurs cas d'utilisation. Elle n'est donc pas propre à un cas d'utilisation ou à une couche particulière. Une entité "Voiture" dans le cas d'utilisation "Réserver" par exemple, n'est pas une couche spécifique mais plutôt une couche non spécifique car elle se retrouve dans d'autres cas d'utilisation de la même application, tels que "Louer" ou "Retourner".

Contrairement à une couche spécifique, les couches non spécifiques dans un cas d'utilisation ne portent qu'une partie de leurs informations, car les autres parties sont portées par la même entité ou classe mais dans un cas d'utilisation différent.

Pour illustrer l'identification des couches, considérons un exemple de deux cas d'utilisation, "Réserver une voiture" et "Louer une voiture".

Tableau I

Représentation et identification des couches spécifiques et non spécifiques

	Réservation	**Location**	**Voiture**	**Client**
Réserver	Réservation	-	Voiture1	Client1
Louer	-	Location	Voiture2	Client2

Dans la colonne de gauche du tableau I, nous avons les cas d'utilisation, et dans les autres les différentes entités qui constituent ces cas d'utilisation [12]. Pour chaque cas d'utilisation, nous avons sur la même ligne les couches des entités. L'entité qui n'apparaît qu'une fois dans la colonne telle que l'entité "Réservation" est une couche spécifique du cas d'utilisation "Réserver". L'entité qui apparaît plusieurs fois telle que l'entité "Client" est une couche non spécifique à un cas d'utilisation, car elle se retrouve dans plusieurs cas d'utilisation de l'application.

Ainsi, d'après le tableau ci-dessus, les entités "Réservation" et "Location" sont des couches spécifiques respectivement au cas d'utilisation "Réserver" et au cas d'utilisation "Louer". Toutes les autres entités sont des couches non spécifiques aux cas d'utilisation.

1.2.3 Composition chirurgicale de logiciels (Invasive Software Composition)

Invasive Software Composition (ISC) utilise un triplet, c'est-à-dire un ensemble de trois règles initiées par Abmann [7] [14], qui permet de composer par l'adaptation, la jonction ou l'extension, des composants ou des fragments de code destinés à être rassemblés afin d'obtenir une application complète. Selon Abmann Uwe, trois principes fondamentaux sont à la base du concept d'assemblage de composants. En effet, toutes les techniques de composition utilisées présentement pour l'assemblage des composants pour différents niveaux d'abstraction dans différentes techniques suivent ces principes [7]. Bien que l'ISC soit proche du code, les principes mentionnés sont intéressants et s'inscrivent bien dans la démarche de la méthode d'assemblage.

Selon ces principes, il faut pouvoir définir avant tout assemblage, un modèle de composant, une technique de composition et un langage de composition (Voir figure 1.3).

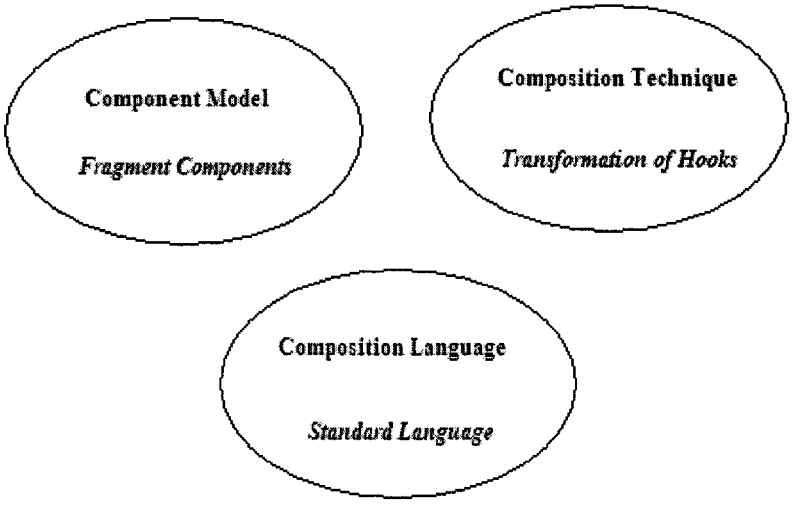


Figure 1.3 Concepts de l'ISC

Par exemple, dans le concept de la programmation par sujet (Subject Oriented Programming) [6], le modèle de composant est celui des vues (des classes C++), la technique de composition est celle des opérateurs d'assemblage qui agissent sur les vues (correspondance, combinaison et remplacement), et le langage d'assemblage est le langage utilisé dans le script des opérateurs d'assemblage.

L'ISC permet l'assemblage de composants par la transformation de ces derniers. En agissant de la sorte, les composants assemblés sont mieux adaptés pour un meilleur fonctionnement.

Le modèle de composant pour l'ISC, utilisé dans l'assemblage, n'est pas différent de celui de l'objet dans le concept de la programmation orientée objet. En effet, le composant est ici un ensemble de fragments de programmes. Ces fragments de programmes possèdent des crochets où peuvent s'accrocher d'autres fragments de programmes. Ces crochets correspondent dans la terminologie MDA aux ports du composant.

Le concept de composant dans l'ISC est très important dans l'application de ce dernier. En effet, dans l'ISC, le composant est une entité logicielle destinée à une composition. Un fragment de code est un composant potentiel dans cette approche. Le fragment de code peut être une classe et les crochets sont les méthodes. Il peut également être une méthode, et à ce moment-là les crochets deviennent les attributs de la méthode. Il y a donc différents niveaux de fragment de code. L'idée du composant en boîte noire n'est pas applicable car le code du composant ou représentant le composant est retouché. Quoi qu'il en soit, le composant dans l'approche ISC est du niveau code au même titre que le paquetage ou l'objet dans l'AOP [6].

La technique de composition pour l'ISC, repose sur des procédés de transformation de code permettant de lier des fragments de code à travers leurs crochets. Ces procédés, nommés composeurs, ne se contentent pas de faire une jonction entre deux composants voués à être assemblés, mais transforment leurs crochets afin que ceux-ci respectent entièrement les exigences, même dans un contexte de réutilisation.

Les composeurs qui sont utilisés pour transformer les crochets sont des programmes. Ils utilisent les règles élémentaires de la construction des programmes logiciels pour arriver à leur fin (Voir figure 1.4).

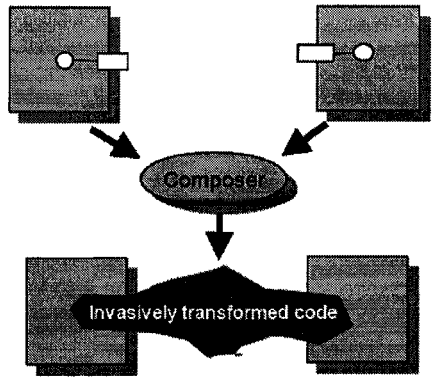


Figure 1.4 Assemblage par le concept ISC

Le composeur transforme les composants en instanciant, en modifiant, en étendant et en adaptant leurs crochets. Le composant est ainsi changé chirurgicalement. Les fragments ainsi impliqués dans cet assemblage deviennent un seul élément entièrement adapté à leur nouveau mode de fonctionnement, ce qui donne la composition "chirurgicale" de logiciels.

Le langage de composition pour l'ISC dépend du langage par lequel les composeurs, qui sont des unités de programmes, ont été développés. Ce langage est lié à celui des fragments de code car les composeurs auront besoin de réadapter le code de ces derniers.

Par ailleurs le COMPOST [6, 15], qui est une bibliothèque écrite en java, permet d'appliquer les principes de l'ISC. Il peut être ajouté dans un programme si celui-ci nécessite l'assemblage de fragments de code par l'approche ISC. Son fonctionnement est similaire à tous les paquetages et classes des bibliothèques de classes java. Il suffit de les appeler et d'en utiliser les composeurs afin d'obtenir une composition chirurgicale des éléments pour lesquels nous désirons un assemblage.

1.3 Concepts de modélisation

Dans cette section, nous introduisons deux concepts importants pour la suite de ce rapport, à savoir l'UML 2 qui est la nouvelle version d'UML adaptée aux concepts de l'approche MDA, et les patrons d'analyse (en l'occurrence les archétypes) qui seront très utilisés dans l'ensemble des modèles qui seront conçus.

1.3.1 UML 2 (Unified Modeling Language)

Les modèles PIM que nous utilisons sont conçus à l'aide du langage UML. En effet, UML est le langage proposé par l'OMG (Object Management Group) [16] pour la modélisation des applications logicielles. Ce langage, lancé par l'OMG en 1997, a été

adopté par l'industrie du logiciel et permet la modélisation des processus d'affaires dans bien des domaines autres que le génie logiciel.

En regard de l'approche MDA, la version antérieure d'UML n'avait pas tous les éléments nécessaires pouvant servir à la transformation des modèles. La précision qu'impose une transformation d'un modèle à un autre ne pouvait être obtenue à l'aide de cette version. Ainsi, une nouvelle version, UML 2 [17], qui étend la précédente et sortie récemment, permet d'appliquer le procédé. Cette nouvelle version possède une nouvelle structure dans l'assemblage de ses nouveaux éléments. Parmi ceux-ci, nous pouvons compter :

a. les interfaces;
b. les ports;
c. les parties ou sous composants;
d. les composants;
e. les connecteurs.

Dans les paragraphes suivants, les éléments introduits dans la version 2 d'UML seront présentés.

Une interface représente entre autres les services d'un composant. Elle peut être fournie ou requise. Elle sert de lien entre un composant et son environnement. L'environnement ici sous-entend les autres composants avec lesquels un composant peut communiquer.

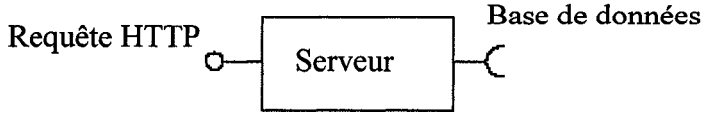


Figure 1.5 Illustration des interfaces d'un composant

Dans la figure ci-dessus le composant "Serveur" possède deux interfaces : une de type fourni ("Requete HTTP") et de type requis ("Base de donnees"). L'interface de type fourni est celle par laquelle le composant fournit ses services aux autres composants, et l'interface de type requis est celle par laquelle le composant consomme les services des autres composants.

Un port [17] est le point d'interaction entre le composant et son environnement immédiat. Les interfaces du composant associées au port, spécifient la nature des interactions qu'il traite. Il y a deux types d'interfaces au niveau d'un port : l'interface fournie qui caractérise une requête fournie par le composant, et l'interface requise qui caractérise une requête demandée par le composant.

Figure 1.6 Illustration des ports d'un composant

Lors de l'introduction de la méthode d'assemblage, nous proposerons une catégorisation des types de ports qui servira à faciliter le processus d'assemblage.

Une partie (sous-entendu une partie d'un composant) ou un sous-composant est une unité qui compose le composant principal. Elle contribue au fonctionnement global du composant et est partie intégrante de celui-ci.

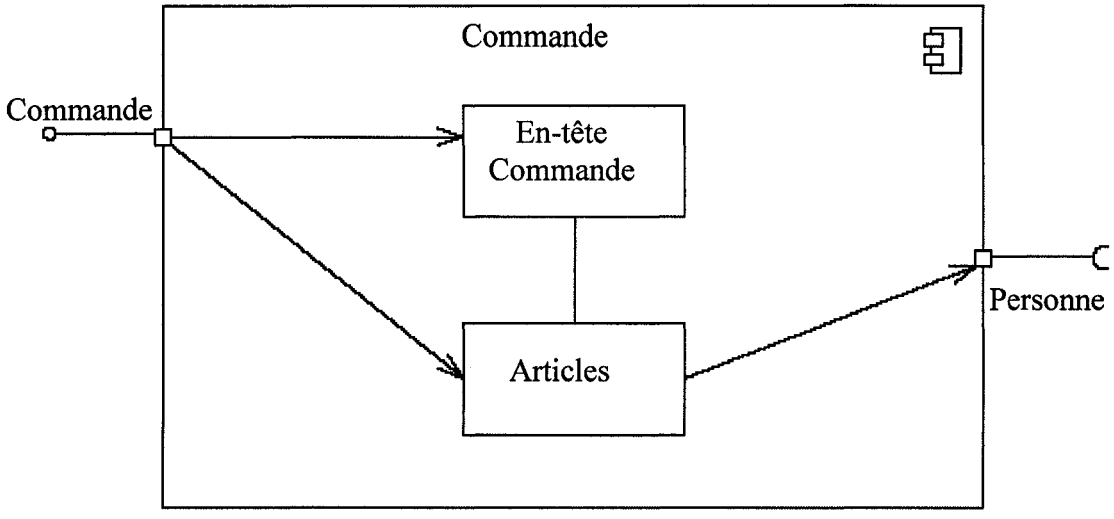


Figure 1.7 Illustration des parties d'un composant en UML 2

Dans la figure ci-dessus, nous pouvons voir que deux sous parties, "En-têteCommande" et "Articles", forment le composant principal "Commande". En d'autres termes, ce sont des sous composants du composant principal "Commande".

Le composant dans UML 2 est une unité indépendante pouvant être branchée ou débranchée à volonté à travers ses interfaces. Il y a plusieurs manières de représenter le composant UML 2. Le composant, en-dessous des trois autres, présenté sur la figure 1.8 sera utilisé dans ce projet car il met bien en évidence les interfaces du composant et permet de faire aisément le lien entre les composants qui doivent être assemblés.

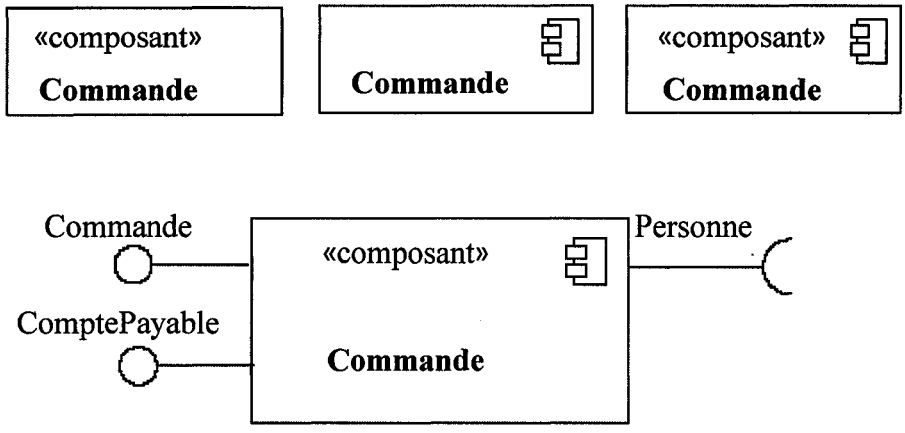


Figure 1.8 Illustration de composant en UML 2

Un connecteur représente le lien entre un composant et un autre, ou un composant et son sous-composant (voir figure 1.7). Le connecteur est dit connecteur de délégation s'il lie un composant avec son sous-composant, et est dit d'assemblage s'il lie un composant avec un autre composant. Le lien effectué via le connecteur se fait à l'aide des interfaces que compte un composant.

Le connecteur [18] ne peut être associé à un comportement ou attribut qui caractérise la connexion. En d'autres termes, le connecteur reste subjectif lorsqu'il s'agit du code. Il n'est ni une méthode ou fonction, ni un attribut.

Bien qu'il ne soit qu'un lien, le connecteur peut cacher d'énormes difficultés du point de vue de son rôle dans l'assemblage car étant le lien entre les composants, il peut avoir la charge du bon déroulement des échanges entre les deux parties. Il contient une partie appelée "rôle" qui effectue effectivement la liaison entre deux composants.

Le rôle définit l'interaction entre composants. C'est un élément essentiel dans l'étude de l'assemblage.

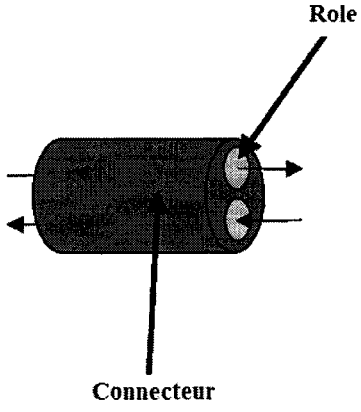


Figure 1.9 Le connecteur

Le rôle est la fonction que doit avoir le composant dans un contexte donné d'assemblage de composants. Il est caractérisé par l'interface du composant mentionné plus tôt dans cette section.

1.3.2 Le patron MVC

Différents patrons ont été développés en génie logiciel afin de permettre l'amélioration et la compréhension des modèles d'implémentation d'application. Les plus connus sont les patrons GOF [19] qui proposent, à un plus bas niveau d'abstraction que MDA, des solutions connues à des problèmes communs dans la conception de modèles d'implémentation d'application. Ces patrons sont très utilisés dans les architectures comme J2EE ou .NET et ne sont pas applicables aux modèles PIM à cause du niveau d'abstraction élevé de ces derniers. Mais un des patrons nommé dans ce document est le patron MVC [1]. Il permet la séparation des modules, catégorisant un logiciel en trois parties : le "Modèle", qui correspond à un support de données, la "Vue", qui détermine l'interface d'utilisation du logiciel et le "Contrôleur", qui gère l'ensemble des activités du logiciel. Le découpage proposé par ce patron est intéressant car il divise un logiciel en plusieurs parties. Nous obtenons ici trois couches qui sont l'interface, le contrôleur (processus intermédiaire entre l'interface et le modèle) et le modèle (support de

données). Ces trois notions sont bien ancrées dans le monde de développement du génie logiciel.

1.3.3 Les archétypes

Par définition, un archétype est une forme modèle que les objets de même genre ou de même catégorie suivent plus ou moins [20]. En d'autres termes, un archétype est un objet ou un concept générique auquel nous pouvons associer les objets de même type ou de même genre. Il permet d'obtenir des modèles de domaine ou tout autre modèle de conception à partir d'exigences données.

Coad *et al.* [21] distinguent quatre archétypes :

- le "moment interval";
- le rôle;
- la description;
- un groupe de personnes, un endroit ou une chose.

Pour bien identifier ces archétypes dans un modèle, des couleurs sont attribuées à chacun. Les paragraphes qui vont suivre donnent une description de chacun des archétypes.

Le "moment interval" représente une entité qui est utilisée à un moment donné ou dans un intervalle de temps donné. Il est le point central des autres entités impliquées dans un modèle d'application. Par exemple, le processus "Louer une voiture" est un "moment interval" car il intervient à un moment donné. Le processus "Réserver une voiture" est un "moment interval" pour les mêmes raisons que "Louer une voiture". Dans une application, l'enchaînement des "moment interval" détermine les activités de celle-ci. Le "moment interval" a la couleur rose dans un diagramme de classes.

Le rôle est l'archétype qui identifie la manière dont un groupe de personnes, un endroit ou une chose participe à une activité donnée. Cet archétype a la couleur jaune dans un diagramme de classes.

La description est un ensemble d'éléments qui sont réutilisés. Par exemple, le catalogue des modèles de voitures à louer dans une location est un archétype description. La couleur bleue est associée à cet archétype.

Un groupe de personnes, un endroit ou une chose, comme les noms le soulignent, est un archétype qui se référe à une entité des noms évoqués. Un groupe de personnes peut représenter une personne ou une organisation. Cet archétype est, la plupart du temps, appelé à jouer un rôle dans un modèle donné, et celui-ci est signifié par l'archétype rôle dont il a été question plus tôt. La couleur associée à cet archétype est le vert.

L'intérêt des archétypes dans ce projet est qu'ils sont effectivement impliqués dans la construction des modèles PIM et que leur niveau d'abstraction correspond à celui des PIM décrits dans l'approche MDA. Ces quatre archétypes seront utilisés tout au long de ce projet pour modéliser les composants.

1.4 Conclusion

Chaque approche a sa particularité et son intérêt dans cette étude.

L'approche MDA est une approche qui permet la transformation des modèles PIM en modèles PSM et ensuite en code. Elle ne spécifie pas, en soi, l'assemblage des modèles PIM mais énonce l'idée d'obtenir l'implémentation d'une application à partir de modèles.

L'expression de composition propose une approche d'assemblage. Elle englobe trois règles qui sont :

a. les règles de correspondance;
b. les règles de combinaison;
c. les règles de remplacement.

Elles établissent les correspondances, les combinaison et les remplacements entre les différentes entités de cas d'utilisation.

L'approche de la théorie des couches, qui se base sur l'AOP, est une approche intéressante, dans la mesure où elle permet le regroupement des entités, qui sont ici des classes, appartenant à des cas d'utilisation différents. L'utilisation d'une matrice permet d'identifier les couches qui doivent être regroupées.

L'approche ISC permet d'assembler des composants de fine granularité en les adaptant afin qu'ils puissent fonctionner dans leur nouvel état. Cette adaptation nécessite d'avoir accès au code du composant et ne permet pas un assemblage en boîte noire de composants. Cette approche préconise le fait que pour qu'un assemblage ait lieu, trois principes doivent être suivis, qui sont la définition du composant, la technique de composition et le langage de composition.

Au niveau des outils, le langage de modélisation est le langage UML 2 introduit par l'OMG dont la nouvelle version est adaptée à l'approche MDA. Ce langage introduit de nouveaux concepts comme le port et le connecteur qui pourraient aider à l'assemblage de composants.

Dans le deuxième chapitre, nous verrons comment peuvent s'unir ces concepts par rapport à l'assemblage des modèles PIM.

CHAPITRE 2

EXPÉRIMENTATION DES TECHNIQUES D'ASSEMBLAGE

Dans le chapitre précédent, nous avons décrit les techniques ou approches qui traitent de l'assemblage des modèles. Ce chapitre les expérimente en vue de les évaluer. Pour ce faire, il faudrait utiliser la même application pour toutes les techniques. C'est une application de location de voitures [22] et la raison du choix de cette application est son accès libre donné par le groupe Business Rules.

Il faut signaler qu'il n'est pas question de concevoir une application, mais de faire une expérimentation. De ce fait, nous ne traiterons que quatre cas d'utilisation de cette application qui sont considérés comme acquis. Ces cas d'utilisation sont 1) Réserver, 2) Louer, 3) Allouer et 4) Retourner une voiture.

2.1 Présentation de l'application

Le diagramme des cas d'utilisation représente les cas d'utilisation qui sont traités dans cette application. Le cas d'utilisation "Allouer" est déclenché à la fin de la journée par un acteur particulier qui est l'horloge. Sa représentation en tant qu'acteur se fait comme un acteur normal.

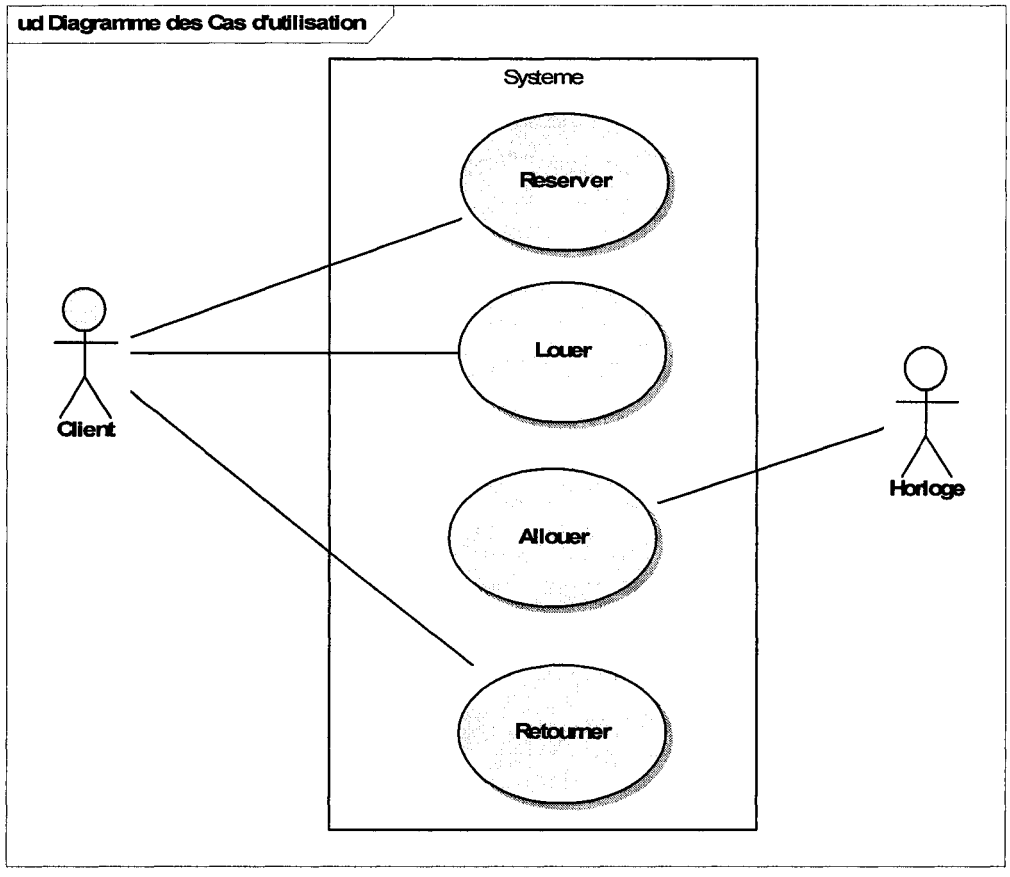


Figure 2.1 Diagramme des cas d'utilisation de l'application

Le tableau ci-dessous donne les éléments et la description des cas d'utilisation mentionnés.

Tableau II

Les cas d'utilisation de l'application

No	Nom	Description
1	Réserver	Un client fait une réservation.
2	Louer	Un client demande une voiture réservée à une succursale.
3	Allouer	À la fin de chaque journée de travail, des voitures sont allouées aux locations des jours suivants.
4	Retourner	Le client retourne la voiture en avance, à l'heure ou en retard.

Pour mieux comprendre le processus de découpage jusqu'au mécanisme d'assemblage des composants, nous traitons les cas d'utilisation séparément car cela permet d'isoler les entités impliquées dans le cas d'utilisation sans se préoccuper de savoir si ces dernières existent dans les autres cas d'utilisation ou non. D'ailleurs, dans le développement de logiciels, il en est généralement ainsi.

Les cas d'utilisation seront traduits en modèle PIM, incluant le diagramme de séquence, qui fait partie de ce dernier et qui donne un aperçu de l'enchaînement des échanges entre les objets.

2.2 Traitement du cas d'utilisation "Réserver"

Le tableau ci-dessous nous donne le déroulement des étapes de ce cas d'utilisation.

Tableau III

Cas d'utilisation "Réserver"

Nom du cas d'utilisation	**Réserver**	**Version 1.1**
Problème du domaine	**Réserver**	
Objet	Les clients ont besoin de faire une réservation pour assurer la disponibilité de la catégorie et du modèle de la voiture qu'ils veulent à la succursale qu'ils veulent.	
Acteurs	Dans le contexte d'une réservation par téléphone, le client parlerait avec un préposé qui sert d'intermédiaire entre le système et le client.	
Evénement déclencheur	Le client désire faire une réservation	
Pré conditions		
Post-conditions	- le client a une réservation pour un groupe particulier, une durée donnée, avec une succursale donnée et une succursale de dépôt donnée ; - l'inventaire de voitures disponibles aux critères mentionnés est diminué pour la durée de la location ; - l'inventaire de la succursale de dépôt est incrémenté de un pour la date donnée.	
Etape	**Actions (Evénements externes)**	**Réponses du Système**
1	Le client envoie la période de réservation, la succursale, la succursale de dépôt, les critères de la voiture.	
2		Le système vérifie la disponibilité des voitures à la succursale pour le jour de la location.
3		(Disponibilité Ok). Le système demande les informations personnelles du client.
4	Le client envoie les informations personnelles.	
5		Le système vérifie les informations du client.
6		Le système confirme la réservation.
7		Le système met à jour l'inventaire de la succursale d'emprunt et dépôt.
Etape	**Action alternative**	**Réponses alternatives du système**
3. **3.a.1**		Pas de voiture à la succursale pour la date donnée, le système demande à une succursale différente ou une date de réservation différente.
3.a.2	Le client envoie de nouvelles données; retour point 4.	
6. **6.a.1**		Le client est sur la liste noire, le système explique la situation et termine la transaction.

Le modèle PIM "Réserver"qui suit est obtenu en se basant sur les scénarios donnés par le cas d'utilisation et l'utilisation des archétypes qui ont été présentés dans le chapitre précédent.

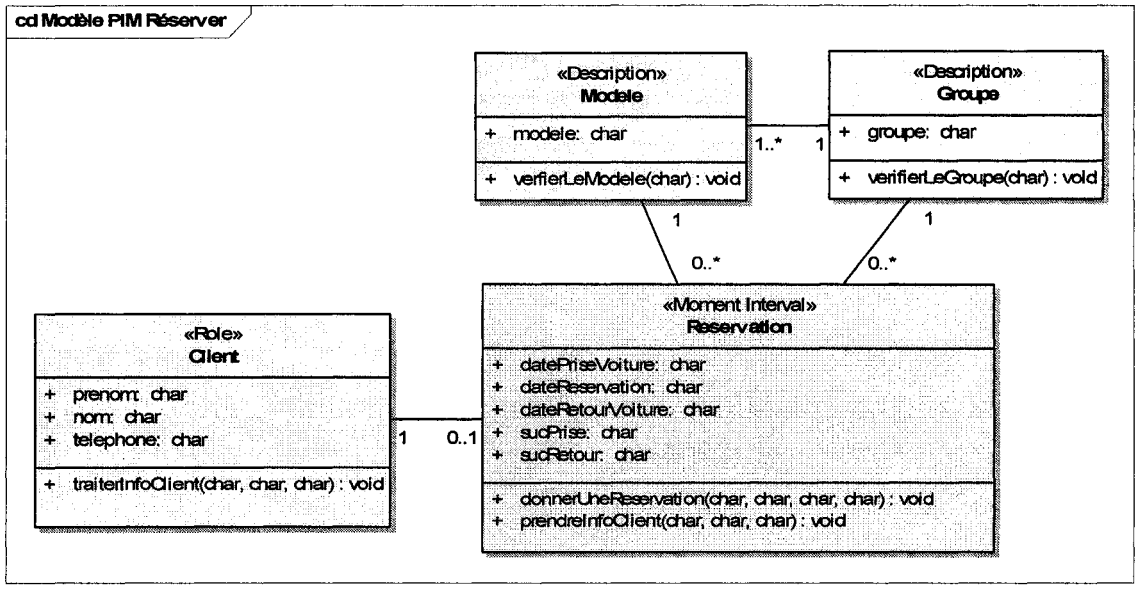


Figure 2.2 Modèle PIM "Réserver"

Dans les scénarios, le préposé déclenche les activités en effectuant une réservation. Le "moment interval" est l'entité "Réservation" qui se fait à un moment donné [20].

Le diagramme de séquence (voir la figure 2.3) illustre la communication entre les objets dans le modèle PIM. Le préposé déclenche le processus en communiquant avec un client au téléphone ou sur place. Le processus "Reservation" génère ensuite tous les autres objets.

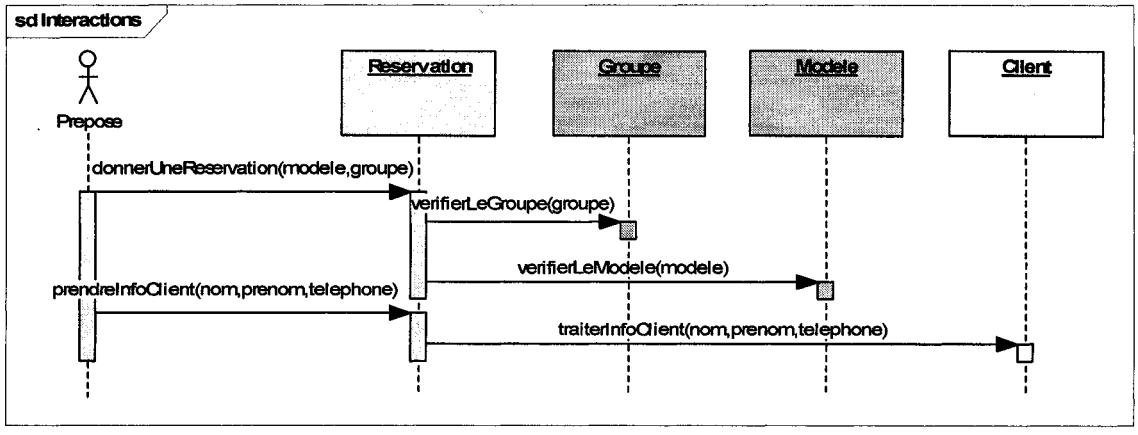


Figure 2.3 Diagramme de séquence du modèle PIM "Réserver"

2.3 Traitement du cas d'utilisation "Louer"

Après le traitement du cas d'utilisation "Réserver", vient ensuite celui de "Louer".

Tableau IV

Cas d'utilisation "Louer"

Nom du cas d'utilisation	**Louer**	Version **1.0**
Problème du domaine	**Louer**	
But	Le client finalise la location pour la voiture réservée. Ce cas d'utilisation décrit la séquence d'interactions.	
Acteurs	Le client et le préposé. Pour des raisons pratiques, le préposé est considéré comme une interface du système.	
Déclencheur	Le client se présente pour sa réservation à la succursale donnée et à la date donnée.	
Pré conditions	- Le client a une réservation à la date donnée et à la succursale donnée.	
Post-conditions	- Le contrat est signé, - La voiture est remise au client, - La carte de crédit du client est débitée au coût de la location.	
Etape	**Actions**	**Réponses du Système**
1.	Le client se présente à la succursale et fournit le numéro de réservation.	
2.		Le système valide le numéro de réservation.
3.	Le client envoie les informations du conducteur.	
4.		Le système valide les informations du conducteur.
5.		Le système présente les critères de la voiture.
6.	Le client confirme les critères.	
7.		Le système valide la police d'assurance.
8.	Le client choisit les options de l'essence.	
9.	Le client envoie les informations de crédit de prépaiement.	
10.		Le système valide les informations de crédit.
11.		Le système vérifie la disponibilité des fonds.
12.	Le client signe le contrat.	
13.		Le système finalise le contrat de location.

Tableau IV (suite)

Scénarios alternatifs		
Etape 2	**Actions alternatives**	**Réponses alternatives du système**
8. 8.a.1.	Le client désapprouve les critères de la voiture et choisit un modèle ou un groupe différent.	
8.a.2.		Le système vérifie la disponibilité d'une voiture; aller au point 8.
8. 8.b.1.	Il y a plusieurs conducteurs. Le client confirme les critères.	
8.b.2.	Le client fournit les informations des conducteurs supplémentaires.	
8.b.3.		Le système valide les informations.
8.b.4.		Le système valide l'éligibilité des conducteurs.
8.b.5.		Le système valide les données des conducteurs; aller au point 9.
9. 9.a.1.		Le système valide la police d'assurance. La couverture est insuffisante, le système propose une couverture additionnelle.
9.a.2.	Le client choisit la couverture additionnelle; aller au point 9.	

Dans ce modèle PIM "Louer", nous retrouvons deux "moment interval" qui sont "Location" et "Paiement". L'un, "Location", déclenche l'autre, "Paiement". Ce détail est important et sera abordé plus loin dans ce document.

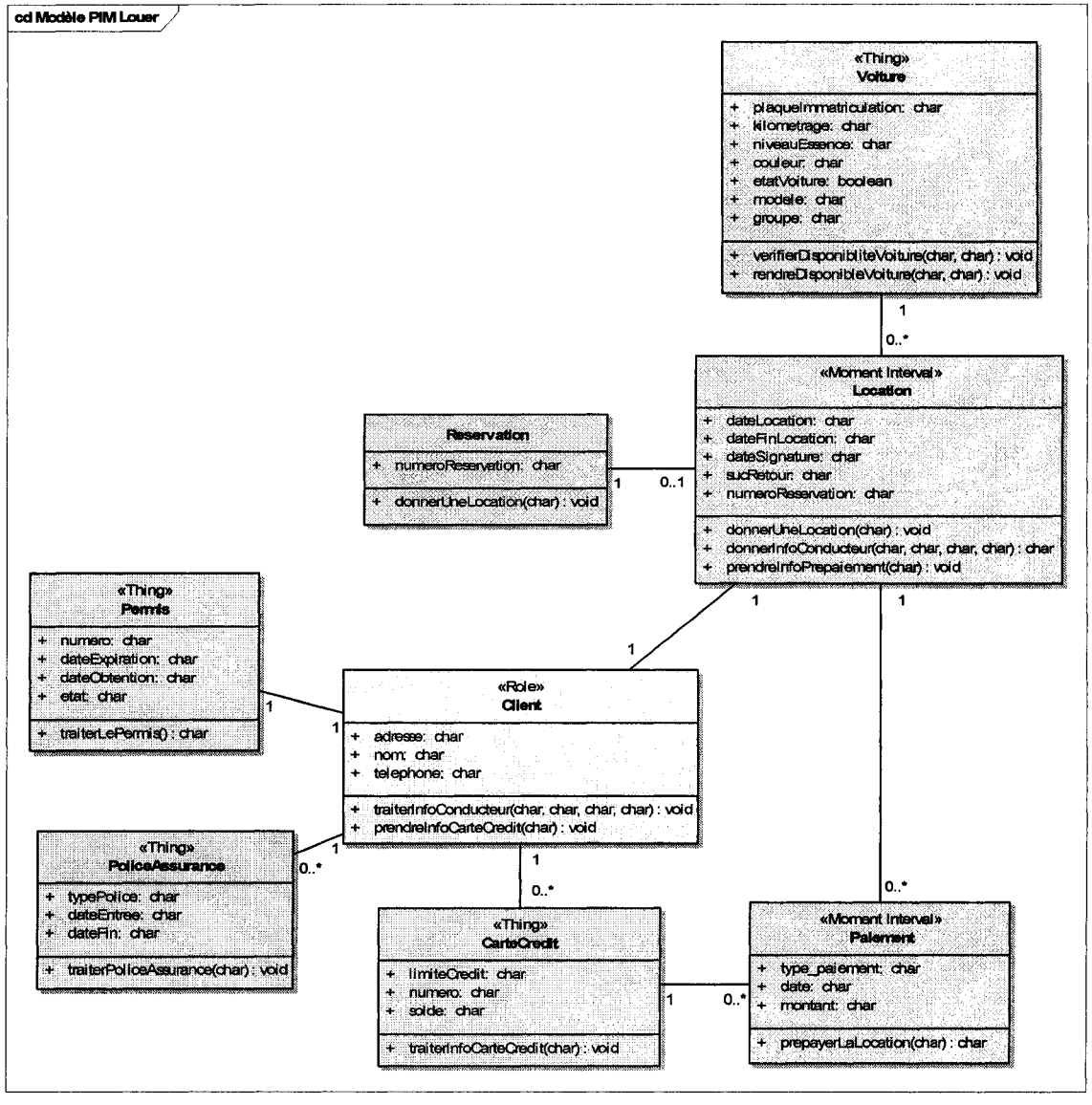


Figure 2.4 Modèle PIM "Louer"

La classe "Reservation" enclenche le processus "Location".

Dans le diagramme de séquence du modèle PIM "Louer", le préposé demande une location suite à sa réservation à l'objet "Reservation" qui, par délégation demande à ce dernier de le faire car cela va de sa compétence.

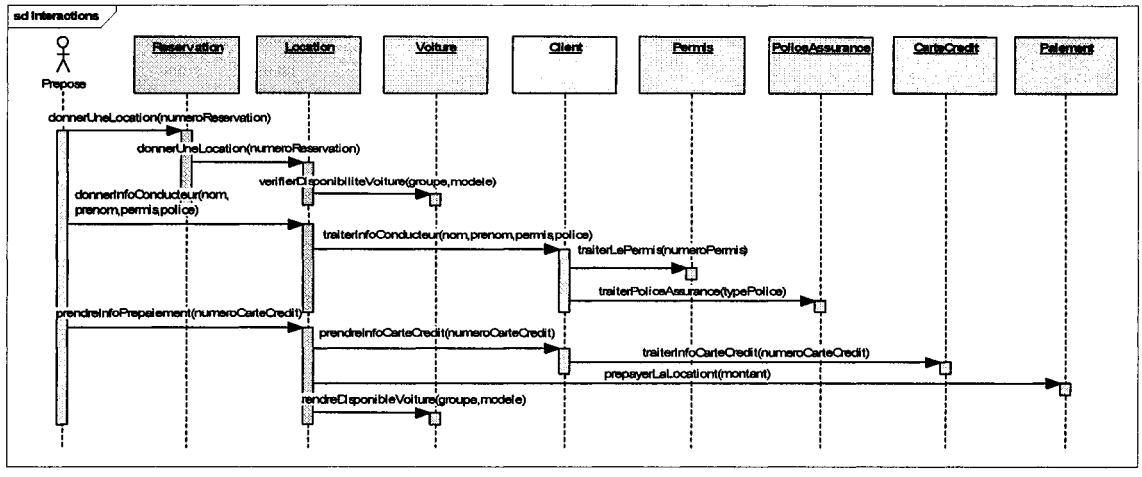


Figure 2.5 Diagramme de séquence du modèle PIM "Louer"

2.4 Traitement du cas d'utilisation "Allouer"

Ce cas d'utilisation, déclenché par le système lui-même à la fin d'une journée de travail, permet d'assigner des voitures aux réservations des journées suivantes.

Tableau V

Cas d'utilisation "Allouer"

Nom cas utilisation	**Allocation de voitures**	**Version 2.0**
Problème du domaine	**EU Rental – Réservation d'avance**	
Objectif	À la fin d'une journée de travail, nous avons besoin d'allouer des voitures aux réservations pour la prochaine journée afin de savoir: 1) quelle voiture doit être assignée ou a besoin d'un entretien ; 2) quelles voitures peuvent être rendues disponibles pour une location spontanée.	
Acteurs	**Système**	
Evénement déclencheur	La journée est terminée. Les réservations sont traitées une par une.	
Pré-conditions		
Post-conditions	Toutes les réservations de la prochaine journée ont des voitures assignées ou en demande aux autres succursales.	
Etape	**Actions**	**Réponses Système**
1.	Fin de journée; il y a une réservation d'un client pour le jour suivant.	
2.		Identifier le lot de voitures pouvant être louées.
3.		Identifier les réservations du lendemain.
4.		Assigner les voitures en utilisant les règles élémentaires.
5.		Réserver les voitures pour les locations spontanées.
6.		Réserver des voitures pour les mises à niveau.
		S'il y a encore des réservations sans voiture assignée alors,
7.		Assigner des voitures en utilisant la mise à niveau.
		S'il y a encore des réservations n'ayant pas de voiture assignée,
8.		Utiliser les règles exceptionnelles s'il y a encore des réservations sans voiture assignée.
9.		Utiliser la procédure in extremis.

Le "moment interval" dans ce cas d'utilisation est déclenché par le système, d'où l'utilisation de la classe "Allocation" comme étant le centre de toutes les activités. En tant que représentant du système dans ce cas d'utilisation, la classe "Allocation" a la responsabilité de l'assignation et la réservation des voitures selon les différentes règles mentionnées dans le cas d'utilisation.

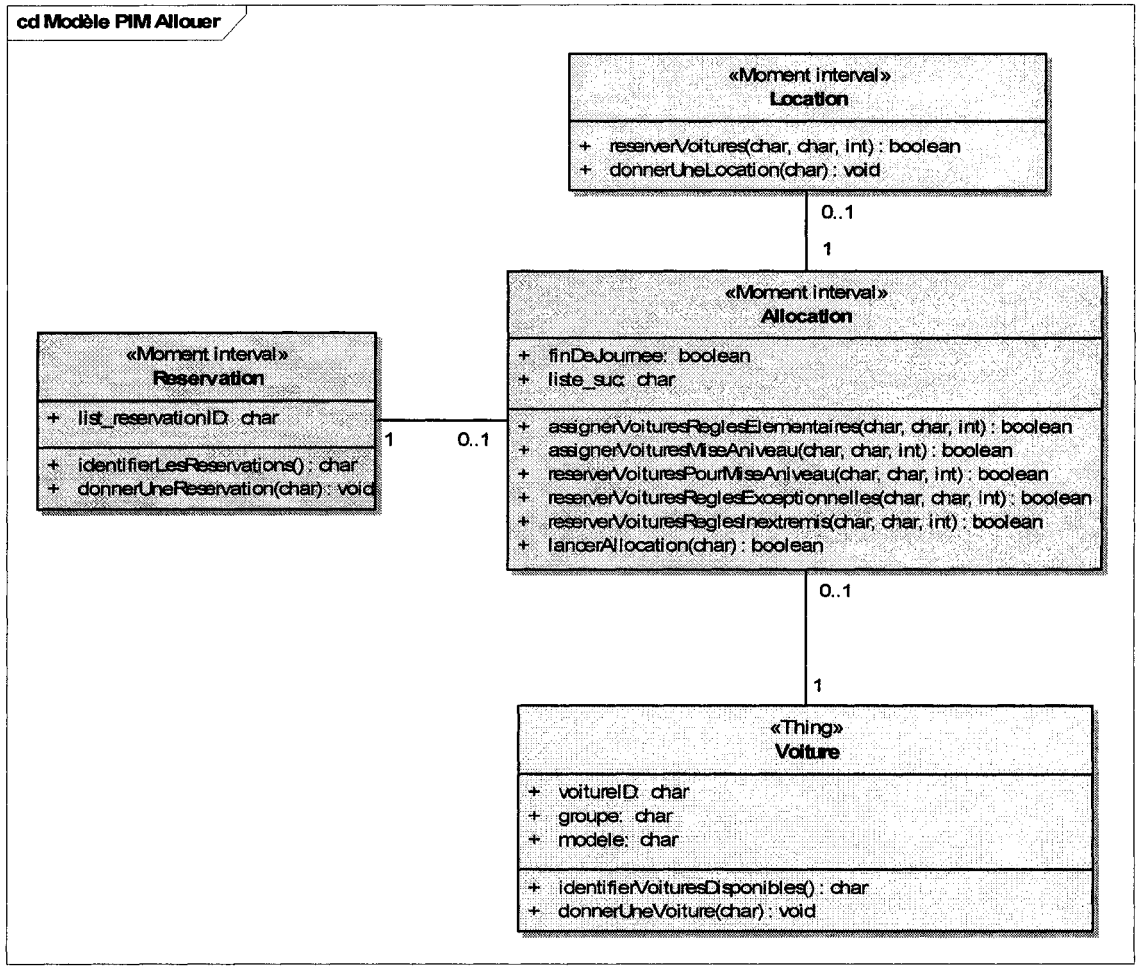


Figure 2.6 Modèle PIM "Allouer"

Ce modèle ne fait qu'établir un lien entre une réservation et une voiture. Il pourrait être tout simplement un groupe de méthodes de la classe d'analyse "Reservation" dans le modèle de l'application.

2.5 Traitement du cas d'utilisation "Retourner"

Le dernier cas d'utilisation concerne le retour d'une voiture après sa location.

Tableau VI

Cas d'utilisation "Retourner"

Nom du cas d'utilisation	**Retourner**	Version **1.0**
Problème domaine	**Retourner**	
But	A la fin de la période, le client retourne la voiture à la succursale.	
Acteurs	Le client à travers le préposé.	
Déclencheur	Le client se présente à la succursale de dépôt.	
Pré conditions	- Le client détient une voiture louée de EU Rent.	
Post-conditions	- Le contrat de location est payé et fermé, - La location est payée, - La voiture est retournée, - Les données du client sont mises à jour, - L'inventaire de la succursale est mis à jour.	
Etape	**Actions (externes)**	**Réponses du Système**
1.	Le client entre dans la succursale avec la voiture.	
2.		Le système vérifie la succursale de dépôt.
3.		Le système vérifie la date de retour du contrat de location.
4.		Le système vérifie le niveau de l'essence.
5.		Le système vérifie l'état de la voiture.
6.	Le client présente les garanties de sa carte de crédit.	
7.		Le système donne le coût de la location.
8.	Le client signe l'imprimé de débit de la carte de crédit.	
9.		Le système ferme le contrat.
10.		Le système met à jour l'inventaire.

Scénarios Alternatifs		
Etape	**Actions alternatives**	**Réponses alternatives du système**
6. 6.a.1.	Le client présente une carte de crédit différente pour le paiement.	
6.a.2.		Le système valide la carte de crédit.
6.a.3.		Le système vérifie la disponibilité des fonds ; aller au point 7.
6. 6.b.1.	Le client paie comptant; aller au point 9.	
10. 10.a.1.		La voiture est louée d'un concurrent, donc retour planifié chez le concurrent.
10.a.2.		Le système ferme le contrat et met à jour l'inventaire.

Le cas d'utilisation ci-dessus décrit les scénarios d'une transaction qui concerne le retour d'une voiture après sa location dans une succursale donnée.

Le modèle PIM "Retourner" est intéressant car il implique deux "moment interval" dans la même transaction selon ce découpage comme dans le cas de la location. Le "moment interval" "Paiement" est au fait ici la confirmation du paiement faite au moment de la réservation.

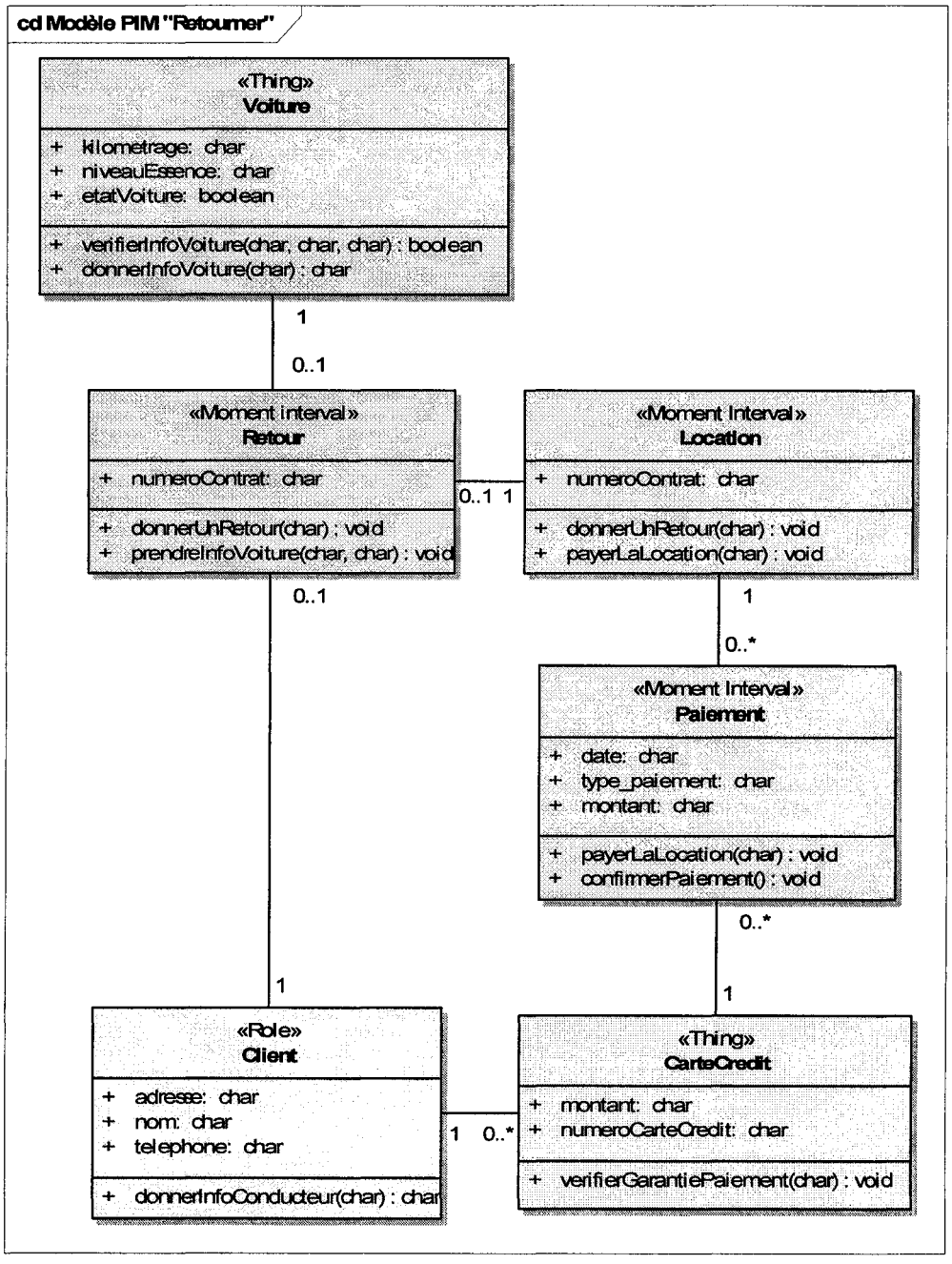


Figure 2.7 Modèle PIM "Retourner"

Dans le diagramme de séquence du modèle PIM "Retourner", tout comme pour la location, le préposé lance le processus en se servant du numéro de la location pour

enclencher le processus de retour. Ce dernier est créé par la location car en réalité, c'est la location qui entraîne le retour de la location.

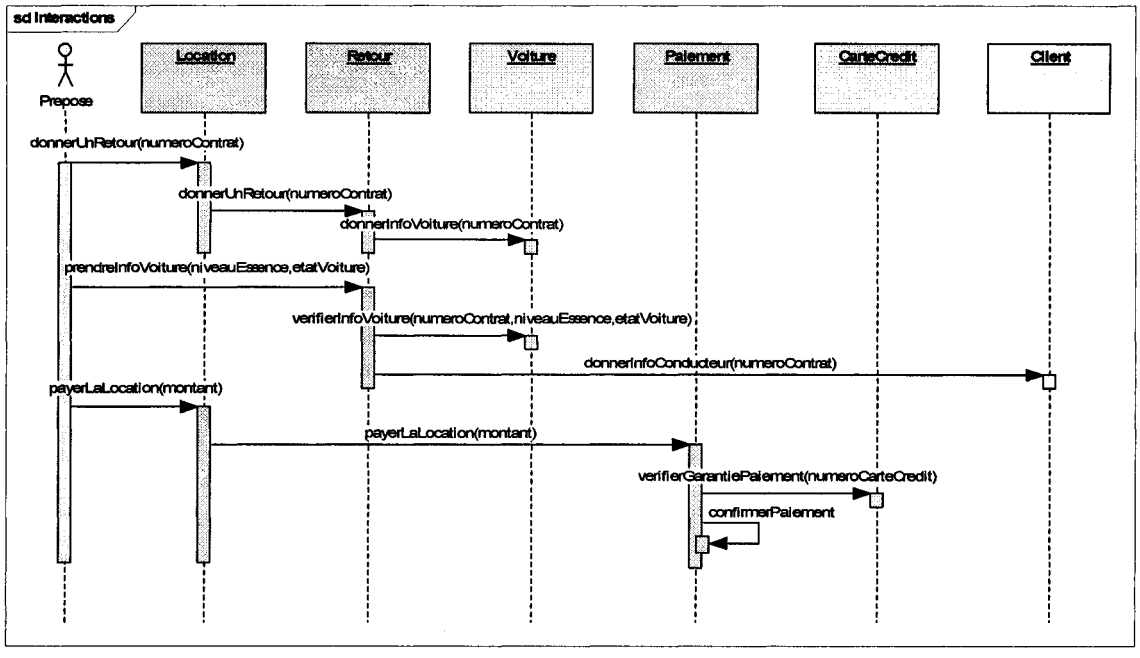


Figure 2.8 Diagramme de séquence du modèle PIM "Retourner"

2.6 Expérimentation des techniques vues

Dans cette section, nous allons expérimenter les techniques et les approches d'assemblage mentionnées. Toutefois, l'approche MDA n'est pas comprise dans cette expérimentation car elle ne spécifie pas l'assemblage des composants.

Nous verrons trois approches qui sont :

a. l'approche de la glu, qui, bien que traitant l'assemblage des modèles PSM, propose une expression servant à assembler des modèles PIM, ce qui peut donner les moyens d'établir une technique d'assemblage;

b. l'approche selon la théorie des couches. Mais dans cette théorie nous nous limiterons à l'utilisation des matrices et nous garderons l'approche de la glu pour l'assemblage car elle formalise la méthode proposée dans ce document;
c. l'approche ISC qui est orientée code mais dont la démarche d'assemblage de composants est intéressante.

Dans le cas de l'approche ISC, la démarche vers un assemblage est à explorer de plus près car elle peut servir à établir une chronologie entre les étapes d'assemblage de modèles.

2.6.1 Approche de la glu

L'approche de la glu est régie par l'expression de composition. Celle-ci comporte un ensemble de règles qui permettent de relier les modèles PIM à travers leurs éléments. Cette approche comprend trois types de règles qui sont les règles de correspondance, les règles de combinaison et les règles de remplacement.

Les règles de correspondance établissent une correspondance entre les différents éléments des modèles. La correspondance des paquetages doit être établie car dans l'application de location de voitures, il y a trois cas d'utilisation qui utilisent la classe "Voiture".

CorrespondPackage [Reserver, Louer, Retourner]

Cette instruction établit la correspondance entre les paquetages "Reserver", "Louer", et "Retourner" dans l'hypothèse où le découpage est ainsi fait.

La correspondance des classes est établit entre les différentes classes de "Client".

CorrespondClasses[Reserver.Client, Louer.Client, Retourner.Client]

Nous pouvons faire de même pour l'entité "Voiture" ce qui nous donne le résultat qui suit.

CorrespondClasses[Louer.Voiture, Retourner.Voiture]

La correspondance de chaque opération ou méthode des classes doit être traitée. Par voie de conséquence, nous pouvons nous retrouver avec une expression assez longue.

Dans notre cas, la classe "Client" ne présente aucune correspondance entre les méthodes se trouvant dans les modèles PIM. La correspondance des opérations n'est donc pas traitée. Par exemple, dans le modèle PIM "Reserver", le client a pour méthode "traiterInfoClient" et qui ne correspond pas à "traiterInfoConducteur" se trouvant dans le PIM "Louer".

Le nombre de correspondances des opérations dépend du nombre d'opérations ou de méthodes qu'une classe peut avoir.

Pour finir, nous présentons la correspondance des attributs qui est plus longue si l'on considère qu'une classe peut comporter de nombreux attributs.

CorrespondAttributs[Reserver.Client.prenom, Louer.Client.prenom, Retourner.Client.prenom]

CorrespondAttributs[Reserver.Client.nom, Louer.Client.nom, Retourner.Client.nom]

CorrespondAttributs[Reserver.Client.telephone, Louer.Client.telephone, Retourner.Client.telephone]

CorrespondAttributs[Louer.Client.adresse, Retourner.Client.adresse]

Ce sont là les correspondances entre les éléments qui doivent être assemblés dans cette application.

D'autre part, nous traitons les règles de combinaison dont la démarche est de retenir un seul élément parmi ceux qui sont mis en correspondance. Toutefois, dans le cadre des opérations, l'intégration est caractérisée par l'ordre de leur exécution.

Pour la combinaison des paquetages, nous voudrions combiner les classes qui se trouvent dans les paquetages "Reserver", "Louer" et "Retourner". L'ordre de priorité des paquetages fait que la combinaison se fera dans le paquetage "Réserver".

JoinPackages[Reserver,Louer,Retourner]

Pour la combinaison des classes, la logique exprimée dans la combinaison des paquetages reste valable également. La définition de la priorité au niveau des classes prévaut sur la priorité des paquetages.

JoinClasses[Reserver.Client,Louer.Client,Retourner.Client]

La combinaison des opérations est en fait l'ordre d'exécution des opérations dans chacun des composants. Ainsi, une nouvelle opération, appelée "ControleOperation", doit être réalisée afin de coordonner l'ordre d'exécution des opérations. Toutefois, dans notre projet, les méthodes n'ont pas trouvé de correspondance. Si par exemple une correspondance avait été trouvée entre traiterInfoClient et traiterInfoConducteur, on aurait l'expression :

JoinOperation[ControleOperation, Reserver.Client.traiterInfoClient,
Louer.Client.traiterInfoConducteur]

Nous reprendrions cette opération de jointure pour chaque méthode des classes que compte le composant.

Selon cet exemple et selon l'ordre chronologique des événements, il faudrait d'abord avoir une réservation ("Reserver") avant de se présenter pour une location ("Louer").

Dans le cadre de la combinaison des attributs, un seul doit être représenté pour les éléments qui sont en correspondance.

JoinAttributs[Reserver.Client.prenom,Louer.Client.prenom,
Retourner.Client.prenom]
JoinAttributs[Reserver.Client.nom,Louer.Client.nom,Retourner.Client.nom]
JoinAttributs[Louer.Client.adresse,Retourner.Client.adresse]

Concernant les règles de remplacement, elles établissent les remplacements qui devront être faits dans une application. Les remplacements effectués seront par la suite sauvegardés dans un autre modèle de modification.

Suite à l'application de la règle de la glu, le modèle de l'application est le diagramme de classes de la figure 2.9. Ce diagramme est le modèle final de l'application.

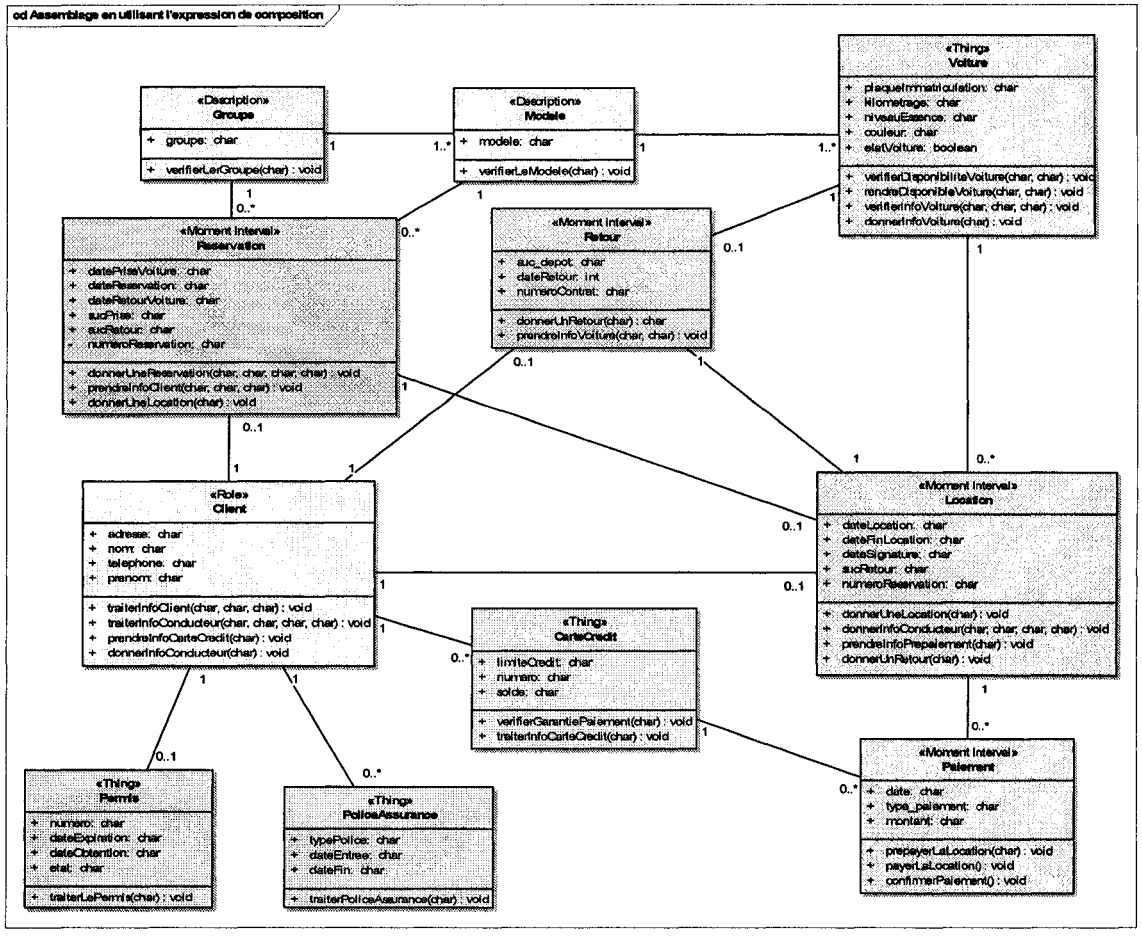


Figure 2.9 Assemblage en utilisant l'expression de composition

L'application de la technique de la glu est précise et efficace en ce sens qu'elle traite élément par élément chaque cas d'utilisation. Cela demande que les cas d'utilisation aient été bien documentés.

Par ailleurs, elle peut être laborieuse pour une application d'envergure en raison du nombre important d'éléments qu'il peut y avoir à traiter dans une classe.

Les différentes règles étudiées conviennent bien à une solution d'assemblage car elles permettent de regrouper les éléments de différents composants pour n'avoir qu'un composant final.

2.6.2 Assemblage par les couches des cas d'utilisation

L'assemblage par couche spécifique et non spécifique selon les techniques de l'orienté aspect, est une approche qui permet de faire le regroupement de tous les éléments ou entités identiques dans les cas d'utilisation d'une application. Une fois ce regroupement obtenu, on obtient l'assemblage des cas d'utilisation.

Cependant, avant d'effectuer un quelconque regroupement, il faut identifier les différentes couches des cas d'utilisation, d'où l'utilité de la matrice dont il a été question dans le premier chapitre. Cette matrice, qui est un tableau des éléments d'une application, permet de mettre en évidence les relations qu'il y a entre ces éléments.
Pour cette application, voici la matrice obtenue après l'analyse des cas d'utilisation.

Tableau VII

Identification des couches spécifiques et non spécifiques

	Cas d'utilisation		
Couches spécifiques et non spécifiques	**Réserver**	**Louer**	**Retourner**
Reservation	Reservation1	Reservation2	–
Location	–	Location1	Location2
Retour	–	–	Retour1
Voiture	–	Voiture1	Voiture2
Client	Client1	Client2	Client3
Paiement	–	Paiement1	Paiement2
CarteCredit	–	CarteCredit1	CarteCredit2
Permis	–	Permis1	–
PoliceAssurance	–	PoliceAssurance1	–
Modèle	Modèle1		–
Groupe	Groupe1		–

Chaque fois qu'un élément se retrouve dans un autre cas d'utilisation, nous l'identifions avec un numéro, et nous le plaçons dans la colonne des cas d'utilisation respectifs. D'autre part, l'élément "Retour" ne se retrouve que dans le cas d'utilisation "Retourner". Ainsi, nous avons trois couches qui correspondent aux trois cas d'utilisation "Réserver"," Louer" et "Retourner".

Pour obtenir un assemblage par cette technique, il suffit de fusionner les lignes, donc les couches. Par exemple, les éléments "Client1 et "Client3", une fois fusionnés donneront l'entité "Client". Dans la figure ci-dessous, nous avons simulé un regroupement pour deux cas d'utilisation (Réserver et Louer) pour des raisons d'encombrement de la figure. Le diagramme illustre ce qui a été dit plus tôt sur le groupement, et cela, sur chacun des éléments des cas d'utilisation impliqués dans l'application.

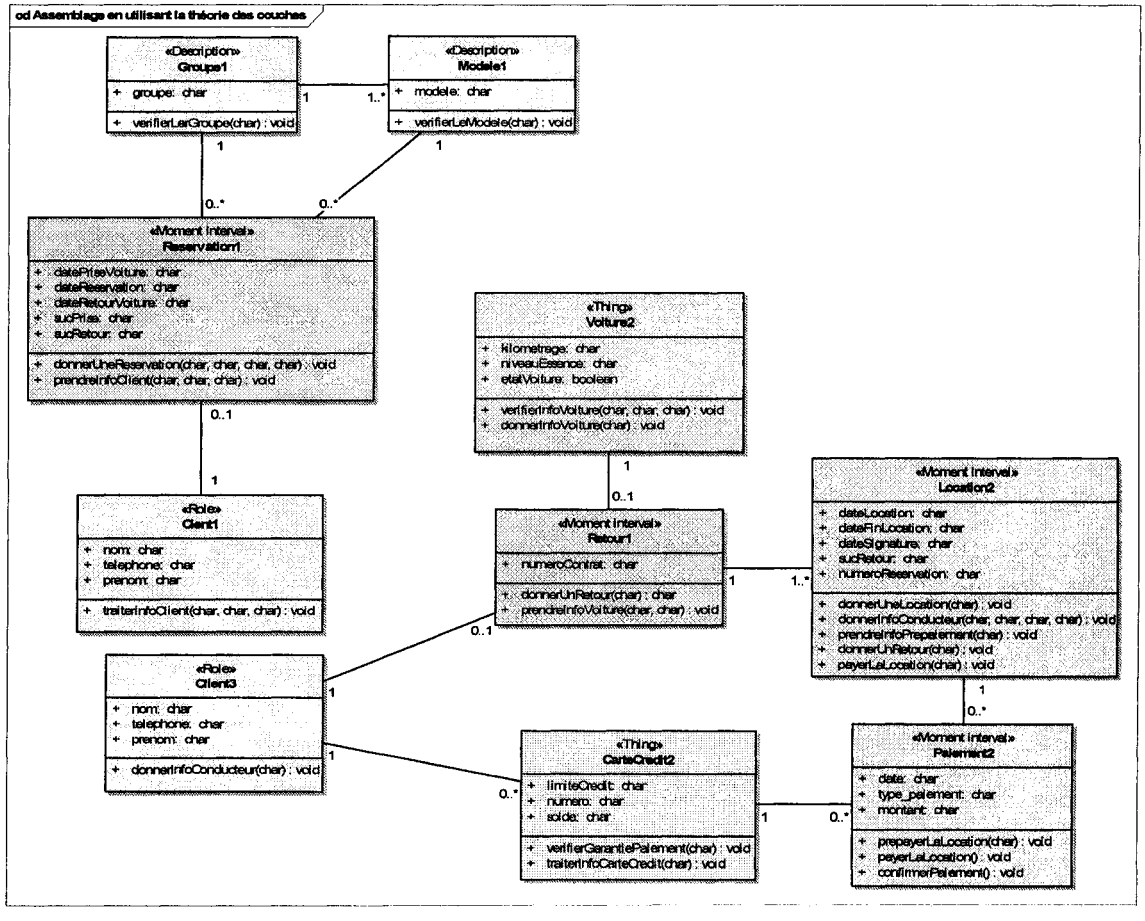


Figure 2.10 Simulation de l'assemblage par la théorie des couches

La théorie des couches est efficace dans le regroupement des entités. Nous pouvons aisément distinguer les entités qui sont impliquées dans d'autres cas d'utilisation.

Cette approche permet également de reconnaître le composant dans son intégralité. Chaque cas d'utilisation détient une partie d'un composant (couche) et les tableaux ont permis de regrouper les couches en une seule afin d'avoir le composant.

D'autre part, la théorie des couches est essentielle dans l'identification des composants et leur regroupement.

2.6.3 Composition chirurgicale de composant (ISC)

La composition ISC est un assemblage qui nécessite des objets de niveau code car elle se sert de certaines subtilités des concepts de la programmation pour établir un environnement d'assemblage. Par exemple, une classe "Employe" qui hérite d'une autre "Client" permet de dire que la classe "Employe" possède un port (*hook*) qui la lie à la classe "Client" pour un assemblage. De même, une classe ayant une méthode ou une fonction nécessitant un paramètre, possède un port qui la lie à la classe fournissant le paramètre.

Ensuite des fonctions d'association (*binding operations*) s'occupent d'assembler en adaptant, en renommant ou en modifiant les composants. L'assemblage se fait ainsi lors de la compilation.

Cet assemblage n'est pas envisageable dans notre cas car il s'agit de composants de niveau modèle et non de niveau code. Toutefois, nous pourrions appliquer cette approche en l'adaptant à des composants modèles tels que les cas d'utilisation. Pour que cela fonctionne, les éléments redondants du cas d'utilisation sont considérés comme étant les éléments pouvant être adaptés, renommés ou modifiés (*hooks*). Ces éléments sont ceux qui appartiennent au domaine du problème. Le langage utilisé pour l'assemblage est l'UML.

Pour assembler les composants selon l'ISC, il faut un composeur qui adaptera, renommera ou modifiera les éléments des modèles en conséquence afin d'obtenir une composition ISC. Nous pouvons ainsi utiliser une classe de type interface de qui héritera d'autres classes (hook). Lorsque deux classes héritent d'une interface elles deviennent de même type que l'interface et forment ainsi un seul élément à travers cette interface [23]. Les éléments regroupés ont le suffixe ISC dans la figure ci-dessous qui donne une illustration de l'assemblage obtenu en utilisant une adaptation de l'approche ISC.

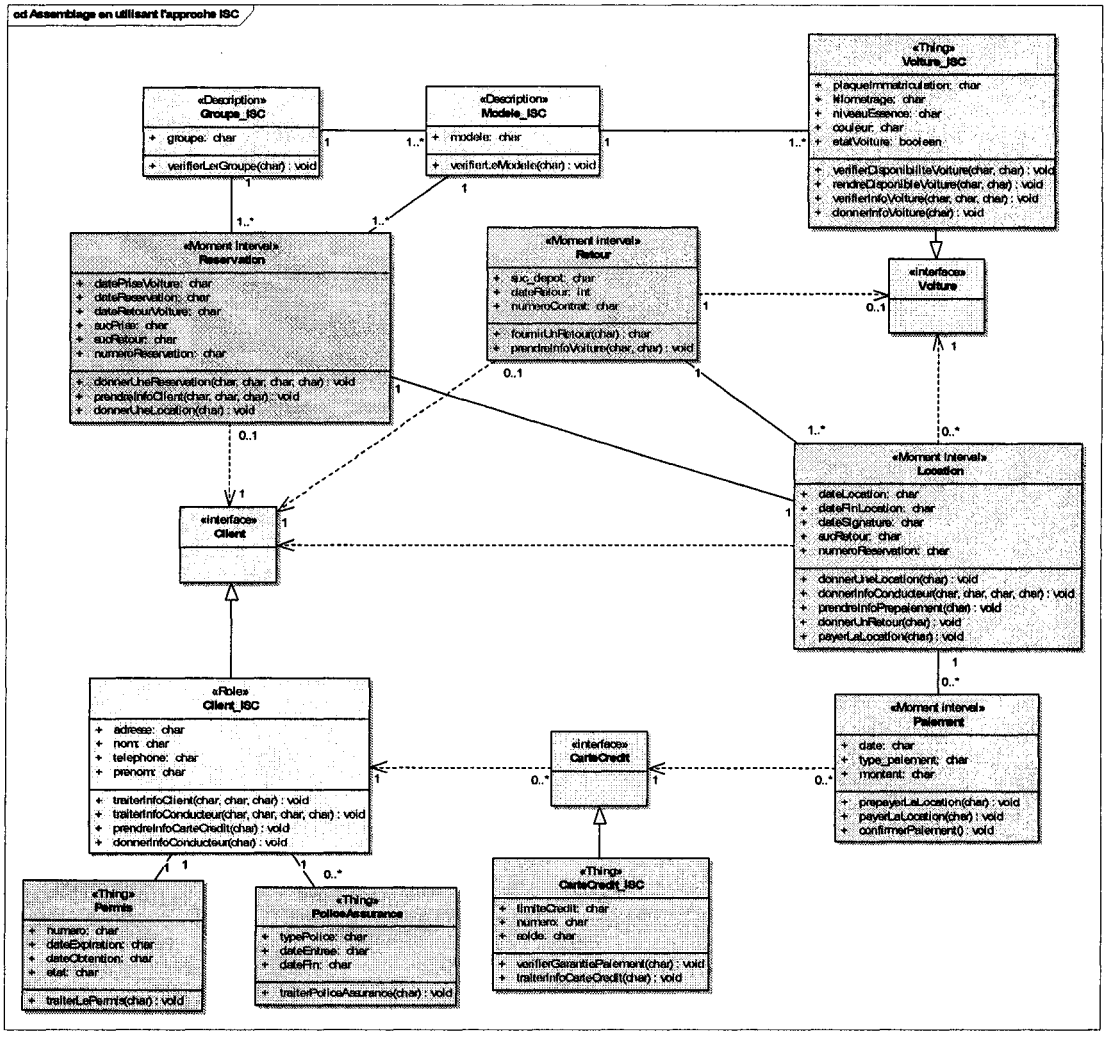


Figure 2.11 Assemblage par l'adaptation de l'approche ISC

Les classes renommées sont les classes ayant des ports, c'est-à-dire des classes présentant des caractéristiques communes, qui ont été modifiées et regroupées, d'où le suffixe ISC. Puisque le nom a été changé, cette modification entraîne l'introduction d'une interface qui garde l'identité de l'entité que nous voulons affecter à l'ensemble des éléments regroupés. Nous partons du principe que le regroupement de classes ayant des ports et l'introduction des interfaces sont faits par le composeur dont le mécanisme n'est pas défini ici.

L'assemblage est obtenu par une adaptation de l'ISC afin de faire une démonstration de son efficacité dans le cadre de cette étude. Cette efficacité reste subjective car il existe plusieurs manières d'adapter cette approche afin d'obtenir un assemblage. Cependant, la méthode impliquée dans cette technique est intéressante car elle décrit bien une démarche d'assemblage, à savoir : le modèle de composants qu'il faut définir, la technique de composition qui sera utilisée pour l'assemblage des composants et le langage de composition utilisé pendant l'assemblage des composants. Ces trois principes sont essentiels dans la méthode d'assemblage de modèles.

2.7 Conclusion

Les techniques recensées présentent chacune une particularité dans le mode d'assemblage. Cet aspect ne peut être que profitable car il permet d'établir dans cette diversité, une méthode d'assemblage de modèles. L'assemblage par l'expression de composition a permis d'obtenir un modèle final à travers l'expression de composition en établissant les liens entre les différents éléments des cas d'utilisation. La théorie de couches a également donné un modèle final des cas d'utilisation par la fusion des couches. Mais nous retenons l'efficacité de l'utilisation de la matrice permettant le regroupement des éléments de modèles. L'ISC est orienté code et a été adapté afin de voir comment elle pouvait se comporter dans l'assemblage de modèles. Bien que subjectif, le modèle résultant est moins évident à obtenir en respectant les règles de cette approche.

CHAPITRE 3

MÉTHODE PROPOSÉE

Nous venons de voir des techniques ou approches d'assemblage dans les chapitres précédents. Nous avons également expérimenté ces techniques et vu comment elles fonctionnaient. Dans ce chapitre, nous introduisons notre proposition concernant l'assemblage de composants. Cette proposition se base sur les travaux que nous avons effectués sur les autres techniques : 1) l'approche de la glu, 2) la théorie des couches qui donnera le moyen de déterminer avec efficacité l'interaction entre composants et leurs interfaces, 3) l'approche ISC qui permettra d'établir un lien entre les différentes étapes de la méthode d'assemblage.

La méthode proposée sera ensuite expérimentée à travers le même exemple de l'application de location de voiture dans le prochain chapitre.

3.1 Principes de la méthode d'assemblage

L'approche ISC [6] (section 1.1.5) induit les principes pour l'assemblage de modèles PIM :

a. avoir un modèle de composant : permet d'établir l'identification des composants;
b. se munir d'une technique de composition : implique la préparation des composants pour être assemblés;
c. avoir un langage de composition : spécifie l'opérateur d'assemblage qui permet le regroupement des composants.

3.1.1 Modèle de composant : identification des composants

Le premier principe dans le processus d'assemblage de modèles PIM, consiste à avoir un modèle de composant. Ce principe implique la première étape de la méthode d'assemblage proposée, soit l'identification des composants dans les modèles à composer. Dans cette section, nous introduisons la catégorisation des composants en composants processus et en composants du domaine, selon que les entités des modèles soient dynamiques ou statiques.

Dans une transaction d'affaire, il y a plusieurs éléments qui contribuent à la solution du problème pour lequel cette transaction est active. Il y a des éléments dynamiques qui, à travers plusieurs calculs et opérations, permettent de solutionner un problème. Il y a également des éléments statiques qui sont des éléments réels qui font partie du problème à résoudre. Les éléments statiques sont utilisés par les éléments dynamiques afin d'apporter une solution à un problème. Par exemple, l'opération "Retrait d'argent" est un processus ou un cas d'utilisation dans une application bancaire. Cette opération utilisera un guichet automatique, qui est un objet du domaine, comme un outil lui permettant d'accomplir la requête du client. Ces deux types d'éléments englobent à eux seuls le concept de composant que nous retrouvons généralement dans un problème à solutionner [24, 25].

Dans cette étude, les éléments dynamiques seront appelés les composants processus et les éléments statiques, les composants du domaine. Les termes utilisés ne sont pas nouveaux et ont déjà été employés dans d'autres études [12, 21].

Tout d'abord le composant processus est le composant qui englobe les processus se déroulant dans une application. En terminologie des archétypes (section 1.3.3), c'est le "moment interval" de l'application. Dans le patron MVC (section 1.3.2), il représente le contrôleur des actions entre les différentes entités.

Un composant processus est une suite d'opérations initiée par une entité tel qu'un acteur dans le sens d'un cas d'utilisation. C'est une fonctionnalité existante d'une application ou d'un système. Il est cependant indépendant du contexte de son utilisation.

Le composant du domaine, quant à lui, est un élément qui fait partie du problème du domaine, et n'est pas spécifique à un modèle PIM. Nous l'avons évoqué dans la présentation des couches spécifiques et non spécifiques plus tôt dans ce document (section 1.2.2).

Un composant du domaine est une entité statique spécifiquement liée au domaine du problème à résoudre. Par exemple un "guichet automatique" est spécifique au domaine bancaire, et est un composant du domaine dans une application de type bancaire. Ces composants ne communiquent généralement pas entre eux directement car ils sont utilisés par les composants processus.

Bien que les composants ainsi que leurs différents éléments aient été définis, et étant donné que le niveau de granularité peut influer sur la complexité de la réutilisation ou de l'assemblage d'un composant, il faut donc définir ce niveau de granularité. Selon la nature des composants dont nous avons parlé, c'est-à-dire un processus et un objet du domaine, qui, tous deux, se retrouvent généralement dans un modèle PIM, le niveau de granularité du composant sera plus fin que celui du modèle PIM. En d'autres termes, les modèles PIM se composent la plupart du temps d'un ou plusieurs composants processus et du domaine. La granularité d'un composant sera donc plus fine que celle d'un modèle PIM.

Selon les spécifications d'UML 2 [17], un composant est constitué de plusieurs éléments qui lui permettent de fournir des services et de communiquer avec son environnement immédiat. L'interface est le service que peut fournir ou recevoir un

composant vis-à-vis d'un autre composant ou d'un système. Ces interfaces sont de type fourni ou de type requis.

D'autre part, la nomination des interfaces est importante car cela peut permettre d'identifier les composants qui peuvent être assemblés. C'est pourquoi le nom d'une interface sera le nom du composant qui fournit ses services à d'autres composants précédé de la lettre "I". En d'autres termes, si un composant "Location" fournit des services, le nom de son interface sera "ILocation". Si ce même composant requiert un service d'un composant tel que "Retour", le nom de cette deuxième interface sera "IRetour".

Pour le cas du composant du domaine, seule une interface aura la charge de la connexion avec les autres composants de l'application. Par exemple pour le composant du domaine "Voiture", l'interface sera nommée "IVoiture". Cette façon de procéder permet de minimiser les erreurs lors de la nomination des interfaces et lors de l'assemblage des composants.

Un autre élément du composant est le port. Dans les spécifications de l'OMG sur l'UML 2 [4], cet élément a été introduit et est le moyen par lequel le composant peut communiquer avec son environnement immédiat. C'est une propriété du classificateur auquel il est attaché.

Le port supporte deux types d'interfaces qui sont ceux qui ont été présentés plus tôt. Il peut y avoir une certaine confusion au niveau des ports des composants, car un composant peut avoir plusieurs ports. À un moment donné, il sera difficile de s'y retrouver, surtout si nous travaillons dans un projet qui regroupe plusieurs composants. C'est dans ce sens que, tout comme avec les interfaces, deux appellations sont introduites, à savoir le "port de synchronisation" et le "port composant", qui permettent d'identifier plus facilement les ports dont il est question.

Le port de synchronisation est une classe d'analyse qui est liée au "moment interval" de l'application. C'est par ce port que les "moment interval" communiquent entre eux (voir figure ci-dessous). Par exemple, le composant "Location" aura une interface de type fourni "ILocation", qui est l'interface via laquelle les autres composants qui requièrent ses services communiquent avec lui. Le port auquel est associée cette interface est un port de synchronisation.

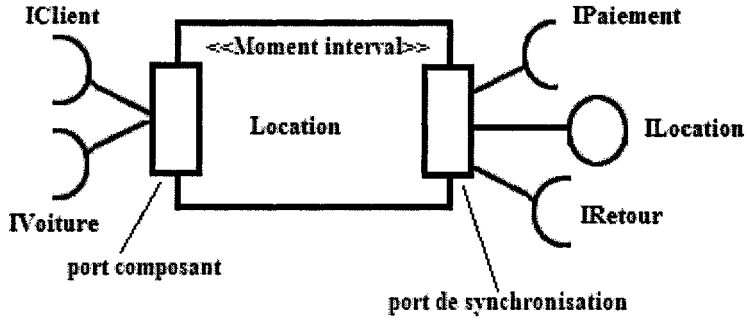


Figure 3.1 Port de synchronisation et port composant.

Le port composant est une classe d'analyse qui lie le composant processus au composant du domaine qu'il utilise. Il n'est autre qu'un port simple au sens de l'OMG sur la spécification d'UML 2 permettant de lier deux composants. L'appellation de port composant indique simplement de quel port il s'agit au sujet des articulations d'un composant. Le port composant peut supporter n'importe quelle interface, que celle-ci soit de type fourni ou requis [17].

Dans la figure 3.1, l'interface "IClient" est associée à un port composant. Ces notions aident tout simplement à s'orienter dans un modèle qui pourrait être complexe, et à identifier les différents points de jonction du composant.

Le composant processus peut avoir deux types de ports, qui sont le port de synchronisation et le port composant. Le port composant lui sert à communiquer avec

les composants du domaine. Les composants du domaine quant à eux ne possèdent en général que des ports composants. Cependant, bien qu'ils soient statiques, ils peuvent à l'occasion déclencher un composant processus.

Le connecteur est un autre élément du composant qui permet de lier deux composants entre eux, que ceux-ci soient des composants processus ou du domaine. L'association entre les composants se fait à travers les interfaces. Cette association est faite sur la base de l'identification des services fournis d'un composant et des composants qui utilisent ces services.

L'une des difficultés du développement par composants est associée à la notion de connecteur qui va lier les composants. Par exemple de quelle manière le connecteur devra lier deux composants en tenant compte des contraintes telles que la performance, la sécurité et bien d'autres lors de l'assemblage? Le connecteur est donc un élément clé dans l'assemblage des composants.

3.1.2 La technique de composition

Dans cette section, nous introduisons le deuxième principe qui porte sur la technique à utiliser dans l'approche d'assemblage proposée. Cette technique d'assemblage se déroule en deux phases qui sont dans un premier temps le regroupement des composants qui se retrouvent dans plusieurs modèles PIM de l'application, et dans un deuxième temps, la détermination des interfaces d'interactions entre les composants concernés et l'attribution des ports à ces derniers.

La technique d'assemblage se résume donc à l'identification des composants, à leur regroupement via le langage de composition qui sera vu dans la section suivante, à l'attribution des ports et ensuite à leur assemblage.

3.1.3 Le langage de composition : l'opérateur d'assemblage

Le dernier principe de la méthode proposée est le langage de composition permettant de regrouper les composants définis plus tôt. Ce langage est l'expression de composition dont il a été question dans le chapitre précédent [11] (section 2.2.1). Il ne s'agit pas d'un langage évolué tels que JAVA ou C++, mais dès lors que cette expression contient des instructions, des règles et une syntaxe, celle-ci peut être considérée comme un langage. De plus, les langages qui ont été mentionnés sont des langages de programmation, et il s'agit ici d'assemblage.

Il faut noter que ce troisième principe ne contient pas une étape quelconque du processus d'assemblage car c'est un outil qui est utilisé dans une étape de la démarche d'assemblage, à savoir celle du regroupement des composants, qui a été décrite dans la section précédente.

3.2 La méthode proposée

Après avoir défini les principes de l'approche proposée, nous pouvons introduire les étapes qui conduisent à l'assemblage des composants. Les principes évoqués plus tôt bâtissent un lien entre les différentes étapes de la méthode (voir figure 3.2). Cette méthode, qui apporte une formalisation dans la méthode proposée, permet de retracer ses points forts et ses points faibles et laisse entrevoir les possibilités de son amélioration.

Outre les étapes déjà présentées, cette section relate l'ensemble des étapes qui doivent mener à un assemblage de modèles PIM. Ces derniers sont respectivement représentés dans le deuxième chapitre par les figures 2.2, 2.4, 2.8.

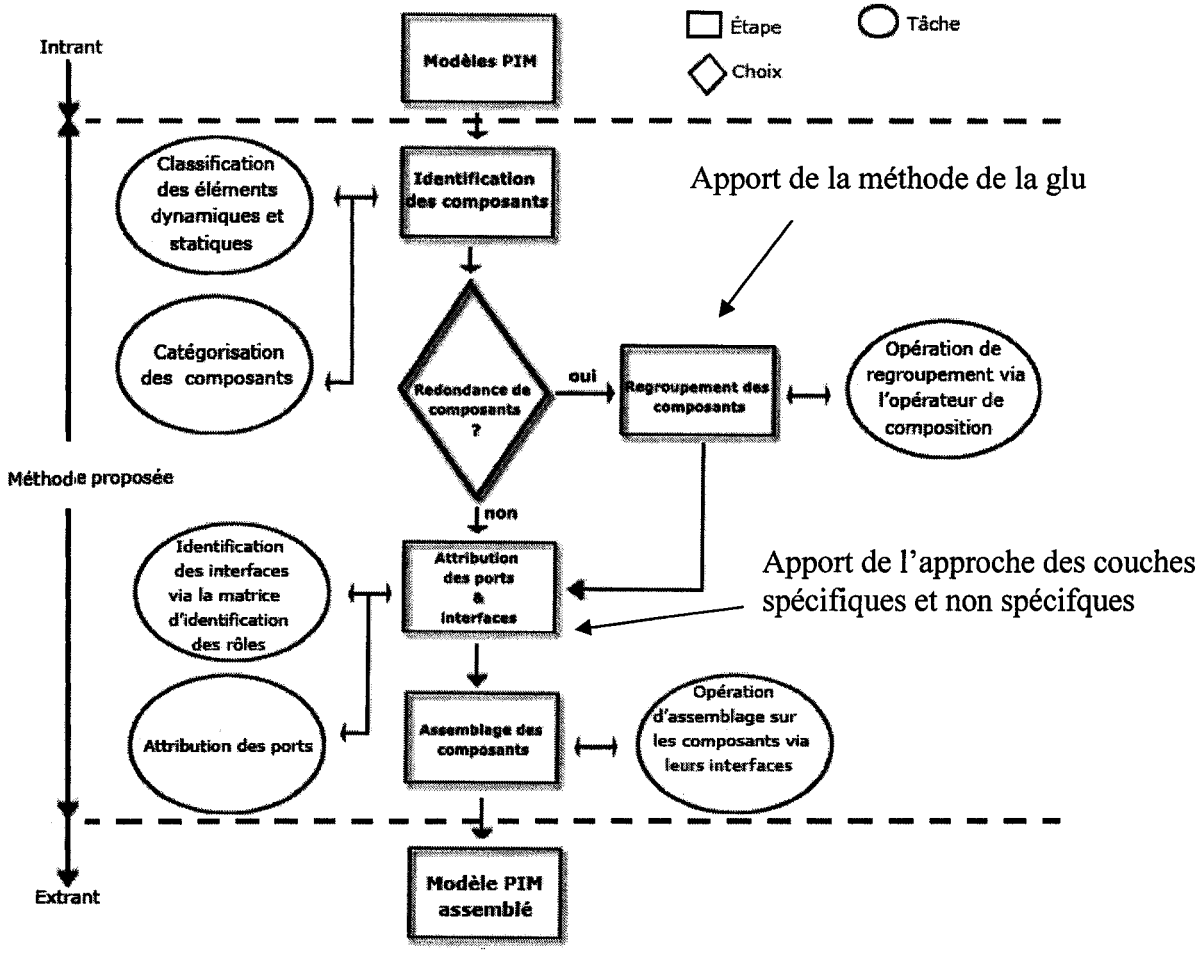


Figure 3.2 Organigramme de la méthode d'assemblage proposée

La figure 3.2 représente l'ensemble des activités ayant lieu dans la démarche d'assemblage proposée (zone de la méthode proposée). Dans cette figure, les formes rectangulaires représentent les différentes étapes de la démarche, les formes ovales représentent les tâches accomplies lors de ces étapes et le losange représente le choix à faire lorsqu'il y a une redondance ou un éparpillement de composants dans plusieurs modèles PIM qui doivent être assemblés. Si tel est le cas, il faudra procéder à un regroupement de composants, sinon passer directement à l'attribution des ports.

Il faut remarquer que le point de départ de la démarche est d'avoir un ensemble de modèles PIM à assembler (les intrants de la figure 3.2). Dans ce projet, l'existence de ces modèles PIM est une hypothèse qui est posée car ils ont été conçus par nous-mêmes pour l'expérimentation. La zone délimitée par les lignes en pointillés représente la zone d'étude de ce projet, et la dernière étape représente le modèle final (l'extrant) obtenu suite à la méthode d'assemblage.

L'apport des approches que nous avons vues se situe au niveau du regroupement pour la méthode de la glu (section 1.2.1)(voir figure 3.2), de l'attribution des ports et interfaces pour l'approche des couches spécifiques et non spécifiques (section 1.2.2). Le concept ISC a pour sa part mené aux principes de la méthode (section1.2.3).

3.2.1 Première étape : Identification des composants

L'identification des composants se fait en les classifiant selon qu'ils soient statiques (composants du domaine), ou dynamiques (composants processus). Pour ce faire, nous répertorions tous les modèles PIM à assembler dans un tableau et nous écrivons pour chacun d'eux, les éléments statiques et dynamiques qu'ils contiennent. Les éléments statiques redondants sont les composants du domaine et les éléments dynamiques deviennent les composants processus.

3.2.2 Deuxième étape : Regroupement des composants

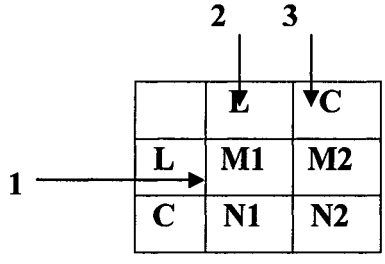
Dans l'étape d'identification des composants, il peut arriver d'avoir une redondance de composants dans les modèles PIM. Ceci fait qu'il peut y avoir plusieurs fois le même composant, même si ces composants ne possèdent pas forcément les mêmes attributs ou méthodes. Il faut donc procéder au regroupement de ces composants. Ce regroupement s'obtient en utilisant l'opérateur d'assemblage sur les classes des composants.

3.2.3 Troisième étape : Attribution des ports et interfaces

Une fois que les composants ont été identifiés et regroupés, nous pouvons passer à l'attribution des ports et des interfaces aux composants. Pour cela, nous utilisons la matrice d'attribution des rôles. Cette matrice illustre les interactions entre composants en indiquant comment un composant communique avec un autre.

Tableau VIII

Matrice d'identification de rôles



Le sens de lecture de cette matrice est celui indiqué par la flèche 1, le sens horizontal. C'est ainsi que le composant L (dans le sens de la flèche 1) a une interaction "M1" avec le composant L (dans le sens de la flèche 2), donc lui-même, (croisement de la flèche 1 et de la flèche 2) et a une interaction M2 avec le composant C (dans le sens de la flèche 3).

Les interfaces des composants contiennent les méthodes dont se sert le composant pour communiquer avec son environnement de travail. Les méthodes utilisées à l'interne ne sont pas exposées au niveau des interfaces.

Le nom d'une interface est le nom du composant qui fournit les services à un autre composant. Ce dernier a une interface de même nom mais de type requis. Nous le verrons dans les sections du prochain chapitre.

3.2.4 Quatrième étape : assemblage des composants

L'opération d'assemblage des composants a lieu après que toutes les étapes précédentes aient été respectées. Lors de l'assemblage, les interfaces d'un composant sont associées en fonction des services fournis et requis tels qu'établis à l'étape précédente. Cet assemblage valide donc les résultats de l'étape précédente.

3.3 Conclusion

La méthode proposée se base sur les trois principes fondamentaux issus de l'approche ISC, qui sont d'avoir un modèle de composant, une technique de composition et un langage de composition. Ces principes induisent quatre étapes pour la méthode d'assemblage proposée et permettent par la même occasion d'établir une chronologie de ces étapes. C'est ainsi que le modèle de composant regroupe l'identification des composants qui est la première étape. Ensuite la technique de composition inclut la préparation des composants, c'est-à-dire le regroupement, la détermination des interfaces et l'attribution des ports, qui représentent les deuxième et troisième étapes. La dernière étape consiste à assembler les composants.

Dans le chapitre qui suit, nous ferons une expérimentation de l'application de la méthode d'assemblage proposée.

CHAPITRE 4

EXPÉRIMENTATION DE LA MÉTHODE PROPOSÉE

Ce chapitre a pour objectif d'appliquer la méthode proposée dans le chapitre précédent et de valider son efficacité et la qualité du modèle PIM final obtenu avec un modèle PIM de l'application conçu par toute autre technique du génie logiciel.

4.1 Expérimentation

Dans cette expérimentation, les trois modèles PIM traités sont issus des cas d'utilisation "Réserver", "Louer" et "Retourner" de l'application de location de voitures qui a été présentée dans le deuxième chapitre.

Nous partons de l'hypothèse qu'il y a un ou plusieurs modèles PIM d'une application auxquels les principes de la méthode d'assemblage sont appliqués pour obtenir un nouveau modèle de l'application à base de composants.

4.1.1 Identification des composants

Dans le tableau ci-dessous sont regroupées les entités tirées des modèles PIM. Cela donne une vue d'ensemble des classes de l'application, une connaissance des objets du domaine et les éléments redondants des modèles PIM.

Tableau IX

Classification des entités des modèles PIM

Modèles PIM	**Eléments statiques**	**Eléments dynamiques**
Réserver (Figure 2.2)	Client,Modèle, Groupe	Réservation
Louer (Figure 2.4)	Voiture, Client,Carte de crédit, Permis de conduire, Police d'assurance	Réservation, Location, Paiement
Retourner (Figure 2.8)	Voiture, Client, Carte de crédit	Retour, Location, Paiement

Nous pouvons remarquer, dans ce cas d'étude, que les classes telles que "Client" et "Voiture" sont des composants du domaine. Le parallèle entre ce qui se passe réellement dans une entreprise de location de voitures et le tableau permet d'en arriver à cette conclusion. Le même constat peut être fait pour les composants processus. Une location, un paiement ou une réservation invoquent tous un processus ou une transaction entre deux entités. Ils sont donc des composants processus potentiels.

Après avoir répertorié l'ensemble des objets du domaine ou processus, les composants de l'application peuvent être déduits et sont donnés dans le tableau X.

Tableau X

Les composants de l'application

Composants du domaine	Composants Processus
Voiture[Voiture,Groupe,Modele]	Réservation
Client[Client,Permis,CarteDeCredit,PoliceAssurance]	Location
	Retour
	Paiement

Le découpage de ces composants est basé sur le fait que les entités "Carte de crédit", "Police d'assurance" et "Permis de conduire" sont des entités qui apportent de l'information sur ce qui est associé au "Client". Il semble donc opportun de les grouper sous une seule entité qui devient alors le composant du domaine "Client". Le même raisonnement s'applique sur l'entité "Voiture", qui regroupe les entités "Groupe" et "Modele" pour former le composant du domaine "Voiture".

Dans les figures ci-dessous, le découpage permet de voir les frontières (les cadres en pointillés) entre chaque composant et où seront affectés les ports dans les prochaines étapes.

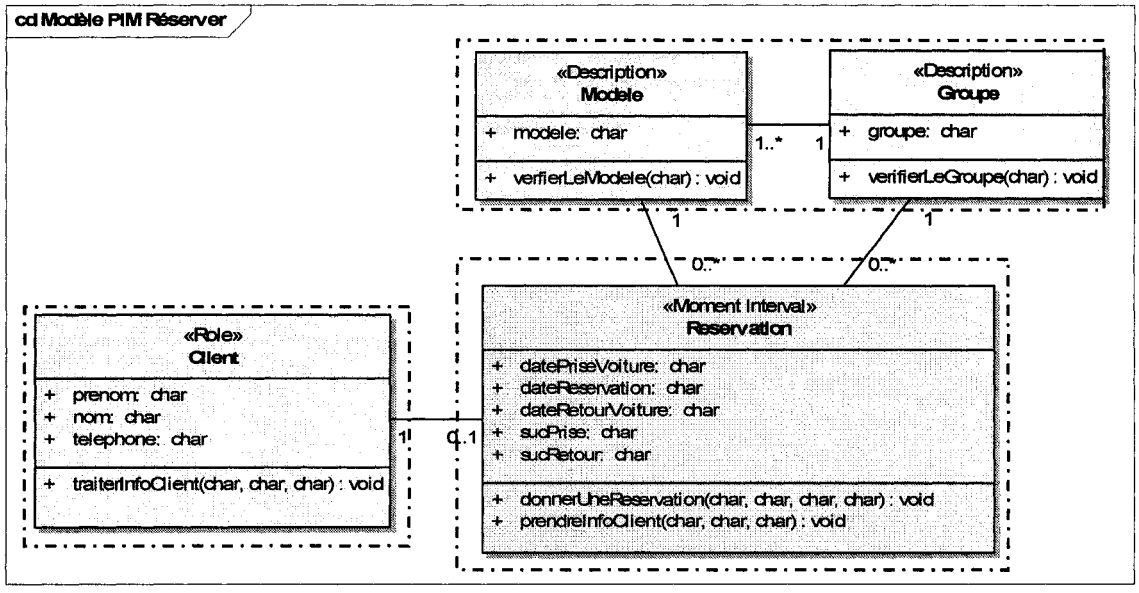


Figure 4.1 Identification des composants du modèle PIM "Réserver"

Dans le modèle PIM "Réserver", le composant "Voiture" est représenté par les deux classes "Modele" et "Groupe".

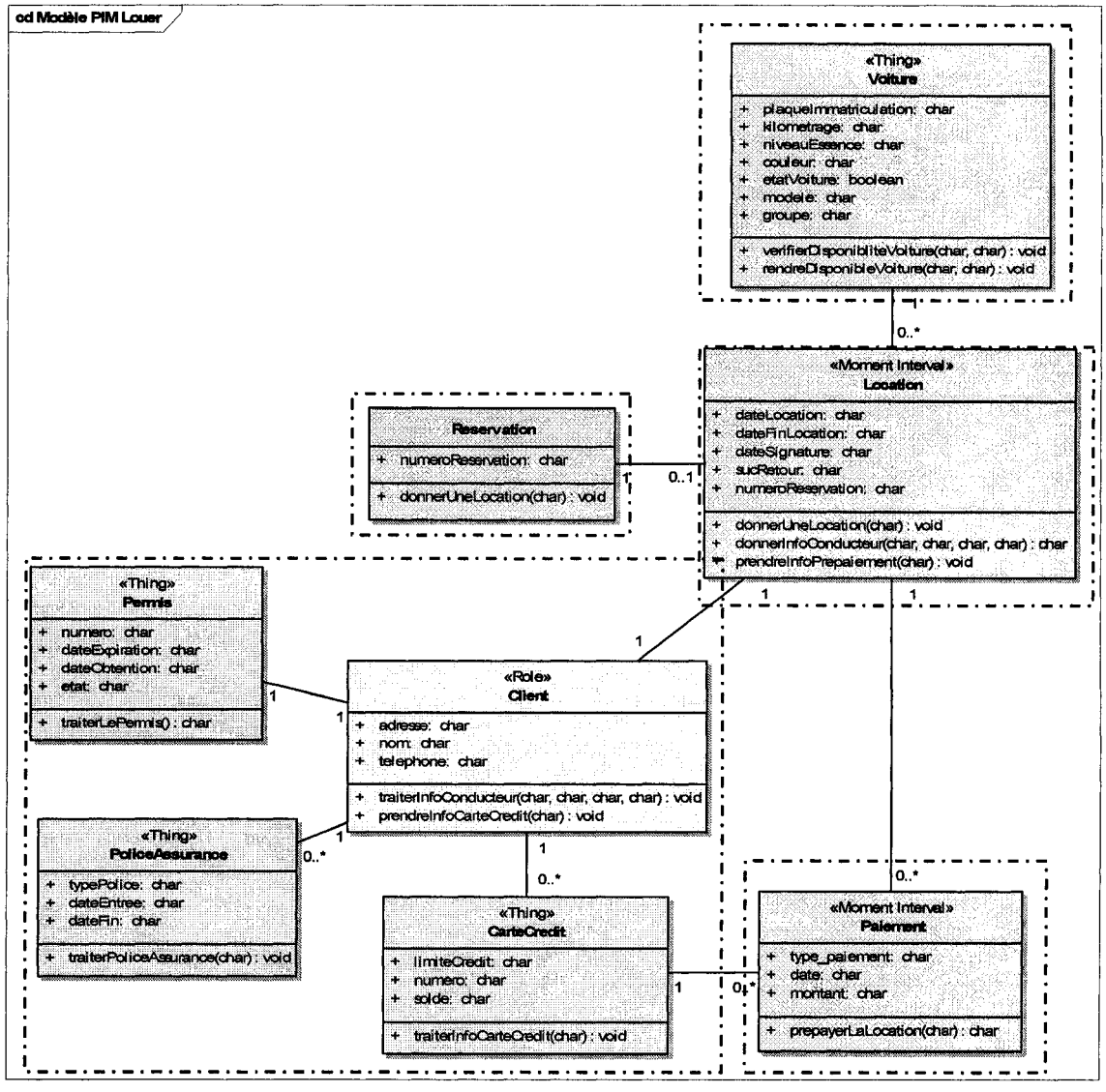


Figure 4.2 Identification des composants du modèle PIM "Louer"

Pour le modèle PIM "Louer", le composant "Voiture" apparaît finalement dans son entier puisque d'autres éléments ont été introduits suite à la location.

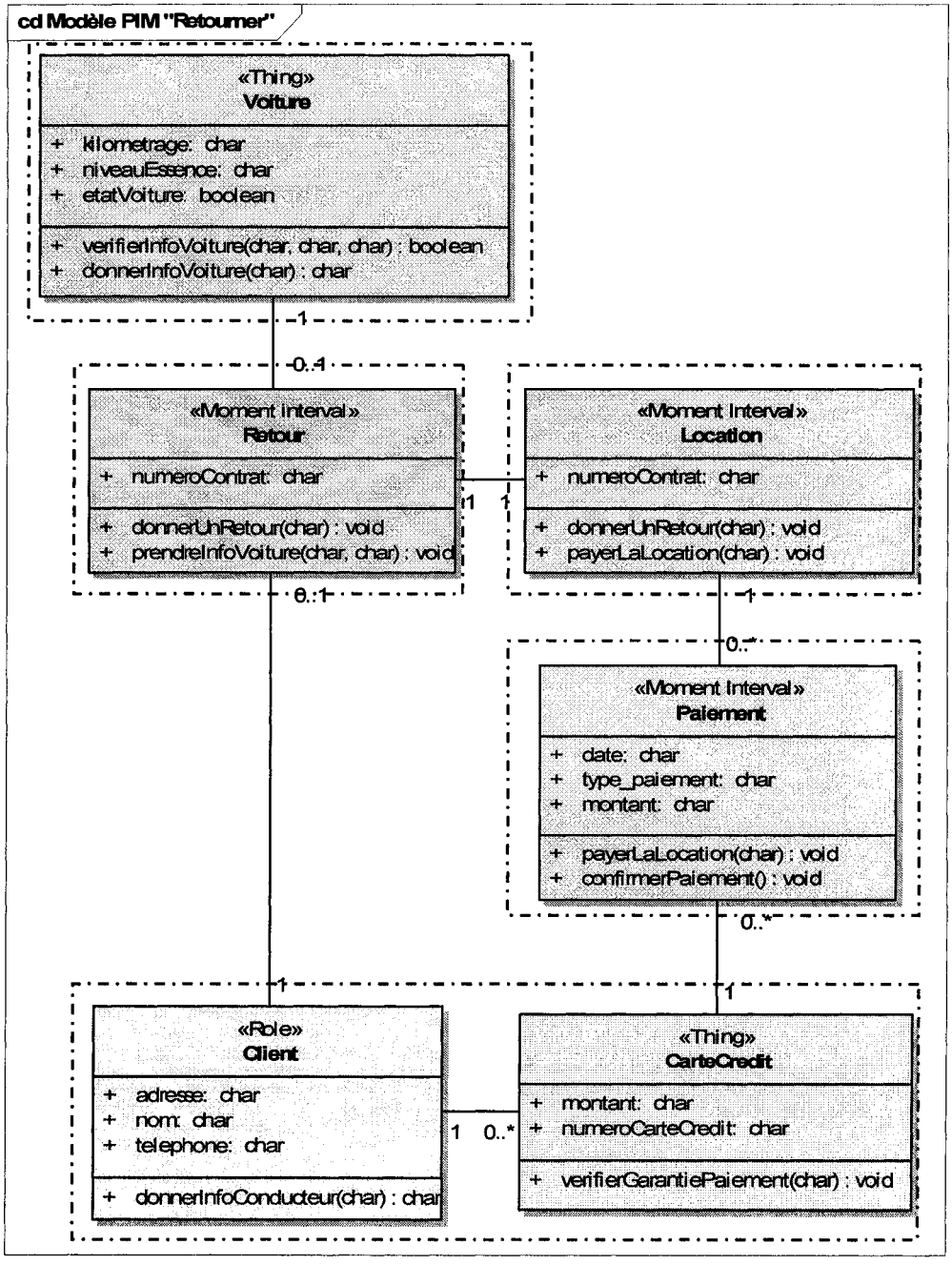


Figure 4.3 Identification des composants du modèle PIM "Retourner"

Dans la figure ci-dessus, le composant "Paiement" est réutilisé par le composant "Location". Il convient de signaler que l'entité "Paiement" est un "moment interval" au

même titre que "Location", car son action intervient à un moment précis de la transaction. De ce fait, il est considéré comme un composant processus à part entière.

4.1.2 Regroupement des composants

Le regroupement des composants concerne les composants se trouvant dans les différents modèles PIM. L'opération décrite dans cette sous section consiste à regrouper toutes les entités semblables en une seule et unique entité. Pour ce faire, l'opérateur d'assemblage est appliqué sur les entités mêmes, c'est-à-dire sur les paquetages, les classes, les méthodes et les attributs.

Pour éviter une longue instruction de regroupement par l'opérateur d'assemblage, il ne sera appliqué que sur un composant, à savoir "Voiture".

L'instruction ci-dessous prépare les paquetages de l'application.

CorrespondPackage [Reserver, Louer, Retourner]

Nous établissons la correspondance entre les différentes classes du composant "Voiture".

CorrespondClasses[Reserver.Groupe]
CorrespondClasses[Reserver.Modele]
CorrespondClasses[Louer.Voiture, Retourner.Voiture]

Toutes les méthodes du composant doivent être traitées. Elles sont regroupées en une seule entité qui est désormais le composant "Voiture".

CorrespondOperations[Reserver.Modele.verifierLeModele]
CorrespondOperations[Reserver.Groupe.verifierLeGroupe]

CorrespondOperations[Louer.Voiture.verifierDisponibiliteVoiture]
CorrespondOperations[Louer.Voiture.rendreDisponibleVoiture]

CorrespondOperations[Retourner.Voiture.verifierInfoVoiture]
CorrespondOperations[Retourner.Voiture.donnerInfoVoiture]

Nous pouvons noter que les méthodes du composant "Voiture" ne trouvent pas de correspondance dans les différentes classes des modèles PIM.

Chaque attribut des éléments du composant "Voiture" doit être pris en compte.

CorrespondAttributs[Reserver.Groupe.groupe, Louer.Voiture.groupe]
CorrespondAttributs[Reserver.Modele.modele, Louer.Voiture.modele]

L'attribut groupe dans la classe "Groupe" du modèle PIM "Reserver" est le même que celui dans la classe "Voiture" du modèle PIM "Louer" car les classes "Modele" et "Voiture" appartiennent au même composant.

CorrespondAttributs[Louer. Voiture.plaqueImmatriculation]
CorrespondAttributs[Louer. Voiture.couleur]
CorrespondAttributs[Louer.Voiture.niveauEssence,
Retourner.Voiture.niveauEssence]
CorrespondAttributs[Louer.Voiture.etatVoiture, Retourner.Voiture.etatVoiture]
CorrespondAttributs[Louer.Voiture.kilometrage, Retourner.Voiture.kilometrage]

Nous obtenons ainsi le regroupement du composant "Voiture". Il faut répéter la même opération pour les autres composants.

4.1.3 Attribution des ports et interfaces

À ce niveau de la méthode, les composants ont été identifiés et regroupés au moyen de l'opérateur d'assemblage. Il reste maintenant à attribuer les interfaces et les ports aux composants identifiés avant l'assemblage final.

Afin d'assembler deux composants, il est important de déterminer le rôle de ces composants. Pour ce faire, la matrice d'identification de rôles est utilisée (voir tableau XI).

De ce fait, les interactions entre les composants et les fonctions qui les régiront sont déterminées. Dans le tableau XI ci-dessous, les interactions entre composants sont déduites du diagramme de séquence de la figure 2.3. En tenant compte de ce diagramme, le tableau d'interactions des composants devient plus facile à remplir.

Tableau XI

Interaction des composants du modèle PIM "Réserver"

Composants	Réservation	Client	Voiture
Réservation	_	traiterInfoClient	verifierLeGroupe, verifierLeModele
Client	_	_	_
Voiture	_	_	_

Le composant "Reservation" (dans la colonne à gauche) n'a pas d'interaction avec lui-même (la ligne "Réservation" est donc vide). Le composant "Reservation" interagit avec le composant "Client" à travers la méthode "traiterInfoClient" et avec le composant "Voiture" via les méthodes "verifierLeGroupe" et "verifierLeModele". Le composant final "Reservation", aura donc au moins une interface pour son port composant pour ce cas-ci.

Tableau XII

Interaction des composants du modèle PIM "Louer"

Composants	Location	Client	Voiture	Paiement	Reservation
Location	–	traiterInfoConducteur, prendreInfoCarteCredit	verifierDisponibiliteVoiture, rendreDisponibleVoiture	prepayerLaLocation	–
Client	–	–	–	–	–
Voiture	–	–	–	–	–
Paiement	–	–	–	–	–
Reservation	donnerUneLocation	–	–	–	–

Nous reprenons la même démarche pour le modèle PIM "Retourner". Le diagramme de séquence du modèle PIM de la figure 2.5 a permis d'avoir les interactions définies dans le tableau ci-dessus.

Tableau XIII

Interaction des composants du modèle PIM "Retourner"

Composants	Retour	Client	CarteCredit	Voiture	Paiement	Location
Retour	–	donnerInfoConducteur		donnerInfo-Voiture verifier-InfoVoiture		
Client	–	–		–	–	–
CarteCredit	–					
Voiture	–	–		–	–	–
Paiement	–		verifierGarantie Paiement	–	–	
Location	donnerUnRetour	–	–	–	payerLaLocation	–

Tout comme les autres tableaux d'interactions des composants, le diagramme de séquence du modèle PIM de la figure 2.8 a permis d'obtenir les résultats du présent tableau ci-dessus. La méthode confirmerPaiement, qui est à utilisation interne, n'est pas représentée ici.

L'utilisation de la matrice dans la théorie des couches permettait de déterminer les composants du domaine selon que ceux-ci étaient dans plusieurs modèles PIM. Dans la méthode proposée, cette matrice joue toujours un rôle d'identification mais dans l'interaction entre composants et la détermination de leurs interfaces. Elle sera utile plus tard dans l'assemblage des composants pour déterminer quel composant doit être assemblé avec quel autre. Nous pouvons même envisager l'utilisation d'une telle matrice dans l'automatisation de la méthode proposée.

Dans les figures ci-dessous, les différents types de ports évoqués plus tôt sont mis en évidence. Le port de synchronisation est celui qui liera le composant "Reservation" au composant "Location". Et le port composant est celui qui liera le composant "Reservation" au composant "Voiture".

La séparation des composants a commencé par déterminer les frontières de chaque composant. Les classes d'un composant qui sont liées à d'autres classes en dehors des limites de ce composant dans le modèle PIM initial, seront liées au port de celui-ci.

Le composant processus "Reservation" possède deux ports portant des interfaces distinctes : de type synchronisation et de type composant. Ceci est normal car il n'y a qu'un composant processus qui utilise les composants du domaine afin d'accomplir sa mission. Ceci explique qu'il y ait un port ayant une interface de type synchronisation pour la transaction entre processus, et un port ayant une interface de type composant pour l'utilisation d'un ou plusieurs composants du domaine. Le "moment interval" est en général le composant processus dans un modèle PIM.

Le type des interfaces, c'est-à-dire requis ou fourni, est donné par le lien d'association qui peut exprimer respectivement une dépendance ou une réalisation (voir figure 4.4).

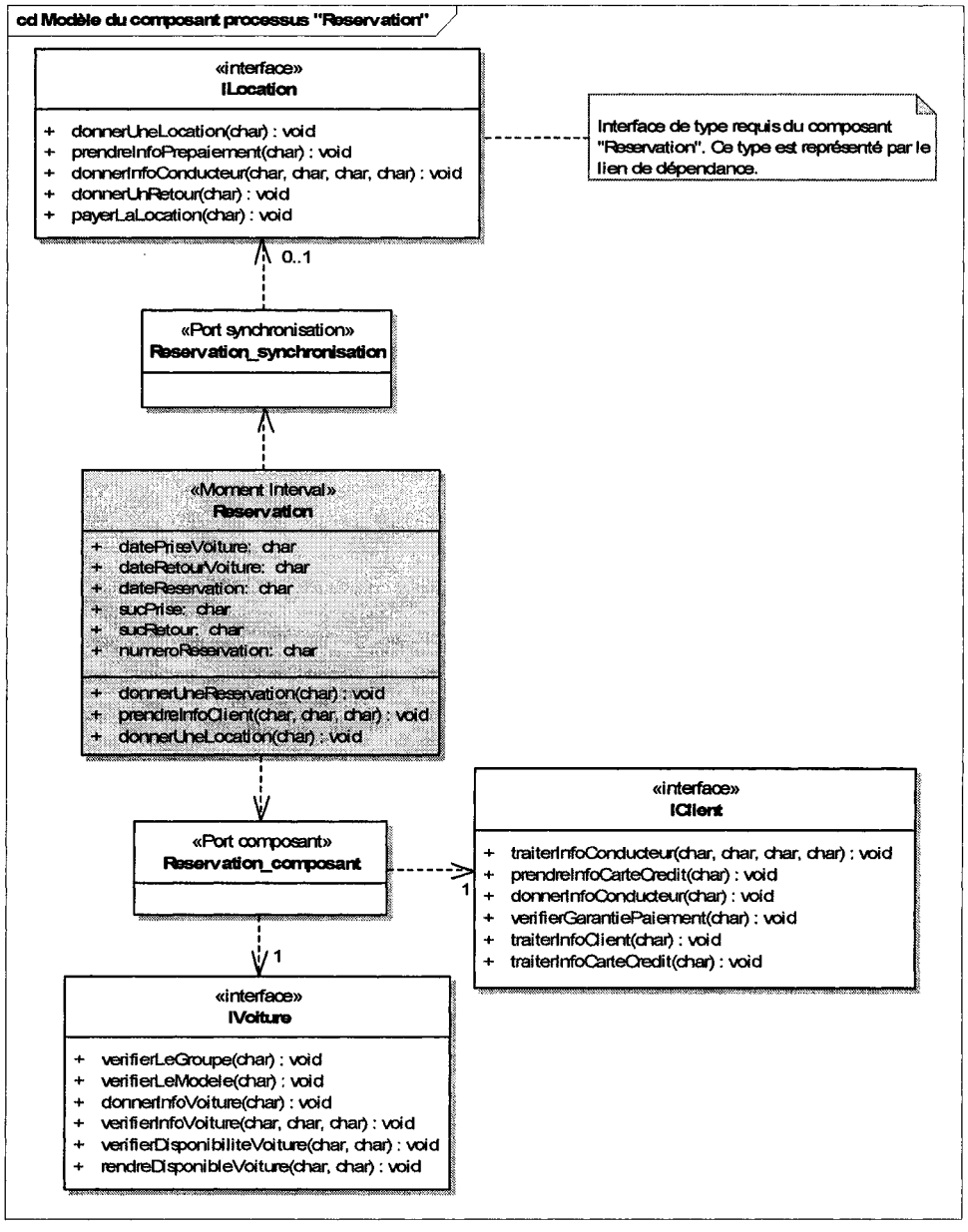


Figure 4.4 Modèle du composant processus "Reservation"

Le diagramme du composant "Reservation" ci-dessous ainsi que les autres qui vont suivre sont obtenus avec la lecture des matrices d'identification.

Pour le composant "Location", tout comme le composant "Reservation", il possède deux types de ports. Ce composant interagit avec la plupart des composants de l'application car il se retrouve dans deux modèles PIM impliqués dans cette expérimentation.

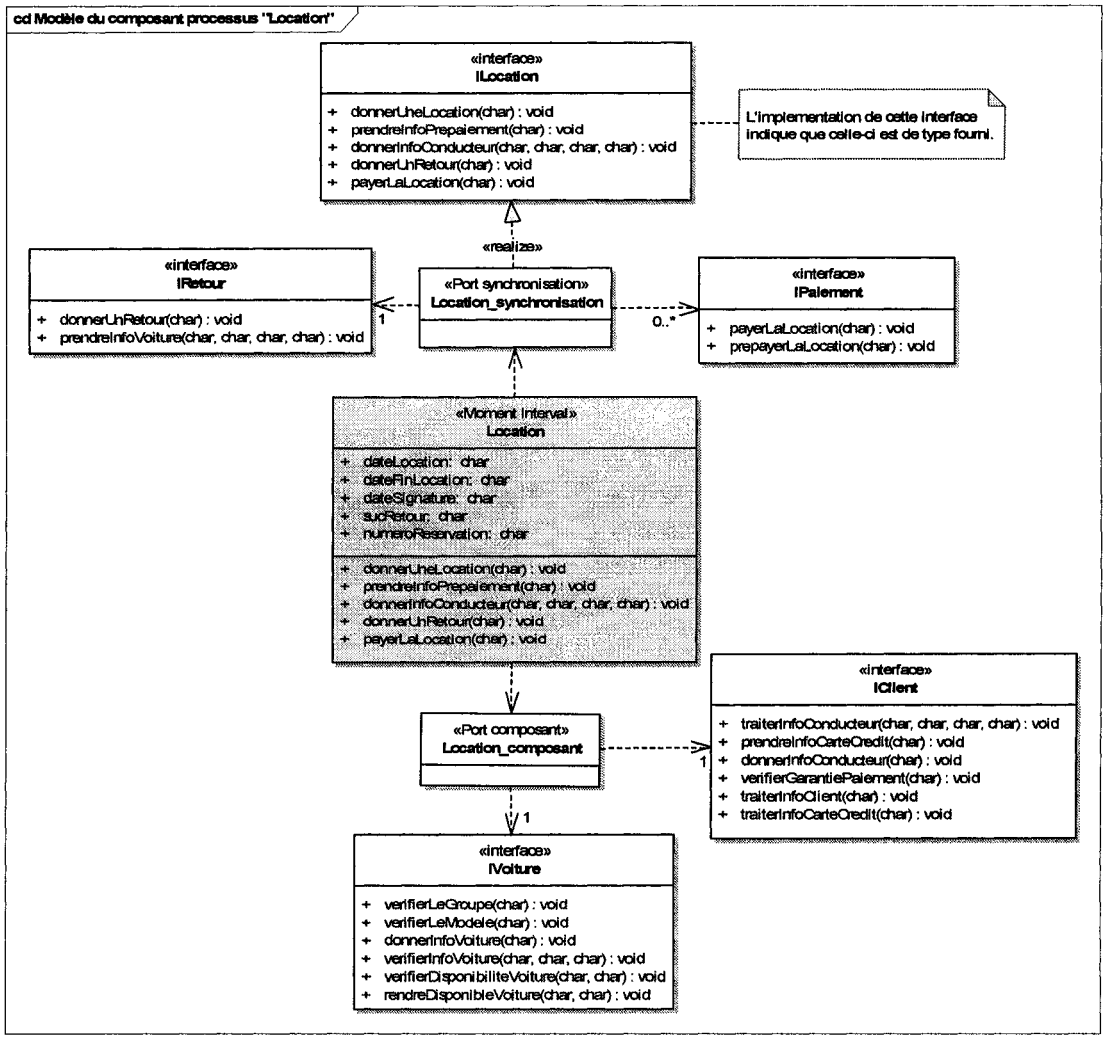


Figure 4.5 Modèle du composant processus "Location"

En ce qui concerne le composant "Retour", il est uniquement composé d'une classe d'analyse "Retour" ayant deux ports comprenant une interface de synchronisation et une interface composant l'interface du composant "Location" interagissant avec l'autre composant processus est de type fourni pour le composant "Retour" car il fournit un retour à une location.

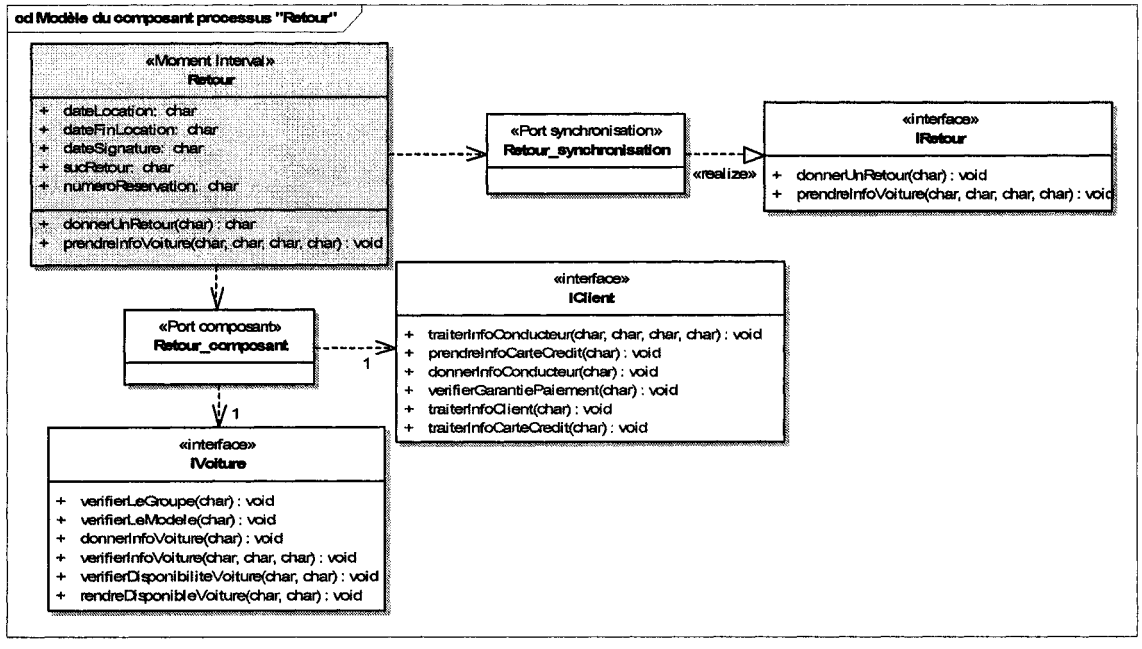


Figure 4.6 Modèle du composant processus "Retour"

Le composant processus "Paiement" est un composant à part entière. Il contient tous les éléments et toutes les actions aboutissant à un paiement. Il se retrouve dans le modèle PIM "Louer" pour un prépaiement demandé par le composant processus "Location", et dans le modèle PIM "Retourner" pour le paiement final de cette même location.

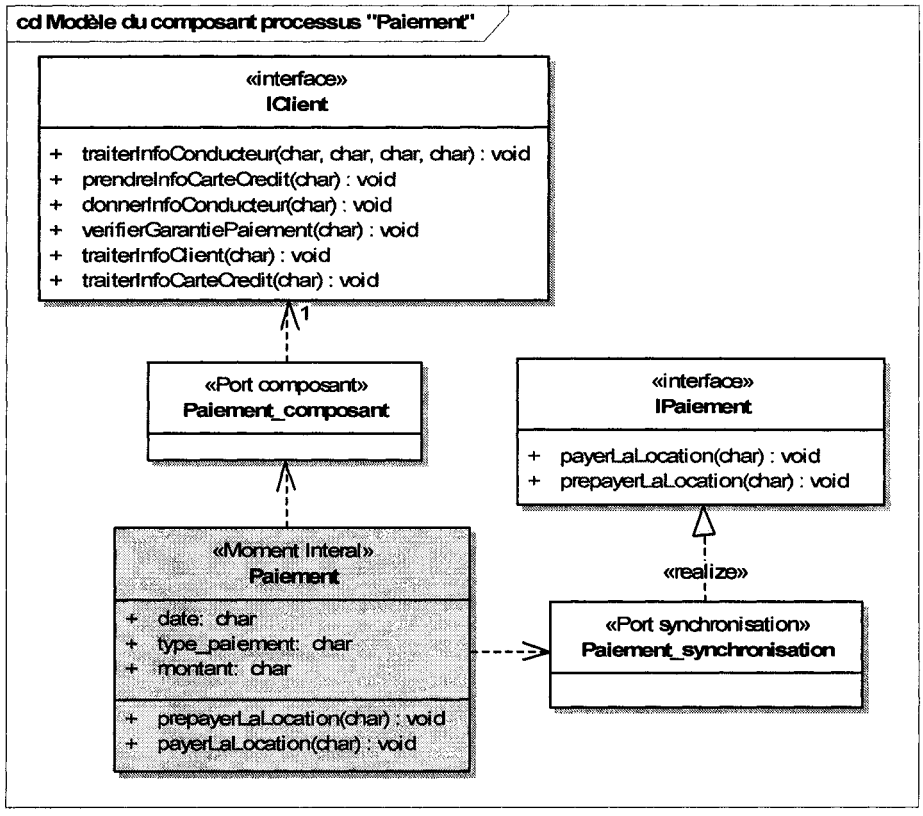


Figure 4.7 Modèle du composant processus "Paiement"

Quant au composant "Voiture", vu que la classe d'analyse "Groupe" était liée au "moment interval" "Réservation", celle-ci devra être reliée au port du composant auquel il appartient afin d'éviter les pertes d'informations lors du découpage. Pour le composant "Voiture", les classes "Groupe" et "Modele" seront liées au port.

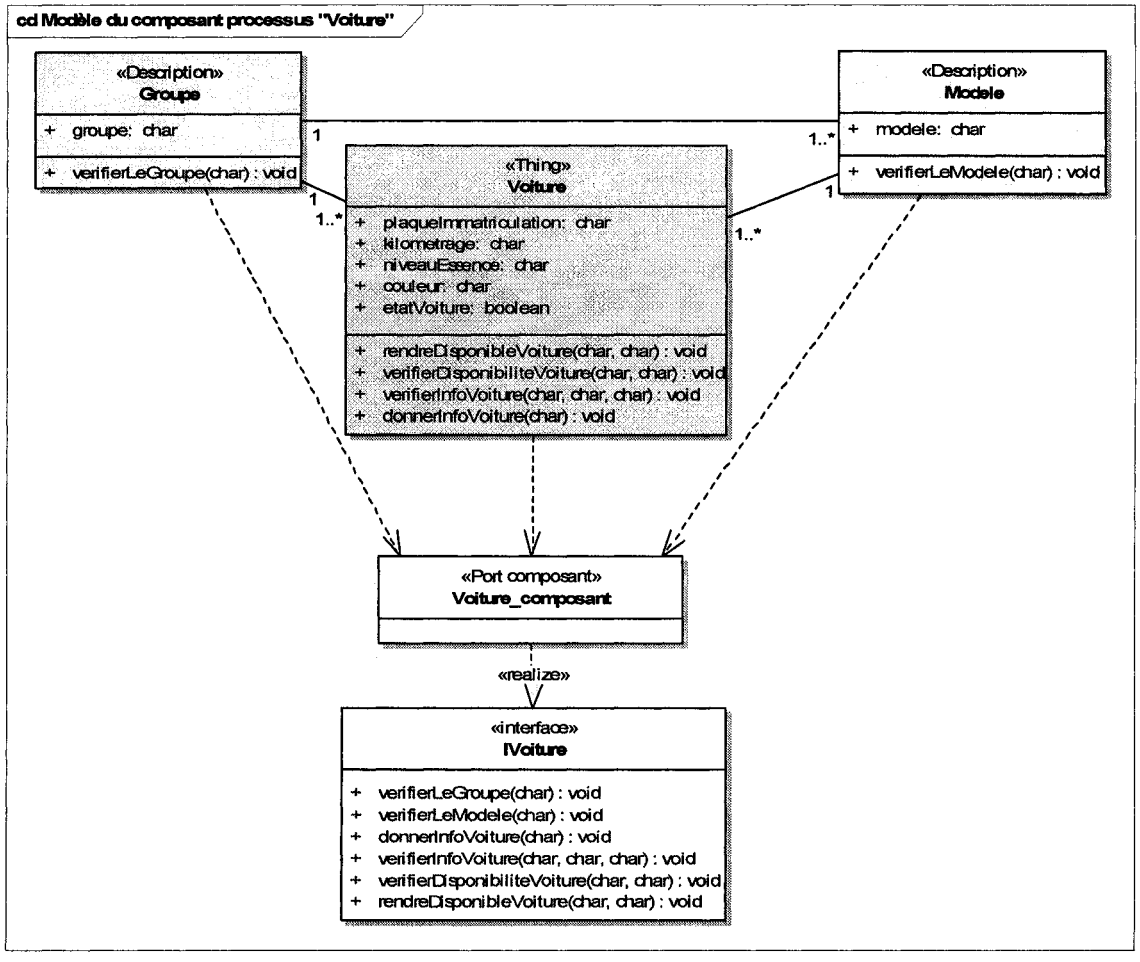


Figure 4.8 Modèle du composant du domaine "Voiture"

À chaque composant, le port du composant accompagné de ses interfaces est ajouté. Celles-ci sont déterminées par le contact entre ce composant et les autres dans son environnement immédiat.

Pour le composant "Voiture", les méthodes qui étaient dans les classes "Groupe" et "Modele" sont désormais dans la classe de l'interface du port car elles font partie du service fourni par le composant.

Le composant "Client" est celui qui interagit avec le composant "Paiement" afin d'effectuer un paiement via une carte de crédit. Ce composant possède des objets tels qu'un permis de conduire, une police d'assurance et une carte de crédit, ce qui correspond tout à fait à la réalité pour un objet de ce type.

C'est également le composant qui est à l'origine de bon nombre d'actions dans cette application, ce qui se confirme par le nombre de méthodes dans le port composant.

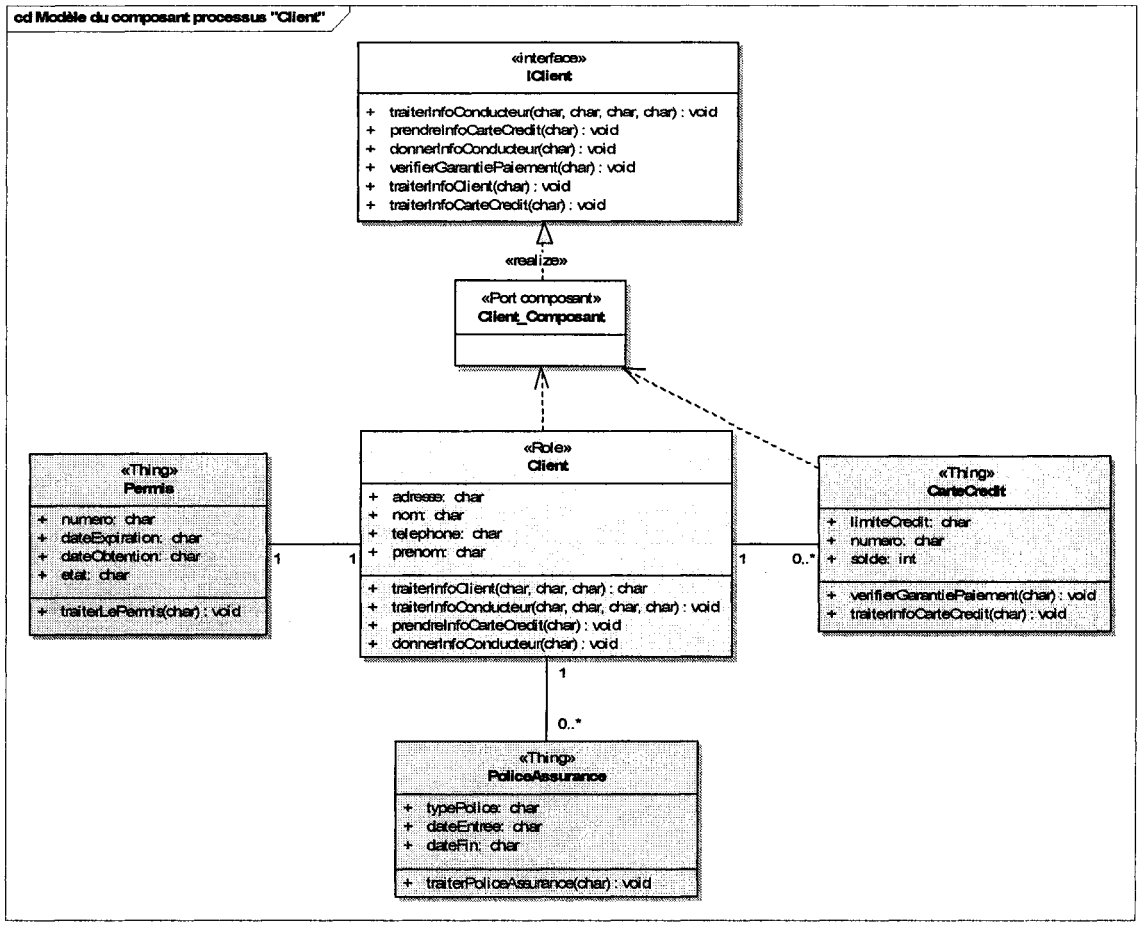


Figure 4.9 Modèle du composant du domaine "Client"

4.1.4 Assemblage des composants

Dans le regroupement des composants de l'application, six composants ont été dénombrés pour l'ensemble de cette application. Parmi ces composants, il y en a quatre de type processus ("Réservation", "Location", "Retour" et "Paiement") et deux de type du domaine ("Client" et "Voiture"). Les composants du domaine se sont avérés être dans les trois modèles PIM.

L'opération d'assemblage des composants se concentre uniquement sur les ports de ceux-ci ainsi que leurs interfaces, car ce sont bien les ports et les interfaces qui permettent de les assembler. Les interfaces portant le même nom sont jointes pour ne former qu'une seule et unique interface dans le modèle de l'application. Concrètement, cette association se traduit par celle des interfaces qui peut être constatée sur le modèle assemblé (figure 4.10) de l'application.

Au vu du modèle complet de l'application, l'objectif que nous poursuivions est atteint. Les limites de chaque composant restent en évidence à travers leurs ports et ce malgré l'assemblage, ce qui peut apporter un avantage au niveau de la flexibilité du modèle, avantage qui caractérise le développement par composants.

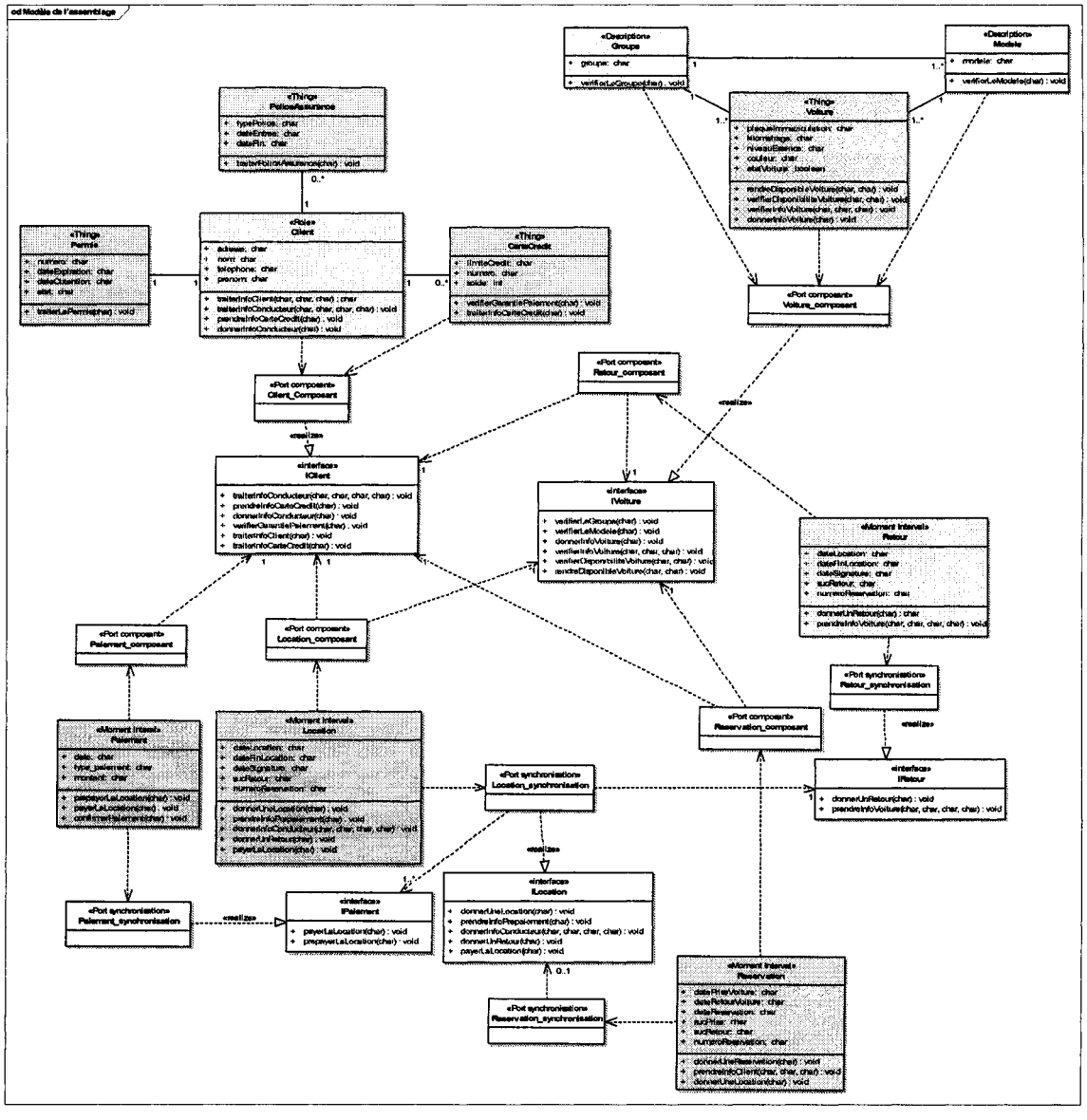


Figure 4.10 Modèle de l'assemblage des composants de l'application

Chaque composant a son point d'entrée ou de communication à partir d'un port, et ensuite, l'entité principale du composant, soit le "Moment Interval", sert de délégation afin d'atteindre les extensions ou les classes subsidiaires. Cette manière de procéder permet de diminuer le couplage et d'augmenter la cohésion [1].

Dans les paragraphes suivants, nous revoyons en forme de boîte noire les modèles PIM vus jusqu'à présent. La version 2 d'UML donne en effet la possibilité d'avoir un modèle final avec des composants en boîte noire, ce qui donne un aperçu de l'interaction entre les composants de l'application.

Le modèle du composant en UML 2 est un modèle en boîte noire car il ne montre pas l'implémentation du composant mais donne une idée de son fonctionnement à travers ses interfaces.

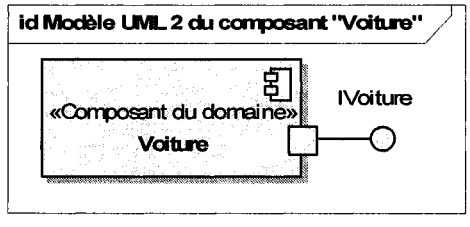


Figure 4.11 Modèle UML 2 du composant "Voiture"

Le composant "Client" possède une interface de type fourni car celui-ci est requis par les composants processus de l'application.

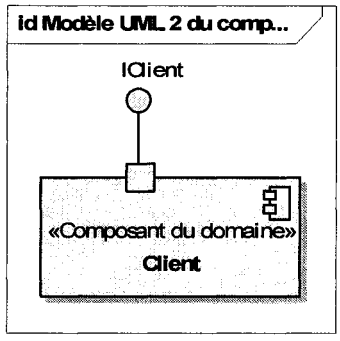


Figure 4.12 Modèle UML 2 du composant "Client"

L'étude des archétypes a montré que le "moment interval" est le point central dans un modèle de conception. Dans cette étude, le "moment interval" est un composant processus. Le schéma de ce composant confirme bien la théorie émise dans l'étude des archétypes [20]. Le port composant portant les interfaces "IVoiture" et "IClient" permet au composant "Reservation" d'être lié aux composants du domaine qu'il utilise.

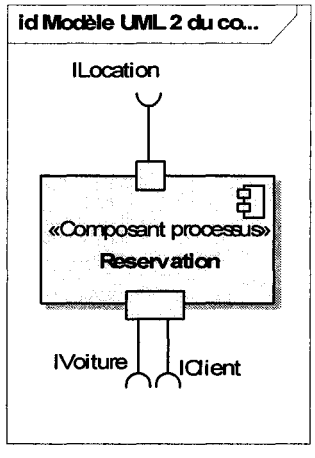


Figure 4.13 Modèle UML 2 du composant processus "Réservation"

Le modèle du composant UML 2 montre bien les interfaces de ce composant.

Sur le modèle du composant ci-dessous, nous pouvons voir les ports composants et le port de synchronisation en remarquant les noms des interfaces. Nous pouvons remarquer également les composants avec lesquels le composant "Location" interagit.

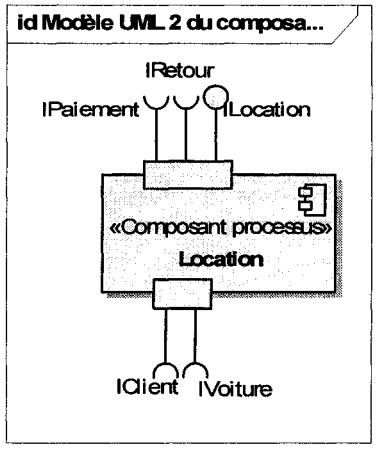


Figure 4.14 Modèle UML 2 du composant processus "Location"

Le composant "Retour", comme l'a indiqué la matrice, possède trois méthodes que nous pouvons voir sur la figure suivante.

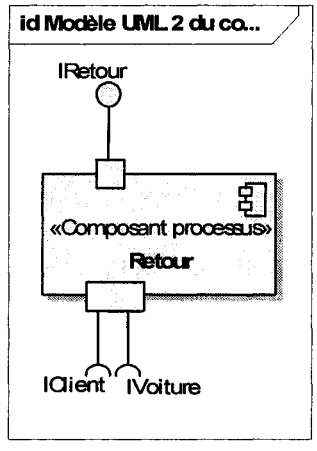


Figure 4.15 Modèle UML 2 du composant processus "Retour"

Le composant "Paiement" fournit les deux modes de paiement et requiert l'autorisation du client pour effectuer le paiement.

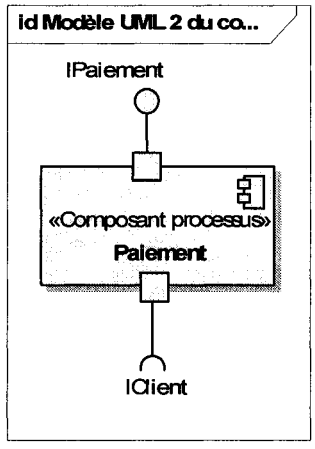


Figure 4.16 Modèle UML 2 du composant processus "Paiement"

Le modèle UML 2 complet de l'application met en évidence les composants de l'application et leurs relations les uns par rapport aux autres. Les liens entre composants reflètent entièrement ce qui a été obtenu dans les matrices d'interactions. Par exemple dans les matrices d'interactions, tous les composants processus ont une interaction avec le composant du domaine "Client", et c'est ce que nous pouvons constater dans la figure 4.17 ci-dessous.

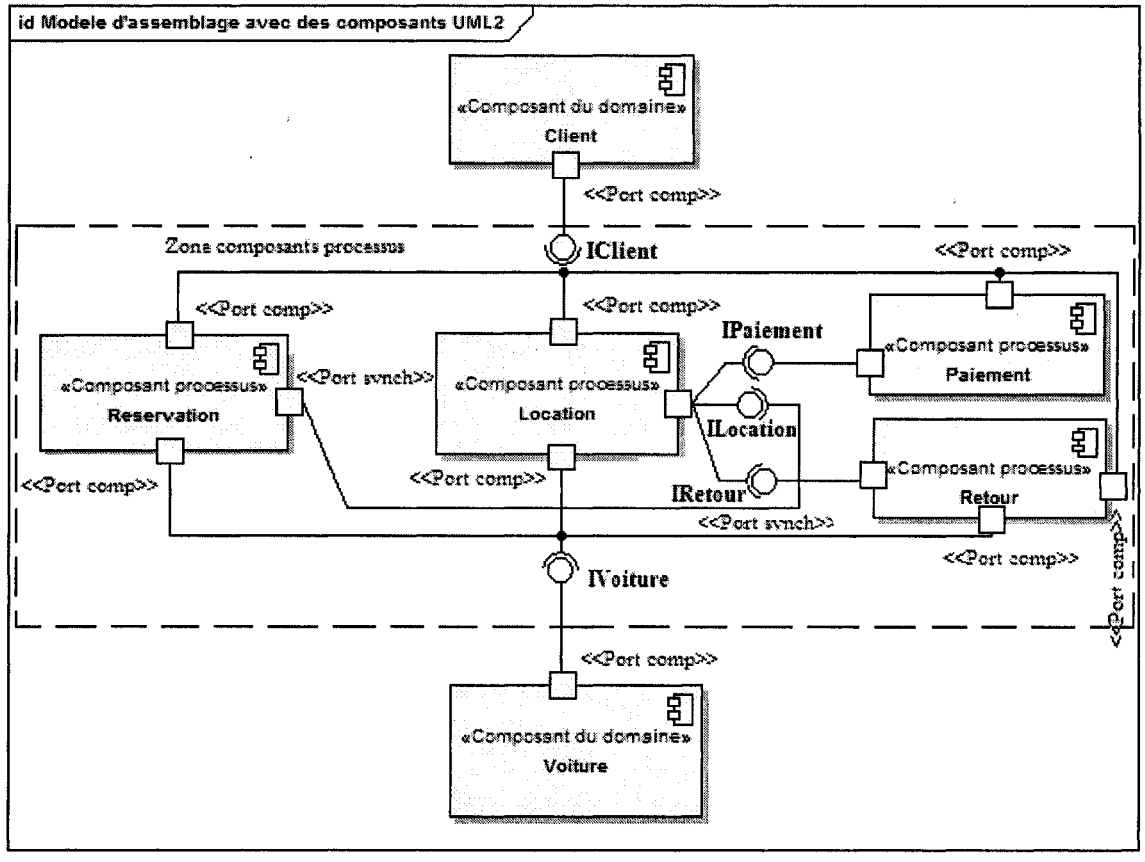


Figure 4.17 Modèle UML 2 de l'assemblage des composants de l'application

Ce modèle confirme celui qui a été déduit précédemment (figure 4.10). Les autres classes des composants sont cachées et seuls les ports et les interfaces indiquent les fonctionnalités que fournit le composant. Par exemple pour le composant "Client", les entités comme la carte de crédit, le permis et la police d'assurance sont omises. Les relations entre les composants se font à travers les ports et les interfaces.

Ce modèle est un aperçu de la future application qui doit être mise en oeuvre permettant de bien assimiler son architecture en cachant les détails que nous avons sur la figure 4.10.

4.2 Validation

Cette section apporte une comparaison entre les techniques qui ont été expérimentées et la méthode proposée. Cette comparaison a pour but d'évaluer l'efficacité, la facilité et la qualité de cette méthode. Il sera aussi question de faire la comparaison entre le modèle assemblé (figure 4.10) et le modèle conçu avec une autre méthode (figure 4.19).

L'expression de composition permet le regroupement des entités qui peuvent être développées séparément, et elle a été adaptée dans l'assemblage de composants en passant par le regroupement des ports des composants respectifs, après avoir défini clairement leurs interfaces. Par exemple le composant "Voiture" qui se trouvait dans les trois modèles PIM, a pu être unifié via cette technique à travers les règles de correspondance, de combinaison et de remplacement.

La technique des couches spécifiques et non spécifiques permet néanmoins de bien identifier les entités d'une application. De plus cette technique peut être utilisée dans l'assemblage des éléments de modèles PIM. La matrice tirée de cette approche a permis de déterminer les interactions entre composants ainsi que leurs interfaces.

L'approche ISC émet bien un assemblage de composants mais leur granularité est du nouveau code et ne correspond pas à l'objectif de ce projet.

De toutes les techniques vues, aucune n'est expressément orientée vers l'objectif de l'assemblage de modèles PIM. Cependant, elles ont apporté chacune une contribution dans l'établissement de la méthode d'assemblage proposée, ce qui la rend plus efficace.

À l'aide de la méthode d'assemblage que nous proposons, nous avons pu reconstituer un modèle global de l'application de location de voitures. Nous sommes partis des différents modèles PIM de l'application (Réserver, Louer et Retourner) auxquels nous avons appliqué les étapes constituant la méthode d'assemblage pour obtenir un modèle

entier de l'application. Pour évaluer le résultat obtenu, nous l'avons comparé à un modèle équivalent de la même application, en partant des mêmes éléments et en utilisant une autre méthode que celle proposée. Les deux modèles offrent fondamentalement les mêmes possibilités d'une éventuelle implémentation. En effet, le modèle assemblé, bien que contenant des concepts nouveaux tels que le port et les interfaces de type fourni ou requis, ne limite pas sa transformation vers des modèles autres que des modèles PIM.

La méthode que nous proposons est relativement simple à suivre et à automatiser. Les étapes évoquées dans cette méthode, en l'occurrence l'identification des composants, le regroupement des composants, l'attribution des ports et interfaces, et enfin l'assemblage, pourraient être automatisées. Ces étapes s'accomplissent en suivant une chronologie, et chacune d'elles regroupe un nombre de tâches bien identifiées qui doivent être accomplies. De plus, ces étapes, tout comme les tâches qu'elles contiennent, rendent plus facile l'application de la méthode proposée. L'organigramme ci-dessous (voir figure 4.18) montre plus clairement ces étapes qui peuvent mener à l'automatisation de la méthode.

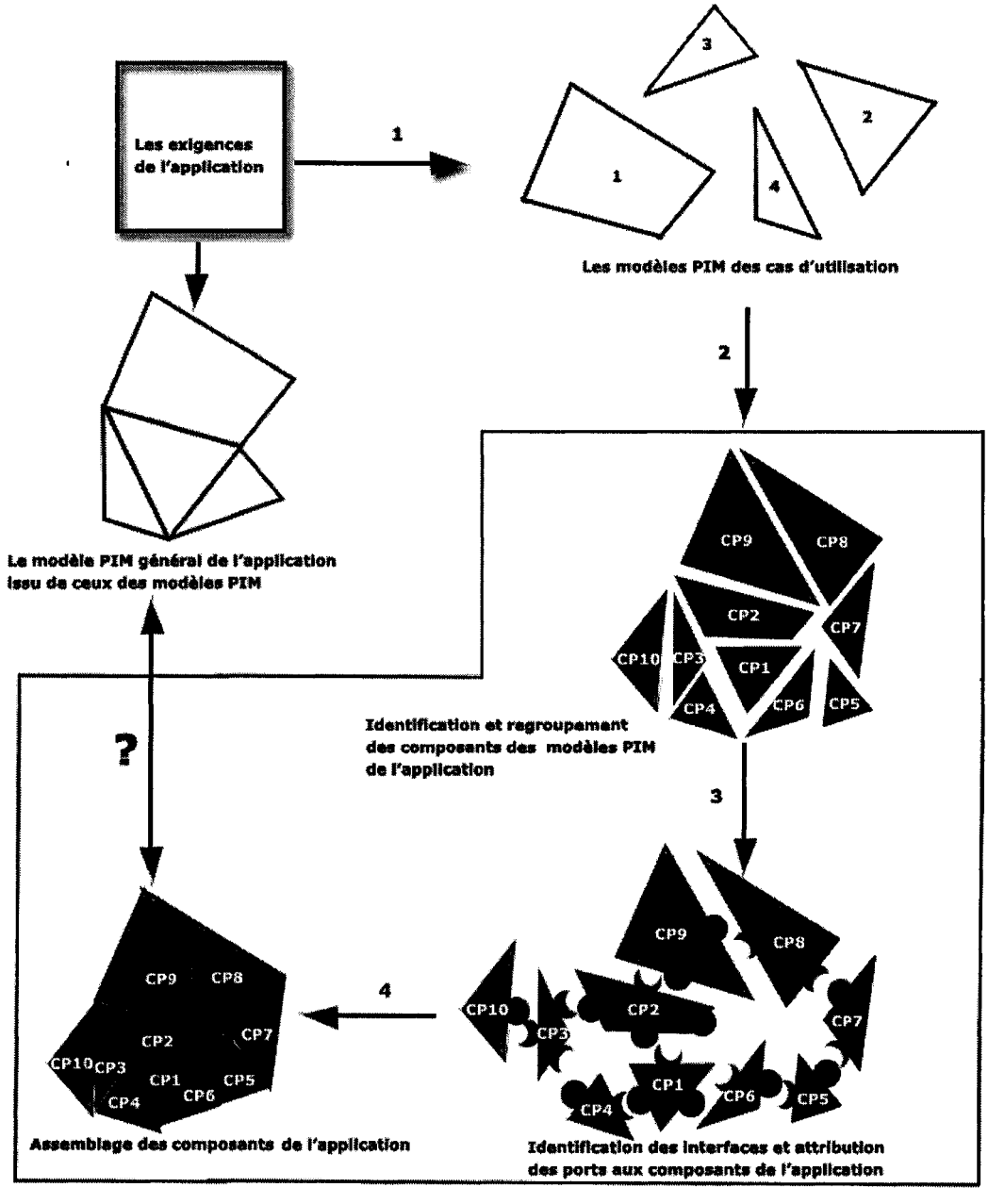


Figure 4.18 Méthode d'assemblage de modèles PIM

Dans la figure ci-dessus, le point un (1) est le passage des exigences de l'application à la modélisation des modèles PIM. Le point deux (2) est le passage des modèles PIM aux composants déduits de ceux-ci. L'élaboration des modèles PIM ne fait pas partie de

cette étude et par conséquent, notre projet suppose que ces modèles sont disponibles en intrant. Dans la figure, les points de la méthode d'assemblage sont les points deux (2), trois (3) et quatre (4).

D'autre part, pour évaluer le modèle composé, nous le comparons avec un autre modèle PIM conçu (figure 4.19) en passant par une autre méthode de conception d'application orientée objet. Bien entendu cela reste subjectif vu qu'une multitude de modèles pourraient être déduits des mêmes cas d'utilisation dépendamment de l'auteur du projet.

En comparant les deux modèles nous pouvons constater deux points importants :

1) dans le modèle conçu, les liens entre les classes d'analyse sont évidents, tandis que dans le modèle composé, ces liens se font à travers les ports. En effet, une classe n'ayant apparemment pas de lien avec une autre dans le modèle conçu, semble en avoir un dans le modèle composé. Ce n'est qu'une façon de voir les choses car ces classes servant de lien peuvent être vues comme des contrôleurs gérant les liens entre composants, ce qui réduit le couplage et augmente la cohésion [1];

2) le modèle composé apporte plus de flexibilité dans la lecture car chaque composant est délimité par le port. De cette façon il est plus facile d'ajouter ou d'enlever des éléments du modèle;

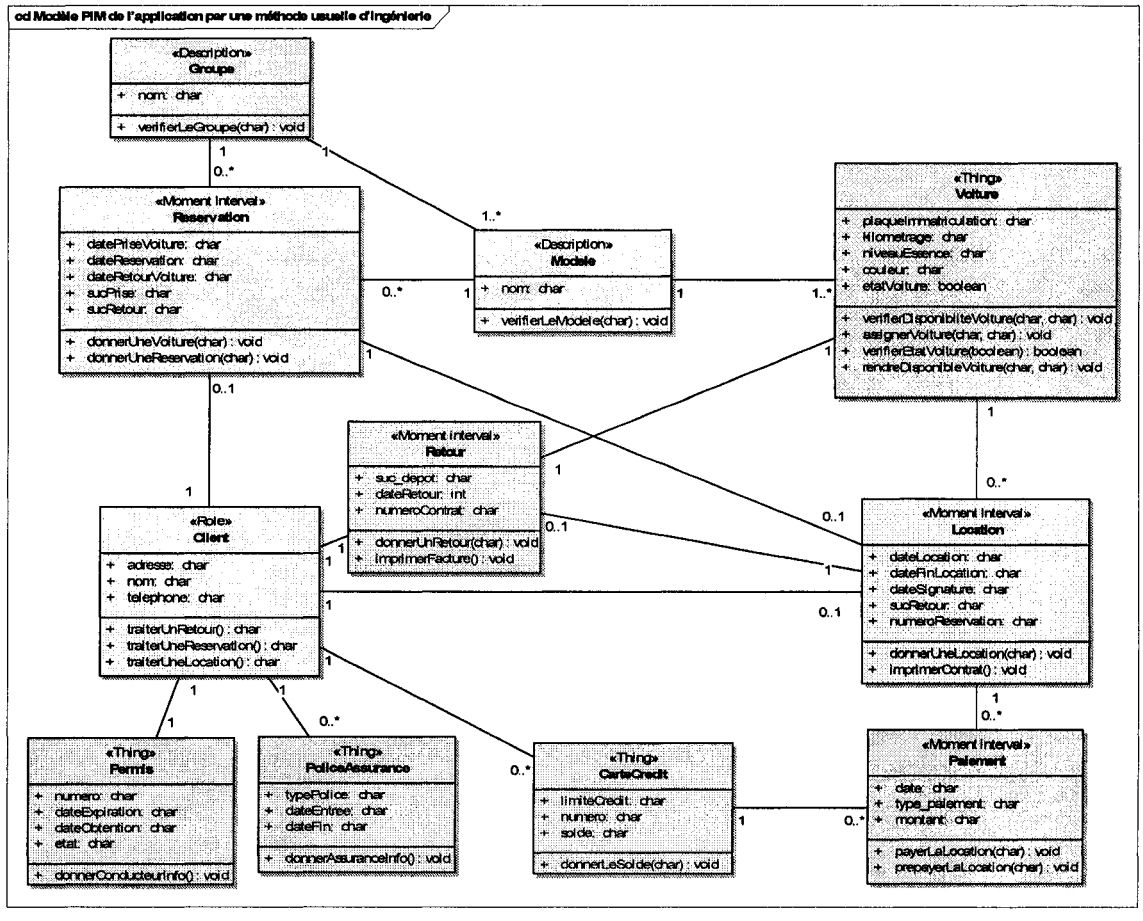


Figure 4.19 Modèle PIM de l'application par une méthode usuelle d'ingénierie

L'utilisation de la méthode d'assemblage a permis d'obtenir un assemblage de modèles dont le modèle final regroupe les éléments fondamentaux de l'application que nous pouvons constater dans le modèle conçu (figure 4.19). Les liens entre les différentes entités sont également présents dans le modèle assemblé.

Dans l'expérimentation de la méthode proposée, nous avons également présenté des modèles de composants en version UML 2. Le modèle final (figure 4.17) que nous avons obtenu montre plus clairement l'architecture de l'application à un plus haut niveau d'abstraction et pourrait être utilisé à un plus haut niveau d'étude du projet. Les

modèles de classes des composants pourraient eux être plus utiles au niveau du développement. Cela souligne l'utilité d'avoir un tel diagramme parmi les diagrammes de classes décrivant les composants comme ceux que nous avons vus.

L'objectif de ce projet était d'arriver à assembler les modèles PIM en nous servant des autres concepts. Le modèle final de la figure 4.10 obtenu en passant par les différentes approches vues telles que la glu, l'approche des couches spéciques et non spécifiques et l'approche ISC permet de conclure que l'objectif recherché est atteint.

CONCLUSION

Dans ce document, nous avons obtenu un assemblage de modèles PIM en combinant plusieurs techniques d'assemblage. Nous avons effectué un exemple d'application pour en voir l'efficacité et en tirer les conclusions. Le développement par composant n'est pas nouveau et a été introduit il y a bien des années dans des concepts de développement [6]. Les différences parmi les techniques considérées se situent au niveau de la granularité du composant ou sa définition. Nous avons également pu constater que pour certaines approches ou concepts, leur application dans les langages de programmation était ancrée et fonctionnait parfaitement [15, 26-28]. Ceci tend à confirmer une maîtrise de la technique d'assemblage d'objets, même si le niveau de granularité est encore celui du code.

L'objectif de notre projet était d'assembler des modèles PIM pour en arriver à un modèle pouvant être par la suite utilisable pour une implémentation future. Pour cela, nous avons utilisé des techniques existantes. Nous avons abordé notamment l'expression de composition qui nous a permis d'avoir une expression permettant le regroupement des composants à travers la correspondance, la combinaison et le remplacement des éléments des composants. Nous avons aussi vu la théorie des couches spécifiques et non spécifiques qui a donné un moyen par lequel nous pouvions attribuer les ports et les interfaces aux composants qui devaient être assemblés. Nous avons aussi vu la technique de la composition chirurgicale qui a apporté une certaine chronologie dans le parcours de la méthode proposée. Ces techniques et approches ont contribué à l'obtention d'un modèle assemblé.

Ce résultat est basé sur la catégorisation des composants en deux types, en l'occurrence le composant processus et le composant du domaine. Ces deux types de composants sont complémentaires dans les transactions généralement rencontrées. Les composants processus utilisent généralement les composants du domaine pour accomplir une tâche.

De plus, nous avons vu l'enchaînement des événements entre les composants processus dans les modèles PIM (la réservation engendre la location et la location engendre le retour). Cela a donné entre autre une chronologie dans l'assemblage des composants.

D'autre part, le découpage des composants en composant processus et en composant du domaine a permis également de déterminer les limites des composants dans les modèles PIM. Ceci a permis de les séparer avec l'aide des archétypes, qui ont apporté une valeur supplémentaire dans l'assemblage de ces composants.

En comparant le modèle assemblé à un modèle de la même application obtenu par une autre méthode de développement, nous avons pu constater que le modèle assemblé montrait une certaine flexibilité. Il faut rappeler que le développement par composants donne cette flexibilité supplémentaire au modèle de conception d'application en ce sens qu'il permet à certains composants assemblés d'être dissociés sans perturber l'apport des autres composants dans la même application. Cet avantage se traduit par un nombre additionnel de classes dans le modèle de l'application. C'est la raison pour laquelle il y a, dans le modèle que nous avons obtenu, un nombre supplémentaire de classes et interfaces.

Ceci étant dit, le développement de logiciels à partir de composants en modèles PIM est possible, si nous considérons le potentiel se trouvant présentement dans bon nombre d'applications de développement, y compris les langages de programmation. En termes de développement par composants, nous avons vu que dès que les cas d'utilisation sont connus, les composants peuvent être identifiés. Cette identification intervient très tôt dans la réalisation de systèmes et peut donc permettre une réduction des erreurs susceptibles d'engendrer des délais et des coûts supplémentaires.

Durant ce projet, nous avons trouvé une importante littérature abordant le sujet du développement par composants mais peu abordant l'assemblage des modèles PIM. Ce

projet nous l'espérons, apportera une valeur supplémentaire dans le développement par composants en général et en développement par composants de modèles PIM en particulier.

RECOMMANDATIONS

L'obtention d'un modèle d'assemblage de l'application de la location de voitures montre que la méthode proposée pour l'assemblage de modèles de composants métier peut fonctionner. Cependant, il faudrait réaliser entièrement une application jusqu'à son implémentation et fonctionnement pour réellement déterminer l'efficacité de cette méthode. Pour ce faire, il faudrait développer séparément les modèles de composants et les assembler par la suite à l'aide de la méthode proposée en respectant les règles énoncées dans ce projet.

D'autre part, des améliorations devront être apportées pour affiner l'assemblage obtenu en approfondissant l'étude sur l'expression de composition. Nous pourrions, par exemple, travailler sur l'automatisation par un logiciel de la technique de la glu et l'utilisation des matrices dans la perspective de la méthode d'assemblage. La condition à cela est que nous puissions nous entendre sur les modèles du composant, la technique de composition et le langage de composition [6].

RÉFÉRENCES

[1] Craig Larman, *Applying UML and patterns: An introduction to Object-oriented Analysis and Design and the Unified Process*. 2nd ed. 2002: Prentice Hall PTR, Upper Saddle River, NJ 07458.

[2] IBM/Rational, www.rational.com, *Rational Unified Process (RUP)*-(Visité en Décembre 2005).

[3] Anneke Kleppe, Jos Warmer,, Wim Bast, *Getting Started with Modeling Maturity Levels.* Dex, 2004.

[4] OMG, *MDA Guide version 1.0.1.* Object Management Group, 2003.

[5] OMG, *Object Management Group, http://www.omg.org/gettingstarted/gettingstartedindex.htm* (Visité en Décembre 2004). 1989.

[6] Uwe. Abmann, *Invasive software composition.* 2003, Berlin: Springer.

[7] Noury M. N. Bouraqadi-Saadani, Thomas Ledoux, *Le point sur la programmation par aspects.* 2001. Volume 20: p. 505-528.

[8] Clemens Szyperski, *Component Software: Beyond Object-Oriented Programming*. 1st ed. 1998: Addison Wesley Professional.

[9] Projet ACCORD, *La démarche MDA.* Projet RNTL ACCORD, 2002.

[10] Software Engineering Institute, *Model-Driven Architecture (MDA).* 2006, Carnegie Mellon University, Pittsburgh. *http://www.sei.cmu.edu/isis/guide/technologies/mda.htm (*Visité en juin 2006*):*

[11] Salim Bouzitouna, *Construction d'applications réparties par réutilisation de modèles dans une approche MDA, Laboratoire d'Informatique de Paris 6.* 2005, Université de Paris 6: Paris.

[12] Ivar Jacobson, Pan-Wei Ng, *Aspect-Oriented Software Development with Use Cases*. Addison-wesley ed. Object Technology series, ed. Addison-wesley. 2005, Upper Saddle River: Pearson Education.

[13] Dean Wampler, *The role of Aspect-Oriented Programming in OMG's Mode-Driven Architecture.* Aspect programming Inc., 2003.

[14] Uwe. Abmann, *Invasive Software Composition, Easycomp journal.* Springer 2003.

[15] The Compost Consortium, *Le Consortium Compost.* http://www.the-compost-system.org/: (Visité en juin 2005), Linköpings Universitet.

[16] OMG, *UML (Unified Management Language),* www.uml.org (Visité en Décembre 2004). 1997.

[17] OMG, *UML 2.0 Sperstructure Specification: Final Adopted specification (2003) document ptc/08-03-02,*. 2004, OMG, http://www.omg.org/cgi-bin/doc?ptc/08-03-02 (Visité en septembre 2005).

[18] James Ivers, Paul Clements, David Garlan, Robert Nord, Bradley Schmerl et Jaime Rodrigo Oviedo Silva, *Documenting Components and Connectors views with UML 2.0.* Technical report CMU/SEI-2004-TR-008, 2004, Carnegie Mellon University : Pittsburgh. Disponible au *http://www.sei.cmu.edu/pub/documents/04.report/pdf/04tr008.pdf.*

[19] Erich Gamma, Richard Helm, Ralph Johnson, *Design Patterns: Elements of Reusable Object-Oriented Software.* Pearson Education ed. 1994: Addison-Wesley.

[20] Eric Lefebvre, *Building MDA Platform-independent Models Using Business Archetypes and Patterns.* 2001: Montréal.

[21] Peter Coad, Eric Lefebvre, Jeff De Luca, *Java Modeling In Color With UML: Enterprise Components and Process.* 1999: Prentice Hall PTR.

[22] Business rules group, *Groupe de chercheurs en système d'information,* www.businessrulesgroup.org. Visité en Juin 2005.

[23] Simon Roberts, Philip Heller, Michael Ernest, *Complete Java 2 Certification (Seconde Edition).* Sybex ed. 2000, Alameda: Sybex.

[24] Emmanuel Renaux, *Définition d'une démarche de conception de systèmes à base de composants, Ecole doctorale des sciences pour l'ingénieur de Lille.* 2004, Université des Sciences et Technologies de Lille: Lille.

[25] Jack Greenfield, Keith Short, *Software factories, assembling applications with patterns, models, frameworks and tools.* 2004.

[26] Aspect team, *The AspectJTM Programming Guide., http://eclipse.org/aspectjs/doc/released/progguide/index.html*, Visité en Septembre 2005.

[27] Sparx Systems, *Enterprise Architect, Logiciel de Modelisation d'applications logicielles.* 2005.

[28] Compuware, www.compuware.com. Visité en Décembre 2004.