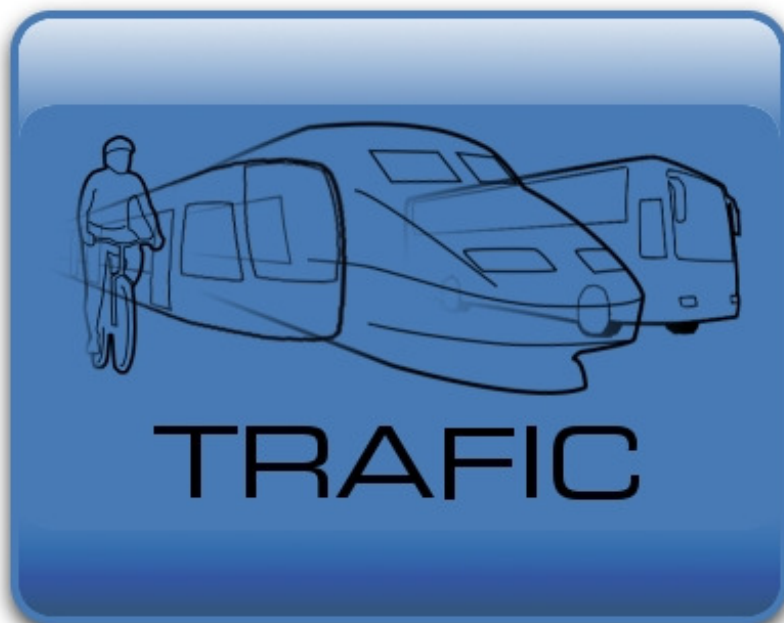


ELEMENTS D'UNE DEMARCHE SPECIFIQUE A L'ETUDE ET AU DEVELOPPEMENT DE SYSTEMES D'INFORMATION TRANSPORT



Livrable 5.2. du projet TRAFIC
Transports : Réutilisation, Adaptation, Fiabilité,
Intermodalité et Coopération inter-systèmes

Projet de la Région Rhône-Alpes 2003-2005

Thématiques prioritaires

Thème 6 : Aide à la décision - Transports

Auteurs : Nicolas Arnaud, Agnès Front, Jean-Pierre Giraudin.

Date : Décembre 2005

TABLE DES MATIERES

Table des Matières	2
Introduction.....	3
Partie A Démarche et processus de développement.....	6
1 Processus de développement en Y.....	6
2 Démarche itérative incrémentale	6
3 Démarche pilotée par les cas d'utilisation	7
4 Démarche orientée composant métier.....	8
Partie B Phases de spécification des besoins, d'analyse et d'analyse de la coopération	9
1 Spécification des besoins	10
2 Spécification organisationnelle des besoins	14
3 Analyse	16
4 Analyse de la coopération.....	17
Partie C Phases d'architecture applicative, technique et coopérative	18
1 Architecture applicative	18
1.1 Description des couches applicatives.....	18
1.2 Etablissement de la cartographie des composants techniques	19
1.3 Description des composants et des classes techniques	20
1.4 Définition des règles de conception et de transformation.....	20
2 Architecture technique	20
3 Architecture coopérative.....	21
Partie D Phase de conception	22
1 Transformation vers le langage Java.....	22
2 Intégration avec le générateur de la société ACTOLL	25
Partie E Conclusion.....	27

INTRODUCTION

La méthode de développement de systèmes d'information transport développée dans le cadre du projet TRAFIC est largement inspirée de la méthode Symphony, une méthode de développement logiciel à l'origine élaborée et industrialisée par la société Umanis¹ en collaboration avec l'équipe SIGMA du laboratoire LSR-IMAG.

La méthode Symphony a pour but de spécifier les différentes phases d'un projet de développement de systèmes d'information et de définir les tâches de chacun des intervenants. Il s'agit d'une démarche **itérative, incrémentale, orientée utilisateur, orientée composant métier et pilotée par les cas d'utilisation**. Elle adopte un modèle de **cycle de vie en Y** permettant de séparer l'étude des besoins fonctionnels (les fonctions attendues de l'application pour répondre aux besoins du métier de l'entreprise) de celle des besoins techniques (les contraintes opérationnelles de l'application telles que les contraintes matérielles et logicielles, la qualité de service en terme de temps de réponse, la tolérance aux pannes, etc.) et ceci dès le début du cycle de vie des applications. Cela permet une meilleure analyse du problème et des risques, mais aussi une meilleure réutilisation de l'existant.

Ainsi, le modèle de cycle de vie de la méthode Symphony présenté dans la Figure 1 s'articule autour de 3 branches.

- **La branche fonctionnelle (gauche)** correspond à la tâche traditionnelle de modélisation du domaine et des besoins des utilisateurs. Les résultats de l'analyse ne dépendent d'aucune technologie particulière et se résument à un modèle de composants métier.
- **La branche technique (droite)** a pour objectif le recensement des contraintes et des choix techniques nécessaires pour la conception du système tels que la sécurité, la montée en charge, l'intégration à l'existant, etc. Ces éléments permettent, par la suite, l'élaboration des architectures applicatives et techniques de l'application. L'architecture applicative spécifie les composants techniques et le *framework* technique à mettre en place pour supporter l'exécution des composants métier. L'architecture technique décrit l'architecture matérielle et réseau du système de production. On y décrit la répartition des logiciels sur les machines physiques ainsi que les protocoles de communication des différents nœuds de l'architecture. Cette élaboration est accompagnée par l'identification des règles de conception et des patrons dont l'objectif est de pallier aux manques de la technologie.
- **La branche centrale** est une **intégration des branches fonctionnelle et technique**. Elle permet de réaliser une conception intégrant le modèle d'analyse dans l'architecture applicative de manière à obtenir un modèle de conception traçant les composants du système à développer. Ce modèle de conception est par la suite traduit dans un langage de programmation en utilisant les outils de développement choisis dans la branche technique. Les tests unitaires des composants sont réalisés au fur et à mesure que leurs unités de code sont développées. La phase de recette permet de tester fonctionnellement l'application en suivant le cahier des charges préalablement rédigé. La phase de déploiement consiste enfin à mettre en production l'application testée techniquement et fonctionnellement.

¹ <http://www.umanis.com>

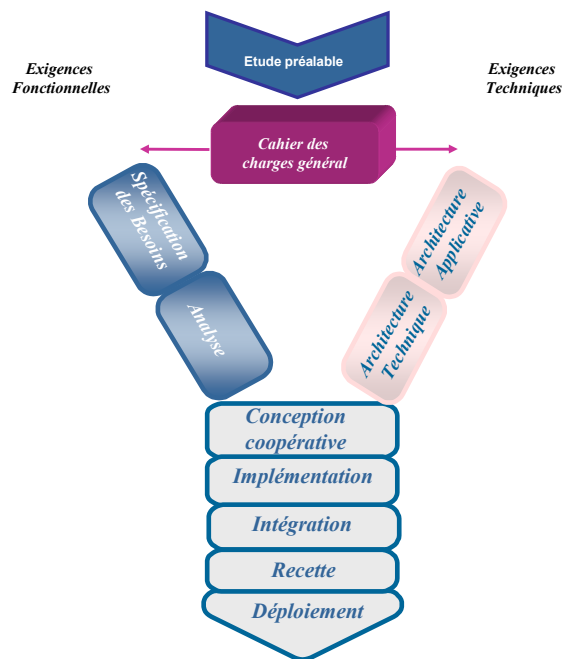


Figure 1 : Cycle de vie en Y de la méthode Symphony

Les résultats obtenus dans les lots 2, 3 et 4 du projet TRAFIC nous ont conduit à modifier la méthode Symphony en l'augmentant de nouvelles phases spécifiques aux SIT, en particulier l'étude de la coopération entre systèmes d'information transport qui joue un rôle prépondérant dans le cadre de l'intermodalité des transports où de nouveaux systèmes doivent pouvoir coopérer avec les systèmes anciens. Ainsi, la nouvelle méthode UP-TRAFIC (cf. Figure 2) est composée d'un ensemble de phases dans lesquelles sont intégrés les patrons, les composants et les technologies développés dans le cadre du projet TRAFIC :

- les composants métiers développés dans le cadre du lot 2 constituent comme dans la méthode Symphony, le mécanisme essentiel utilisé lors des phases de spécification des besoins et d'analyse,
- les approches coopératives développées dans le cadre du lot 3 font l'objet de nouvelles phases :
 - la phase d'analyse de la coopération où les patrons d'analyse de la coopération présentent les différents modes possibles de coopération,
 - la phase d'architecture coopérative intégrant les patrons d'architecture asynchrones, les patrons de protocoles et les patrons synchrones,
 - la phase de conception de Symphony est augmentée et renommée en phase de conception coopérative utilisant les patrons de conception qui représentent le résultat de l'intégration des composants métiers dans les architectures spécifiées dans les patrons d'architectures,
 - enfin la phase d'implantation utilise des patrons de support technique dédiés aux plates-formes distribuées telles que J2EE. En particulier, dans le cadre de la société ACTOLL, elle permet d'introduire en entrée du générateur les composants métiers issus de la phase d'analyse.
- les travaux sur les requêtes visuelles et sur l'accès progressif développés dans le cadre du lot 4 constituent des composants techniques particuliers au domaine des SIT et sont utilisables lors de la phase d'architecture applicative.

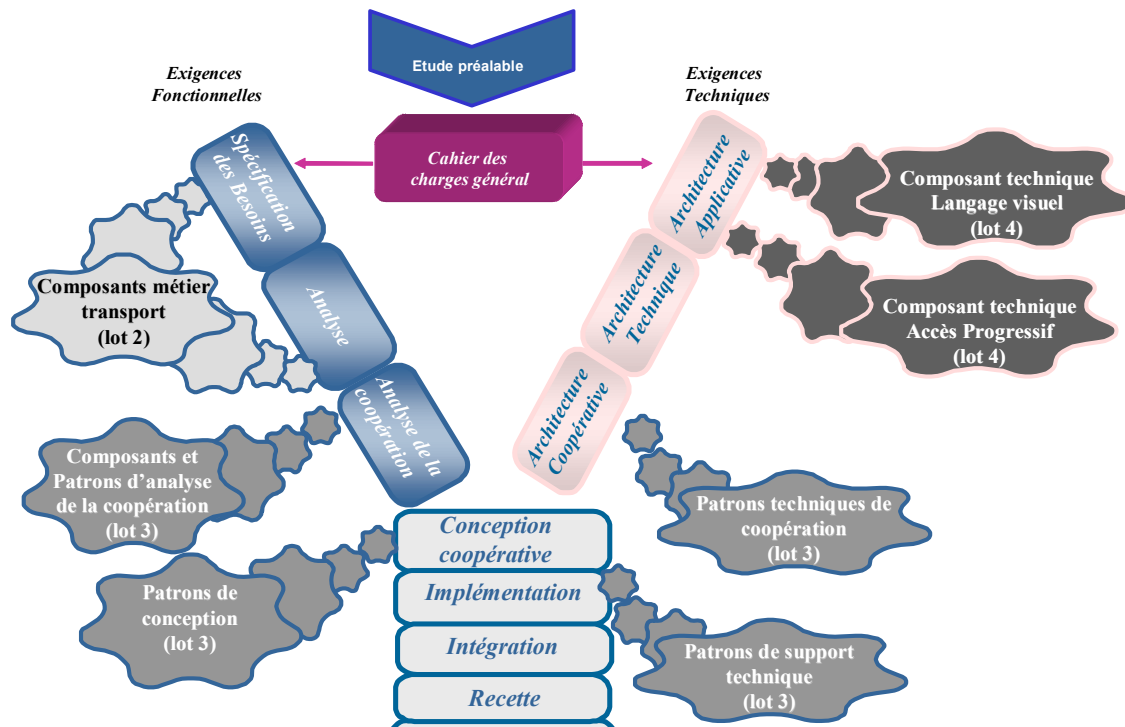


Figure 2 : démarche UP-TRAFIC, positionnement des lots

Comme la démarche Symphony, la démarche UP-TRAFIC s'appuie sur le langage unifié de modélisation UML. Elle a pour but de développer des applications évolutives dans le domaine des systèmes d'information transport et de réduire leurs coûts de développement, de déploiement, d'exploitation et de mise à jour tout en mettant un fort accent sur l'étude de la coopération et de l'interopérabilité entre les systèmes.

Les sections correspondantes sont les suivantes :

- **démarche et processus de développement** : présentation de la démarche, du processus de développement et de son cycle de vie en Y,
- **phases de spécification des besoins, d'analyse et d'analyse de la coopération** : description des phases et des activités formalisant les exigences métier,
- **phases d'architecture applicative, d'architecture technique et d'architecture coopérative** : description des phases et des activités formalisant les exigences techniques, en particulier au niveau applicatif, le besoin ou non d'un langage d'interrogation visuelle ou d'un accès progressif à l'information, et d'un point de vue technique,
- **phases de conception coopérative et d'implantation** : description de la phase et des activités associées.

Partie A Démarche et processus de développement

1 Processus de développement en Y

UP-TRAFIC propose un processus de développement itératif, organisé au travers d'un certain nombre de cycles de développement en Y (cf. Figure 2). Chaque cycle de développement, appelé aussi **itération**, est centré sur un processus métier ou le raffinement d'un processus métier identifié, décrit et pondéré lors de la phase d'étude préalable. Un cycle de développement est une instanciation de la succession des différentes **phases** du processus.

2 Démarche itérative incrémentale

Un cycle de développement peut être lui-même décomposé en d'autres cycles si le processus métier qu'il représente est lui-même décomposable. Dans ce cas, selon leur complexité et leur priorité, de nouvelles étapes d'étude préalable sont initiées avec en entrée un processus métier à décomposer. Une itération est donc centrée sur un processus métier ou le raffinement d'un processus métier. L'enchaînement de plusieurs cycles de développement aboutit progressivement au produit logiciel final.

Le processus global itératif de la démarche UP-TRAFIC relève comme celui de la méthode Symphony d'un nouveau type de cycle de vie généralisant le cycle de développement en Y : le **modèle en flocons**. La Figure 3 présente de manière détaillée la généralisation de ce modèle au processus de développement UP-TRAFIC. Le développement est ainsi fondé sur la croissance et l'affinement successif d'un système par le biais d'itérations et d'**incrément**s multiples. L'intégration de la vision métier s'effectue dans chaque itération de développement.

Le résultat de chaque itération est un système testé, intégré et exécutable, mais incomplet et non encore prêt à être livré (un incrément). Le déploiement dans un environnement de production nécessite plusieurs itérations. Le résultat n'est pas non plus un prototype². En revanche, il représente un sous-système du système final et répond à un sous-ensemble des spécifications du système.

Pratiquement, pour chaque itération, il convient de choisir un ensemble d'exigences métier puis de les concevoir, de les implémenter et de les tester rapidement. Lors des premières itérations, les choix fonctionnels et techniques ne correspondent pas forcément au résultat final souhaité. Mais le fait de commencer la réalisation avant que les spécifications soient complètement finalisées ou que la totalité de la conception ait été pensée permet d'obtenir rapidement un retour d'évaluation des utilisateurs, des développeurs et aussi des testeurs. Le cas échéant, ce retour d'information permet la remise en cause de l'exactitude des spécifications ou de la conception.

² Le développement itératif n'est pas une activité de prototypage.

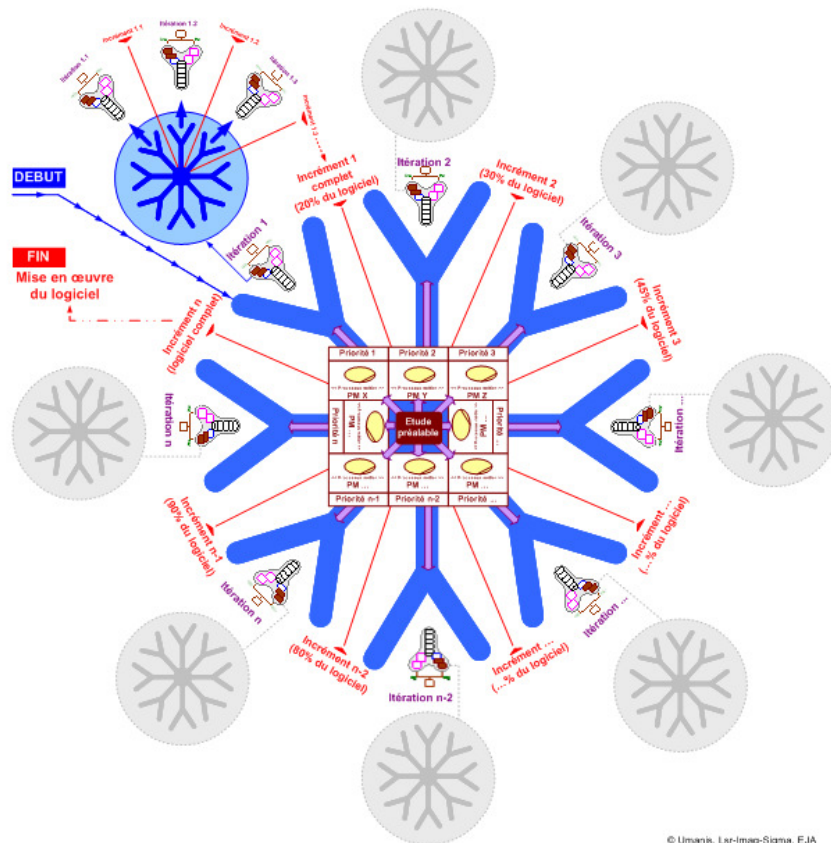


Figure 3 : démarche UP-TRAFFIC, processus de développement itératif incrémental

L'avancement du projet dépend des retours d'évaluations des différents sous-systèmes résultats issus des différentes itérations. L'utilisateur final dispose ainsi de la possibilité d'évaluer rapidement le système partiel et de donner ou non son approbation selon qu'il s'agisse ou non de fonctionnalités attendues du système final. Ainsi son avis intervenant en amont du processus constitue un bon moyen de découvrir en début de projet ce qui est réellement important pour lui.

3 Démarche pilotée par les cas d'utilisation

La démarche est orientée utilisateur et pilotée par les cas d'utilisation. En effet, elle intègre les besoins et les usages réels des utilisateurs dès les phases amont du développement. Ces besoins sont alors modélisés par des cas d'utilisation qui assurent la cohésion des activités et guident le processus de développement dans son ensemble. Les modèles de cas d'utilisation décrivent les fonctionnalités complètes du système. A partir de ces modèles, les concepteurs créent une série de modèles de conception et de développement réalisant les cas d'utilisation. Les testeurs vérifient si les composants métier logiciels développés respectent correctement les cas d'utilisation. Les besoins des utilisateurs sont donc prioritairement traités dans la branche gauche et la branche droite du modèle en Y. Les cas d'utilisation constituent un mécanisme essentiel de traçabilité entre modèles.

Pour apporter des éléments de réponse aux problèmes et contraintes d'évolution du système et de ses besoins métier et technique, l'équipe de développement doit être en mesure de naviguer dans l'ensemble du système et de pouvoir remonter d'un modèle à un autre pour faciliter la propagation des modifications. Cette deuxième forme de traçabilité est garantie par les liens entre les différents éléments des modèles.

4 Démarche orientée composant métier

La démarche UP-TRAFIC est orientée Composant Métier (CM) car l'application est vue, tant au niveau conceptuel que logiciel, comme un assemblage de CM indépendants et interconnectés. Cette pratique garantit une bonne modularité des spécifications et facilite leur réutilisation. Les CM sont représentés selon le modèle conceptuel de composants métier Symphony, tel que présenté dans le livrable 2.2 du lot 2.

Vis-à-vis d'un acteur extérieur, un CM correspond à une entité qu'il peut manipuler. Cette entité est apte à exécuter des opérations et maîtrise des informations qui lui permettent d'assumer des opérations qu'elle s'engage à réaliser. Les informations forment un réseau de concepts qui a un sens pour le métier.

Les parties suivantes se concentrent sur chaque branche du cycle de vie en Y de la démarche UP-TRAFIC, en commençant par les phases de modélisation métier, d'analyse et d'analyse de la coopération (branche fonctionnelle), puis les phases d'architecture applicative, d'architecture technique et d'architecture coopérative (branche technique) et en terminant par la phase de conception coopérative (branche centrale). Pour des raisons de place, les phases succédant à la conception ne sont pas décrites dans ce livrable.

Partie B Phases de spécification des besoins, d'analyse et d'analyse de la coopération

L'objectif de Symphony en tant que démarche à base de composants métier est de conduire les acteurs du processus de développement du système à produire une structure (cartographie) de CM dont le couplage est le plus faible possible. Ces CM doivent supporter l'ensemble des services attendus par le métier générateur de la demande de développement. ».

Dans notre exemple, le découpage fonctionnel global permet de recenser deux PM : « Gestion du Voyage » et « Gestion des Usagers ».

- Gestion des usagers : opérations relatives aux personnes et à leurs cartes qui ne concernent pas le transport directement (achat de titre de transport, achat de carte, etc.)
- Gestion du voyage : opérations ayant trait au voyage, validations des titres de transports, comptage des validations, etc.

Ce découpage fonctionnel global est issu de notre collaboration avec la société ACTOLL. Nous nous intéressons plus particulièrement au PM « Gestion des Usagers » de priorité 1. Ce PM traite des opérations ayant trait aux usagers et à leurs titres de transport (achat de titre, achat de cartes, etc.).

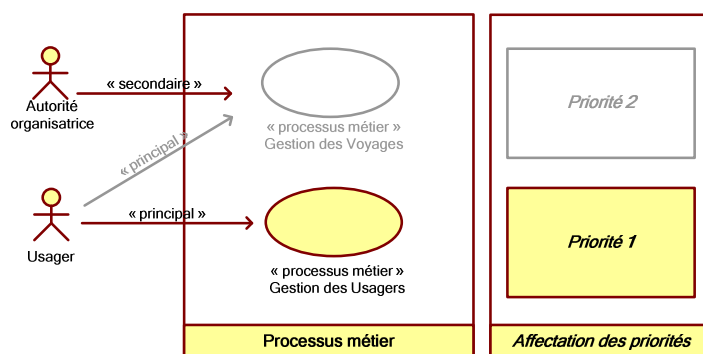


Figure 4 : étude préalable, modèle des processus métier

Un des buts de la phase d'étude préalable est aussi d'identifier les acteurs internes et externes afin de comprendre la finalité attendue par chaque acteur du système. Il s'agit alors de répondre aux questions suivantes : pour qui, pour quoi, dans quelles conditions, comment et quels résultats ? La Figure 5 montre la classification des acteurs internes et externes obtenue en réponse à ces questions. Elle identifie par exemple l'utilisateur comme acteur externe déclencheur des processus Gestion des Usagers et Gestion des Voyages, alors que l'autorité organisatrice qui souhaite obtenir des informations statistiques sur les voyages est un acteur externe secondaire. Le guichetier, les validateurs dans les bus et les transporteurs sont considérés comme des acteurs internes qui devront interagir avec le système informatique à développer.

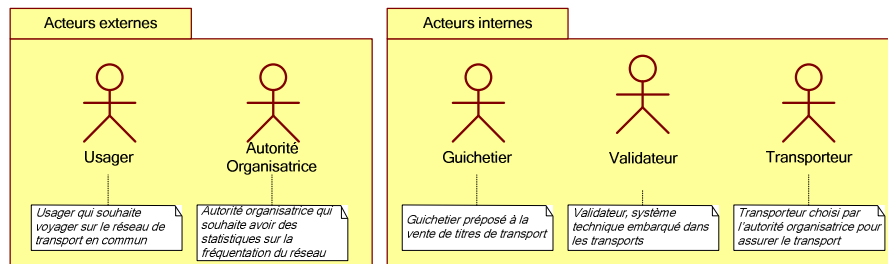


Figure 5 : étude préalable, diagramme des acteurs métier

1 Spécification des besoins

Cette phase correspond à l'identification des finalités de chaque PM identifié précédemment. Il s'agit de décrire le « quoi », de se focaliser sur les objectifs et contraintes du processus lié aux acteurs externes en faisant abstraction des contraintes d'organisation et des acteurs internes au processus.

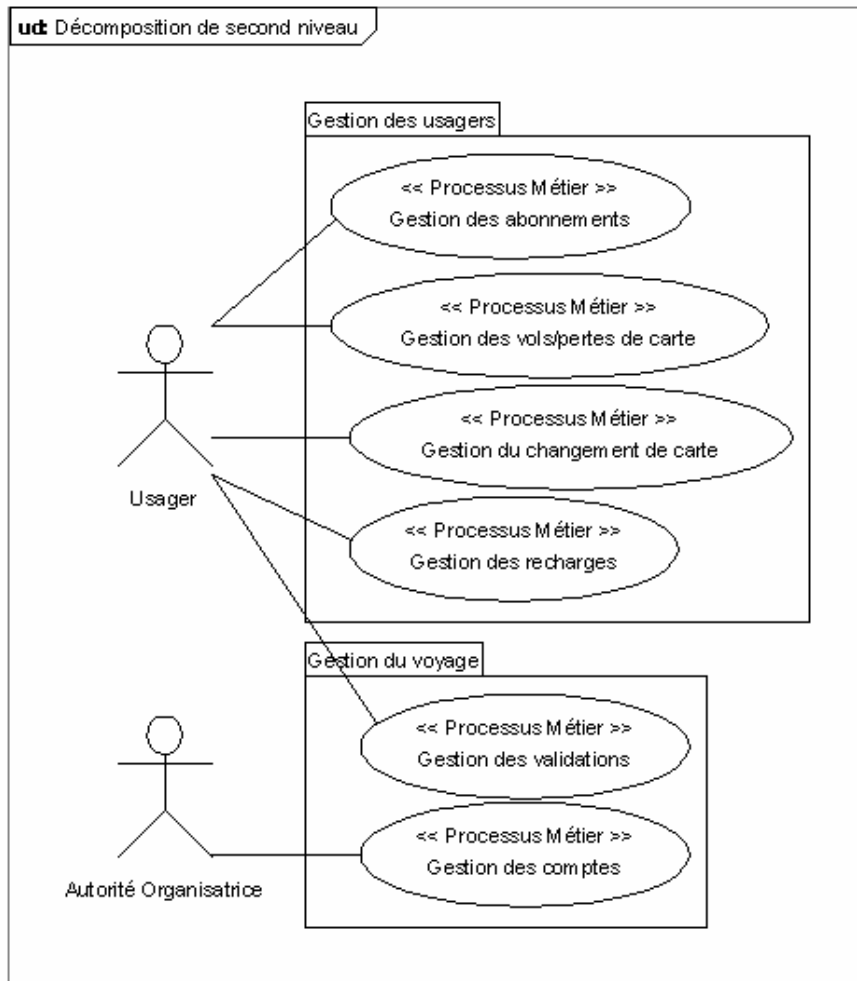
La spécification conceptuelle des besoins comprend les activités suivantes pour chaque PM :

- **description conceptuelle de chaque PM sous la forme d'un scénario principal³** mettant en évidence les interactions entre acteurs externes et PM, les pré et post conditions,
- **description conceptuelle formelle de chaque PM** à l'aide d'un diagramme de séquence conforme au scénario établi dans l'activité précédente,
- **décomposition en Processus Métier Composants (PMC)** le cas échéant.

Sur notre exemple, le but est donc de décomposer les deux processus métiers définis précédemment, c'est à dire la Gestion des Usagers et la Gestion du Voyage en plusieurs PMC (Processus Métier Composant). La Figure 6 montre les processus métiers composants permettant de répondre à l'ensemble des opérations nécessaires à la gestion des usagers et des voyages dans un SIT.

- Gestion des Usagers :
 - Gestion des abonnements : concerne les achats de nouveaux abonnements pour des nouveaux ou anciens clients ne possédant pas de carte.
 - Gestion des vols/pertes de carte : éventuellement un changement de carte avec remplacement des titres présents sur l'ancienne carte et qui n'ont pas encore été validés.
 - Gestion du changement de carte lorsque la carte n'est plus valide (une fois la date de validité expirée ou en cas de dysfonctionnement).
 - Gestion des achats de titre : l'opération la plus courante pour un S.I.T après la validation.
- Gestion du Voyage :
 - Gestion des validations : simple validation, avec toutes les modifications que cela implique (aussi bien sur la carte qu'au niveau de l'historisation).
 - Gestion des comptes : ce processus métier composant est utilisé pour connaître le nombre d'usagers ayant utilisé une borne précise, ceci afin de gérer les remboursements entre les différentes sociétés de transports.

³ Un scénario principal est le cas nominal. Il s'agit souvent du chemin normal d'exécution du processus [KET 01]. Il n'implique pas les flux alternatifs ou les flux d'erreur.



Created with Poseidon for UML Community Edition. Not for Commercial Use.

Figure 6 : Processus métiers composants - Gestion des Usagers et Gestion des Voyages

L'étape suivante consiste pour chaque processus métier composant, à définir et à représenter dans un diagramme de séquence, son événement déclencheur, les acteurs externes participants, les pré-conditions, un scénario d'exécution, et les post-conditions.

Par exemple, le PMC Gestion des abonnements est défini de la façon suivante :

Gestion des abonnements

Événement déclencheur : Emission de la Demande d'Abonnement.

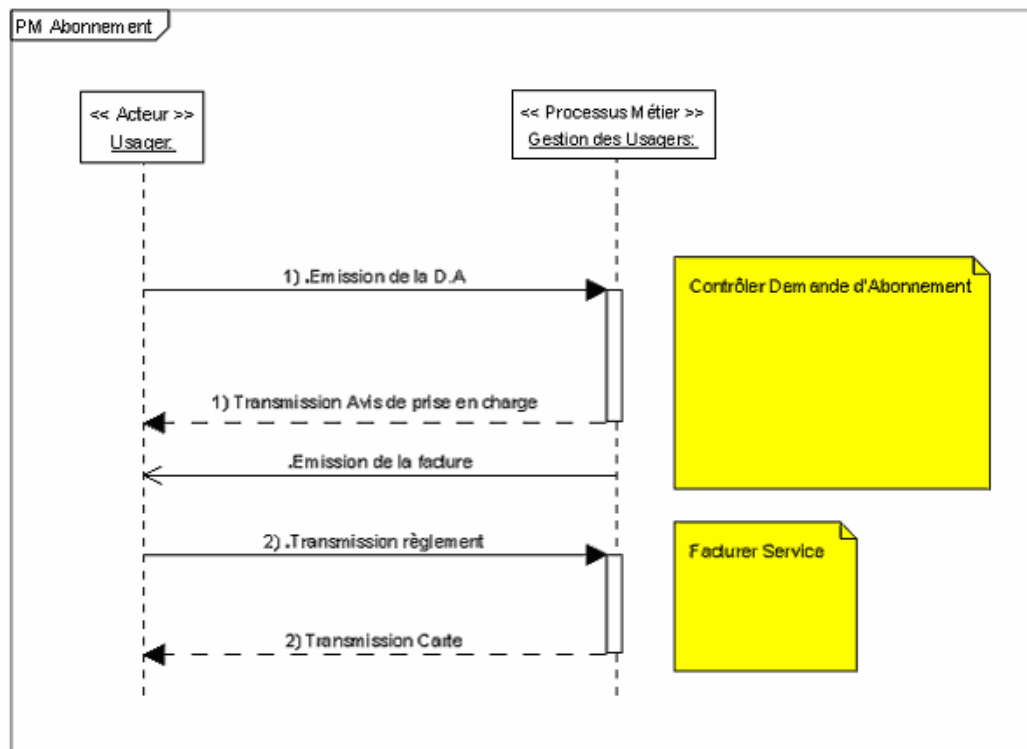
Acteur Externe : Usager.

Pré-condition : aucune.

Scénario :

- *L'utilisateur émet une Demande d'Abonnement.*
- *Le PGU (Processus de Gestion des Usagers) émet un avis de prise en charge.*
- *Le PGU émet la facture.*
- *Le PGU transmet la carte.*

La personne est maintenant considérée comme Usager et possède une carte d'abonnement valide.



Comme nous le voyons sur le diagramme de séquence, le processus métier composant Gestion des Abonnements peut lui-même être décomposé en deux processus métiers composants :

Contrôler Demande d'Abonnement :

Événement déclencheur : *Emission d'une Demande d'Abonnement.*

Acteur externe : *Usager.*

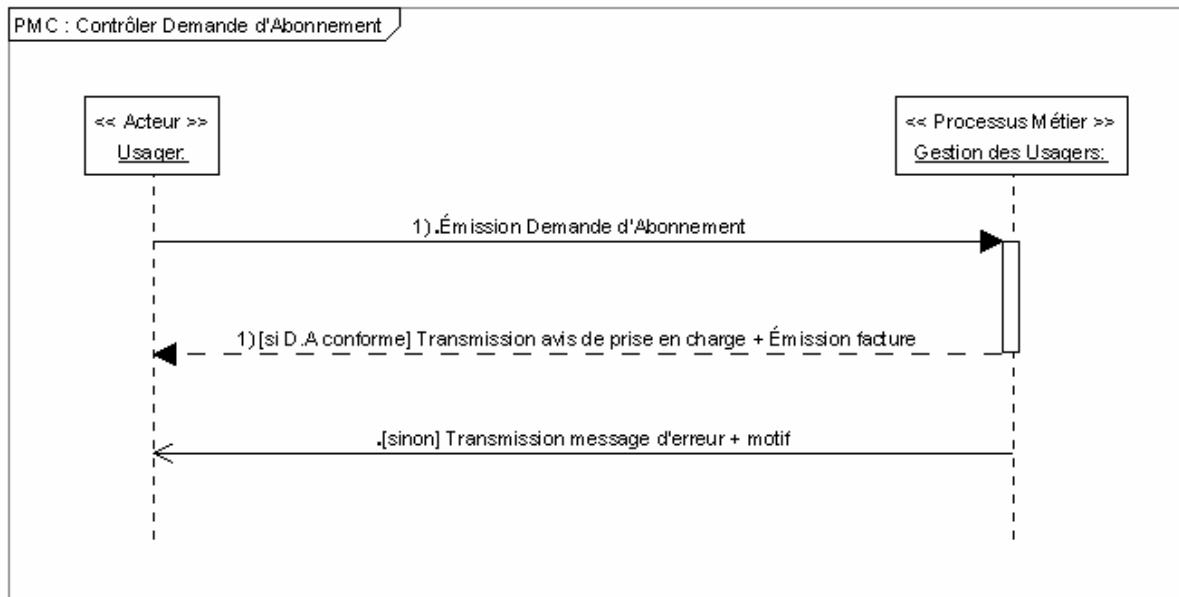
Pré-condition : *Aucune.*

Scénario :

- *Emission d'une Demande d'Abonnement (D.A.)*
- *Vérifier que la D.A est correcte et que l'utilisateur ne possède pas déjà une carte.*
 - o *Si les informations sont erronées ou que l'utilisateur possède déjà une carte : refusé.*
 - o *Sinon : transmettre à l'assuré une facture pour le règlement de sa carte.*

Post-conditions :

- *Si l'utilisateur n'a pas de contrat et si les informations sont correctes, on émet une facture*
- *Sinon, on lui transmet un avis de refus en expliquant le motif.*



Created with Poseidon for UML Community Edition. Not for Commercial Use.

Facturer service d'abonnement

Événement déclencheur : *Transmission règlement.*

Acteur externe : *Usager.*

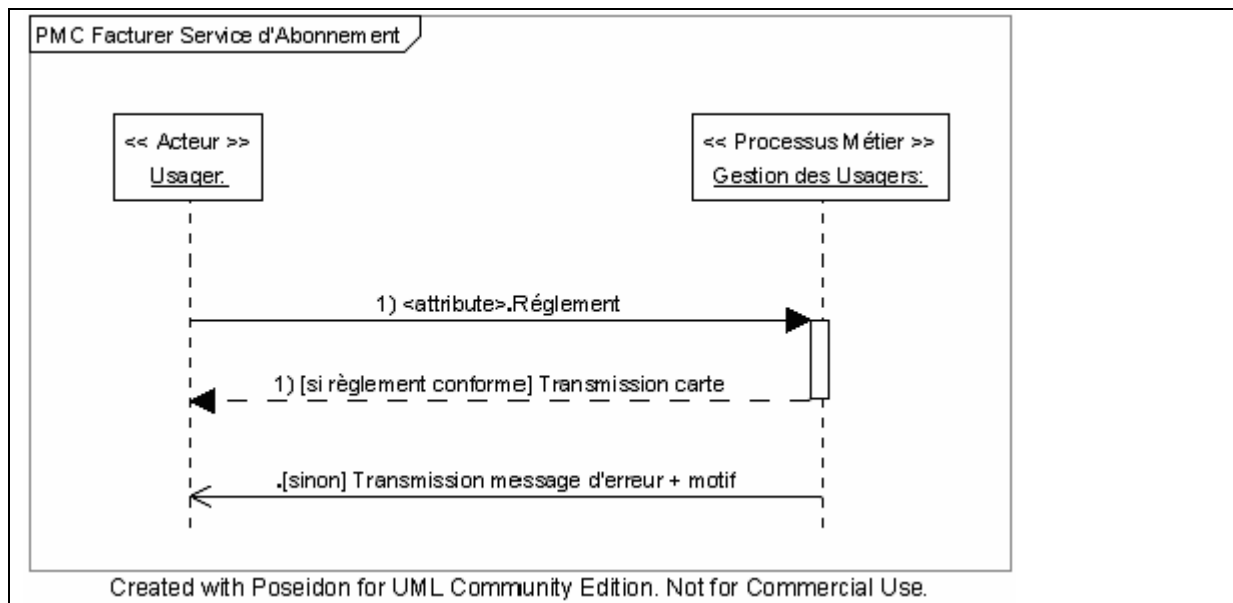
Pré-condition : *L'utilisateur a reçu sa facture.*

Scénario :

- *L'utilisateur règle la facture.*
 - o *Si le paiement n'est pas correct : on transmet à l'utilisateur un refus ainsi que son motif.*
 - o *Sinon, on accède à sa demande (rechargement de carte, transmission de sa carte d'abonnement, etc.)*

Post-conditions :

- *L'utilisateur a reçu ce qu'il désirait ou un avis de refus.*



La décomposition en PMC est régie par les flux externes. Un PMC élémentaire est donc ininterrompible : il est déclenché par un flux externe et se déroule sans interruption jusqu'à émission de résultats. Cette décomposition est similaire à celle préconisée dans Merise/2 [PAN 94] pour déterminer les activités dans un modèle de flux conceptuels ou les opérations d'un Modèle Conceptuel de Traitements (MCT). Dans le cas d'un évènement temporel, l'« horloge » correspond à un acteur externe.

Chaque PMC est ensuite décrit, en spécifiant son scénario, ses pré et post conditions ainsi que son diagramme de séquence en tenant compte, cette fois-ci, des différents flux alternatifs. Cette activité est suivie par un bilan permettant de spécifier si les PMC sont eux mêmes décomposables ou pas. Si c'est le cas et que le PMC est assez complexe, une nouvelle phase d'étude préalable complète est démarrée. En revanche, pour un PMC décomposable non complexe, on reprend l'activité de description conceptuelle formelle du PMC pour effectuer sa décomposition. Dans le cas d'un PMC non décomposable, on enchaîne sur la phase de spécification organisationnelle des besoins.

2 Spécification organisationnelle des besoins

Dans cette phase, il s'agit de décrire « qui fait quoi et quand ». L'objectif est de prendre en compte les choix de l'organisation en faisant intervenir les acteurs internes identifiés dans la phase d'étude préalable.

La spécification organisationnelle des besoins comprend les activités suivantes pour chaque PMC :

- **décomposition organisationnelle des PMC** en fonction de la répartition des processus entre acteurs internes et de la distinction entre les processus informatisés et ceux manuels,
- **génération et mise en relation des cas d'utilisation** à partir de Processus Métier Informatisés (PMI) et sous la forme d'un diagramme de cas d'utilisation,
- **établissement de la cartographie des CM Processus** réalisant les cas d'utilisation.

Dans notre exemple, la décomposition organisationnelle des deux PMC « Contrôler Demande d'Abonnement » et « Facturer Service » donne lieu à un diagramme d'activités (cf. figure suivante) montrant l'acteur interne Guichet ainsi que l'enchaînement des processus métiers informatisés et manuels.

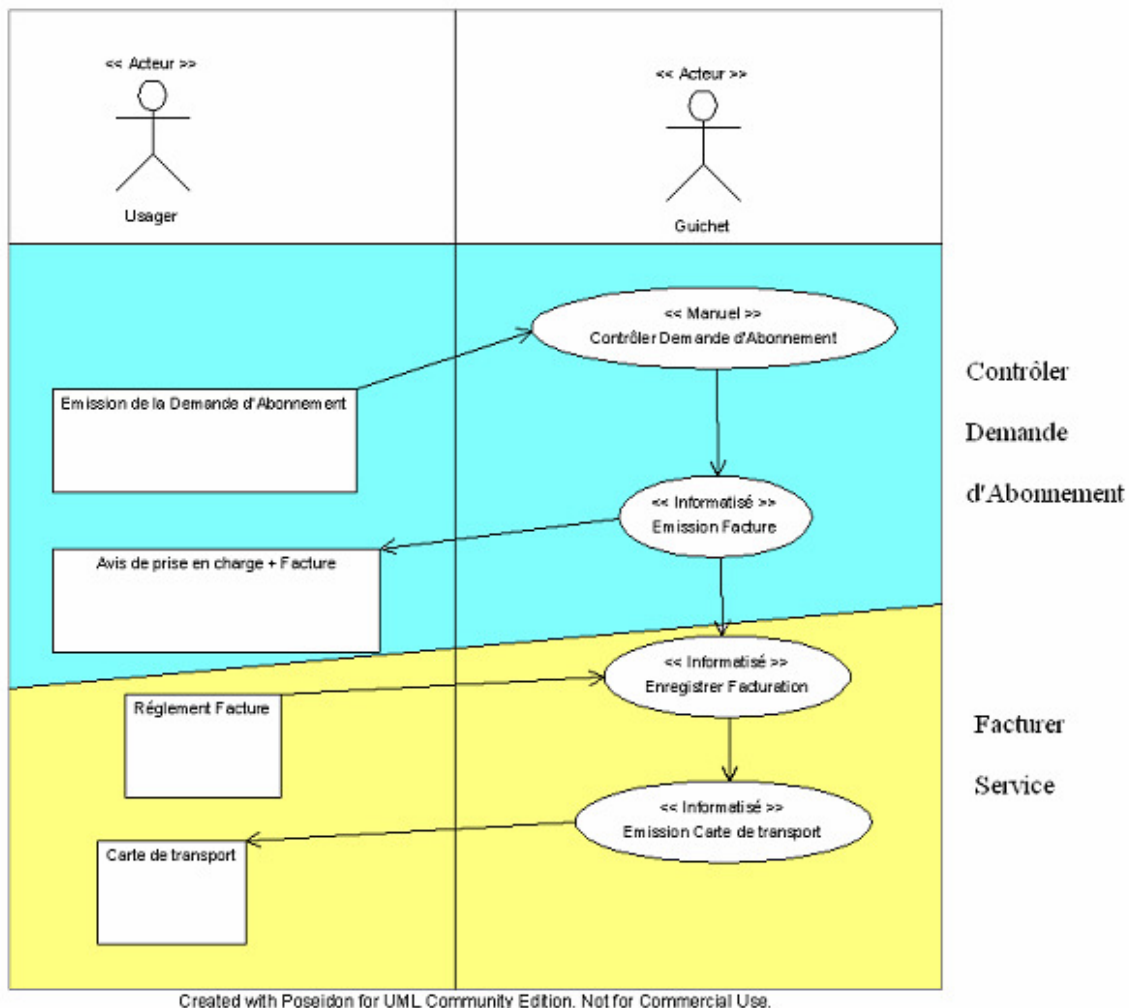


Figure 7 : spécification organisationnelle des besoins, décomposition organisationnelle de PMC

Le diagramme des cas d'utilisation résultant (cf. Figure 8) prend en compte tous les PMI (un cas d'utilisation pour chaque processus). La figure suivante montre l'ensemble des processus informatisés réalisables par les acteurs internes pour les deux PM Gestion des Usagers et Gestion des Voyages. L'activité « Emission Carte de Transport » constitue par exemple un cas d'utilisation déclenché par l'acteur (interne) « Guichet ».

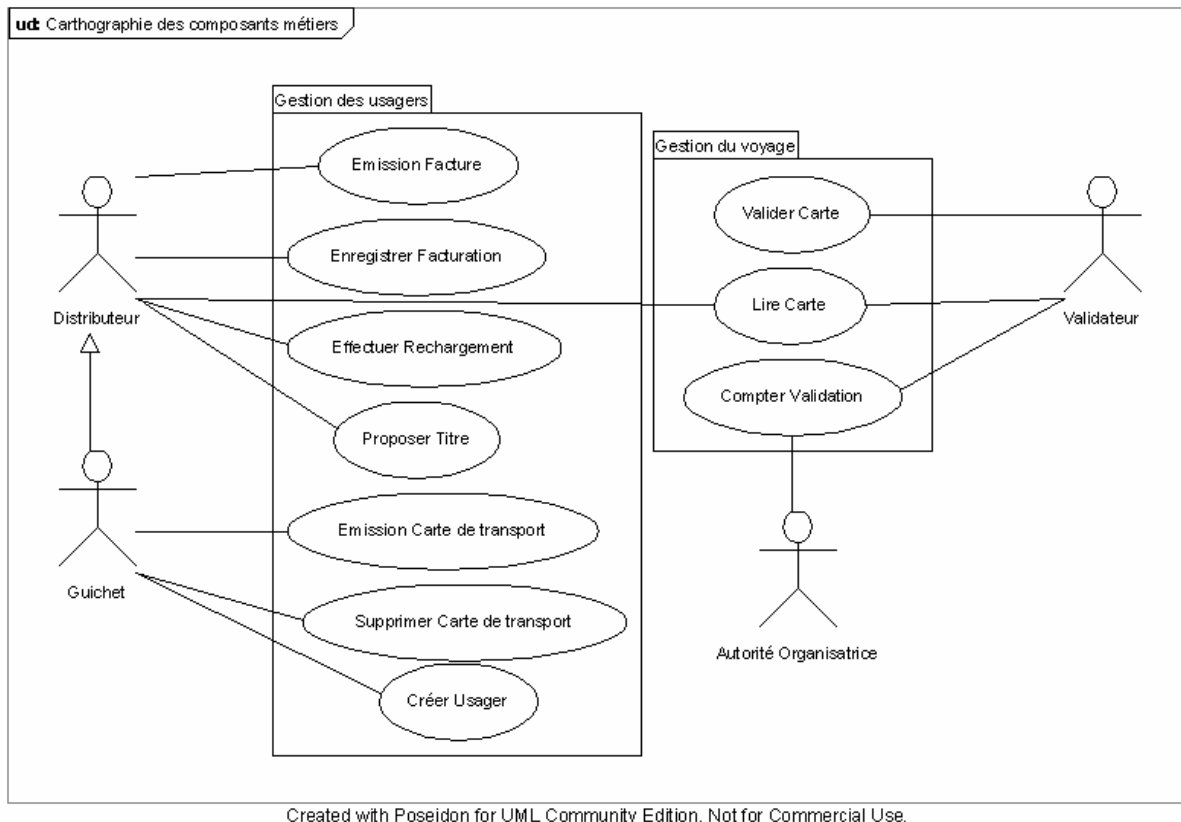


Figure 8 : spécification organisationnelle des besoins, cas d'utilisation

3 Analyse

L'objectif de la phase d'analyse consiste à mettre en évidence ce que doit faire le système sans tenir compte de la manière de le réaliser. Il est question de modéliser le comportement et la structure des CM Processus, Entité et Données de référence, sachant que les CM déjà identifiés correspondent plus à des CM Processus, chargés de réaliser un certain nombre de cas d'utilisation. La phase d'analyse permet d'identifier et de modéliser de nouveaux CM de type Entité et Données de référence. Le modèle d'analyse résulte d'une analyse dynamique (orientée diagrammes dynamiques UML) puis d'une analyse structurelle (orientée diagramme statiques UML).

Pour l'analyse dynamique, les activités sont les suivantes :

- formalisation de haut niveau du scénario principal de chaque cas d'utilisation fonctionnel,
- formalisation détaillée du scénario principal et des scénarii secondaires de chaque cas d'utilisation,
- description des cycles de vie des CM,
- modélisation du comportement⁴ (pour les scénarii complexes),
- définition des contrats d'opérations.

Les activités de l'analyse structurelle sont les suivantes :

- modélisation des interfaces des CM (classe « interface » de chaque CM),

⁴ La modélisation du comportement concerne les CM Processus complexes. Les activités complexes sont alors décomposées en sous-activités dont l'enchaînement des activités (séquence, conditions, parallélisation) doit être précisé et formalisé (par des diagrammes d'activités par exemple).

- modélisation des collaborateurs (classes « rôle » de chaque CM),
- description des CM (en langage naturel),
- établissement des relations inter CM (via les classes « rôle »).
- élaboration de la cartographie des CM (Processus, Entité et Données de référence).

Par souci de simplicité, nous ne présentons pas toutes les étapes qui nous ont permis d'aboutir à la cartographie des 9 composants métiers mis en œuvre dans les processus métiers Gestion des Usagers et Gestion des Voyages. Ces 9 composants ont été présentés en détail dans les livrables 2.2 et 2.3. Nous rappelons uniquement dans le tableau ci-dessous leur brève description.

Composant	Description
<i>Client</i>	<i>Composant principal, désigne les personnes clientes d'une société de transport.</i>
<i>Usager</i>	<i>Désigne l'utilisateur d'une société de transport, il possède une carte de transport valide.</i>
<i>Employé</i>	<i>Représente tout employé de la société de transport.</i>
<i>Contrat</i>	<i>Concerne l'intégralité des contrats effectués entre un usager et la société de transport.</i>
<i>Type_Contrat</i>	<i>Représente les différents types de contrats existants.</i>
<i>Type_Produit</i>	<i>Contient les informations relatives à un type de produit. Par exemple un ticket unitaire est consommable alors qu'un abonnement est non-consommable.</i>
<i>Carte</i>	<i>Représente la carte à puce de l'utilisateur</i>
<i>Support_Produit</i>	<i>Contient les produits possédés par l'utilisateur sur carte.</i>
<i>Historisation</i>	<i>Permet de garder une trace de toutes les transactions ou modifications ayant eu lieu. Il peut retenir tous les types d'informations, aussi bien les changements de nom d'un client que les validations effectuées par une carte.</i>

4 Analyse de la coopération

Cette phase utilise les patrons et les composants d'analyse de la coopération définis dans le cadre des livrables 3.2 et 3.3.

Partie C Phases d'architecture applicative, d'architecture technique et d'architecture coopérative

Cette partie présente les règles de bonne pratique préconisées par UP-TRAFIC pour construire une architecture applicative et technique à base de composants. Ces règles de bonne pratique sont pour la plupart proposées par la méthode Symphony, mais certains des composants techniques utilisés sont spécifiques à des systèmes d'information transport, et donc introduits explicitement dans UP-TRAFIC : il s'agit des composants d'accès progressif à l'information et d'interrogation visuelle.

1 Architecture applicative

L'objectif de l'architecture applicative est de spécifier les composants techniques à mettre en place pour supporter l'exécution des composants métier. Cette phase est réalisée en parallèle avec les exigences fonctionnelles. Elle permet d'anticiper les développements des composants génériques (ne contenant aucun savoir métier).

Elle représente la spécification des composants techniques et de leur collaboration les uns avec les autres.

Les activités de cette phase sont les suivantes :

- décrire les différentes couches applicatives,
- recenser et cartographier les composants techniques,
- décrire les composants et classes techniques,
- définir les règles de conception et de transformation.

1.1 Description des couches applicatives

La spécification de ces couches consiste à appliquer des patrons de conception (tels que Façade, *Layers* par exemple) pour spécifier l'ensemble des couches applicatives de l'architecture logicielle. Une architecture multi-couches permet alors de répondre à des objectifs qualitatifs permettant d'atteindre de hauts niveaux de productivité, de maintenance et de réutilisation grâce à la décomposition du système en sous systèmes.

La relation de dépendance entre les différentes couches signifie qu'un composant de la couche inférieure ne peut pas accéder à un composant de la couche supérieure : on parle de découplage. Cette notion correspond au patron de conception *Layers* et permet :

- d'attribuer aux équipes de réalisation des responsabilités sur le développement de chaque couche,
- de masquer les détails d'implantation d'une couche, ce qui permet de ne pas être affecté par son évolution à venir.

Chaque couche peut être modélisée, développée et testée individuellement. Cette architecture sépare la logique de présentation, la logique applicative, la logique transactionnelle et de persistance (mise à jour des bases de données).

Cette architecture multi-couches est aussi celle mise en œuvre dans la société ACTOLL. C'est pourquoi, dans le cadre de la démarche UP-TRAFIC destinée aux SIT, nous préconisons d'utiliser une architecture multi-couches telle que celle présentée dans la Figure 9 qui peut

donc être mise en place dans de nombreux projets de développement de systèmes d'information transport.

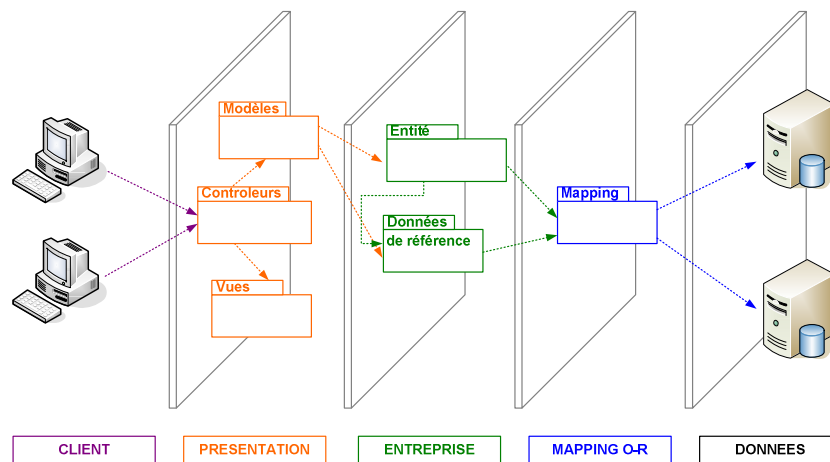


Figure 9 : architecture multi-couche préconisée par UP-TRAFIC

Les couches sont détaillées de la manière suivante :

- **La couche Client** est relative aux « composants » (navigateur Internet par exemple) permettant aux utilisateurs d'utiliser l'application.
- **La couche Présentation** s'assure de la définition des composants permettant de gérer la présentation et l'interaction avec les couches inférieures.
- **La couche Entreprise** représente l'ensemble des **objets métier Entités et Données**. Cette couche représente les entités qui seront persistantes, donc projetées dans la base de données.
- **La couche Projection (*mapping*) Objet-Relationnel** permet de transformer les objets métier Entité et Données de référence en requêtes SQL. Cette couche peut être prise en charge par un *framework* (open source, commercial ou « maison »).
- **La couche Données** est généralement relative au SGBDr et aux différentes bases de données dans lesquelles sont sérialisés les objets métier de la couche Entreprise (via la couche Projection Objet-Relationnel).

1.2 Etablissement de la cartographie des composants techniques

Une fois le cadre architectural posé, l'objectif est de définir la cartographie des composants techniques. Il s'agit en particulier de choisir des composants techniques et de décrire leurs interactions. Cette activité constitue le cœur de la phase d'architecture applicative car il faut définir les technologies à mettre en place en convergence avec les besoins fonctionnels.

Dans le cadre des applications n-tiers, Symphony recense plusieurs catégories de composants nécessaires pour le bon fonctionnement et la maintenance de ces applications. Les catégories présentées ci-dessous sont souvent présentes dans les applications de gestion :

- composants ou *frameworks* de présentation (Struts, JSF par exemple),
- composants de projection objet relationnel (LiDO, TopLink, par exemple),
- composants d'authentification (JAAS par exemple),
- composants de notification (JavaMail, NetSize, par exemple),
- composants d'édition (FOP par exemple),
- composants de recherche (Lucène par exemple),
- composants de gestion des *pools* de connexions.

Dans le cadre des applications systèmes d'information transport, UP-TRAFIC recense deux catégories supplémentaires de composants utiles pour le développement d'un système d'information transport :

- composants d'interrogation visuelle (langage LVIS par exemple, développé dans le cadre du lot 4),
- composants d'accès progressif à l'information (développé dans le cadre du lot 4).

Cette cartographie est souvent décrite par des schémas spécifiques du fait d'un manque dans la notation des diagrammes de composants d'UML. La **Erreur ! Source du renvoi introuvable.** illustre un exemple de cartographie.

1.3 Description des composants et des classes techniques

L'activité suivante consiste à détailler ces composants techniques, leurs interfaces en particulier, les classes spécialisables, si nécessaire leur fonctionnement interne notamment pour les *frameworks* techniques. En effet, certains composants nécessitent un enrobage pour faciliter leur utilisation dans le cadre des développements. Par exemple, l'utilisation du composant FOP pour les éditions nécessite la définition de façades pour faciliter l'appel de la commande d'édition (comprenant par exemple la recherche automatique de la feuille de style, les facilités de génération du XML, ...). La conception de ces classes d'enrobage se fera dans la phase de conception. Nous retrouvons des situations similaires pour l'utilisation des composants d'accès aux bases de données (accès à une connexion, exécution d'une requête, ...). Même l'utilisation d'un outil de projection nécessite un enrobage certes très simple, mais utile pour éviter la redondance de code.

Dans la réalisation de l'architecture applicative, l'architecte applicatif doit également définir le cadre d'utilisation du composant (d'origine ou spécialisé) dans l'application. Cette description peut se faire par des exemples concrets d'utilisation. Dans le cas d'un composant complexe (ou *framework* technique), une documentation doit être annexée au dossier d'architecture applicative.

1.4 Définition des règles de conception et de transformation

Une des dernières activités de la phase d'architecture applicative dans le cycle en Y consiste à spécifier et / ou identifier les règles de conception et de transformation applicables dans la conception de l'application telles que :

- la transformation du modèle d'analyse en modèle de conception,
- la génération des interfaces des composants distribués,
- la génération des classes de composants (inutile si la génération est réalisée par un outil),
- la génération du schéma relationnel,
- la traduction des modèles d'états.

Cette activité permet également de spécifier les patrons à utiliser (généraux et spécifiques à la plate-forme) notamment dans les phases de conception et d'architecture technique (décrite ci-après).

2 Architecture technique

Dans la continuité de la phase précédente, l'objectif de la phase d'architecture technique consiste à décrire l'environnement de production, l'architecture réseau et matérielle. Cette phase sert également à spécifier les contraintes d'exploitation.

Cette phase comprend une seule activité : la définition de l'architecture technique. Dans le détail, cette activité consiste à :

- Décrire l'architecture existante et les contraintes d'utilisation et d'exploitation.
- Définir l'architecture technique de production répondant aux besoins suivants :
 1. définir les noeuds de l'architecture distribuée,
 2. définir la nature et la cardinalité des connexions des noeuds (spécification des protocoles de communication, exemple : FTP, http, SOAP, etc.),
 3. associer les logiciels et les packages techniques aux noeuds. Positionner les serveurs d'applications, serveurs Web, moteurs de base de données sur les noeuds.
- Définir les autres environnements (de recette, d'intégration, etc.).
- Définir l'environnement de développement.
- Définir les règles d'exploitation.
- Définir la politique d'archivage.
- Définir la politique de supervision.
- Définir la volumétrie des données.
- Définir les batchs d'alimentation (ordonnancement).

A titre d'exemple, la Figure 10 illustre l'architecture technique existante d'un portail Intranet d'applications hospitalière sous la forme d'un diagramme de déploiement UML.

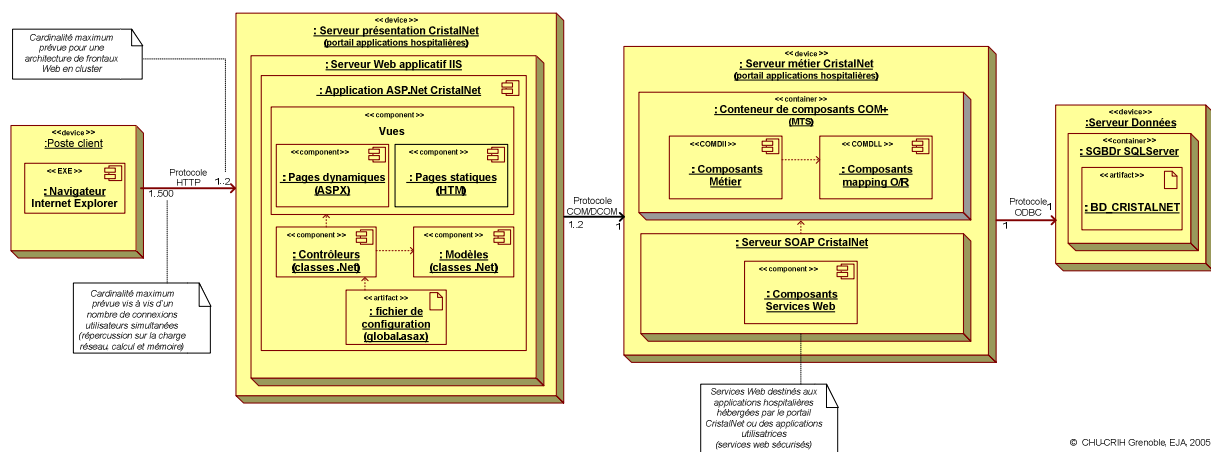


Figure 10 : architecture technique, exemple d'architecture existante

Ce diagramme montre la collaboration et la répartition des nœuds d'exécution matériels (poste client, serveur), les nœuds d'exécution logiciels (navigateur client, serveur Web, serveur de composants, serveur de base de données), et les composants métier et techniques existants. L'architecture technique cible peut être représentée de la même manière en positionnant les nouveaux objets / composants techniques nécessaires et les objets métier potentiellement.

Les phases d'architecture applicative et technique aboutissent respectivement à un dossier d'architecture applicative et à un dossier d'architecture technique.

3 Architecture coopérative

Cette phase utilise les patrons techniques de coopération définis dans le cadre des livrables 3.2 et 3.3.

Partie D Phase de conception

La conception représente la convergence des choix technologiques réalisés dans la phase d'architecture applicative et des spécifications produites dans la phase d'analyse. Cette tâche permet de réaliser la conception technique du système.

Le modèle de conception est obtenu en appliquant des règles de conception / transformation et un ensemble de patrons de conception liés à la méthodologie. Ces patrons ont été répertoriés lors de la phase d'architecture applicative. Les activités de transformation se basent également sur les patrons de conception tels que ceux de Gamma, ceux de J2EE, ou bien ceux définis lors de la phase d'architecture applicative (cf. patrons de transformation d'un modèle objet vers un modèle relationnel selon les préconisations du *framework* retenu).

Cette section détaille l'activité principale de la phase de conception : la conception des CM. D'autres activités existent qui ne seront pas présentées ici car elles ne diffèrent pas de la démarche Symphony : il s'agit de la conception des scénarii détaillés et de la conception de la persistance.

Concevoir un CM consiste à appliquer des transformations sur les diagrammes de classes du modèle d'analyse pour obtenir les diagrammes de classes du modèle de conception. Les règles de conception permettent de :

- décomposer le contrat pour affecter, à partir de la responsabilité d'un CM, les opérations élémentaires aux classes du CM et placer les opérations sur toutes les classes,
- ajouter les types aux attributs et les méthodes d'accès, de création, de destruction et de recherche,
- déterminer les classes abstraites,
- appliquer les différents patrons de conception (Fabrication, *Data Access Object*, etc.) pour concevoir les différents types d'OM et obtenir une meilleure conception (évolution, maintenance),
- choisir les structures de données ensemblistes (*collection*, *array*, *vector* ...),
- traduire les associations (en particulier les associations multi-valuées et les objets associatifs),
- définir la visibilité des attributs, méthodes et rôles des objets,
- définir la navigabilité des associations et s'assurer que les associations sont monodirectionnelles (sauf nécessité),
- enfin définir les identificateurs techniques (ex : compteur numérique).

Dans le cadre du projet TRAFIC, nous avons identifié des règles de conception dans deux technologies : d'une part vers le langage Java, un langage orienté objet très bien adapté pour implémenter des composants, d'autre part vers le générateur de la société ACTOLL.

1 Transformation vers le langage Java

Chaque composant métier doit être implémenté dans un package composé de plusieurs classes. Nous présentons dans la suite l'exemple du package Client, les packages étant tous construits de manière quasi similaire.

- L'Interface.

L'interface sert à communiquer avec le package en cours, elle va définir l'ensemble des méthodes et des fonctions définies pour la classe d'implémentation (Impl). L'interface peut être utilisée par :

- la classe Factory.
- d'autres composants via l'utilisation de leur classe « rôle ».

➤ La classe d'Implémentation

Cette classe implémente la classe Interface. De plus, elle possède en attribut un objet du type Value pour le package correspondant ainsi qu'un objet de type DAO. C'est donc la seule classe à pouvoir appeler la classe DAO, afin de modifier ou de lire la base de données.

➤ La classe Factory

C'est une classe unique qui ne sert qu'à créer des objets de type Impl pour le package en cours. La plupart du temps, elle peut le créer de deux manières différentes :

- En chargeant l'objet à partir de la base de données à l'aide de son identifiant (clé primaire).
- En créant un nouvel objet à partir de tous les paramètres nécessaires à celui-ci (par exemple, pour le Client, il faudra identifiant, nom, prénom, date de naissance et lieu de naissance.

➤ La classe DAO

Cette classe est la passerelle entre la base de données et le package. Pour accéder à la base de données, nous utilisons des ODBC (Open DataBase Connectivity), les ODBC étant une sorte de JDBC (Java DataBase Connectivity) utilisés pour relier les bases de données.

Une classe DAO comprend toujours trois procédures et une fonction :

- Store pour ajouter un nouvel élément, il prend en paramètre un objet de type Impl du package en cours.
- Delete pour supprimer un élément à partir de son identifiant. L'identifiant étant généralement l'ID ou le Nom.
- Update pour mettre à jour les valeurs d'un élément.
- Load qui, à partir de l'identifiant, renvoie un objet de type Impl du package en cours.

➤ La classe Value

Cette classe contient les attributs du composant. Ainsi, pour le client, on retrouvera son ID, nom, prénom, date de naissance, lieu de naissance... Bien entendu, elle pourra posséder des procédures et fonctions.

➤ Les classes parties

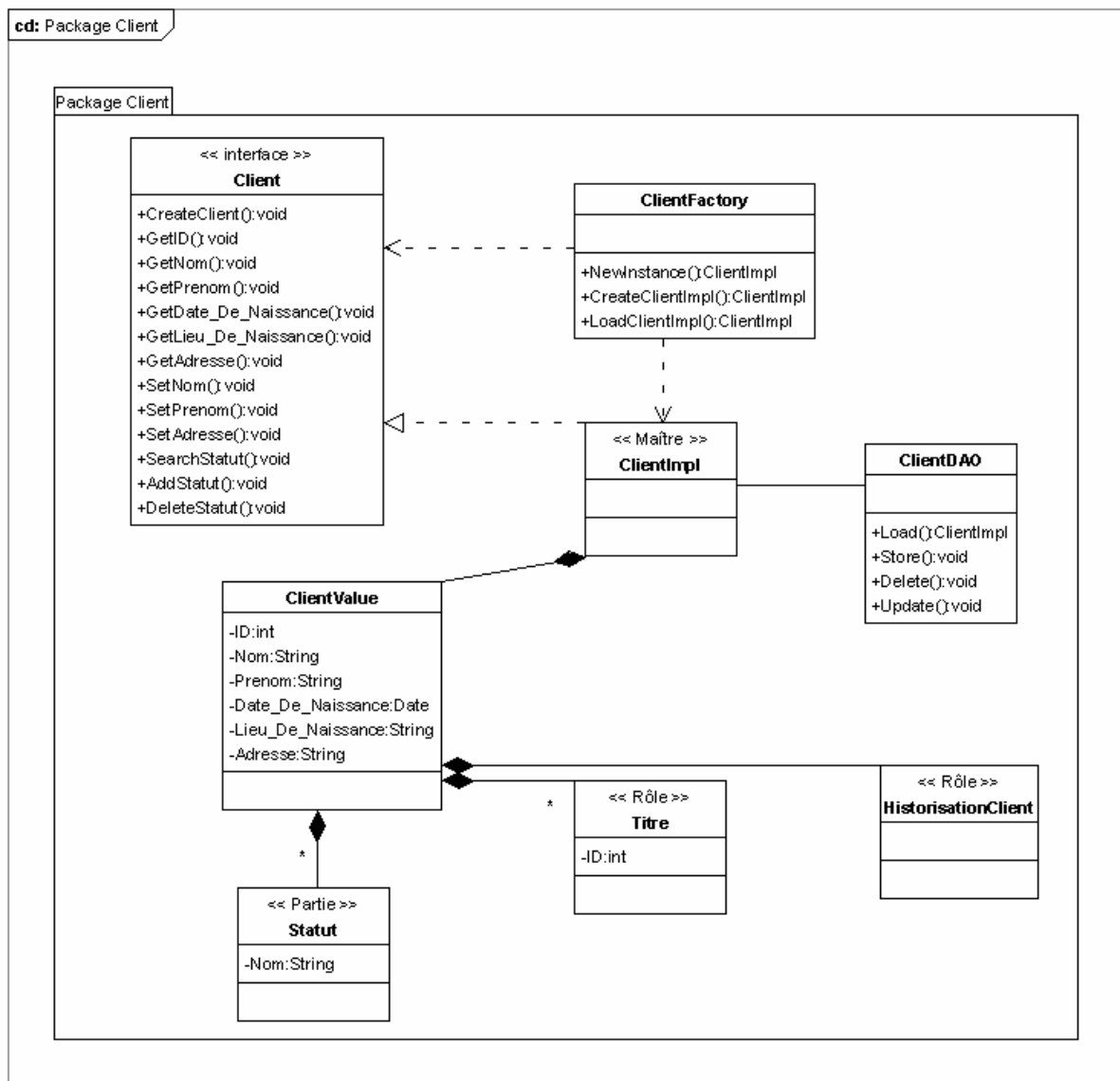
Chaque objet maître peut contenir des objets « parties ». Ces objets sont des suppléments à la classe Value. Par exemple, la classe ClientValue peut contenir plusieurs objets de type Statut ou Titre.

➤ Les classes rôles

Les classes rôles sont similaires aux classes partie, à la différence près qu'elles communiquent avec une interface d'un autre package.

Exemple : Dans le package Client, une classe rôle est HistorisationClient. Elle sert à sauvegarder les changements à chaque fois qu'un nouvel attribut est donné au client. Ainsi, pour tout ce qui concerne les changements de nom, prénom, d'adresse et d'ajout et de suppression, on passera par la classe HistorisationClient qui, via l'interface du package Historisation, sauvegardera les changements.

Le composant métier Client issu de la phase d'analyse est donc progressivement transformé dans le package suivant :



Created with Poseidon for UML Community Edition. Not for Commercial Use.

La classe rôle HistorisationClient communique avec le package Historisation et la classe rôle Titre communique avec le package Contrat.

Cette transformation devra s'appliquer sur le diagramme de classes de chaque CM (de type Entité et Donnée de référence).

D'autres règles de transformation pour une architecture applicative mettant en œuvre des composants EJB (*Entreprise JavaBeans*) sont données dans a méthode Symphony, mais n'ont pas été utilisées pour le projet TRAFIC et ne sont donc pas présentées dans le cadre de la démarche UP-TRAFIC.

2 Intégration avec le générateur de la société ACTOLL

Dans le cadre de l'application de la démarche UP-TRAFIC au sein de la société ACTOLL, nous pouvons reconsidérer le générateur et sa démarche comme des outils d'automatisation intervenant au cours de la phase commune du cycle de vie en Y, à partir de la phase de conception.

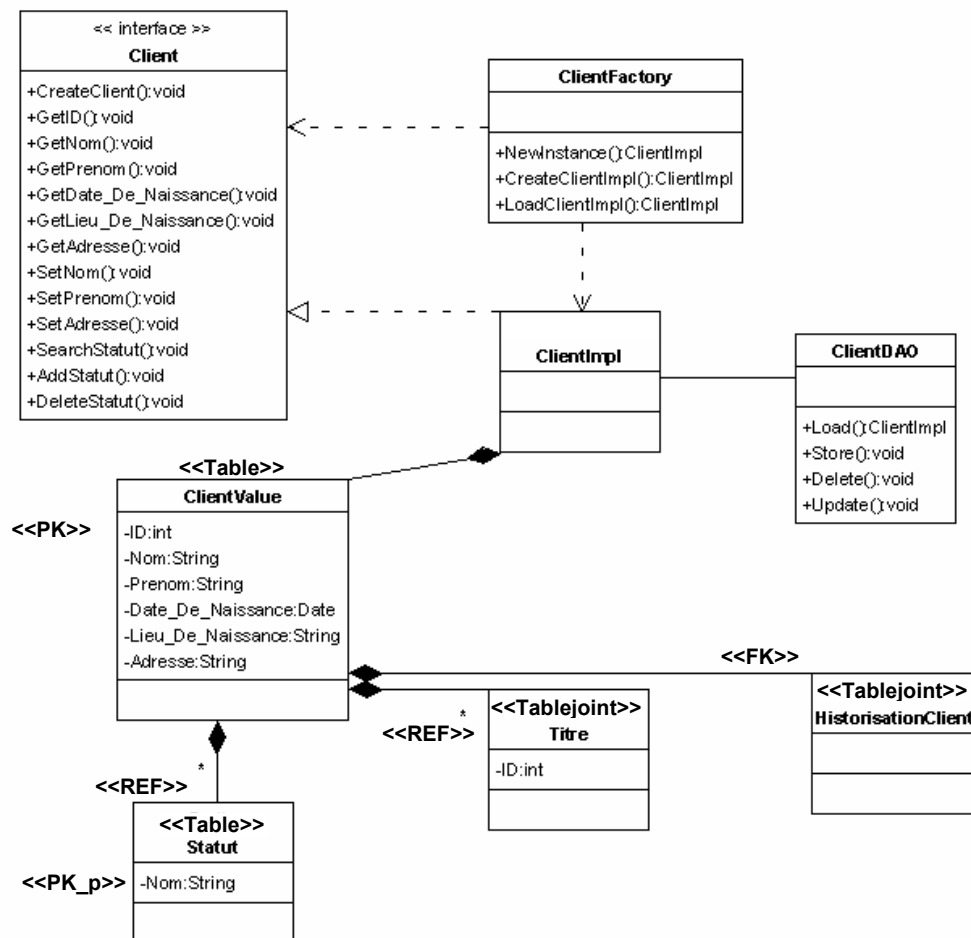
L'intégration de la démarche du générateur (voir livrable 2.3) à UP-TRAFIC implique quelques modifications au niveau des premières phases de cette démarche actuellement utilisée par ACTOLL. En effet, la branche gauche du cycle de vie en Y engendre un modèle métier du domaine plus complet qu'un « simple » modèle conceptuel de données (MCD). De plus, même si ce modèle peut avoir été réalisé en partie par réutilisation, il est déjà spécifique à l'application en cours de développement.

Ainsi, la partie « générateur » de la démarche UP-TRAFIC/ACTOLL débute par l'extraction des modèles physiques et du fichier XML à partir du modèle métier. En effet UP-TRAFIC propose déjà un patron pour transformer le modèle métier en modèle de conception. Pour assembler les deux démarches nous proposons deux patrons permettant de définir le modèle physique à partir du modèle de conception (par stéréotypage) et un patron de transformation de ce modèle « étiqueté » en modèle physique directement utilisable dans le fichier XML nécessaire au générateur (non présenté ici).

Nom	Définition du modèle physique
Contexte	Ce patron requiert un modèle de conception de l'application
Problème	Ce patron permet de définir le modèle physique en appliquant des stéréotypes sur un modèle de conception de l'application.
Solution Démarche	<ol style="list-style-type: none"> Déterminer, pour chaque classe, le besoin de persistance et lui appliquer le stéréotype <<Table>> <ul style="list-style-type: none"> Les classes « Value » est « Partie » issues de la modélisation métier sont concernées, Les classes « Rôle » également. Pour chaque classe non « Rôle » élue, définir la clé primaire (bien souvent l'identifiant issu du modèle métier). Appliquer le stéréotype <<PK>> (primary key) a (aux) attribut(s) concerné(s). <ul style="list-style-type: none"> Eventuellement définir des clés secondaires avec les stéréotypes <<SK0>>, <<SK1>>, <<SKx>> Les attributs étant considérés persistants par défaut, si un attribut n'est pas persistant, le préciser avec le stéréotype <<NP>>. Utile pour les attributs dérivés. Incompatible avec l'appartenance à une clé. Un attribut ne devant pas être vide doit être marqué par <<NNULL>>. N.b. Induit par <<PK>>. Si une classe est composée d'une autre, définir une <<PK_p>> dans a classe composant. Elle servira de partie de la clé primaire qui sera augmentée de la <<PK>> de la classe composite.

3. On applique aux classes « Rôle » le stéréotype <<TableJoint>>
 - Il n'y a pas à appliquer <<PK>> dans une <<TableJoint>>
4. Pour chaque association reliant des <<Table>> ou <<TableJoint>>
 - Si l'association a deux rôles complètement spécifiés, il faut déterminer dans quel sens la référence se fera.
 - Si le rôle sélectionné est monovalué
 - S'il est total, appliquer le stéréotype <<FK>>
 - S'il est partiel et si on autorise les valeurs nulles appliquer le stéréotype <<FK>>, sinon le stéréotype <<NNULL>>
 - Si le rôle sélectionné est multivalué, appliquer le stéréotype <<REF>>

Cas Application



Partie E Conclusion

En terme de processus de développement, UP-TRAFIC est, comme beaucoup de méthodes actuelles basées sur un processus unifié, une démarche itérative incrémentale orientée utilisateur et pilotée par les cas d'utilisation basée sur un cycle de vie en Y.

Le modèle en flocons généralisant le cycle de vie en Y de UP-TRAFIC, et les phases de modélisation métier permettent désormais une décomposition systématique d'un système d'information en autant de processus métier que d'itérations de développement. Les phases de modélisation métier (étude préalable, spécification conceptuelle et organisationnelle des besoins) sont destinées à faciliter l'identification des composants métier et l'élaboration de la cartographie fonctionnelle du système informatique cible. Cette identification se situe en amont du processus de développement, au début de l'expression des besoins.

Le modèle conceptuel de composants métier décompose chaque composant métier en trois parties : le contrat avec l'extérieur (ce que je sais faire), la partie structurelle (ce que je suis), et la partie collaboratrice (ce que j'utilise). Ce modèle prend en compte trois types de composant métier : Processus, Entité et Données de référence.

Que ce soit au niveau de la branche fonctionnelle ou de la branche technique de son cycle de vie, la **démarche** est **orientée** respectivement **composant métier** et composant technique puisque l'application est vue, tant au niveau conceptuel que logiciel, comme un **assemblage de composants métier** et de composants techniques **indépendants et interconnectés**. Cette orientation a pour objectif de garantir une bonne modularité des spécifications et de faciliter ainsi leur réutilisation.

La démarche prend en compte la coopération en intégrant dans chacune des phases les patrons et composants de coopération définis dans le cadre du lot 3.

Enfin, compte tenu des contraintes d'exploitation du système cible, **les phases d'architecture applicative et technique** contribuent à mettre en œuvre une **architecture à base de composants techniques** répartis sur un **modèle d'architecture multi-couches**. Dès la **phase d'architecture applicative**, UP-TRAFIC met l'accent sur l'**application de patrons d'architecture et de conception** pour spécifier l'ensemble des couches applicatives. De plus, la démarche UP-TRAFIC recommande fortement l'utilisation de deux types de composants techniques réalisés dans le cadre de ce projet : un composant technique langage visuel d'interrogation (utile par exemple pour interroger de façon visuelle les horaires de transport, etc.), et un composant technique d'accès progressif à l'information. Cette phase se préoccupe également de **définir les règles de conception et de transformation** applicables lors de la phase de conception. Dans le cadre du projet TRAFIC, d'une part des règles de transformation ont été définies vers le langage orienté objet Java, d'autre part, un patron a été défini permettant d'introduire dans le générateur de la société ACTOLL les composants métiers issus de la phase d'analyse. Ainsi, la démarche couvre un cycle complet de développement de systèmes d'information transport, partant de la spécification des processus métiers, et allant jusqu'à l'implantation de ces processus métiers dans des technologies classiques de développement de systèmes d'information (par exemple le langage orienté objet Java) ou dans des technologies spécifiques aux SIT (par exemple le générateur mis en œuvre dans la société ACTOLL). La formalisation sous forme de patrons de ces règles ainsi que la description des composants techniques peuvent augmenter de manière conséquente la capacité à maximiser la réutilisation de ces composants.