Conception préliminaire

Objectifs du chapitre

Nous allons maintenant étudier le rôle d'UML lors de l'étape de conception préliminaire. Les diagrammes UML servent ici plus particulièrement à construire et à documenter la façon dont le développement de la solution doit être organisé.

Nous allons voir comment :

- élaborer la conception préliminaire avec UML ;
- développer les vues préconisées pour cette étape ;
- organiser un développement objet et d'intégration EAI avec UML ;
- construire les composants d'une architecture 3-tiers ;
- identifier et construire les applications d'un système d'entreprise ;
- identifier et construire un composant métier.

Quand intervient la conception préliminaire ?

La conception préliminaire est certainement l'étape la plus délicate du processus 2TUP car elle en représente le cœur. C'est en effet à cette occasion que s'effectue la fusion des études fonctionnelles et techniques. En conséquence, plusieurs activités doivent coexister. Il convient de :

- passer de l'analyse objet à la conception,
- intégrer les fonctions métier et applicatives du système dans l'architecture technique,
- adapter la conception générique aux spécifications fournies par l'analyse.

La conception préliminaire est avant tout affaire d'organisation ; elle s'appuie sur les points de vue de spécification fonctionnelle et structurelle de l'analyse, mais également sur les *frameworks* de la conception technique. Elle se termine lorsque la conception est organisée suivant :

- son déploiement cible,
- son modèle d'exploitation,
- son modèle logique.

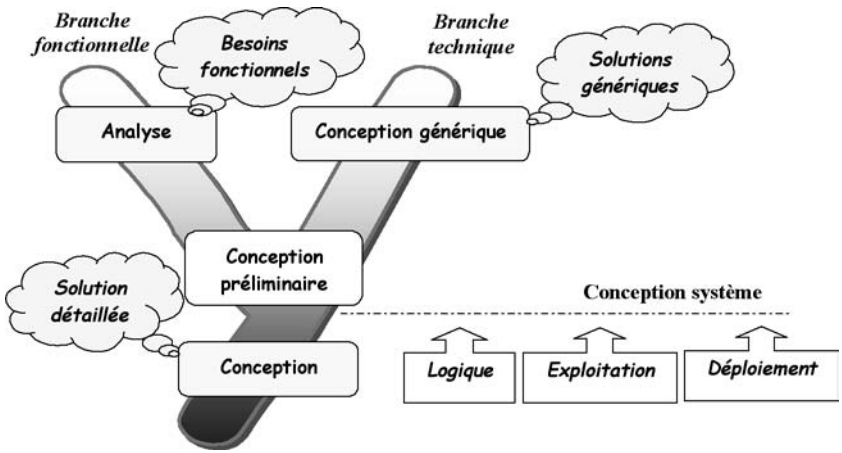

Figure 10-1 : Situation de la conception préliminaire dans 2TUP

Le processus global de conception est un processus à information croissante dont l'étape de conception préliminaire est la phase d'organisation. Le processus de conception combine différents points de vue de modélisation qui, à partir du résultat que l'on souhaite obtenir, remontent jusqu'aux détails de fabrication en langage objet.

- Le premier niveau de conception d'un système est son déploiement car c'est généralement l'organisation des environnements de travail sur un réseau qui est la plus immédiatement définie. Le diagramme de déploiement UML suffit ici à documenter ce niveau.
- À partir du déploiement, on est capable de définir les composants qui seront administrés par l'exploitant du système. On conçoit ici le modèle d'exploitation en intégrant les résultats de la conception générique. Les développeurs doivent également définir les interfaces qui constituent le lien entre les composants, et énumérer les interfaces utilisateur qui correspondent aux besoins exprimés par l'analyse.
- La fabrication des composants d'exploitation recourt à différents outils, en l'occurrence Java, SQL et un atelier de fabrication d'écrans. La conception s'effectue ici avec des diagrammes de classes UML et les classes doivent

être organisées suivant des catégories de conception. La conception préliminaire s'arrête à la définition de cette organisation et c'est en conception détaillée que l'on développe précisément le contenu de chaque catégorie. À ce niveau, le développeur doit encore définir les interfaces des catégories et apporter plus de précisions aux interfaces utilisateur. L'ensemble de ces interfaces assure le passage de l'analyse à la conception.

- L'organisation des classes de conception, ainsi que la connaissance des composants d'exploitation à développer, permettent en final d'organiser la configuration logicielle en sous-systèmes.

Les liens avec l'analyse consistent à élaborer les interfaces à partir des cas d'utilisation, et à intégrer les classes d'analyse dans le modèle logique de conception. Ceux avec la conception générique résident dans l'insertion des éléments du modèle déjà conçus, suivant les points de vue logique et d'exploitation. À l'image du cône qui vous a été présenté pour illustrer le développement incrémental, le schéma ci-dessous résume le processus envisagé pour la conception.

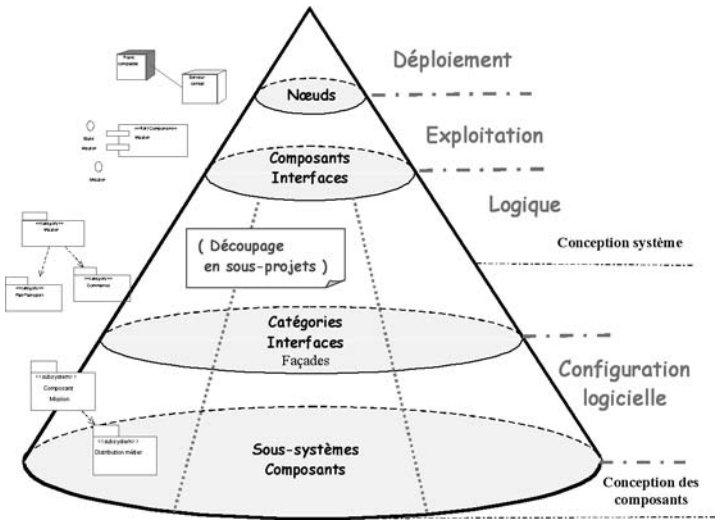

Figure 10-2 : Le processus de conception à information croissante suivant les points de vue

L'organisation suivant le point de vue logique permet d'isoler les éléments de développement. Cette caractéristique qui suit les principes d'architecture et d'orientation composants peut aussi devenir une opportunité de développement parallèle en sous-projets.

Éléments mis en jeu

- Modèle de déploiement, postes de travail, style d'architecture à niveaux,
- modèle d'exploitation, applications, composants métier et instances de base de données,
- interfaces de composants, interfaces utilisateur, interfaces EAI, façade,
- modèle logique, diagramme de classes, classes et catégories de conception, distribution du modèle logique sur les couches,
- modèle de configuration logicielle.

Développement du modèle de déploiement

Le déploiement d'une solution client/serveur se construit sur la définition des postes de travail.

Définition

POSTE DE TRAVAIL

Le poste de travail représente un ou plusieurs acteurs pouvant être localisés sur une machine d'un type particulier et remplissant une fonction identifiée dans l'entreprise. Le poste de travail ne représente pas forcément une machine physique, mais peut consister en plusieurs machines, à condition qu'elles donnent lieu au même type de déploiement.

À un acteur de l'analyse correspond généralement un poste de travail, mais ce n'est pas toujours le cas :

- le chauffeur disposant d'un terminal portable correspond à un poste de travail ;
- si l'on imagine qu'il existe deux types de terminaux nécessitant deux configurations logicielles différentes, deux postes de travail correspondent à l'environnement du chauffeur ;
- l'administrateur du système n'a pas de poste de travail particulier mais peut intervenir à partir de n'importe quel poste.

La notion de poste de travail peut cependant être quelque peu bouleversée par la généralisation des déploiements en client léger. En effet, au travers de la notion de portail plusieurs applications sont potentiellement accessibles, voire atteintes de manière transparente à l'utilisateur par des techniques de syndication. Cette évolution technologique nous amène à associer la notion de poste de travail à l'ensemble des applications Web que l'on désire rendre accessibles pour un acteur particulier du système. La définition des postes de travail dans ce cadre constitue alors une excellente analyse des acteurs et des droits que

l'on doit déclarer au travers des mécanismes de « single sign-on » d'un portail.

Les modèles de déploiement et de configuration matérielle s'expriment tous deux à l'aide d'un diagramme de déploiement. Cependant, ils n'expriment pas tout à fait le même niveau de description.

- Le modèle de configuration matérielle a été utilisé au chapitre 5 pour exprimer les contraintes de mise en œuvre au niveau physique. On y trouve les nœuds et les connexions physiques du système, qui sont les différents types de machine connectés par des moyens divers. Le modèle de configuration matérielle permet de spécifier, de documenter et de justifier tous les choix d'organisation physique en fonction des machines dédiées aux diverses fonctions techniques du système.
- Le modèle de déploiement considère plutôt chaque nœud comme un poste de travail. Il exprime la répartition physique des fonctions métier du système et permet de justifier la localisation des bases de données et des environnements de travail. Le modèle de déploiement aide à préciser la qualification des postes client, des réseaux et de leur sécurité physique, par rapport à des critères fonctionnels.

ÉTUDE DE CAS : DESCRIPTION D'UN DÉPLOIEMENT À 3 NIVEAUX

Le déploiement se répartit suivant les deux niveaux central et départemental de l'entreprise [Orfali 94]. Les postes de travail s'articulent ainsi soit autour du serveur central, soit autour d'un serveur d'agence. Un serveur Web est dédié à la consultation distante des clients.

Vous remarquerez la capacité d'exprimer l'imbrication de différents nœuds (depuis UML 2.0), utilisée ici pour montrer ce qui est déployé en DMZ.

- En rapprochement avec la configuration matérielle :
- les environnements de travail « Comptable » et « Logistique » sont chacun déployés sur un des PC centraux – l'environnement comptable correspond au déploiement de l'IHM de SAP R3 sur poste client lourd ;
- les environnements de travail « Réceptionniste » et « Répartiteur » sont mis en œuvre sur des PC agence – au travers d'un portail le réceptionniste accède à la fois à des fonctions propres à SIVEx et aux IHM de SIEBEL ;
- L'environnement de travail « Opérateur de quai » est implanté sur un PC agence spécifique qui devra être muni d'un lecteur de codes-barres et d'une balance de pesée ;
- L'environnement de travail « Chauffeur » est déployé sur un terminal spécifique en termes de matériel, mais prenant en charge l'exécution d'une machine virtuelle Java.

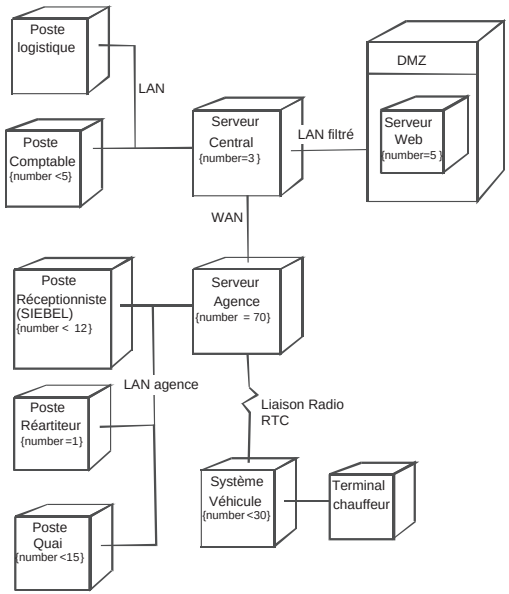

Figure 10-3 : Modèle de déploiement SIVEx

Développement du modèle d'exploitation

L'élaboration d'une architecture d'exploitation a déjà commencé en conception générique. À partir du déploiement, il est possible de la compléter en fonction des machines et des postes de travail, tout en intégrant les besoins exprimés en analyse. Conformément à ce qui vous a été expliqué en conception générique, le modèle d'exploitation va définir les applications installées sur les postes de travail, les composants métier déployés sur les serveurs et les instances de bases de données implantées sur les serveurs également.

Les applications se déterminent par regroupement des fonctions de l'utilisateur, tout en respectant la définition des postes de travail. On partira du modèle de spécification fonctionnelle pour définir les applications du système – à cette occasion, les choix d'utilisation de progiciels du marché et l'anticipation des impacts sur l'architecture pourront être affinés. Un découpage idéal en cas d'utilisation permet en effet d'affecter une application à la réalisation d'un nombre entier de ces derniers. Il faut cependant tenir compte des regroupements qu'a pu réaliser l'analyste, car un cas d'utilisation peut concerner plusieurs acteurs différents sur des postes de travail séparés. On rencontre donc les cas de figure suivants :

- un même cas d'utilisation donne lieu à plusieurs applications. Cette situation survient lorsque plusieurs acteurs y participent simultanément ou au travers de postes de travail différents ou encore par la réutilisation des fonctions d'un progiciel ;
- un ou plusieurs cas d'utilisation recourent à la même application. C'est un choix d'ergonomie du poste de travail, lorsque les cas d'utilisation concernent le même acteur.

ÉTUDE DE CAS : DÉFINITION DES APPLICATIONS

L'exemple ci-après illustre la réalisation des cas d'utilisation par des applications, à partir d'un extrait du modèle de spécification fonctionnelle.

Nous avons attribué un nom à chaque application de SIVEx. L'application *ConVEx* servira aux exemples de conception préliminaire et détaillée. Elle correspond aux fonctions de planification et de contrôle des missions.

- *ConVEx* implémente tout ou partie des cas d'utilisation *Gestion de mission* et *Suivi de mission* ;
- *ConVEx* est destinée au poste de travail du répartiteur.

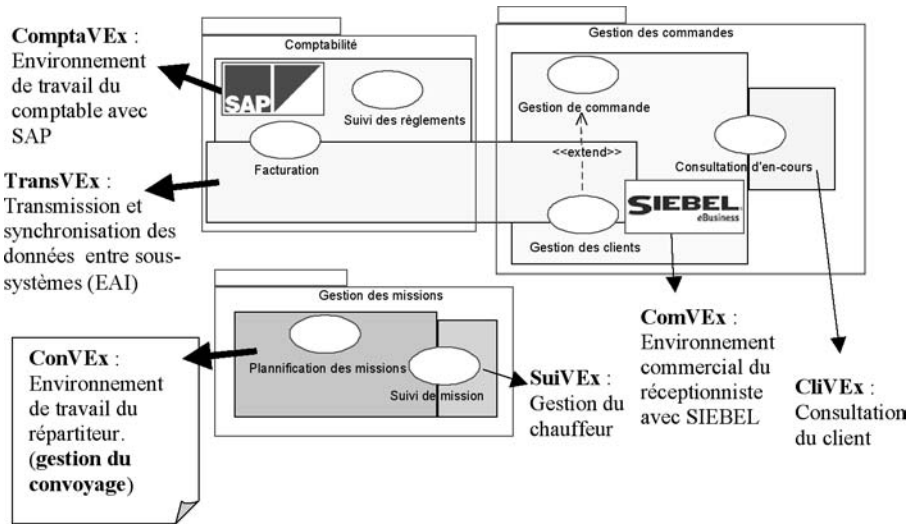

Figure 10-4 : Identification des applications du système SIVEx

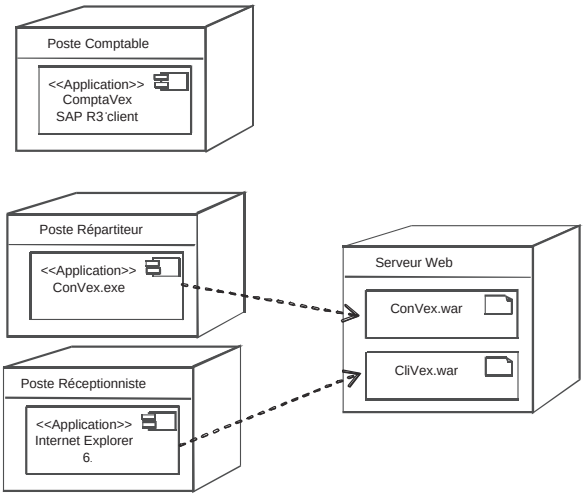

Figure 10-5 : Définition des applications dans le modèle d'exploitation

La distribution d'un composant facilite sa réutilisation, puisque les mêmes services sont accessibles depuis différentes applications. La première façon d'identifier les composants distribués consiste donc à recenser les catégories d'analyse partagées par plusieurs applications. Le critère de partage et de réutilisation n'est a posteriori pas suffisant : les composants distribués offrent en effet un découplage logiciel entre les applications et le métier. Ce découplage facilite la réutilisation, mais également la maintenance et l'évolution du système global. Nous conseillons donc de transformer chaque catégorie d'analyse en composant d'exploitation, dès lors que celle-ci représente des concepts du domaine. L'architecte logiciel peut ensuite arranger ce découpage pour intégrer des critères de conception. Dans la perspective d'optimisation du système, il peut être en effet judicieux de regrouper au contraire plusieurs catégories d'analyse dans le même composant.

L'introduction de progiciels dans le système est une forme de réutilisation qui bouleverse notre approche d'architecture orientée objet. En effet, le progiciel introduit à la fois des concepts fonctionnels parfois orthogonaux à l'analyse effectuée et des contraintes techniques d'interopérabilité. Dans ce cadre, nous allons bien entendu procéder à l'assimilation du progiciel à un composant supplémentaire de l'architecture logicielle. Pour couvrir l'ensemble de la problématique d'intégration, nous rattachons cependant notre méthode aux techniques d'analyse et de conception employées dans le domaine de l'EAI.

Pour compléter la définition des composants, il est ensuite nécessaire d'en énumérer les interfaces. Il s'agit d'un travail d'approche consistant à produire une description sommaire de ce que réalise un composant dans le système. Le premier objectif de ces interfaces vise à donner à l'ingénieur d'exploitation une vision fonctionnelle de ce que réalisent les différents composants afin de lui permettre d'améliorer son diagnostic de pannes. Les interfaces servent également à préciser les dépendances logicielles qui vont s'établir entre composants. Notez bien que nous sommes encore loin de définir des opérations et des signatures précises. Dans le cadre de notre processus à précision croissante, c'est seulement en conception détaillée que l'on aboutira à ce résultat.

Des interfaces particulières sont également définies pour permettre l'encapsulation des progiciels en composants logiciels. Techniquement, ces interfaces correspondent soit à des fonctions distribuées synchrones qui s'apparentent aux opérations EJB que l'on désire mettre en œuvre dans notre architecture, soit à des messages de données transmis de manière asynchrone. Dans tous les cas, les techniques d'interopérabilité proposées par les éditeurs de progiciels sont plus assimilables à des flux de synchronisation de données qu'à des services proprement encapsulés. Dans le cadre de notre architecture orientée objet, une interface de progiciel, que l'on nomme « interface EAI » tout au long de cet ouvrage, représente des flux de synchronisations d'objets.

ÉTUDE DE CAS : DÉFINITION DES COMPOSANTS MÉTIER

Pour SIVEx, toutes les catégories donnent lieu à un composant distribué, sauf la catégorie *Transmission comptable* car elle ne décrit pas proprement de nouveaux objets métier mais correspond exclusivement à des concepts applicatifs d'échanges de données.

D'autre part, le couplage des concepts entre les catégories *Réseau* et *Ressource* a poussé le concepteur à les regrouper dans un même composant distribué. Le diagramme ci-dessous illustre l'identification des composants sur la base du modèle structurel d'analyse.

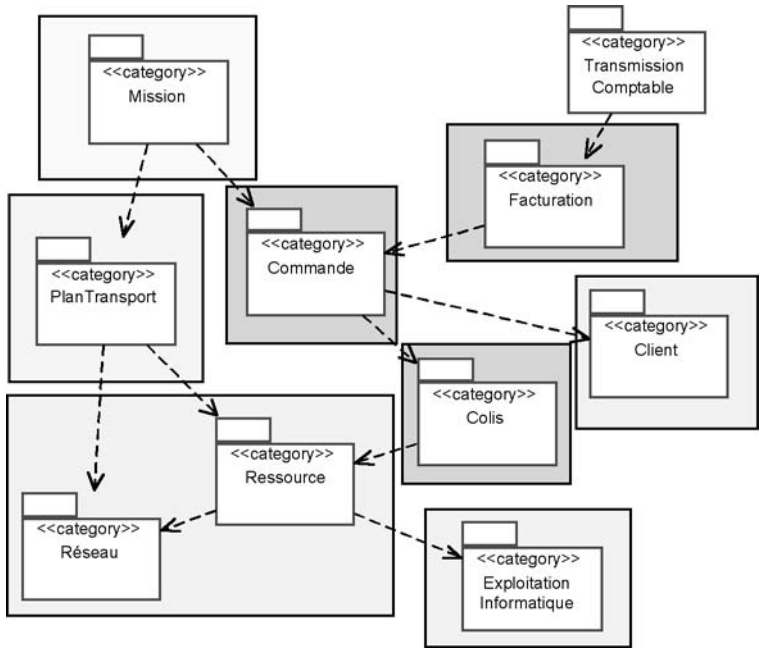

Figure 10-6 : Identification des composants métier du système SIVEx

Tous les composants identifiés sont ajoutés, avec leurs dépendances, au modèle d'exploitation.

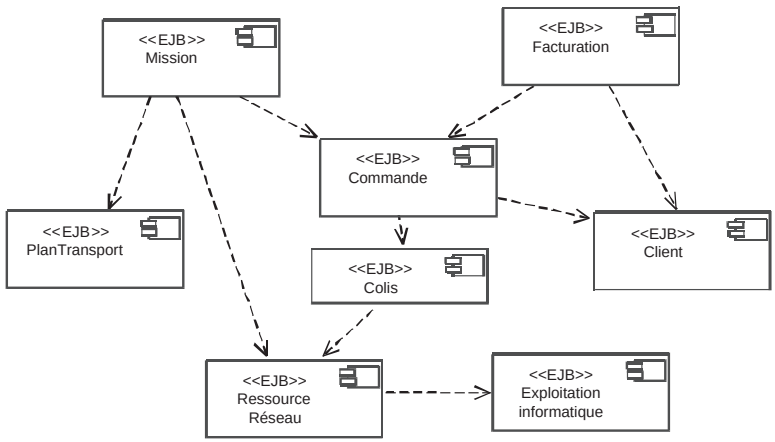

Figure 10-7 : Schéma de dépendances entre composants métier du modèle d'exploitation

Dans un second temps, les interfaces sont définies en termes de regroupement de responsabilités. Conformément à ce que nous avons déjà mentionné, il s'agit juste de préciser un premier niveau de définition qui devra être affiné en conception détaillée. Le tableau ci-dessous illustre cette première ébauche de définitions pour les interfaces des composants *Commande* et

Mission. Vous remarquerez que, par convention, tous les noms d'interfaces sont précédés d'un I, comme suggéré dans [UML-UG 99].

Composant distribué	Interface	Description de ses responsabilités
Mission	IMission	Gestion distribuée d'une entité mission : création, modification, validation, archivage.
	ISuiviMission	Distribution des informations d'une mission en exécution. Notification d'alertes, calcul des durées estimées de parcours, rafraîchissement des informations d'avancement. Distribution automatique des informations pour le rafraîchissement des en-cours de commandes.
Commande	ICommande	Gestion distribuée des entités commande. Création, modification, validation, annulation, archivage et suppression d'une commande.
	IEnCoursDeCmd	Distribution des informations d'en-cours des commandes en exécution. Réception des informations de mission, calcul des horaires estimés de passage.

Tableau 10-1 : Définition sommaire des interfaces métier

Pour compléter l'architecture logicielle, il reste à identifier et définir les interfaces EAI afin de préparer leur rattachement aux concepts orientés objet de notre processus de développement. Nous verrons par la suite les avantages à appliquer une méthode orientée objet à un domaine du logiciel, l'intégration des progiciels d'entreprise, dans lequel une approche purement fonctionnelle est généralement employée.

ÉTUDE DE CAS : DÉFINITION DES INTERFACES EAI

Cette définition consiste à recenser dans un premier temps les objets métier de SIVEx et à les projeter sur chacun des composants du système : SAP R/3, Siebel, Mission, Commande, Facturation, etc. Dans un second temps, chacune des intersections du tableau est renseignée avec les opérations de synchronisation que réalisent les composants.

Composant	SAP R3	Siebel	Client	Mission
Client	Consultation Renseignement des données comptables Interdiction de vente	Consultation Renseignement des données commerciales	Consultation Suppression Archivage	Consultation Renseignement des adresses de livraison
Commande	Consultation Consolidation	Création Consultation Modification des conditions commerciales, Suppression		Consultation Renseignement des en-cours

Tableau 10-2 : Identification des interfaces EAI

Pour chaque case renseignée de cette matrice, une ou plusieurs interfaces EAI sont identifiables. Par exemple, l'interdiction de vente d'un client donne lieu à une interface EAI entre les composants SAP R3 et Siebel. Dans l'optique d'en donner une première définition, nous nous sommes inspirés des PIPs (Partner Interface Processes) du consortium RosettaNet (*www.rosettanet.org*) chargé de définir des standards d'interfaces d'échanges pour l'industrie électronique.

Chaque PIP constitue effectivement une spécification d'interface, dont la modélisation UML est compatible avec le profil « UML for EAI », proposé par l'OMG en janvier 2002.

Dans ce cadre, une spécification métier de l'interface utilise un diagramme d'activité pour décrire l'usage fonctionnel de l'interface EAI.

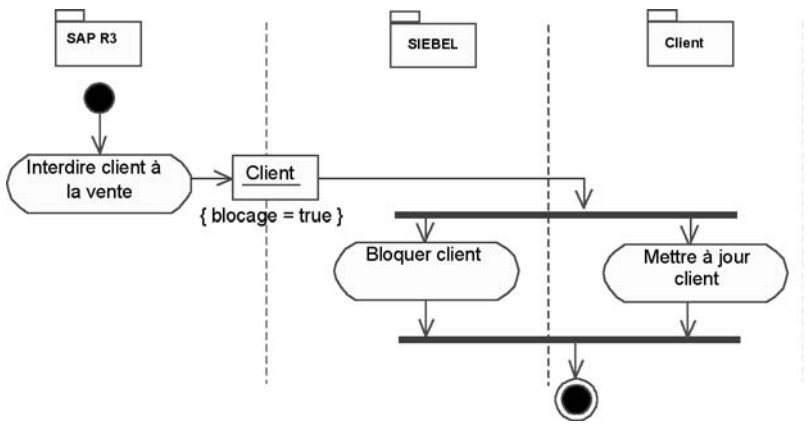

Figure 10-8 : Schéma de spécification métier d'une interface EAI

Par la suite, un diagramme de séquence concerne l'analyse du protocole réalisant la synchronisation d'information. Remarquez la notation des messages asynchrones utilisés par les mécanismes EAI de SIVEx, ainsi que la numérotation décimale des messages.

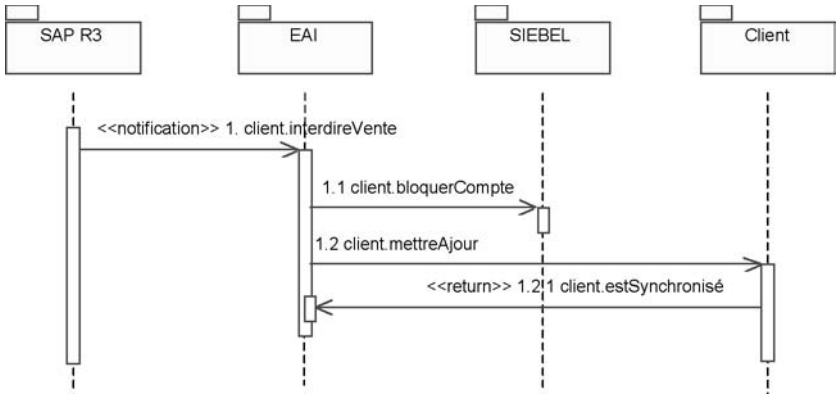

Figure 10-9 : Analyse du protocole d'une interface EAI

Dans le cadre d'une architecture 3-tiers : application, composant distribué, stockage des données, il reste maintenant à identifier les instances de base de données pour optimiser la distribution du système. Ces dernières se définissent en fonction de critères de répartition de données soit pour accélérer les temps d'accès, soit pour intégrer d'autres systèmes, soit pour isoler une partie du système en vue de sa réutilisation.

ÉTUDE DE CAS : DÉFINITION DES INSTANCES DE BASE DE DONNÉES

Dans le cadre de SIVEx, l'optimisation des temps d'accès est gérée par les mécanismes de la couche d'accès aux données et ne nécessitent pas une séparation en plusieurs instances. Seules les données de la catégorie *Client* doivent être isolées en vue de leur réutilisation au sein du système d'entreprise. Nous disposerons donc de deux instances de base de données : les données *SIVEx* et une base *Clients*.

Il reste à répartir les données entre agences et siège. La base *Clients* doit rester unique car c'est le meilleur moyen de garantir la cohérence de ce référentiel métier. De plus, la fréquence de mise à jour des clients depuis les agences est relativement faible et permet de conserver ces informations au niveau central. En revanche, les données traitées en agence et en central n'étant pas les mêmes, il semble naturel que leurs schémas soient différents, suivant qu'il s'agit d'une agence ou du central. Une fréquence de mise à jour également différente fait pencher la balance en faveur d'une conception des bases séparées en deux types d'instances : *Agence* et *Central*.

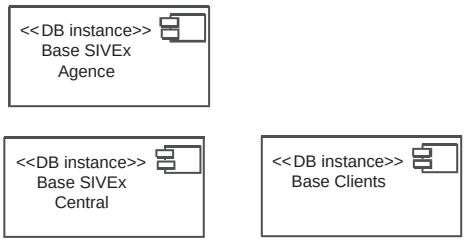

Figure 10-10 : Modèle d'exploitation des instances de base de données

CONSOLIDEZ LE MODÈLE D'EXPLOITATION POUR CHAQUE APPLICATION

Une fois les composants d'exploitation identifiés – applications, composants distribués et instances de bases de données – avec leurs interfaces métier et EAI définies, il convient de tracer un diagramme des dépendances du point de vue de chacune des applications du système. Vous verrez alors se dessiner le rôle de chaque interface par rapport aux applications, les rôles que jouent les composants entre eux et les contextes d'accès aux instances de base de données.

Ces diagrammes vont vous permettre de consolider les définitions du modèle d'exploitation et de cartographier les flux de données distribuées qui utilisent le réseau.

ÉTUDE DE CAS : CONSOLIDATION DE L'APPLICATION *CONVEX*

Le modèle d'exploitation permet de tracer sur plusieurs diagrammes les dépendances entre les composants. La figure suivante livre un aperçu des composants impliqués par l'application *ConVEx*.

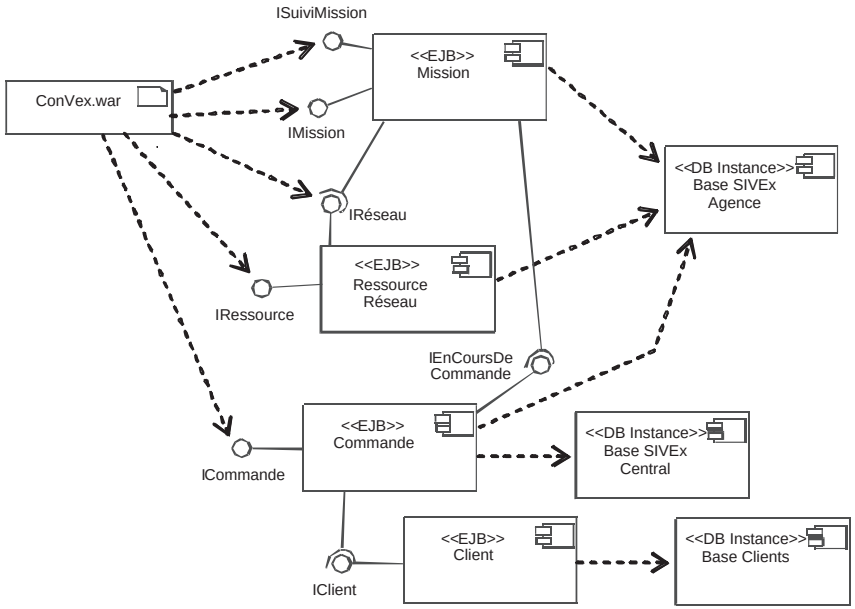

Figure 10-11 : Diagramme de consolidation de l'application ConVEx

Pour terminer le paragraphe sur le modèle d'exploitation, il convient de ne pas oublier le travail commencé en conception générique. En effet, le modèle d'exploitation comporte déjà les instances de base de données et les composants offrant les services techniques transverses. Ces composants restent évidemment sous-jacents au travail de définition que nous venons d'effectuer. Ils définissent des protocoles et des interfaces implicitement accessibles par tous les composants d'exploitation du système. Vouloir les intégrer dans un travail de consolidation aurait complexifié les ramifications des dépendances sans apporter de progrès aux éléments de notre conception.

Énumération des interfaces utilisateur

Si les composants communiquent par le biais de leurs interfaces, les applications, quant à elles, sont utilisables par le biais de leurs interfaces utilisateur ou IHM (Interface Homme Machine). Afin de compléter l'architecture d'exploitation, les concepteurs peuvent dresser la liste des vues d'IHM dont dispose chaque application. Il n'y a ici aucune technique propre à UML : une liste des vues attendues et de leurs principales fonctions suffit à ce niveau de description. En outre, ces définitions servent à identifier des composants

d'IHM transverses aux applications, augmentant ainsi la réutilisation dans la conception de la couche de présentation.


Conseil

EXPLOITEZ LES DÉFINITIONS D'IHM TOUT AU LONG DU PROCESSUS 2TUP

Dans le cadre d'entreprises dont la culture de spécification est fortement axée sur la définition d'écrans, il est fort judicieux d'accompagner le développement 2TUP de la définition des IHM.

- À l'étape de capture des besoins fonctionnels, ceux d'IHM peuvent se transformer en maquettes pour chaque cas d'utilisation. Leur présentation aux utilisateurs pourra susciter des remarques et avoir des répercussions sur les règles de gestion métier.
- À l'étape d'analyse, au niveau de l'analyse de l'application plus particulièrement, les messages produits ou reçus par les acteurs peuvent être attachés aux vues déjà définies. Cette pratique permettra de mieux fixer le rôle fonctionnel de chaque vue et d'améliorer éventuellement leur ergonomie.
- À l'étape de conception préliminaire, les vues de la maquette sont formalisées et réparties sur les différentes applications du modèle d'exploitation. L'identification des composants d'IHM et la réutilisation au niveau de la couche de présentation s'en trouveront affinées. Leur présentation auprès des utilisateurs permettra enfin d'améliorer l'ergonomie de fonctionnement.

Bien que l'écran constitue un excellent support de communication avec les utilisateurs, à exploiter tout au long du processus, nous ne sommes pas ici les promoteurs d'une méthode pilotée par les écrans, car inversement, le processus 2TUP prône un développement axé sur la définition du métier. Il ne faut donc pas confondre succession d'écrans avec processus d'entreprise, ni ergonomie avec plus-value métier.

ÉTUDE DE CAS : ENUMÉRATION DES VUES DE L'APPLICATION *CONVEX*

Cette énumération définit les interfaces exploitées par l'application de gestion du convoyage (missions) et donne une idée plus précise de l'environnement de travail du répartiteur.

Vue d'IHM	Description
Sélection mission	Sélection d'une mission dans une liste filtrée et triée des missions de l'agence courante. Les critères sont les suivants : affectation, site, client, poids, chauffeur, véhicule.
Sélection commande	Sélection d'une commande dans une liste filtrée et triée des commandes présentes dans l'agence courante. Les critères sont les suivants : affectation, site, client, poids, mission, quai.

Édition mission	Feuille d'édition d'une mission. Création, modification, suppression, validation et annulation.
Tableau suivi de mission	Tableau de suivi des missions en cours d'une agence, affichage des retards, des événements et des messages.
Planning des ressources	Tableau de planning style GANTT d'affectation des chauffeurs et des véhicules aux missions. Cette vue se présente suivant deux options qui sont la matrice de planification des véhicules ou la matrice de planification des chauffeurs.

Tableau 10-3 : Extraits des définitions d'IHM de l'application ConVEx

L'énumération des vues permettra de développer la structure des classes de la couche de présentation.

Développement du modèle logique

Nous avons jusqu'ici défini les modèles de déploiement et d'exploitation en déterminant les postes de travail et les composants de la solution visée. Nous devons cependant développer les classes nécessaires au codage. Le modèle logique est précisément celui de la représentation des classes organisées en catégories. Comme nous allons le voir, ce modèle dérive du modèle structurel d'analyse d'une part et du modèle logique de conception technique d'autre part.

Une catégorie de conception a des objectifs analogues à une catégorie d'analyse : c'est un regroupement de classes à fort couplage. Les objectifs poursuivis ne sont cependant plus les mêmes ; les catégories de conception organisent des classes techniques et contribuent à la réutilisation et à l'optimisation d'un système à développer.

Les éléments qui servent à identifier les catégories de conception sont à la fois :

- les composants d'exploitation, qui définissent un ensemble cohérent de classes à assembler ;
- les *frameworks* techniques, conservant la structure des couches logicielles, regroupent les classes travaillant dans le même domaine de responsabilité technique et mettant en œuvre les mêmes types de technologies ;
- les catégories d'analyse qui structurent le métier en plusieurs domaines de spécialisation logique et qui rassemblent des classes collaborant étroitement entre elles.

INFLUENCE DES COMPOSANTS D'EXPLOITATION SUR L'ORGANISATION DU MODÈLE LOGIQUE

Premièrement, les composants d'exploitation peuvent constituer autant de sous-projets dans le développement global du système. On pourrait dès à présent découper le développement global de SIVEx en différents sous-projets et chaque équipe se verrait confier la tâche de concevoir des applications et des composants distribués. Dans ce cadre, chaque équipe en charge de sa conception pourra développer des catégories de classes parallèles et indépendantes les unes des autres.

Deuxièmement, une catégorie de conception ne peut regrouper des classes destinées à fonctionner pour moitié sur une application et pour moitié sur un composant distribué : il faut choisir, ou l'une ou l'autre. Un composant d'exploitation est en effet réalisé par un nombre entier de catégories de conception. Cela n'exclut pas pour autant qu'une même catégorie puisse participer à l'élaboration de plusieurs composants par réutilisation.

Retenez donc que le modèle d'exploitation guide l'organisation du projet au travers de son modèle logique, car une même catégorie de conception ne peut être découpée entre les réalisations de différents composants.

INFLUENCE DES FRAMEWORKS TECHNIQUES SUR L'ARCHITECTURE LOGIQUE DE CONCEPTION

Rappelons que les *frameworks* techniques peuvent être abstraits. Nous effectuons par exemple une distinction entre :

- le *framework* de journalisation auquel correspond un composant d'exploitation réutilisable ;
- et le noyau applicatif ou la définition d'un mécanisme modèle vue contrôleur (MVC) qui apporte une structure de conception générique, mais qui nécessite des compléments de code.

Les *frameworks* concrets font déjà l'objet de catégories dans le modèle logique de conception technique. Ces catégories continuent d'exister dans la nouvelle organisation pour fournir leurs services techniques.

Les *frameworks* abstraits vont permettre d'organiser les catégories de conception. Chacun d'eux correspond généralement à une couche logicielle du système. Dans cette optique, nous organisons le modèle logique suivant les cinq couches : Présentation, Application, Métier, Accès aux données et Stockage de données.

À chacune des couches correspond un package qui englobe les catégories de conception. Si on leur applique les règles de nommage UML, on distingue alors la catégorie *Métier::Mission* de la catégorie *Accès aux données::*

Mission. La figure 10-10 illustre l'organisation en couches des catégories de conception.

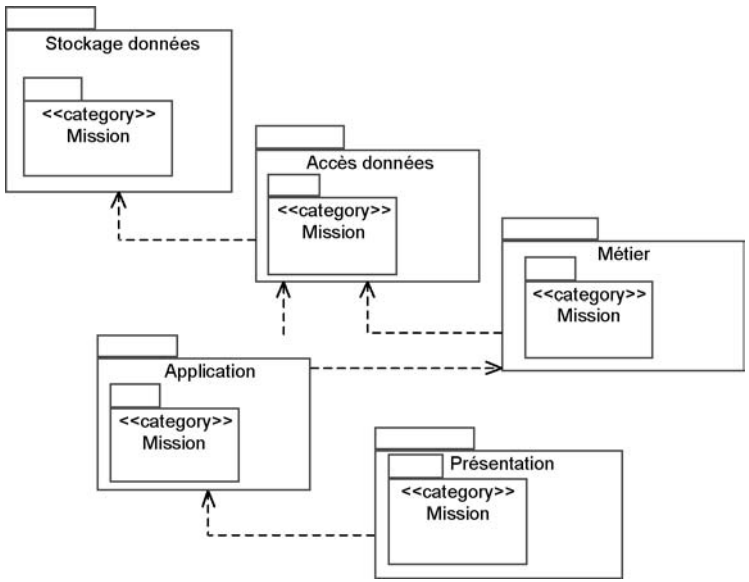

Figure 10-12 : Projection de la catégorie «Mission» sur les couches logicielles

Par définition, la couche métier est implicitement structurée suivant les catégories d'analyse. Afin de formaliser l'influence des *frameworks* abstraits sur le découpage en catégories de conception, la figure 10-13 illustre la réalisation concrète du *framework noyau métier*, suivant les domaines métier définis par l'analyse.

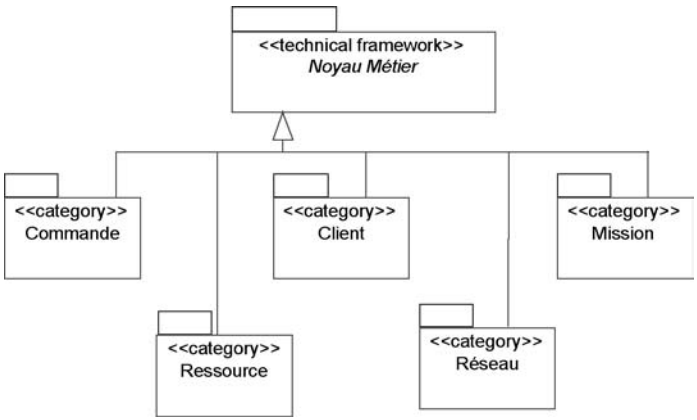

Figure 10-13 : Relations entre catégories dans le package Métier

Chaque catégorie de ce schéma correspond à l'une des catégories de conception qui réalisent la couche métier. La relation de généralisation entre packages UML est moins formelle que celle entre classes ; elle signifie simplement que le contenu d'un package est la spécialisation d'un super-package. Considérées de l'extérieur des packages, les définitions publiques du super-package, principalement ses interfaces, sont héritées par spécialisation. Ainsi, la catégorie *Métier::Client* en tant que spécialisation du framework *Noyau Métier* hérite de comportements techniques homogènes et répandus à toutes les autres catégories de la couche Métier. C'est ainsi que nous appliquons les informations issues de l'analyse objet, en projetant les catégories d'analyse sur les couches logicielles.

Conseil

TECHNIQUE D'ORGANISATION DU MODÈLE LOGIQUE DE CONCEPTION SYSTÈME

On rappelle que la conception système est le niveau d'abstraction visé par l'étape de conception préliminaire. Pour obtenir le découpage en catégories de conception, notre approche consiste à prendre tour à tour chaque composant d'exploitation et à considérer le devenir de chaque catégorie d'analyse par rapport aux *frameworks* techniques abstraits. C'est ce que nous avons réalisé pour l'étude de cas.

À l'échelle d'un système tel que SIVEx, cette technique fait apparaître des redondances entre les différentes catégories. Ces redondances conduisent à identifier les parties réutilisables que l'on isolera dans des catégories spécifiques pour en faciliter la réutilisation.

ÉTUDE DE CAS : ORGANISATION DU MODÈLE LOGIQUE

Le découpage présenté ici correspond à la façon dont les concepteurs envisagent d'organiser leur conception. Notez qu'il ne s'agit pas de la seule et bonne organisation car nous sommes ici dans le domaine subjectif des choix de conception.

Découpage pour l'application *ConVEx* :

En tant qu'application, *ConVEx* n'est concernée que par les *frameworks* issus des couches de présentation et d'application. Dans le tableau 10-3, chacune des cases remplies correspond à une catégorie de conception, qui définit brièvement le rôle qu'elle doit occuper au sein du système.

Catégorie	Noyau présentation	Noyau applicatif
Mission	Gestion de mission	Mission et Suivi de missions
	Tableau de suivi de mission	
Commande	*Consultation des commandes**	Sélection des commandes
Ressource	*Consultation des ressources**	Sélection des ressources

Réseau	Gestion des mises à disposition	Mise à disposition des ressources
	*Consultation des sites et des parcours**	Sélection des sites et des parcours
Exploitation informatique	-	Gestion des habilitations

Tableau 10-4 : Projection des catégories d'analyse dans le cadre de l'application ConVEx

Les catégories marquées d'un astérisque représentent des mécanismes réutilisables dans d'autres contextes applicatifs. Ainsi, le concepteur a intérêt à isoler ces mécanismes dans des catégories séparées, de manière à en favoriser la réutilisation.

Découpage pour le composant métier *Mission* :

Le composant *Mission* implémente son propre noyau métier et s'appuie également sur la couche d'accès aux données qui lui est propre. En règle générale, la trame des catégories identifiée en analyse est similaire à celle de la couche métier en conception.

Framework / Catégorie	Noyau métier	Accès aux données
Mission	Implémentation des interfaces IMission et ISuiviMission.	Accès aux informations des missions
Commande	Utilisation de l'interface ICommande	-
Plan Transport	Utilisation de l'interface IPlanTransport	-
Ressource	Utilisation de l'interface IRessource	-
Réseau	Utilisation de l'interface IReseau	-
Exploitation informatique	Utilisation de l'interface IHabilitation	-

Tableau 10-5 : Projection des catégories d'analyse dans le cadre du composant métier Mission

Les catégories résultantes pour la couche présentation s'établissent en fonction des besoins afférents des différentes applications. Les dépendances entre catégories pourront être ensuite définies en fonction des structures des différentes IHM conçues. Ainsi, on distingue une catégorie *Mission* d'une catégorie *SuiviMission* car les besoins de présentation seront différents. On différencie également *EnCoursCommande* de *EnCoursCommandeWeb* car les techniques employées seront cette fois-ci différentes.

Notez que les catégories de la couche Application seront identiques. Il existe en effet un fort couplage entre l'application et sa présentation, qui correspond au couplage existant entre la vue et le contrôleur du modèle MVC.

Les catégories résultantes pour la couche métier reflètent le modèle structurel d'analyse, d'où sont absents les concepts purement applicatifs. Dans l'exemple de SIVEx, l'analyste a déjà isolé dans la catégorie *Transmission Comptable* des concepts propres à l'application ; cette catégorie disparaît de la couche métier.

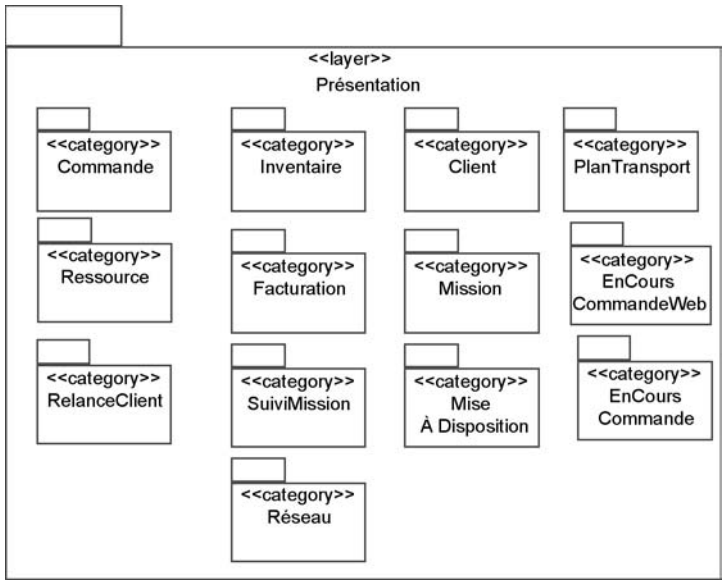

Figure 10-14 : Identification des catégories de la couche Présentation

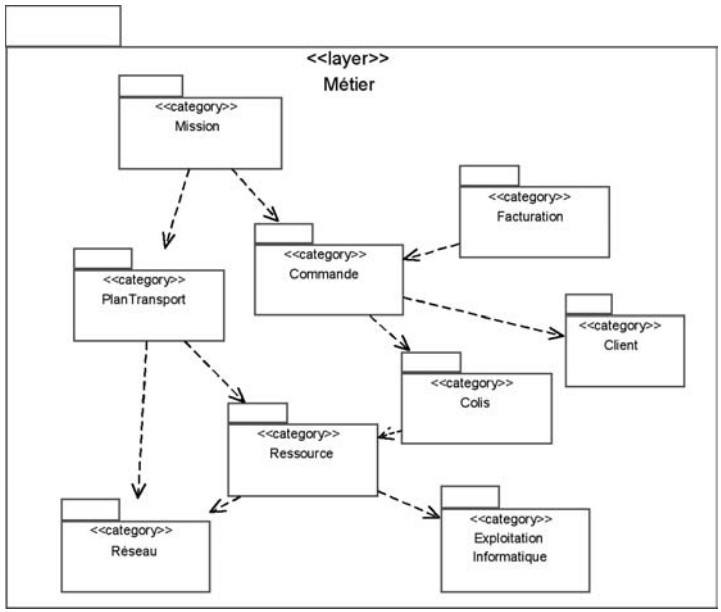

Figure 10-15 : Identification des catégories de la couche Métier

Notez que les catégories de la couche d'accès aux données seront également identiques. Il existe effectivement un couplage fort entre les concepts métier et leurs données[1].

1.Couplage que le mécanisme de persistance EJB rend inutile de décrire dans le cas où le concepteur opte pour des composants gérés par leur conteneur EJB (EJB CMP).

Définir l'interface des catégories

Rappelons que chaque catégorie regroupe des classes qui fournissent un ensemble cohérent de services aux autres parties du logiciel. Comme nous vous l'avons déjà signalé, il est possible de considérer la catégorie comme niveau d'encapsulation supérieur aux classes, avec d'une part l'interface constituée des classes publiques et d'autre part l'implémentation comportant des classes privées.

Dans un modèle, l'interface d'une catégorie se présente donc sous la forme des classes et des interfaces utilisables de l'extérieur de la catégorie. Nous avons introduit le stéréotype *interface* qui, appliqué aux packages, isole ces définitions particulières.

Les interfaces des catégories se construisent à partir des interfaces déjà identifiées des composants d'exploitation. Leur définition nécessite cependant d'être précisée et doit prendre en compte les opérations identifiées dans le modèle d'analyse. Ces opérations se répartissent sur les couches application, métier ou accès aux données en fonction de leur spécialité. D'autres opérations d'analyse correspondent plutôt à des mécanismes internes à la catégorie et ne se positionneront pas dans une interface. Le tableau suivant vous donne un exemple de répartition des opérations d'analyse.

Opération d'analyse	Description	Positionnement
Affecter commande	Associer une commande à une mission, ce qui déclenche la vérification de la charge du véhicule et la création d'une étape, s'il s'agit d'une mission de tournée.	C'est un service métier de la catégorie *Mission* qui entre dans la définition de l'interface *IMission*.
Vérifier tonnage	Opération déclenchée par l'ajout d'une commande.	C'est une opération métier exclusivement déclenchée par l'ajout d'une commande. Il s'agit donc d'une opération interne à la catégorie *Métier::Mission*.
Éditer bordereau étape	Opération déclenchée à la validation d'une mission.	C'est d'une part une opération qui gère un aspect applicatif. D'autre part, puisqu'on désire localiser cette opération sur le poste client il s'agit également d'un service de *Application::Mission*.

Tableau 10-6 : Exemple de positionnement des opérations d'analyse

Outre la prise en compte des opérations d'analyse, les *frameworks* techniques définissent eux-mêmes des interfaces identifiées lors de la conception générique. Celles-ci vont évidemment influencer l'identification et la structure des

interfaces des catégories de conception. En partant de la conception générique des couches Présentation et Métier, nous allons illustrer comment apparaissent les interfaces des catégories de conception.

Il nous paraît cependant essentiel d'introduire auparavant la définition d'un *design pattern* connu, parce qu'il formalise le concept d'interface entre catégories.


Définition

LE *DESIGN PATTERN* « FAÇADE »

Le *design pattern* « Façade » [Gamma 95] constitue une technique complémentaire pour organiser le modèle logique. Une façade a pour fonction de produire une classe qui constitue le seul point d'entrée aux services offerts par la catégorie. L'objectif de cette classe est de mieux contrôler les échanges d'une catégorie avec le reste du système et de faciliter le bon usage d'une catégorie.

La réalisation d'une façade consiste en la définition d'une classe dont les opérations reproduisent les services offerts par l'ensemble d'une catégorie. De manière à apporter plus de cohérence par rapport aux interfaces identifiées, on va développer une façade par interface de la catégorie. L'utilisation d'une ou de plusieurs façades par catégorie simplifie la compréhension des objectifs que réalisent les classes de cette catégorie. La figure suivante montre la façade de la catégorie *Métier::Mission* qui réalise l'interface *IMission*.

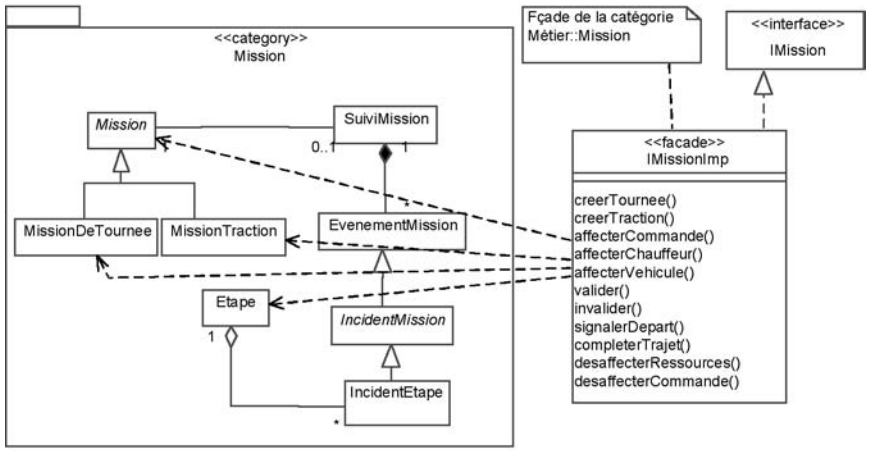

Figure 10-16 : Structure d'une façade pour la catégorie Métier::Mission

La façade masque au code client la complexité interne de la catégorie qu'il utilise. Elle réduit le nombre de classes publiques et favorise une utilisation correcte des services offerts. Elle permet également de formaliser la

dépendance entre catégories et minimise les couplages entre composants. En d'autres termes, la façade contribue à transformer les catégories en modules réutilisables et aide à mettre en œuvre le leitmotiv « réutiliser pour concevoir et concevoir pour réutiliser », exposé au chapitre 9. En conséquence, le concept de façade correspond exactement aux principes que nous cherchons à instaurer pour développer un modèle logique de conception, à la fois robuste et évolutif.


Conseil

UTILISEZ DES FAÇADES POUR CONCEVOIR LES COMPOSANTS DISTRIBUÉS

Lorsque l'on utilise un middleware de distribution tel que WebServices, EJB, .Net, Corba, DCOM ou RMI, il est indispensable de réduire le nombre d'interfaces [Orfali 98] en identifiant des façades sur les couches concernées par la distribution. Cela réduit en conséquence le nombre de références d'objets distribués et améliore sensiblement les performances de l'infrastructure de distribution.

ÉTUDE DE CAS : DÉVELOPPEMENT DES INTERFACES ET DES FAÇADES

Nous allons passer en revue tour à tour les interfaces d'un *framework* technique, une catégorie de la couche d'application et une catégorie métier.

Interface du *framework* technique *Noyau applicatif* :

Le *Noyau applicatif* est issu de la conception générique. Il est calqué sur la classe *Document* et les interfaces *CommandeApp* et *Vue*.

La classe *Vue* représente toutes les fenêtres d'un IHM qui seront porteuses d'informations provenant d'un ou de plusieurs objets du modèle.

La classe *Document* représente les entités qu'un utilisateur manipule au travers d'une ou de plusieurs vues. Le document constitue le cache des informations présentées dans une vue. Au sein d'une application, il joue généralement le rôle de conteneur des données nécessaires aux fonctions CRUD (Create, Retrieve, Update, Delete) d'un objet métier, ou le rôle de graphe d'objets permettant d'agir sur des relations entre objets métiers, ou encore celui d'ensemble d'objets métier de même type pour les recherches et la définition de listes. Ce concept est associé au design pattern *Observateur* qui vous sera présenté au chapitre 11.

La classe *CommandeApp* (commande applicative) représente les processus de l'application. Ce sont toutes les exécutions déclenchées par un utilisateur depuis une vue. Ce concept est associé au design pattern *Commande* qui vous sera également présenté au chapitre 11.

L'interface du *framework* technique est donc composée des trois classes représentées à la figure suivante.

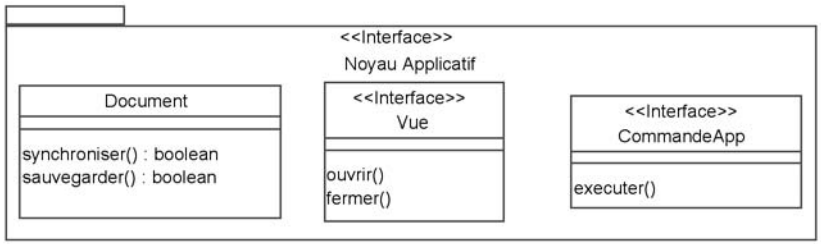

Figure 10-17 : Interface du noyau applicatif

Les classes publiques de la catégorie *Application::Mission* :

Il y aura autant de classes publiques qu'il y a de documents et de commandes applicatives, les panneaux d'IHM correspondant aux vues appartiennent quant à eux à la couche de présentation. Le diagramme ci-après représente les documents utilisés par l'utilisateur, dont les deux plannings d'affichage possibles : le planning d'affectation des chauffeurs corrélé avec le celui des véhicules.

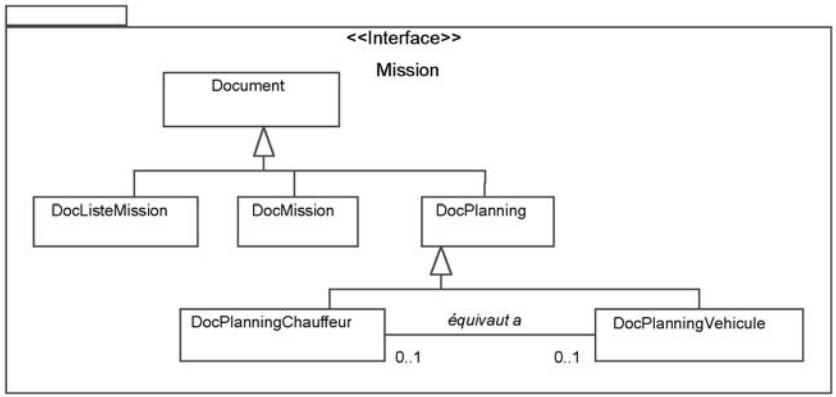

Figure 10-18 : Interface des documents de la catégorie Application::Mission

Les commandes applicatives sont de deux ordres : celles issues d'une action portant sur une classe de la couche métier, et celles qui n'ont qu'une portée applicative. Les premières proviennent d'opérations déjà identifiées en analyse, ce sont par exemple toutes les opérations d'affectation sur une mission. Les secondes peuvent résulter soit de l'identification des besoins d'IHM, soit de l'analyse de l'application ; il s'agit par exemple des opérations de sélection, de recherche ou d'impression.

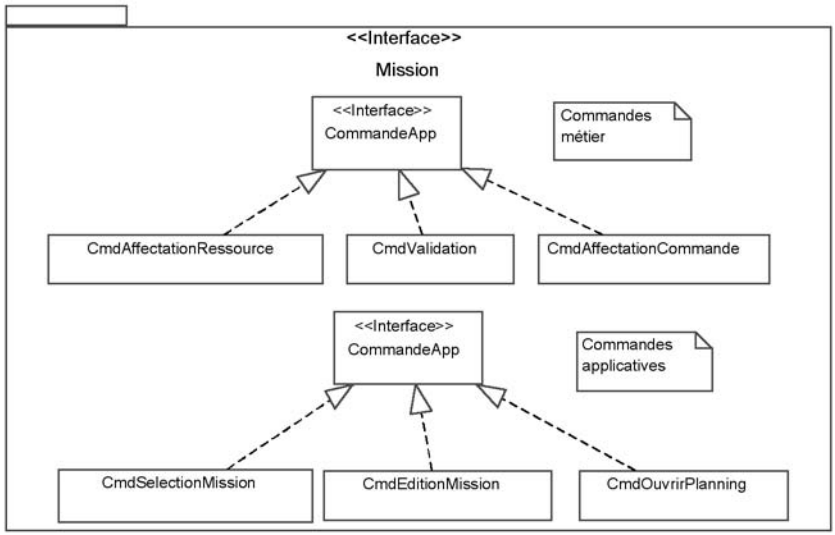

Figure 10-19 : Interface des commandes de la catégorie Application::Mission

Interface de la catégorie *Métier::Mission* :

Les façades de cette catégorie sont identifiées à partir des opérations que le composant métier doit mettre à la disposition des applications. Conformément aux interfaces déjà identifiées sur le composant distribué *Mission*, nous développons les deux façades *IMission* et *ISuiviMission*. Encore une fois, il faut tenir compte ici du *framework* technique *Noyau Métier*. Ce dernier calqué sur le framework EJB définit l'interface, l'implémentation et le gestionnaire (classe *home*) pour chacun des composants métier décrits par le mécanisme EJB. L'architecte logiciel a d'autre part conçu pour chaque composant métier, une interface « critères » représentant les différentes façons de retrouver, d'ordonner et de lister les objets métier en fonction de requêtes qui ont été prédéfinies par l'analyse des utilisations possibles du composant métier. Cette conception est expliquée plus en détail au chapitre 11.

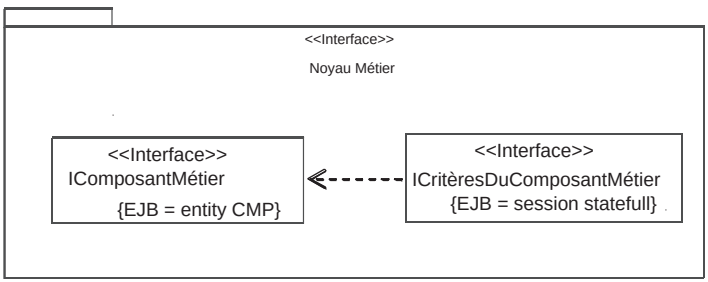

Figure 10-20 : Interface du framework technique Noyau Métier

- Le composant métier réalise un objet distribué associé directement à un objet d'analyse. On peut y recourir lorsque plusieurs applications doivent se synchroniser parallèlement sur le même objet. Le composant métier est donc un EJB dans notre conception, comme c'est le

cas pour l'en-cours de commande qui est mis à jour par les processus d'acheminement des colis, tout en étant observé par les clients et les répartiteurs.

- Le gestionnaire du composant métier (classe *home* du mécanisme EJB), implicite par la déclaration de la valeur étiquetée « EJB », pilote les cycles de vie des instances. C'est notamment à lui que s'adressent les applications pour contrôler la création et la modification des objets.
- Les critères du composant métier correspondent à des services utilitaires : échanger une référence d'objet métier par l'intermédiaire d'une clé de référence métier ou exprimer un critère de sélection d'objets prédéfini par l'analyse des cas d'utilisation et de leurs besoins d'IHM.
- Les objets *Mission* et *SuiviMission* sont distribués suivant ce framework, comme l'illustre le diagramme ci-après. La figure 10-21 schématise la structure de l'interface du package *Métier::Mission* et illustre la signification d'une généralisation entre une catégorie métier et un framework abstrait provenant de la conception générique.

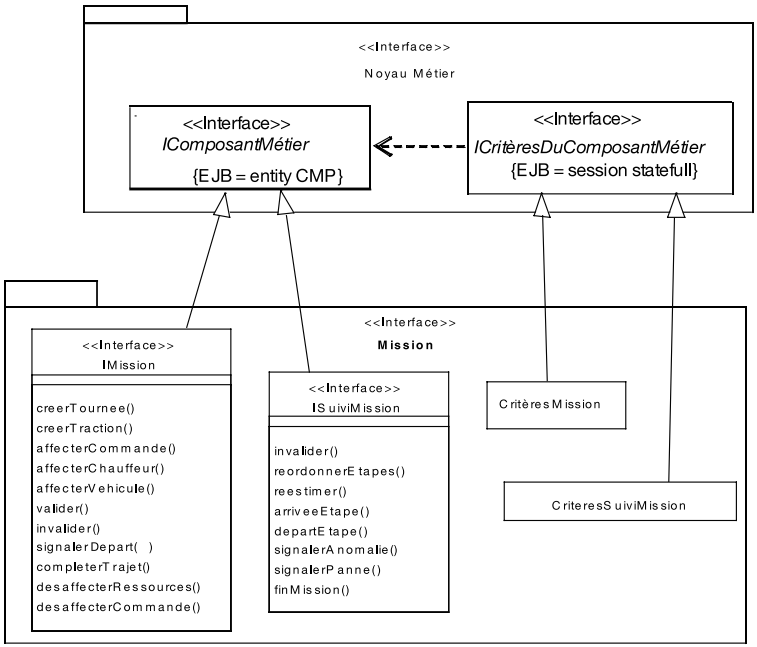

Figure 10-21 : Interface de la catégorie Métier::Mission

Le lecteur peut se demander à juste titre, l'intérêt de concevoir le framework noyau métier, alors qu'EJB en soi représente déjà une conception « prête-à-réutiliser ». Nous avons souvent ressenti le besoin d'aller un peu plus loin dans nos conceptions, en introduisant par exemple des mécanismes généralisés pour lister les objets suivant des clés différentes, ou bien pour « réserver » l'objet à l'usage d'une session particulière – créant ainsi un mécanisme de verrouillage métier au-dessus des simples mécanismes techniques fournis par les moniteurs transactionnels et les bases de données.

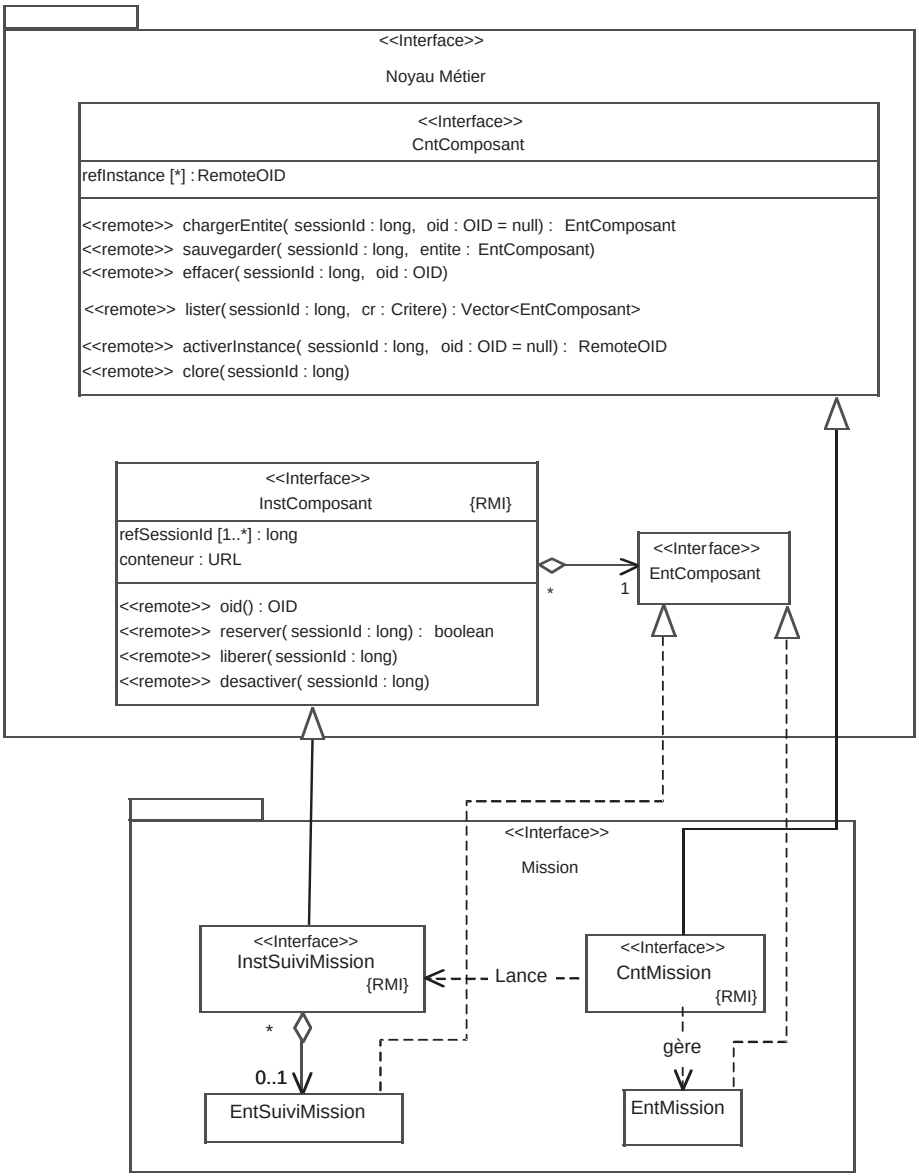

Figure 10-22 : Interface de la catégorie Métier::Mission héritée du noyau métier

Concevoir la structure objet des IHM

Nous sommes maintenant à un état d'avancement où la structure de la conception est complètement définie, ainsi que les interfaces qui permettent aux différentes catégories de communiquer entre elles. Pour compléter la vision du système, il est nécessaire d'en concevoir les IHM. Leur conception ne fait pas spécifiquement appel à des techniques UML, mais plutôt à des règles d'ergonomie accompagnées d'outils de fabrication d'écrans.

La conséquence notable est l'apparition des classes correspondant aux vues panneaux ou pages d'IHM de la couche de présentation. D'une part ces panneaux contribuent à préciser l'identification des documents et des commandes de l'application, d'autre part on peut estimer que ces classes constituent en quelque sorte les interfaces des catégories de la couche de présentation.

ÉTUDE DE CAS : CONCEPTION DES IHM DU RÉPARTITEUR

Les panneaux de l'application *ConVEx.exe* - qui représente l'application cliente du répartiteur développée en client lourd (Java Swing) pour des raisons d'ergonomie - correspondent aux classes Documents identifiées dans le package *Application::Mission*. Le framework technique *Noyau Présentation* consiste simplement à établir le lien entre la classe *JFrame* de Java Swing et l'interface *Vue* nécessaire au pilotage des applications.

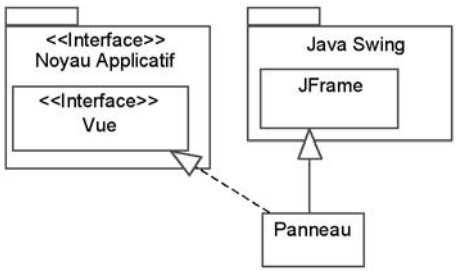

Figure 10-23 : Définition de la classe technique Panneau (Java Swing)

À chaque panneau identifié correspond une classe d'IHM. La conception détaillée des protocoles entre l'application et l'utilisateur pourra ensuite être formalisée par des diagrammes d'interactions UML (séquences ou collaborations) mettant en jeu l'acteur, le panneau, les commandes et les documents.

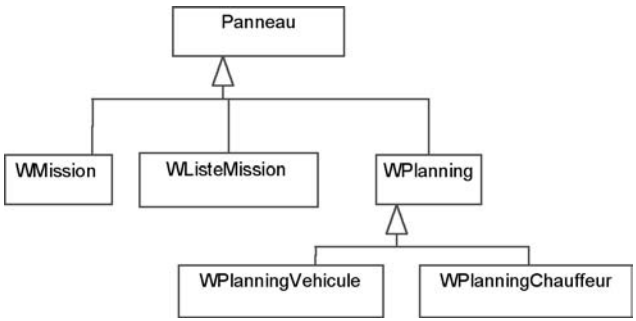

Figure 10-24 : Structure objet des panneaux de l'application ConVEx

Dans le cas des postes de travail déployés en client léger, chaque page JSP représente une classe. En l'occurrence, l'utilisation de Struts nécessite d'associer à chaque page une classe JavaBean. Le bean définissant exhaustivement l'ensemble des données échangées avec l'utilisateur joue en fait le rôle de document vis-à-vis de notre conception.

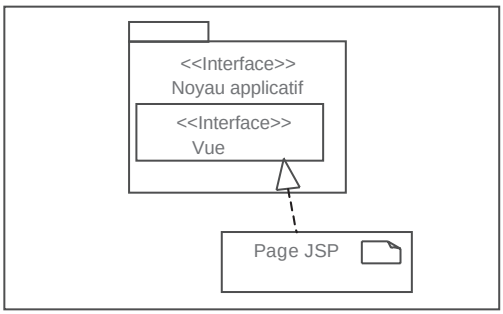

Figure 10-25 : Définition de la classe technique Page JSP

Organiser la configuration logicielle

Le travail de conception préliminaire s'achève par la définition des unités de fabrication du logiciel. Il s'agit ici de consolider le modèle logique avec les réutilisations de code détectées et de compléter la configuration logicielle déjà commencée en conception générique.

A priori, chaque catégorie se transforme en un sous-système de configuration logicielle. Il faut cependant prendre en compte les parties à extraire pour bénéficier de la réutilisation de code. Par exemple, le sous-système de présentation des commandes doit isoler spécifiquement un sous-système regroupant les classes nécessaires à la construction de la fenêtre de recherche et de sélec-

tion de commandes. Ce sous-système est en effet réutilisé pour la présentation de l'application *ConVEx*.

Inversement, plusieurs catégories peuvent être regroupées dans le même sous-système de configuration logicielle. Les couches présentation et application sont par exemple souvent indissociables et regroupées dans la même unité de fabrication. C'est le cas des catégories *Présentation::Mission* et *Application::Mission* qui participent à la définition d'un même sous-système de constitution de l'application *ConVEx*. C'est seulement suite à la conception détaillée que chaque sous-système pourra être précisé avec les composants correspondants aux classes Java qu'il faudra coder.

ÉTUDE DE CAS : ORGANISATION DE LA CONFIGURATION LOGICIELLE

L'application *ConVEx* est élaborée à partir de deux sous-systèmes de construction du client ; l'environnement d'édition des missions et l'environnement de suivi des missions. Les deux sous-systèmes correspondant aux panneaux de sélection ont été isolés afin de permettre leur réutilisation. Les sous-systèmes issus de la conception générique sont intégrés.

Côté serveur, il existe un sous-système de fabrication du composant distribué *Mission* ainsi, qu'un sous-système pour l'accès aux données. Les deux sous-systèmes sont séparés par choix d'architecture technique. Il est en effet nécessaire de ménager un serveur spécifique gérant les connexions aux bases de données.

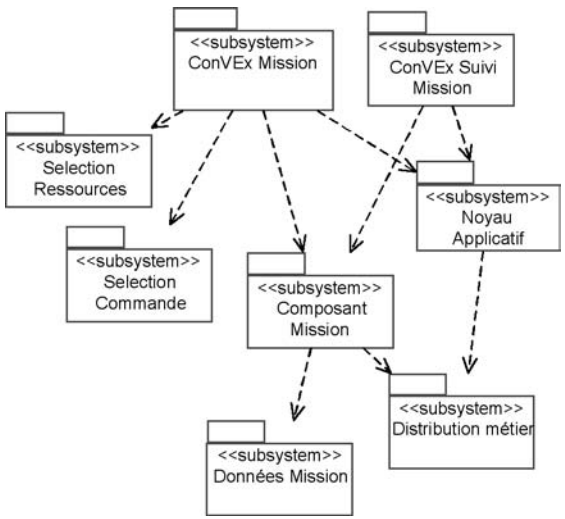

Figure 10-26. : Organisation des sous-systèmes de l'application ConVEx

Pour la suite, chacun des sous-systèmes donne lieu à une subdivision en composants représentant l'implémentation Java des classes de conception.

IL Y A COMPOSANT ET COMPOSANT !

Pour terminer ce chapitre, il nous semble nécessaire de revenir sur la notion de composant et d'y apporter quelques précisions en matière de terminologie.

Un composant au sens d'UML représente un élément physique à partir duquel on construit le logiciel. Il peut s'agir d'un fichier de code tout comme d'un exécutable. Comme nous vous l'avons expliqué, ce concept est trop vaste pour être exploitable, et dans le cadre des projets, nous avons été amenés à distinguer les composants d'exploitation des composants de la configuration logicielle.

On parle également de composant pour la distribution. Dans ce cas, un composant distribué est un composant de la vue d'exploitation, au même titre qu'une application ou qu'une instance de base de données. Le composant distribué participe au logiciel par l'ensemble des services accessibles *via* un middleware. Le composant distribué correspond donc à une même unité d'exécution qui déclare ses services par l'intermédiaire de différentes interfaces. L'interface d'un composant distribué ne doit pas être confondue avec le stéréotype *interface* d'UML qui correspond plutôt à l'interface de Java. En conception préliminaire, l'interface du composant distribué est plutôt une ou plusieurs catégories qui permettent de déclarer des services. En conception détaillée, les interfaces du composant distribué doivent ensuite coller à la notion d'interface EJB.

On parle enfin de composant métier, ce qui intègre une dimension fonctionnelle. Il s'agit cette fois d'un composant de configuration logicielle qui regroupe les services issus d'une même partie spécifique d'un métier. Un composant métier s'identifie à l'aide des catégories d'analyse et sert à réutiliser les notions communes à plusieurs applications. Un objet métier correspond à une classe de l'analyse du domaine et participe bien entendu à la réalisation d'un composant métier. Un composant distribué peut regrouper plusieurs composants métiers. En les localisant sur le réseau, la distribution facilite la réutilisation des composants métier.

Phases de réalisation en conception préliminaire

La conception préliminaire est avant tout affaire d'organisation ; il s'agit de préparer le modèle de conception en intégrant les résultats provenant à la fois

de l'analyse et de la conception générique. Dans le cadre d'une application client/serveur, la conception du déploiement des postes de travail et des composants d'exploitation constitue un premier guide d'organisation.

La vue d'exploitation est constituée d'applications, de composants distribués et d'instances de base de données nécessaires au système. L'identification de ses interfaces précise la définition d'un composant distribué, tout comme les interfaces utilisateur précisent celle d'une application. Des interfaces EAI peuvent également être définies lorsque l'on doit intégrer des progiciels dans son système.

Les catégories de conception sont identifiées à partir des *frameworks* abstraits de la conception générique et des catégories de l'analyse. Une première liste des catégories de conception fait apparaître les éléments communs que l'on isolera en vue d'une réutilisation. Chaque catégorie doit ensuite définir son interface. Dans le cadre d'une distribution, le recours au *design pattern* Façade permet de réduire la multiplicité des interfaces et facilite le protocole d'utilisation du composant distribué. Pour les applications, il est utile de concevoir les IHM afin d'avoir une vision précise des interfaces des couches présentation et application.

La conception préliminaire se termine en organisant la configuration logicielle du développement.

Le détail du processus suggéré pour la conception préliminaire est le suivant :

1. Concevez le déploiement :
- identifiez les postes de travail ;
- déployez-les sur le réseau physique ;
- commentez et justifiez les caractéristiques opérationnelles du déploiement : dimensionnement des réseaux, dispositifs physiques de sécurité, localisation des bases de données, etc.

2 Élaborez le modèle d'exploitation :
- identifiez les applications à partir des cas d'utilisation ;
- identifiez les composants distribués à partir des catégories d'analyse ;
- ébauchez les interfaces des composants distribués ;
- identifiez et spécifiez le cas échéant les interfaces EAI ;
- identifiez les instances de base de données afin d'optimiser la distribution ;
- énumérez les interfaces utilisateurs des applications ;
- complétez la vue d'exploitation.

3. Organisez le modèle logique de conception :

- identifiez les catégories de conception à partir des catégories d'analyse et des *frameworks* techniques abstraits ;
- isolez les mécanismes communs dans des catégories séparées afin d'organiser leur réutilisation ;
- structurez le modèle logique suivant les couches logicielles et disposez-y les catégories identifiées.

4. Créez les interfaces des catégories :

- répartissez les opérations d'analyse suivant les couches ;
- identifiez les opérations accessibles depuis l'extérieur de la catégorie ;
- concevez l'interface des catégories conformément aux *frameworks* techniques réalisés.

5. Mettez au point la présentation des applications :

- élaborez une maquette d'IHM pour les interfaces des applications ;
- reportez la structure des classes d'IHM dans les catégories des couches de présentation et d'application.

6. Structurez la configuration logicielle :

- identifiez les sous-systèmes à partir des catégories de conception ;
- complétez la configuration logicielle déjà commencée en conception générique.

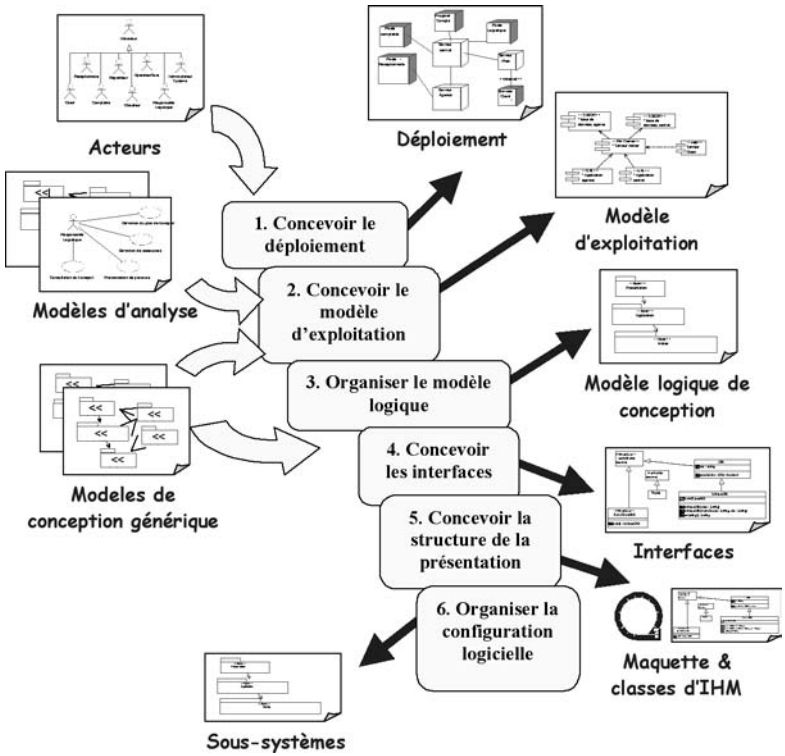

Figure 10-27 : Construction de l'étape de conception préliminaire

Dans le chapitre 11 « Conception détaillée », nous aborderons en détail l'élaboration des catégories suivantes :

- *Présentation::Mission*, pour illustrer la documentation d'un IHM avec UML ;
- *Application::Mission*, pour montrer comment l'IHM se couple avec les concepts de l'application ;
- *Métier::Mission*, pour étudier un modèle de distribution avec UML ;
- *StockageDonnées::Mission*, pour examiner un modèle de persistance relationnel à partir d'un schéma objet.