UML en action

De l'analyse des besoins à la conception en Java

Pascal ROQUES
Franck VALLÉE

Deuxième édition 2003

ISBN : 2-212-11213-0

EYROLLES

Chapitre 2

Processus et architecture

Une introduction aux processus unifiés

La complexité croissante des systèmes informatiques a conduit les concepteurs à s'intéresser aux méthodes. Bien que ce phénomène ait plus de 30 ans, nous ne pouvons constater aujourd'hui l'existence d'une règle qui soit à la fois très formelle et commune aux différentes cultures. On a par exemple comptabilisé en 1994 jusqu'à 50 méthodes objets différentes. Chaque méthode se définit par une notation et un processus spécifique, mais la plupart convergent en ce qui concerne la sémantique de leur notation. Néanmoins le travail de définition d'un processus est toujours resté vague et succinct. UML a ouvert le terrain de l'unification en fusionnant les notations et en apportant précision et rigueur à la définition des concepts introduits. L'introduction d'UML a apporté un élan sans précédent à la technologie objet, puisqu'elle y propose un standard de niveau industriel.

Il reste cependant à définir le processus pour réellement capitaliser des règles dans le domaine du développement logiciel. Définir un seul processus universel serait une grave erreur car la variété des systèmes et des techniques ne le permet pas. Dans la lancée d'UML, les 3 amigos[1] ont donc travaillé à unifier non pas les processus, mais plus exactement les meilleures pratiques de développement orienté objet. Le résultat de ces travaux est actuellement disponible dans [Jacobson 99] et [Kruchten 99] et surtout dans le produit commercial RUP de Rational (*www.rational.com*).

1. Dénomination de Grady Booch, James Rumbaugh, et Ivar Jacobson qui sont les « pères » d'UML.


Définition

PROCESSUS DE DÉVELOPPEMENT LOGICIEL

Un processus définit une séquence d'étapes, en partie ordonnées, qui concourent à l'obtention d'un système logiciel ou à l'évolution d'un système existant.

L'objet d'un processus de développement est de produire des logiciels de qualité qui répondent aux besoins de leurs utilisateurs dans des temps et des coûts prévisibles. En conséquence, le processus peut se décomposer suivant deux axes de contrôle sur le développement :

- l'axe de développement technique, qui se concentre principalement sur la qualité de la production ;
- l'axe de gestion du développement, qui permet la mesure et la prévision des coûts et des délais. Nous ne traitons pas cet aspect dans « UML en *Action*… ».


Définition

PROCESSUS UNIFIÉ (UNIFIED PROCESS)

Un processus unifié est un processus de développement logiciel construit sur UML ; il est itératif et incrémental, centré sur l'architecture, conduit par les cas d'utilisation et piloté par les risques.

La gestion d'un tel processus est organisée suivant les 4 phases suivantes: préétude (*inception*), élaboration, construction et transition.

Ses activités de développement sont définies par 5 disciplines fondamentales qui décrivent la capture des besoins, l'analyse et la conception, l'implémentation, le test et le déploiement.

Le processus unifié doit donc être compris comme une trame commune des meilleures pratiques de développement, et non comme l'ultime tentative d'élaborer un processus universel. La définition d'un processus UP est donc constituée de plusieurs disciplines d'activité de production et de contrôle de cette production. Tout processus UP répond aux caractéristiques ci-après.

- Il est itératif et incrémental. La définition d'itérations de réalisation est en effet la meilleure pratique de gestion des risques d'ordre à la fois technique et fonctionnel. On peut estimer qu'un projet qui ne produit rien d'exécutable dans les 9 mois court un risque majeur d'échec. Chaque itération garantit que les équipes sont capables d'intégrer l'environnement technique pour développer un produit final et fournit aux utilisateurs un résultat tangible de leurs spécifications. Le suivi des itérations constitue par ailleurs un excellent contrôle des coûts et des délais.

- Il est piloté par les risques. Dans ce cadre, les causes majeures d'échec d'un projet logiciel doivent être écartées en priorité. Nous identifions une première cause provenant de l'incapacité de l'architecture technique à répondre aux contraintes opérationnelles, et une seconde cause liée à l'inadéquation du développement aux besoins des utilisateurs.
- Il est construit autour de la création et de la maintenance d'un modèle, plutôt que de la production de montagnes de documents. Le volume d'informations de ce modèle nécessite une organisation stricte qui présente les différents points de vue du logiciel à différents degrés d'abstraction. L'obtention de métriques sur le modèle fournit par ailleurs des moyens objectifs d'estimation.
- Il est orienté composant. Tant au niveau modélisation que production, c'est une garantie de souplesse pour le modèle lui-même et le logiciel qu'il représente. Cette pratique constitue le support nécessaire à la réutilisation logicielle et offre des perspectives de gains non négligeables.
- Il est orienté utilisateur, car la spécification et la conception sont construites à partir des modes d'utilisation attendus par les acteurs du système.

Le processus 2TUP

2TUP signifie « 2 Track Unified Process ». C'est un processus UP qui répond aux caractéristiques que nous venons de citer. Le processus 2TUP apporte une réponse aux contraintes de changement continuel imposées aux systèmes d'information de l'entreprise. En ce sens, il renforce le contrôle sur les capacités d'évolution et de correction de tels systèmes. « 2 Track » signifie littéralement que le processus suit deux chemins. Il s'agit des chemins « fonctionnels » et « d'architecture technique », qui correspondent aux deux axes des changements imposés au système informatique.

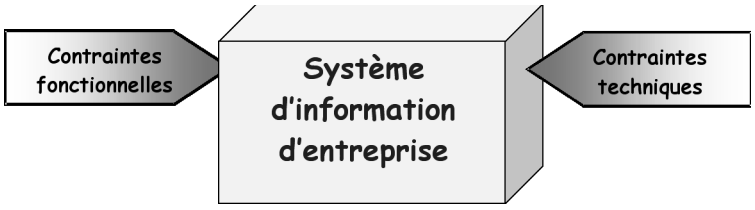


Figure 2-1. : Le système d'information soumis à deux natures de contraintes

L'axiome fondateur du 2TUP consiste à constater que toute évolution imposée au système d'information peut se décomposer et se traiter parallèlement,

suivant un axe fonctionnel et un axe technique. Pour illustrer cet axiome, prenons les trois exemples suivants :

1. une agence de tourisme passe des accords avec une compagnie aérienne de sorte que le calcul des commissions change. En l'occurrence, les résultats issus de la branche fonctionnelle qui évoluent pour prendre en compte la nouvelle spécification ;
2. cette même entreprise décide d'ouvrir la prise de commande sur le Web. Si rien ne change fonctionnellement, en revanche, l'architecture technique du système évolue ;
3. cette entreprise décide finalement de partager son catalogue de prestations avec les vols de la compagnie aérienne. D'une part, la fusion des deux sources d'informations imposera une évolution de la branche fonctionnelle, d'autre part, les moyens techniques de synchronisation des deux systèmes conduiront à étoffer l'architecture technique du système. L'étude de ces évolutions pourra être menée indépendamment, suivant les deux branches du 2TUP.

À l'issue des évolutions du modèle fonctionnel et de l'architecture technique, la réalisation du système consiste à fusionner les résultats des deux branches. Cette fusion conduit à l'obtention d'un processus de développement en forme de Y, comme illustré par la figure 2-2.

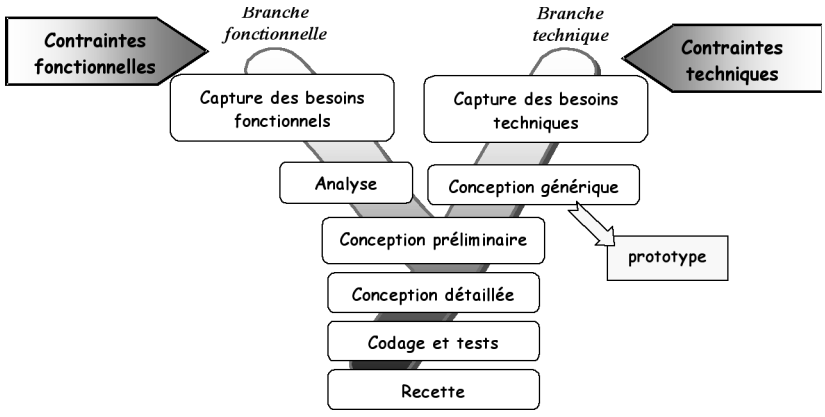


Figure 2-2. : Le processus de développement en Y

La branche gauche (fonctionnelle) comporte :

- la capture des besoins fonctionnels, qui produit un modèle des besoins focalisé sur le métier des utilisateurs. Elle qualifie au plus tôt le risque de produire un système inadapté aux utilisateurs. De son côté, la maîtrise d'œuvre consolide les spécifications et en vérifie la cohérence et l'exhaustivité l'analyse, qui consiste à étudier précisément la spécification fonc-

tionnelle de manière à obtenir une idée de ce que va réaliser le système en termes de métier. Les résultats de l'analyse ne dépendent d'aucune technologie particulière.

La branche droite (architecture technique) comporte :

- la capture des besoins techniques, qui recense toutes les contraintes et les choix dimensionnant la conception du système. Les outils et les matériels sélectionnés ainsi que la prise en compte de contraintes d'intégration avec l'existant conditionnent généralement des prérequis d'architecture technique ;
- la conception générique, qui définit ensuite les composants nécessaires à la construction de l'architecture technique. Cette conception est la moins dépendante possible des aspects fonctionnels. Elle a pour objectif d'uniformiser et de réutiliser les mêmes mécanismes pour tout un système. L'architecture technique construit le squelette du système informatique et écarte la plupart des risques de niveau technique. L'importance de sa réussite est telle qu'il est conseillé de réaliser un prototype pour assurer sa validité.

La branche du milieu comporte :

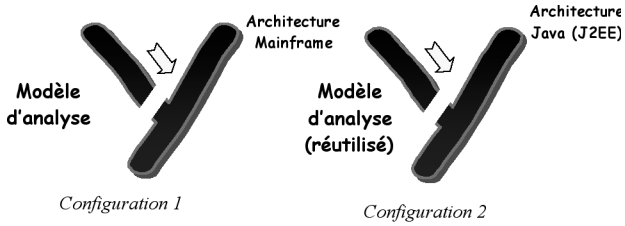
- la conception préliminaire, qui représente une étape délicate, car elle intègre le modèle d'analyse dans l'architecture technique de manière à tracer la cartographie des composants du système à développer ;
- la conception détaillée, qui étudie ensuite comment réaliser chaque composant ;
- l'étape de codage, qui produit ces composants et teste au fur et à mesure les unités de code réalisées ;
- l'étape de recette, qui consiste enfin à valider les fonctions du système développé.

L'ensemble de ces étapes de développement sera illustré tout au long de cet ouvrage par la mise en application du processus 2TUP à l'étude de cas SIVEx. Seules les deux dernières étapes, ne relevant pas de l'utilisation d'UML, ne seront pas abordées dans cet ouvrage.

LES BRANCHES DU "Y" PRODUISENT DES MODÈLES RÉUTILISABLES

La branche gauche capitalise la connaissance du métier de l'entreprise. Elle constitue généralement un investissement pour le moyen et le long terme. Les fonctions du système d'informations sont en effet indépendantes des technologies utilisées. Cette évidence n'a malheureusement pas souvent été mise en pratique, car dans bien des cas, la connaissance fonctionnelle d'un produit se perd dans les milliers de ligne de code de sa réalisation.

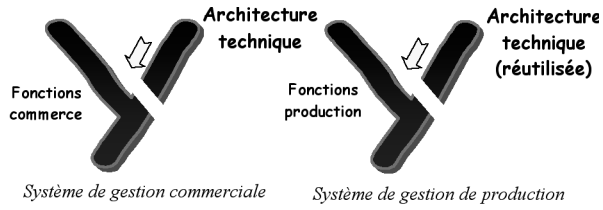
L'entreprise qui maintient le modèle fonctionnel de sa branche gauche est pourtant à même de le réaliser sous différentes technologies. Il suffit de « greffer » une nouvelle architecture technique pour mettre à jour un système existant.



La branche droite capitalise quant à elle un savoir-faire technique. Elle constitue un investissement pour le court et le moyen terme. Les techniques développées pour le système peuvent l'être en effet indépendamment des fonctions à réaliser.

L'architecture technique est d'ailleurs de moins en moins la préoccupation des services informatiques dont l'entreprise n'a pas vocation à produire du code. L'existence de produits tels que les serveurs d'application ou la standardisation des services Web reflète cette tendance à pouvoir disposer sur le marché d'architectures techniques « prêtes à intégrer ».

Une architecture technique est en effet immédiatement réutilisable pour les différentes composantes fonctionnelles d'un même système d'entreprise.



Un processus itératif et incrémental piloté par les risques

Définition

ITÉRATION

Une itération est une séquence distincte d'activités avec un plan de base et des critères d'évaluation, qui produit une release (interne ou externe). Le contenu d'une itération est porteur d'améliorations ou d'évolutions du système et il peut être évalué par les utilisateurs.


Définition

INCRÉMENT

Un incrément est la différence (delta) entre deux releases produites à la fin de deux itérations successives.

Une ou plusieurs itérations s'inscrivent dans chacune des phases de gestion de projet et servent en conséquence à réaliser l'objectif propre à chaque phase.

- En préétude, les itérations servent à évaluer la valeur ajoutée du développement et la capacité technique à le réaliser.
- En phase d'élaboration, elles servent à confirmer l'adéquation du système aux besoins des utilisateurs.
- En phase de construction, elles servent à livrer progressivement toutes les fonctions du système.
- En phase de transition, elles servent à corriger et à faire évoluer le système.

Le processus de développement en Y se reproduit à différents niveaux d'avancement en se terminant sur la livraison d'une nouvelle release. La figure 2-3 illustre un scénario typique d'avancement sur les 3 premières phases de gestion de projet.

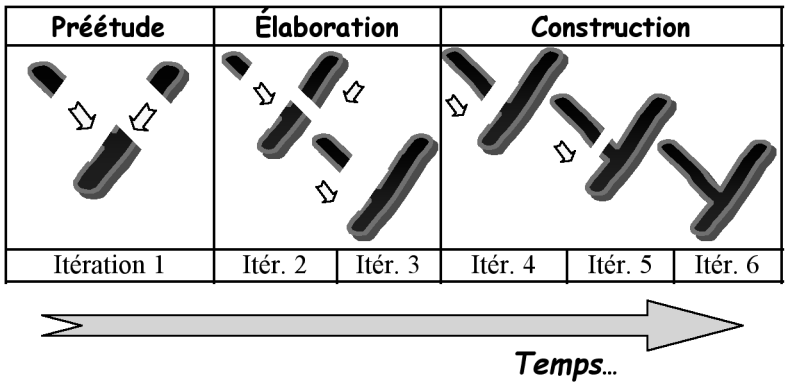


Figure 2-3. : Plan d'itération suivant l'axe de gestion de projet

- L'itération 1 développe les fonctions de validation du principe du système et intègre les outils prévus pour le développement.
- L'itération 2 est focalisée sur l'architecture ; elle peut être considérée comme le prototype de réalisation technique.
- L'itération 3 avance dans la réalisation des fonctions les plus prioritaires de manière à présenter une première version de déploiement pour les utili-

sateurs. Elle permet entre-temps d'améliorer et de compléter l'architecture technique.

- Les itérations suivantes avancent dans la réalisation des fonctions jusqu'à l'obtention complète du système initialement envisagé.

Par définition, chaque itération passe en revue toutes les activités du processus en Y. Il est évident que l'effort consacré à chaque activité n'est pas identique suivant la phase de développement du projet.

Pour reprendre l'exemple précédent, on peut estimer que l'itération de préétude consacre relativement plus d'efforts à la capture des besoins que les itérations d'élaboration. Le tableau ci-dessous établit pour chaque phase, un ordre de grandeur des efforts consacrés. Ces chiffres ne constituent en aucun cas une référence qui puisse être réutilisée telle que dans votre projet.

Effort relatif par activité	Itération 1 (préétude)	Itération 2 (élaboration)	Itération 4 (construction)
Capture des besoins fonctionnels	16 %	1 %	2 %
Analyse	22 %	9 %	4 %
Capture des besoins techniques	2 %	4 %	0 %
Conception générique	5 %	14 %	2 %
Conception préliminaire	10 %	10 %	10 %
Conception détaillée	10 %	14 %	20 %
Codage et tests	28 %	26 %	30 %
Recette	7 %	22 %	32 %
(Total des activités du Y)	100 %	100 %	100 %

Tableau 2-1 : Répartition des efforts par activité suivant les différentes phases du projet

Etude

LES EFFORTS CONSACRÉS À CHAQUE ITÉRATION NE S'ARRÊTENT PAS AUX ACTIVITÉS DU Y

Le processus en Y est focalisé sur les seules activités de développement technique, mais en réalité, la fabrication d'un incrément nécessite deux types d'activités support dont il faut tenir compte dans l'estimation des efforts.

Les tâches d'industrialisation du logiciel concernent la mise en place des moyens qui vont permettre un développement de qualité. On y regroupe les activités d'administration et d'appropriation des outils de développement, ainsi que la gestion de configuration très importante pour le contrôle du changement.

Les tâches de pilotage regroupent les efforts consacrés à contrôler et à anticiper l'avancement du projet. Elles comprennent également les temps de réunion et de coordination.

Les phases définies par un UP sont prévues pour la gestion des risques. Comme on l'a vu au paragraphe précédent, le couplage avec une politique d'itérations permet de traiter en priorité les problèmes présentant le plus de risques.

La configuration du processus en Y a également été conçue pour gérer en priorité et en parallèle les risques de nature fonctionnelle et technique :

- d'une part, les risques d'imprécision fonctionnelle, et d'inadéquation aux besoins sur la branche gauche,
- d'autre part les risques d'incapacité à intégrer les technologies, et d'inadaptation technique sur la branche droite.

Il est d'usage de consacrer la phase de préétude à l'étude des fonctionnalités que produira le système logiciel. Afin de pallier un risque courant d'imprécision dans la définition fonctionnelle du système, les techniques de formalisation du contexte (voir chapitre 4) vous aideront à établir la frontière entre le système et le reste du monde.

L'exigence d'aboutir à une première release permet également d'évaluer très rapidement la capacité à intégrer les technologies nécessaires au projet. À l'issue de la phase de préétude, un choix s'impose : faut-il continuer ou non le projet ? Pour y répondre, le suivi des efforts est un élément clé qui permet d'estimer la rentabilité globale du projet.

La phase d'élaboration est ensuite consacrée au développement de l'architecture technique et à la réalisation des fonctions les plus prioritaires. À l'issue de cette phase, le comportement technique du système (temps de réponse, tenue en charge, robustesse, etc.) et son accueil par les utilisateurs (réponse aux besoins, ergonomie, etc.) doivent écarter définitivement tous les risques d'échec.

Un processus piloté par les exigences des utilisateurs

Comme nous l'avons souligné précédemment, un bon nombre de risques proviennent de la non-adéquation technique et fonctionnelle du système aux besoins des utilisateurs. Les exigences des utilisateurs sont donc prioritairement traitées dans les deux branches du processus en Y en considérant deux types d'acteurs différents du système informatique :

- l'utilisateur consommateur des fonctions du système, qui correspond généralement à un poste, un rôle ou un métier dans l'entreprise. Il convient dans ce cadre de se focaliser sur la plus-value que lui apporte le système dans l'exercice de sa profession ;
- l'utilisateur exploitant le système, qui correspond plutôt à un rôle technique et opérationnel commun à la plupart des systèmes informatiques. Dans le domaine client/serveur, les utilisateurs, considérés au sens large, attendent des performances, une tenue à la charge, une sécurité d'accès, etc. L'axe technique permet également d'introduire le point de vue des exploitants et des administrateurs souvent oubliés dans la livraison finale d'un produit.

L'enjeu du processus en Y est donc de développer le point de vue utilisateur et de construire la spécification puis la conception objet à partir des concepts maniés par les acteurs du système. Les cas d'utilisation sont justement des outils construits pour définir les besoins, développant de surcroît le point de vue des utilisateurs. Il convient par la suite de montrer comment s'établit la traçabilité entre les cas d'utilisation et le modèle de conception. La figure 2-4 montre comment les cas d'utilisation influencent l'analyse et la conception d'architecture du système.

- Sur la branche gauche, pour la capture des besoins fonctionnels, les cas d'utilisation portent uniquement sur la plus-value métier des fonctions du système. Chaque cas d'utilisation met en évidence des classes d'analyse qui sont les concepts utilisés par l'utilisateur et des scénarios qui établissent les comportements attendus du système.
- Sur la branche droite, pour la capture des besoins techniques, la nature des cas d'utilisation a été quelque peu adaptée en fonction des plus-values opérationnelles du système pour ses exploitants. Chaque cas d'utilisation technique structure des spécifications d'architecture que l'on peut par la suite décomposer par couche logicielle. Les cas d'utilisation techniques permettent de concevoir les classes qui vont offrir une réponse aux contraintes opérationnelles du système. Les interactions entre ces classes permettent par ailleurs d'étudier les échanges entre classes, de consolider et de vérifier la conception des cas d'utilisation techniques.
- Lors de la conception préliminaire, les classes obtenues naissent de la distribution des classes d'analyse sur les couches logicielles. Les interactions entre classes de conception permettent de consolider et de vérifier à terme

la conception des cas d'utilisation fonctionnelle tenant compte des contraintes opérationnelles.

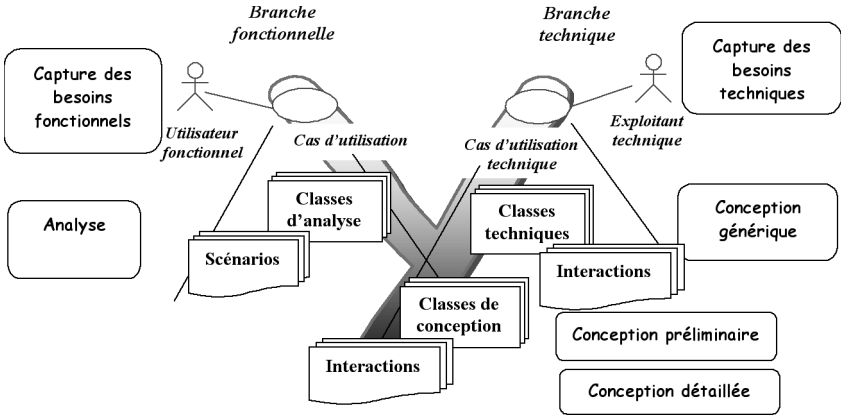


Figure 2-4. : Influence des cas d'utilisation dans le processus en Y

Au travers des cas d'utilisation, le point de vue utilisateur a donc bien le rôle d'initiation souhaité pour une conception. Vous noterez par ailleurs que l'orientation métier imposée par les cas d'utilisation de la branche gauche renforce la plus-value apportée par le système et introduit une dimension d'analyse de la valeur.

Le pilotage par les cas d'utilisation consiste justement à ordonner les cas d'utilisation par priorité, de manière à organiser les itérations par valeur ajoutée. En phase de construction notamment, c'est une bonne pratique d'inclure les cas d'utilisation les plus prioritaires en réalisation des premières itérations, puis de continuer par ordre de priorité. En apportant au plus tôt le maximum de valeur ajoutée au système, on rentabilise plus rapidement le développement, ce qui est encore une manière de réduire les risques.

Un processus de modélisation avec UML

Il nous paraît difficile d'envisager le processus 2TUP sans recourir à UML comme support. Certes, les concepts présentés jusqu'à présent ne sont pas spécifiquement liés à une notation particulière. Nous avons cependant omis de préciser le rôle central et fondamental de la modélisation objet tout au long du développement d'une solution logicielle.

Le recours à la modélisation est depuis longtemps une pratique indispensable au développement, car un modèle est prévu pour anticiper les résultats du développement. Un modèle est en effet une abstraction du résultat, dont le but est de documenter, de prévoir, d'étudier, de collecter ou d'estimer les informations

d'un système. Associé au processus de développement, un modèle représente la vue sur une spécification ou sur une solution de système, pris à un niveau de détail pertinent pour exprimer ou concevoir la cible de l'étape en cours.

Le modèle sert donc des objectifs différents suivant l'étape de développement et sera construit avec des points de vue de plus en plus détaillés :

- dans les activités de capture des besoins, il convient premièrement de considérer le système comme une boîte noire à part entière afin d'étudier sa place dans le système métier plus global qu'est l'entreprise. On développe pour cela un modèle de niveau contexte, afin de tracer précisément les frontières fonctionnelles du système ;
- dans les activités d'analyse, le modèle représente le système vu de l'intérieur. Il se compose d'objets représentant une abstraction des concepts manipulés par les utilisateurs. Le modèle comprend par ailleurs deux points de vue, la structure statique et le comportement dynamique. Il s'agit de deux perspectives différentes qui aident à compléter la compréhension du système à développer ;
- dans les activités de conception, le modèle correspond aux concepts qui sont utilisés par les outils, les langages ou les plates-formes de développement. Le modèle sert ici à étudier, documenter, communiquer et anticiper une solution. Il est en effet toujours plus rentable de découvrir une erreur de conception sur un modèle, que de le découvrir au bout de milliers de lignes codées sans grands horizons. Pour la conception du déploiement enfin, le modèle représente également les matériels et les logiciels à interconnecter.

Pour illustrer au mieux ce qu'est un modèle, Grady Booch [UML-UG 99] a établi un parallèle entre le développement logiciel et la construction BTP. Cette analogie est judicieuse, car les plans tracés pour construire un immeuble reflètent parfaitement bien l'idée d'anticipation, de conception et de documentation du modèle. Chaque plan développe par ailleurs un point de vue différent suivant les corps de métier. Par exemple, le plan des circuits d'eau et le plan des passages électriques concernent le même immeuble mais sont nécessairement séparés. Enfin, chaque plan se situe à un niveau d'abstraction et de détail distinct suivant l'usage que l'on désire en faire. Ainsi, le plan de masse permet d'anticiper les conséquences de l'implantation de l'immeuble sur son environnement, exactement comme le modèle de contexte. Viennent ensuite des plans de construction d'un étage, analogues aux modèles de conception.

Le modèle en tant qu'abstraction d'un système s'accorde parfaitement bien avec le concept orienté objet. L'objet représente en effet l'abstraction d'une entité utilisée dans le système en analyse, puis le modèle d'un composant de solution logicielle en conception. La correspondance est encore plus flagrante, et le modèle encore plus précis, lorsque les outils de développement sont eux-mêmes orientés objet. Aujourd'hui, le standard industriel de modélisation objet est UML.

Définition

« UNIFIED MODELING LANGUAGE »

UML se définit comme un langage de modélisation graphique et textuel destiné à comprendre et décrire des besoins, spécifier et documenter des systèmes, esquisser des architectures logicielles, concevoir des solutions et communiquer des points de vue.

UML unifie à la fois les notations et les concepts orientés objet. Il ne s'agit pas d'une simple notation, mais les concepts transmis par un diagramme ont une sémantique précise et sont porteurs de sens au même titre que les mots d'un langage. UML a une dimension symbolique et ouvre une nouvelle voie d'échange de visions systémiques précises. Ce langage est certes issu du développement logiciel mais pourrait être appliqué à toute science fondée sur la description d'un système. Dans l'immédiat, UML intéresse fortement les spécialistes de l'ingénierie système.

UML unifie également les notations nécessaires aux différentes activités d'un processus de développement et offre, par ce biais, le moyen d'établir le suivi des décisions prises, depuis la spécification jusqu'au codage. Dans ce cadre, un concept appartenant aux besoins des utilisateurs projette sa réalité dans le modèle de conception et dans le codage. Le fil tendu entre les différentes étapes de construction permet alors de remonter du code aux besoins et d'en comprendre les tenants et les aboutissants. En d'autres termes, on peut retrouver la nécessité d'un bloc de codes en se référant à son origine dans le modèle des besoins.

Les diagrammes d'UML

UML s'articule autour de 9 diagrammes différents, chacun d'eux étant dédié à la représentation des concepts particuliers d'un système logiciel. Par ailleurs, UML modélise le système suivant deux modes de représentation : l'un concerne la structure du système pris « au repos », l'autre concerne sa dynamique de fonctionnement. Les deux représentations sont nécessaires et complémentaires pour schématiser la façon dont est composé le système et comment ses composantes fonctionnent entre elles.

Le mode de représentation statique ou structurel s'appuie sur les 5 diagrammes ci-après.

- Le diagramme de cas d'utilisation représente la structure des fonctionnalités nécessaires aux utilisateurs du système. Il est utilisé dans les deux éta-

pes de capture des besoins fonctionnels et techniques. Vous en retrouverez une description détaillée de son usage aux chapitres 4 et 5 de cet ouvrage.

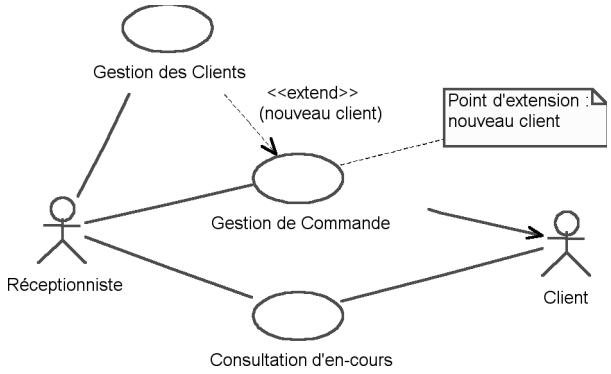


Figure 2-5. : Diagramme de cas d'utilisation

- Le diagramme de classes est généralement considéré comme le plus important dans un développement orienté objet. Sur la branche fonctionnelle, ce diagramme est prévu pour développer la structure des entités utilisées par les utilisateurs. Vous retrouverez les explications relatives à cette utilisation aux chapitres 4, 6 et 7. En conception, le diagramme de classes représente la structure d'un code orienté objet, ou au mieux les modules du langage de développement. Vous retrouverez l'utilisation du diagramme de classes en conception aux chapitres 9, 10 et 11.

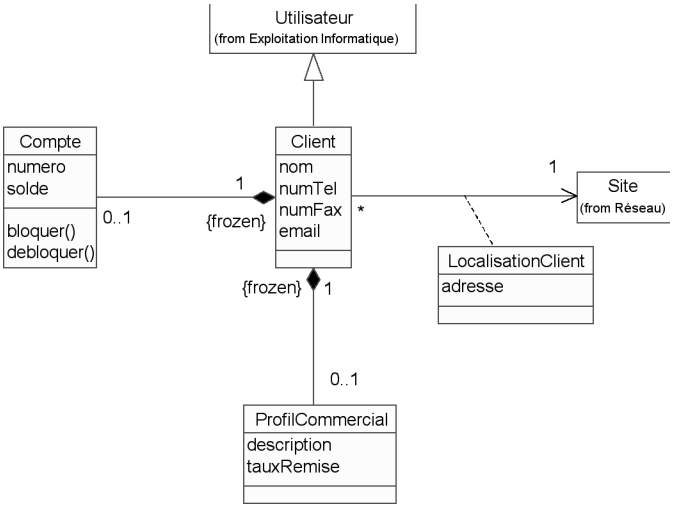


Figure 2-6. : Diagramme de classes

- Le diagramme d'objets sert à illustrer des structures de classes compliquées. Ce diagramme est utilisé en analyse pour vérifier l'adéquation d'un diagramme de classes à différents cas possibles. Vous retrouverez l'utilisation d'un tel diagramme en analyse au chapitre 7.

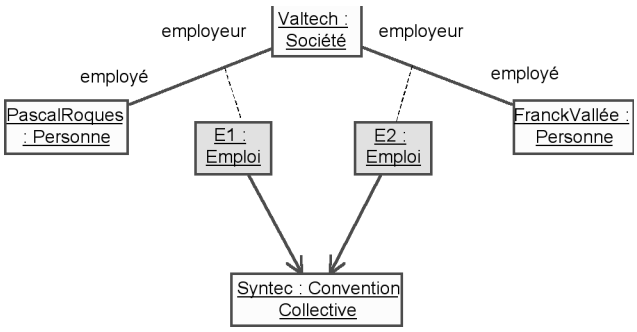


Figure 2-7. : Diagramme d'objets

- Le diagramme de composants représente en premier lieu les concepts connus de l'exploitant pour installer et dépanner le système. Il s'agit dans ce cas de déterminer la structure des composants d'exploitation que sont les librairies dynamiques, les instances de bases de données, les applications, les progiciels, les objets distribués, les exécutables, etc. L'utilisation du diagramme de composants pour l'exploitation est illustré au chapitre 10. Le diagramme de composants représente en second lieu les concepts de configuration logicielle, pour fabriquer une version de composant d'exploitation ou tout autre produit intermédiaire tel qu'une librairie ou un fichier JAR (Java). Il s'agit de montrer comment s'agencent des composants comme les fichiers source, les packages de code ou les librairies. Vous trouverez aux chapitres 9 et 11 l'utilisation d'un diagramme de composants pour la configuration logicielle.

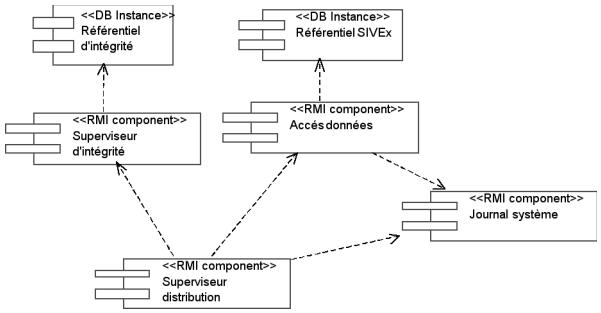


Figure 2-8. : Diagramme de composants

- Le diagramme de déploiement correspond à la fois à la structure du réseau informatique qui prend en charge le système logiciel, et la façon dont les composants d'exploitation y sont installés. Vous trouverez aux chapitres 5 et 10 l'utilisation d'un tel diagramme.

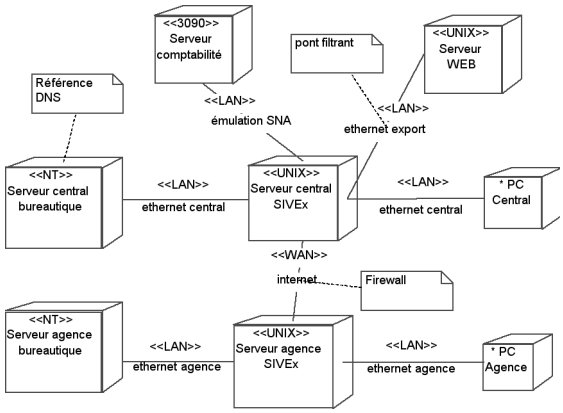


Figure 2-9 : Diagramme de déploiement

Le mode de représentation dynamique ou comportemental s'appuie sur les 4 diagrammes ci-après.

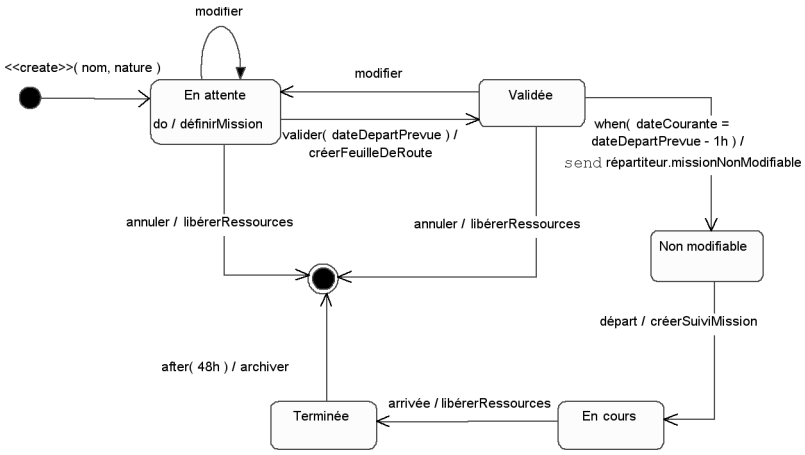


Figure 2-10. : Diagramme d'états

- Le diagramme d'états représente le cycle de vie commun aux objets d'une même classe. Ce diagramme complète la connaissance des classes en analyse et en conception. Le chapitre 8 vous indiquera comment utiliser ce diagramme à des fins d'analyse et le chapitre 11 à des fins de conception.

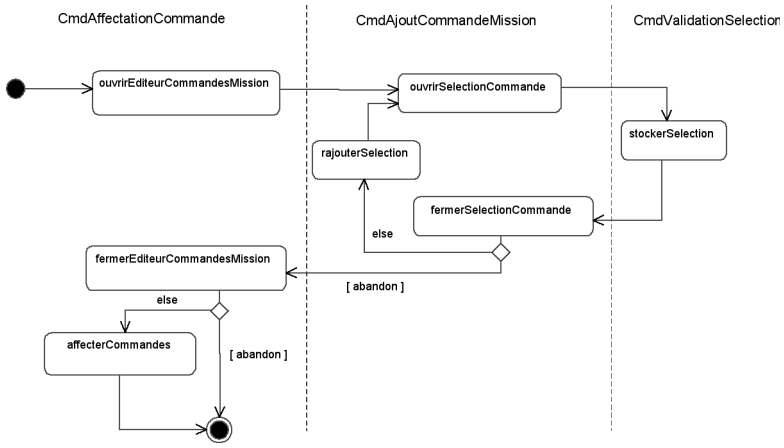


Figure 2-11. : Diagramme d'activités

- Le diagramme d'activités représente les règles d'enchaînement des activités dans le système. Il permet d'une part de consolider la spécification d'un cas d'utilisation comme illustré au chapitre 4, d'autre part de concevoir une méthode comme le montre le chapitre 11.

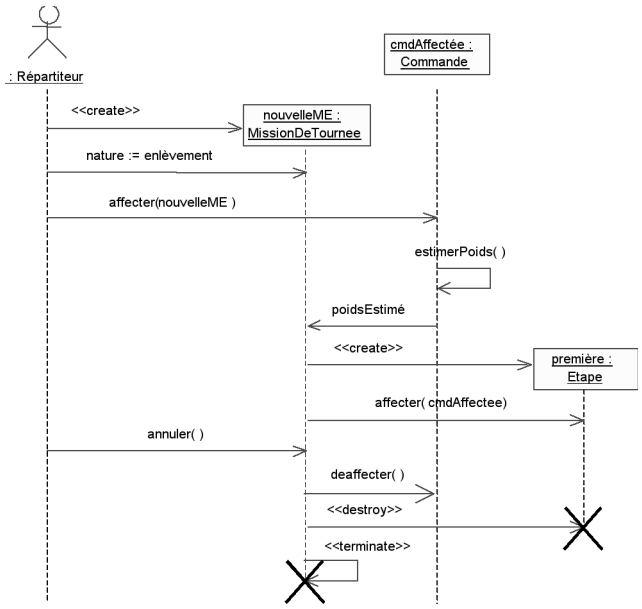


Figure 2-12. : Diagramme de séquence

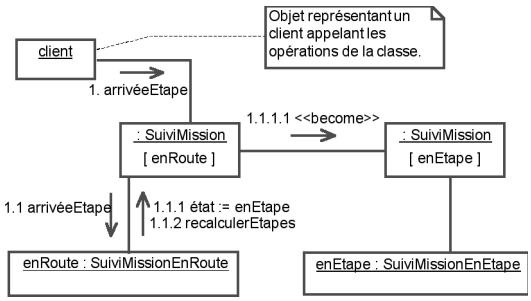


Figure 2-13. : Diagramme de collaboration

- Les diagrammes de collaboration et les diagrammes de séquence sont tous deux des diagrammes d'interactions UML. Ils représentent les échanges de messages entre objets, dans le cadre d'un fonctionnement particulier du système. Le diagramme de collaboration est utilisé pour modéliser le contexte du système, tel qu'illustré au chapitre 3. Les diagrammes de séquence servent ensuite à développer en analyse les scénarios d'utilisation du système. Vous en trouverez des exemples au chapitre 8. Enfin, les diagrammes de collaboration permettent de concevoir les méthodes comme indiqué au chapitre 11.

Un processus par niveaux d'abstraction

La modélisation se construit forcément par étapes successives de plus en plus détaillées. Il est en effet impossible de produire un modèle représentant quelques milliers de lignes de code sans passer par les étapes d'avancement qui permettent d'organiser judicieusement le volume d'informations collectées.

Tout processus construit sur un modèle doit donc se doter d'une tactique d'avancement par consolidation de chaque étape de construction du modèle. Un tel avancement est itératif, car il définit des objectifs à atteindre suivant un découpage en niveaux de détail allant croissant par rapport au modèle de développement. Ces niveaux correspondent à une vision de moins en moins abstraite du développement, c'est pourquoi nous les qualifions aussi de niveaux d'abstraction.

Pour illustrer l'évolution du modèle par niveaux d'abstraction, nous avons recours à un cône dont la surface symbolise à un instant donné le volume d'informations rassemblées par le modèle. Le processus en Y implique une dichotomie initiale du modèle suivant les deux branches fonctionnelle et technique, comme l'illustre la figure 2-14.

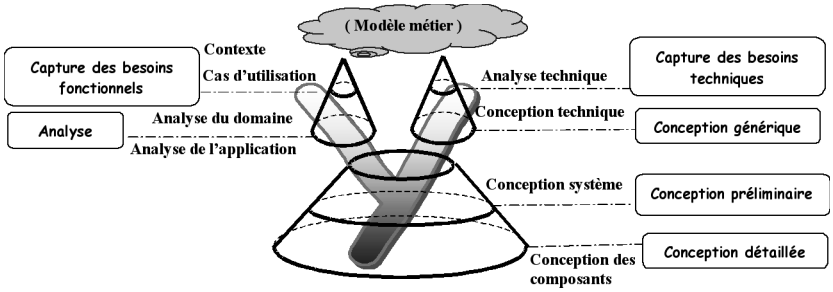


Figure 2-14. : Les niveaux d'abstraction du processus en Y

Chaque niveau d'abstraction du modèle s'inscrit dans une étape du processus et fixe un objectif de description du modèle.

Pour la capture des besoins fonctionnels :

- le niveau du contexte a pour objet de définir la frontière fonctionnelle entre le système considéré comme une boîte noire et son environnement ;
- le niveau des cas d'utilisation définit ensuite les activités attendues des différents utilisateurs par rapport au système toujours envisagé comme une boîte noire. Ce modèle permet de contrôler la bonne adéquation des besoins avec les utilisateurs.

Pour l'analyse :

- on ouvre le système pour établir la structure des objets utilisés. Le modèle d'analyse du domaine définit la structure et le comportement des objets connus dans le métier des utilisateurs du système ;
- le modèle d'analyse de l'application y rajoute, suivant le même processus, les objets qui sont connus des utilisateurs, dans le cadre de la mise en application de leurs besoins.

Pour la capture des besoins techniques :

- le modèle d'analyse technique établit des couches logicielles et y spécifie les activités techniques attendues.

Pour la conception générique :

- le modèle de conception technique définit les composants qui, délivrant des services techniques, assurent la réponse aux exigences opérationnelles du système.

Pour la conception préliminaire :

- le modèle de conception système organise le système en composants, délivrant les services techniques et fonctionnels. Ce modèle regroupe les informations des branches fonctionnelle et technique. Il peut être consi-

déré comme la transformation du modèle d'analyse par projection des classes d'analyse sur les couches logicielles.

Pour la conception des classes :

- le modèle de conception des composants fournit l'image « prêt à fabriquer » du système complet.

Notons que, préalablement au développement, un modèle métier peut établir le contexte organisationnel dans lequel vient s'insérer le système informatique. Ce modèle éventuellement objet constitue un point d'entrée possible au processus en Y. Mais nous n'aborderons pas la problématique de modélisation métier dans cet ouvrage.

Une fois les objectifs établis, l'avancement sur un modèle se fait par itération des mêmes tâches de construction jusqu'à obtention d'un modèle satisfaisant. Nous vous présenterons dans cet ouvrage le workflow de construction associé à chaque niveau d'abstraction. L'animation d'un processus itératif consiste à planifier les tâches nécessaires à l'aboutissement du modèle, à les réaliser, puis à valider le modèle avec les acteurs concernés.

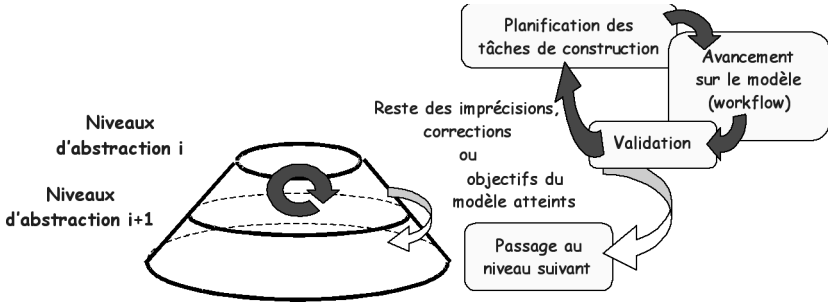


Figure 2-15. : Animation du processus itératif

Les points de vue de modélisation

Le processus en Y itère donc sur la construction d'un modèle. À cet effet, nous avons abordé les niveaux qui servent à fixer des jalons dans l'avancement du développement. Il nous reste maintenant à considérer les points de vue qu'un modèle doit honorer ainsi que les techniques qu'il doit développer pour structurer le volume d'informations croissant qu'il contient.

En tant que support d'étude, d'anticipation, de conception et de documentation du système, le modèle doit représenter les points de vue nécessaires aux différents protagonistes du développement. Ces points de vue représentent autant de vitrines d'observation du même problème en mettant en valeur

certains contenus et en en masquant d'autres. Par analogie au BTP, pensez aux plans tracés pour les différents corps de métier, électricité, plomberie, maçonnerie, etc. Les points de vue du modèle d'un système logiciel sont répertoriés ci-après.

- Le point de vue de spécification fonctionnelle concerne l'organisation du modèle des besoins fonctionnels exprimés par les utilisateurs et étudiés par les analystes. Les éléments de construction correspondent aux cas d'utilisation organisés en packages, aux acteurs, aux activités, aux interactions entre objets et aux contraintes dynamiques.
- Le point de vue structurel a trait à l'organisation du modèle des besoins élaboré en classes par les analystes. Les éléments de construction y sont les catégories, les classes, les associations, les généralisations, les attributs et les contraintes structurelles.
- Le point de vue matériel développe la structure physique des machines et des réseaux sur lequel repose le système informatique. Il concerne les ingénieurs système et réseau. Comme éléments de construction, on compte les nœuds et les connexions qui permettent de prévoir le dimensionnement des processeurs et des bandes passantes.
- Le point de vue de déploiement représente la structure des postes de travail et localise les composants sur le réseau physique. Il concerne les ingénieurs d'exploitation chargés d'installer le logiciel et d'identifier les causes de pannes. Les éléments de construction y sont les postes de travail, les serveurs, les connexions et les composants qui permettent d'étudier les échanges internes et externes du système en développement. En client/serveur, on évoquera souvent les niveaux de répartition locale, départementale et centrale.
- Le point de vue d'exploitation correspond à l'organisation des composants et identifie les fonctions prises en charge par le logiciel installé. Il concerne à la fois les ingénieurs d'exploitation et les concepteurs, les uns pour trouver le composant incriminé par une panne, les autres pour cartographier les dépendances logicielles. Les composants, les interfaces et les dépendances entre composants constituent les éléments de construction. En client/serveur, il est d'usage de répartir les composants suivants des architectures 2-tiers, 3-tiers ou n-tiers.
- Le point de vue de spécification logicielle concerne les architectes qui décident de répartir par couches les exigences techniques, afin de les dissocier par nature de responsabilités. Les éléments de construction y sont les cas d'utilisation techniques organisés en couches, les exploitants, les activités, les interactions entre objets et les contraintes dynamiques. On aura recours aux couches réparties en responsabilités de présentation, de gestion des applications, de gestion du métier, d'accès aux données et de stockage des données.

- Le point de vue logique est relatif à l'organisation du modèle de solution élaboré en classes par les concepteurs. Les éléments de construction y sont les classes regroupées en catégories, les interfaces, les associations, les généralisations, les réalisations, les attributs, les états, les opérations et leurs méthodes. Ce point de vue est incontournable, car il fournit la vision « prêt à coder » de la solution et inversement, documente le code produit.
- Le point de vue de configuration logicielle retrace enfin la manière dont sont construits les composants qui servent à l'élaboration d'un sous-système. Il permet de mesurer l'impact d'un changement sur la construction du logiciel et d'établir les configurations et les compatibilités entre plusieurs éléments de versions différentes. Les éléments de construction y sont les sous-systèmes, les composants de construction et leurs dépendances de fabrication. Pour fixer les idées, sachez que ces informations peuvent servir à construire les fichiers « makefile » d'un projet.

Vous remarquez que tous les points de vue n'interviennent ni au même moment, ni au même niveau d'abstraction dans le processus de développement. Il existe par ailleurs des dépendances entre points de vue, dont il est nécessaire d'avoir une cartographie pour maintenir la cohérence du modèle. La figure 2-16 présente les relations existantes entre les différents points de vue et les situe par rapport aux différents niveaux d'abstraction du modèle.

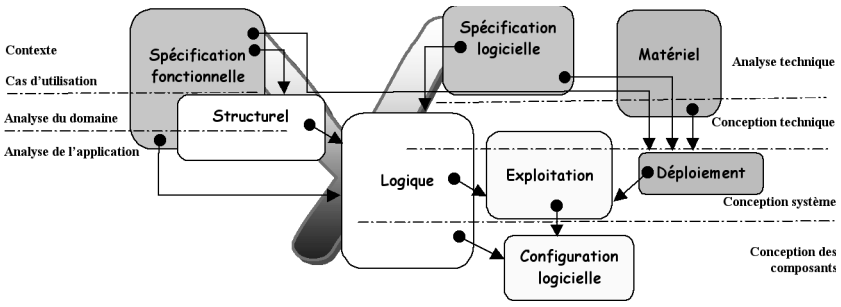


Figure 2-16. : Cartographie des points de vue de modélisation

La spécification fonctionnelle défend le point de vue des utilisateurs, elle pilote le contenu du développement et conditionne les points de vue structurel et logique mais également le déploiement de la conception système de la façon suivante :

- les cas d'utilisation permettent de trouver les classes de la vue structurelle du modèle d'analyse ;
- les scénarios élaborés par cas d'utilisation permettent de trouver les opérations des interfaces de la vue logique du modèle de conception système ;

- les cas d'utilisation identifient des fonctions qu'il faut répartir sur le déploiement du modèle de conception système.

La spécification logicielle se place du point de vue des exploitants et gère le modèle de la façon suivante :

- les cas d'utilisation techniques permettent de trouver les classes de la vue logique du modèle de conception technique. Leurs scénarios permettent de trouver les opérations de ces classes regroupées par couches ;
- les cas d'utilisation techniques identifient les fonctions d'exploitation qu'il faut répartir sur le déploiement du modèle de conception système.

La vue du matériel présente le support du déploiement.

La vue structurelle projette ses classes dans la vue logique au niveau de la conception système.

La vue d'exploitation définit ses composants à partir des interfaces de la vue logique du modèle de conception système et des composants de déploiement.

La vue de configuration logicielle dépend des classes du modèle logique qui réalisent les différents composants d'exploitation.

Le tableau suivant établit la correspondance entre les diagrammes proposés par UML 1.4 et les points de vue de modélisation. Le symbole ✓ signifie que l'utilisation du diagramme est incontournable dans la formalisation du point de vue, tandis que ✓ symbolise une utilisation optionnelle.

Point de vue / Diagramme UML	Spéc. Fonc.	Struct.	Spéc. Log.	Mat.	Log.	Expl.	Dépl.	Conf. Log.
Classes	✓	✓	✓		✓			
Objets		✓			✓			
Cas d'utilisation	✓		✓					
Séquence	✓		✓		✓			
Collaboration	✓		✓		✓			
États		✓			✓			
Activités	✓		✓		✓			
Composants						✓	✓	✓
Déploiement				✓			✓	

Tableau 2-2 : Utilisation des diagrammes UML en fonction des points de vue du modèle

Un processus centré sur l'architecture

Le terme architecture est un mot actuellement en vogue mais utilisé de manière abusive. L'architecture est souvent mal comprise parce qu'on la situe dans la structure résultante d'un modèle. L'architecture n'est pas la conséquence du modèle mais préside au contraire à son organisation. Cette organisation fixe des directives générales au développement, et les structures qu'elle induit aident au maintien de l'intégrité du modèle.

Définition

ARCHITECTURE

L'architecture est l'ensemble des décisions d'organisation du système logiciel qui défend les intérêts de son propriétaire final. Les intérêts s'expriment en termes d'exigences fonctionnelles, techniques et économiques. L'architecture y répond par l'intégration de plusieurs styles de développement informatique qu'elle adapte aux éléments logiciels d'un contexte existant.

Le propriétaire du système logiciel, par définition le maître d'ouvrage, est au premier chef concerné par l'adéquation aux besoins des utilisateurs, la pertinence par rapport à l'organisation de l'entreprise, l'analyse de la valeur qui en résulte, et les qualités de maintenance et d'évolution du logiciel. C'est pourquoi l'architecture du logiciel décrit plusieurs axes de solution générique, définis comme ci-après.

- Les architectures client/serveur en tiers (2-tiers, 3-tiers ou n-tiers) concernent la capacité de montée en charge du système. Le style 2-tiers vise des applications départementales à nombre limité d'utilisateurs. Elles mettent généralement en jeu des clients et un serveur de base de données. Les styles 3-tiers ou n-tiers permettent l'évolution du nombre des utilisateurs par l'introduction d'un middleware, qui distribue les services entre les clients et les serveurs.
- Les architectures en couches visent la distribution des responsabilités techniques sur les parties développées pour le système logiciel. Nous avons déjà cité la répartition suivant les cinq couches : présentation, application, métier, accès aux données et stockages des données. Ces architectures améliorent les qualités d'évolution et de maintenance des aspects techniques du système.
- Les architectures en niveaux correspondent au déploiement des fonctions sur les postes de travail des utilisateurs. Les 3 niveaux considérés pour l'entreprise sont le niveau central, le niveau départemental et le niveau local. Ces niveaux permettent de mieux contrôler l'imbrication des fonctions du système, ce qui améliore ses qualités d'évolution et de maintenance fonctionnelles.

- Les architectures à base de composants consistent à développer les opportunités de réutilisation au sein du système informatique. Les composants sont à la fois spécifiquement développés ou achetés sur étagère. Une telle architecture impose dans tous les cas une définition stricte de la décomposition modulaire. Les composants améliorent non seulement part les qualités d'évolution et de maintenance mais également les coûts de développement du système.

L'architecture implique des décisions d'organisation qui se répercutent sur la structure du modèle lui-même. Les différents points de vue de modélisation cités précédemment deviennent les outils de contrôle de l'architecte logiciel qui permettent de superviser la conformité du développement aux intérêts du maître d'ouvrage. Le tableau suivant illustre l'influence des styles d'architecture sur les différents points de vue du modèle.

Architecture / Points de vue	Multiniveaux	Par composant	En couches	Multitiers
Spécification fonctionnelle	Positionnement des acteurs aux différents niveaux de l'entreprise.	Identification de cas d'utilisation gérés par des composants existants.		
Spécification logicielle		Identification de cas d'utilisation techniques, gérés par des composants existants.	Positionnement des cas d'utilisation techniques sur les différentes couches.	
Structurel et organisationnel	Regroupement des classes par niveau.	Regroupement des classes provenant de composants existants.		
Matériel	Définition des matériels installés par niveaux.			Définition d'un middleware.
Déploiement	Définition des postes de travail par niveau.	Identification et positionnement des composants d'exploitation sur le réseau.		Regroupement des composants par services distribués.
Exploitation	Regroupement des composants par poste de travail.	Identification des interfaces et des composants à exploiter.	Répartition des composants par couche.	Répartition des composants entre tiers.

Architecture / Points de vue	Multiniveaux	Par composant	En couches	Multitiers
Logique		Regroupement des classes communes au même composant.	Identification des classes se situant dans la même couche de services techniques.	Identification des interfaces et des classes distribuées.
Configuration logicielle		Fabrication de chaque composant.	Fabrication des composants spécifiques à une couche.	Fabrication des composants distribués.

Tableau 2-3 : Influence des styles d'architecture sur les vues du modèle

Un processus centré sur l'architecture impose le respect des décisions d'architecture à chaque étape de construction du modèle. L'architecture est donc la condition qui préside à l'intégrité d'un projet complexe, car elle permet la structure et la cohérence des points de vue.

La somme des parties d'un système complexe a en effet pour caractéristique de dépasser la complexité de chacune des parties. L'organisation d'un modèle cohérent permet d'établir des règles précises de croissance, sans dépasser les capacités de compréhension humaine. L'architecte utilise la structure du modèle pour communiquer et contrôler les liens complexes entre ces parties. Le processus centré sur l'architecture permet donc de dégager des opportunités pour renforcer les intérêts économiques du maître d'œuvre :

- les modèles correctement organisés offrent des moyens de réutilisation du logiciel à large échelle ;
- leur découpage facilite l'élaboration de métriques, l'estimation et la répartition du travail entre équipes ;
- les points de vue cohérents facilitent l'étude d'impact suivant les différents axes de changement que sont l'évolution des fonctions, de la structure des entités, des techniques, des configurations d'exploitation, et des versions logicielles ;
- la documentation apportée par les modèles facilite les tests, l'intégration, et aide à identifier la source des erreurs.

Un processus orienté vers les composants

Le respect des règles d'architecture et la structuration du modèle à toutes les étapes du processus tend naturellement à regrouper les concepts à forte cohérence et à identifier scrupuleusement tous les couplages entre parties. L'expression des couplages implique la spécification de règles d'interface et

permet d'évaluer l'opportunité de réutiliser les regroupements de concepts à d'autres contextes de développement.

Au niveau du système d'information d'entreprise, tel que celui présenté dans l'étude de cas, un progiciel doit être considéré comme un composant à part entière, ayant des caractéristiques fonctionnelles et logicielles propres. Cette orientation rejoint clairement les options du « best of breed » qui vise à positionner le meilleur outil à la meilleure place pour répondre aux fonctions du système d'information d'entreprise ; elle se distingue d'une philosophie « ERP centric » qui tend au contraire à ramener toutes les fonctions de l'entreprise dans le même progiciel. Le mariage de plus en plus fréquent de progiciels du marché, de développements spécifiques et de technologies d'intégration (EAI ou WebServices) va introduire des contraintes non négligeables dans l'organisation d'une modélisation UML tant pour les activités d'analyse que pour celles de conception.

Les regroupements de concepts définissent des packages et des composants dans le modèle, leur réutilisation peut se situer à tous les niveaux d'abstraction. De plus, la cohérence du modèle impose d'établir et de suivre les liens entres les structures d'un point de vue à l'autre et d'une étape à l'autre.

- Lors de la capture des besoins fonctionnels, les cas d'utilisation sont regroupés en packages pour organiser le modèle de spécification fonctionnel. Ces packages représentent les besoins d'un métier d'entreprise vis-à-vis d'un système informatique, et peuvent constituer des modèles de spécification à réutiliser par des progiciels métier. Les packages de cas d'utilisation structurent la répartition en applications du système. Ces applications déployées sur les postes de travail constituent une partie des composants du modèle d'exploitation.
- Lors de l'analyse, les classes.sont regroupées en catégories pour organiser successivement le modèle d'analyse métier et le modèle d'analyse de l'application. Les catégories métier représentent la description détaillée des concepts de l'entreprise et peuvent constituer des références, réutilisables par différentes applications. Les catégories d'analyse structurent les catégories de conception ainsi que la répartition en composants métier. Ces derniers constituent éventuellement des composants du modèle d'exploitation.
- Pour la capture des besoins techniques, les cas d'utilisation sont regroupés en couches logicielles en vue d'organiser le modèle de spécification technique. Ces packages représentent les besoins techniques vis-à-vis d'une technologie, et peuvent constituer des modèles de spécification à réutiliser par différentes applications de l'entreprise. Les couches logicielles structurent la création de frameworks techniques qui constituent des mécanismes de conception générique pour la technologie concernée.

- Lors de la conception préliminaire, les classes sont regroupées en frameworks pour remplir des fonctions techniques spécifiques. Les frameworks constituent éventuellement des composants du modèle d'exploitation et participent au modèle de configuration logicielle.
- Pour la conception détaillée, les classes sont organisées en catégories et documentent généralement une librairie de classes réutilisables. Les catégories de conception constituent éventuellement des composants du modèle d'exploitation et structurent les sous-systèmes de configuration logicielle.
- Les composants d'exploitation sont les éléments que l'on déploie pour installer le système complet. Les différentes technologies correspondant à cette notion sont les instances de bases de données, les applications à disposition des utilisateurs, les librairies dynamiques, les objets distribués, les Java-Beans, les ActiveX, etc. Les composants d'exploitation sont des cibles de fabrication du logiciel, de sorte que chaque composant d'exploitation correspond à un sous-système de configuration logicielle.

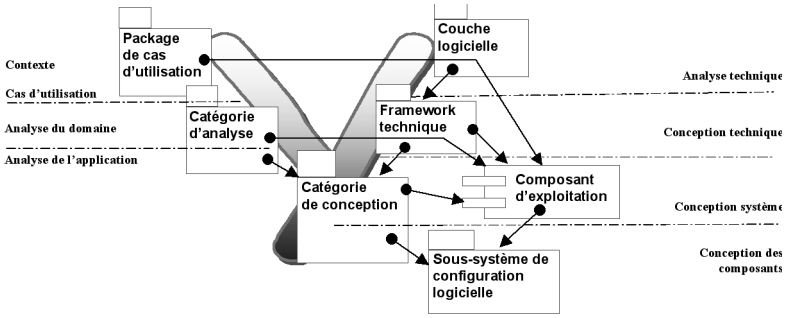


Figure 2-17. : Positionnement et influence des structures réutilisables sur le processus 2TUP

Aujourd'hui, il existe des technologies orientées composant, à savoir CORBA, DCOM, ActiveX, Java RMI, Java Beans et plus récemment les infrastructures EAI et Web Services. De fait, le développement et l'intégration de composants vont de pair et le travail du développeur nécessite un effort encore plus poussé vers le respect du style d'architecture orienté composant. Les types de composants que l'on peut réutiliser au sein de l'entreprise ou du commerce s'intègrent à différents niveaux suivant leur nature.

- Les composants transverses correspondent aux outils offrant des fonctionnalités purement techniques. De tels outils s'intègrent dès la conception technique s'ils se présentent sous la forme d'une bibliothèque de code. Dans tous les cas, ils doivent être considérés suivant les points de vue du déploiement, de leur exploitation et de leur configuration logiciel.

- Les composants verticaux, ou *compogiciels*, apportent à la fois des fonctions métier et une architecture technique. Ils doivent d'une part se représenter sous la forme de classes et de services pour être intégrés dans les points de vue dynamique et statique du modèle d'analyse. Si leurs mécanismes d'interface ne sont pas standard, ils font d'autre part l'objet d'une couche logicielle à intégrer dans la conception technique. Les *compogiciels* doivent ensuite être considérés suivant les points de vue du déploiement et de leur exploitation.
- Les progiciels apportent des fonctions métier, une architecture technique et des interfaces utilisateur. S'ils sont prévus pour fonctionner en complète autonomie, leur intégration dans un système d'information existant requiert une étude fonctionnelle et technique analogue à l'intégration d'un compogiciel. Les progiciels ont par ailleurs leur propre déploiement et exploitation.
- Les bus d'intégration EAI et B2B apportent un framework d'échanges qui a des implications non négligeables sur les architectures technique et fonctionnelle. Ce domaine a fait l'objet d'un « Profile » particulier édité en février 2002 par l'OMG.

Résumé du chapitre

La famille des « Unified Process » constitue une trame commune pour intégrer les meilleures pratiques de développement. Un processus UP est itératif et incrémental, centré sur l'architecture, conduit par les exigences des utilisateurs, piloté par les risques et orienté composants. Le processus 2TUP se situe dans cette lignée, en insistant sur la non-corrélation initiale des aspects fonctionnel et technique. Les deux branches d'étude fusionnent ensuite pour la conception du système, ce qui donne la forme d'un processus de développement en Y. La dichotomie initiale permet à la fois de capitaliser la connaissance métier sur la branche gauche et de réutiliser un savoir-faire technique sur la branche droite.

UML est le langage de modélisation objet standard du 2TUP. Chacun des 9 diagrammes de UML 1.4 est en effet pertinent pour représenter les étapes de développement et les points de vue de modélisation préconisés. 2TUP est construit autour de la construction et du maintien d'un modèle qui permet de contrôler l'adéquation du développement aux règles d'architecture et qui favorise la conception d'un système orienté composants.

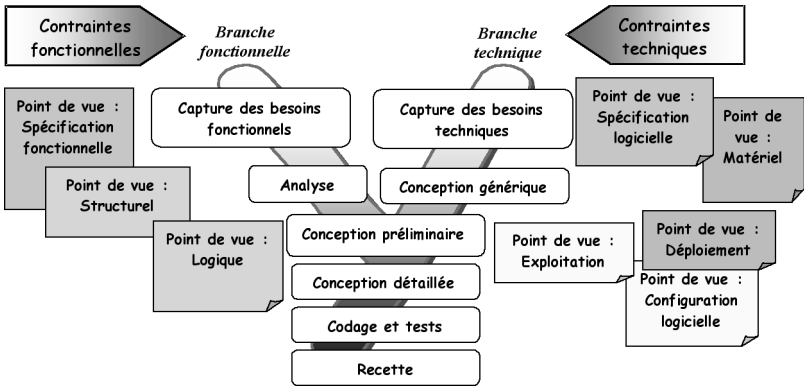


Figure 2-18. : Rappel des étapes et des points de vue du 2TUP