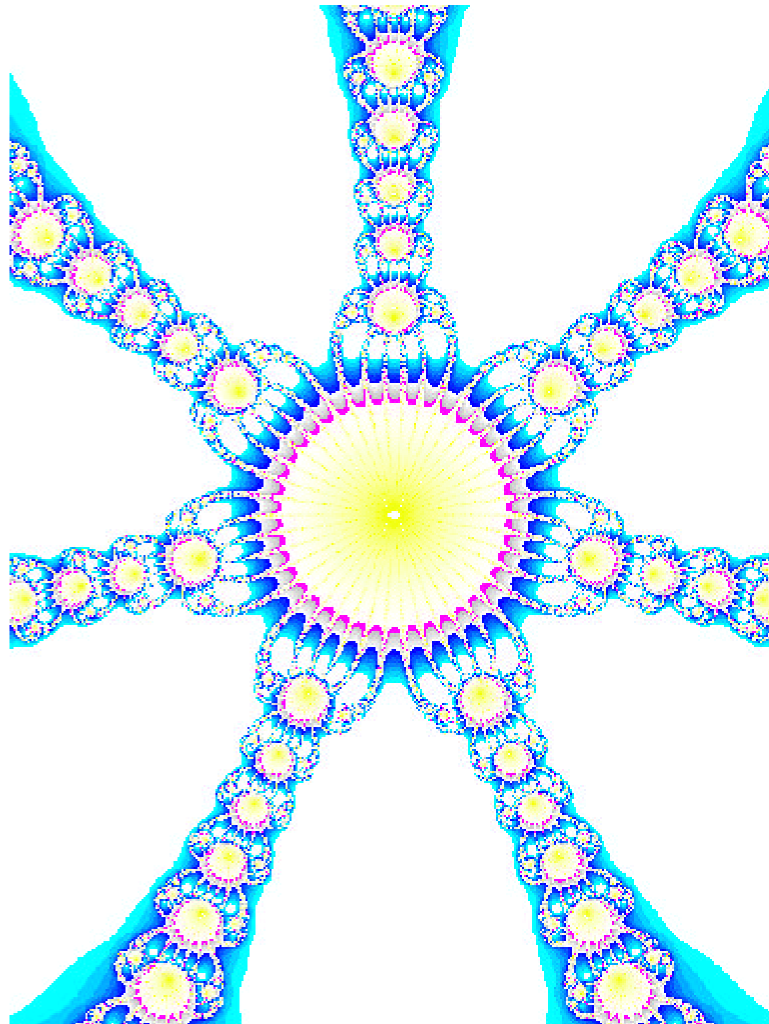
CHAPITRE 7

Introduction à UML

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com



7.1 *Introduction*

UML est un langage qui permet de représenter des modèles, mais il ne définit pas le processus d'élaboration des modèles. Cependant, dans le cadre de la modélisation d'une application informatique, les auteurs d'UML préconisent d'utiliser une démarche :

- itérative et incrémentale,
- guidée par les besoins des utilisateurs du système,
- centrée sur l'architecture logicielle.

D'après les auteurs d'UML, un processus de développement qui possède ces qualités devrait favoriser la réussite d'un projet. Mais que recouvre la notion de projet ?

7.1.1 *La notion de projet*

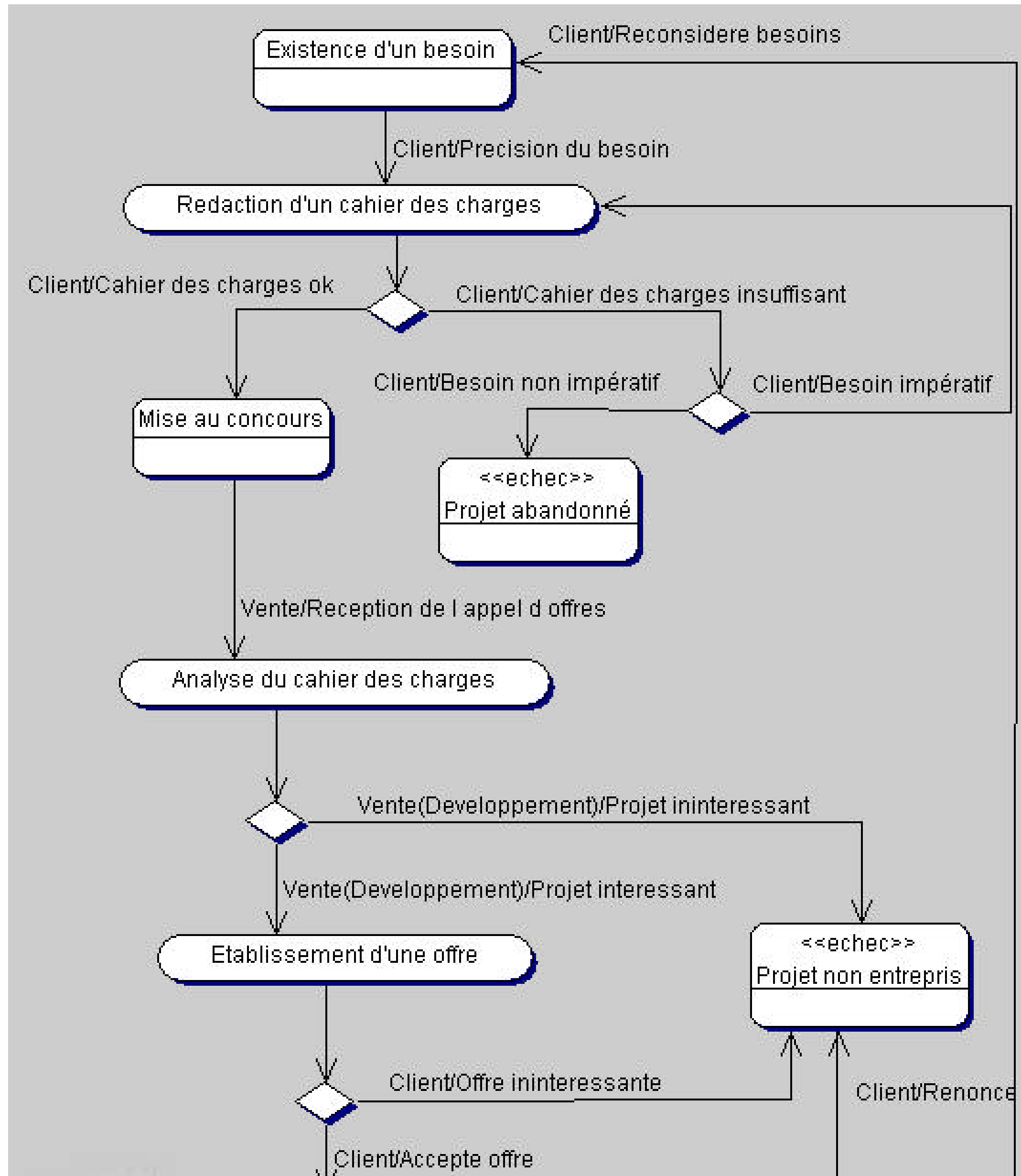
La notion de projet est souvent considérée différemment selon le secteur d'activité dans lequel on se trouve. Ainsi, un "projet" dans le domaine commercial sera-t-il abordé différemment d'un projet dans un développement de microélectronique, et encore différemment dans un projet informatique. Or, il s'agit fondamentalement d'une seule et même chose! Dans ce cas, on devrait pouvoir utiliser des méthodes identiques pour ces diverses modélisations.

C'est effectivement le cas pour les modélisations orientées objets, mais pas pour les modélisations basées sur la notion de procédure : la procédure est une notion trop abstraite pour être aisément traduite en termes matériels. Un PLL (Phase Locked Loop, boucle asservie en phase) est un objet faisant partie, avec d'autres, d'un objet plus complexe qui est un démodulateur FM. En revanche, comment mettre en relation le circuit intégré implémentant la fonction de démodulation avec les fonctions de Bessel décrivant le spectre de la modulation, de manière à ce que cette relation soit utilisable tout au long de la chaîne de développement ? Clairement, il apparaît que les mathématiques (donc, les algorithmes) ne sont nécessaires qu'à un niveau atomique du fonctionnement de notre système, alors que le fonctionnement global doit être décrit d'une autre manière.

En résumé, il faut idéalement pouvoir décrire un projet sans tenir aucunement compte des moyens utilisés pour venir à bout de ce projet; c'est à cette condition que l'on peut valablement conduire un projet intégrant aussi bien du logiciel que du matériel. La syntaxe utilisée pour la modélisation doit tenir compte de cette condition.

Un projet est constitué de diverses phases, dont toutes sont essentielles, mais dont certaines prennent plus d'importance que d'autres selon l'optique dans laquelle on se place; les figures suivantes montrent un tel cycle vu sous l'angle de la documentation de projet. La division en plusieurs figures est uniquement dûe à des contraintes de mise en page. La description du cycle de vie du projet est faite à l'aide de UML (Unified Modeling Language) dont nous reparlerons plus loin.

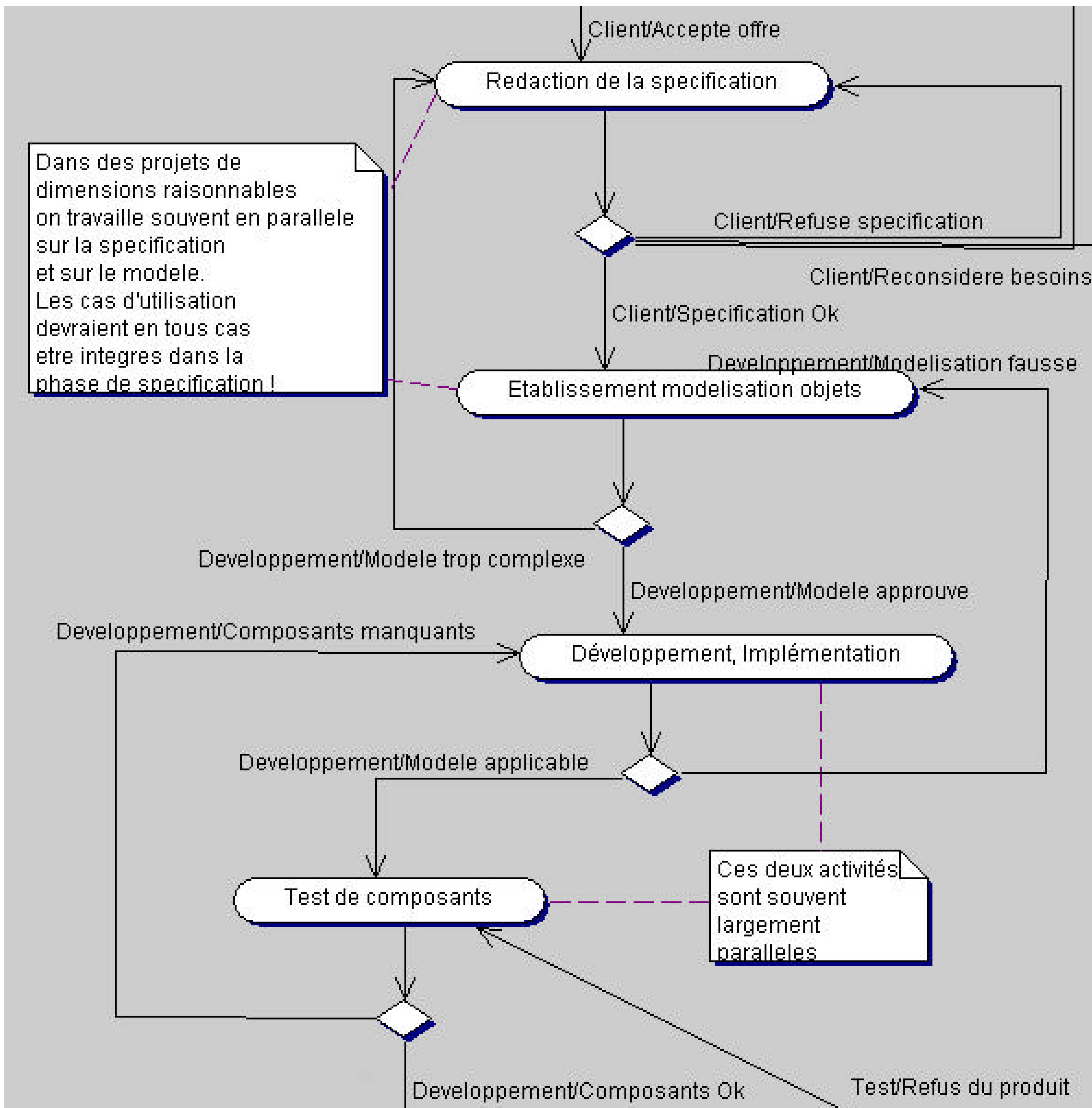
FIGURE 7.1 La première phase du projet



Dans la première phase, seul le client et le vendeur sont concernés, le développement n'intervient qu'en guise de conseil au vendeur, et n'a aucun pouvoir exécutif. Les arguments sont économiques, et le développement doit estimer aussi précisèment que possible l'investissement qu'il devra consentir pour réaliser le projet, de manière à ce que le vendeur puisse rédiger une offre aussi réaliste que possible. Cette phase est extrêmement délicate : l'expérience joue un rôle prépondérant. Le vendeur a toute latitude pour moduler les prix selon l'importance du projet pour la société : il peut donc sous-évaluer le projet alors même que le développeur avait donné un préavis nettement moins optimiste; mais le développeur n'a pas non plus intérêt à surévaluer le travail, puisqu'il risque de faire échouer le projet !

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

FIGURE 7.2 **L'étape intermédiaire du développement**

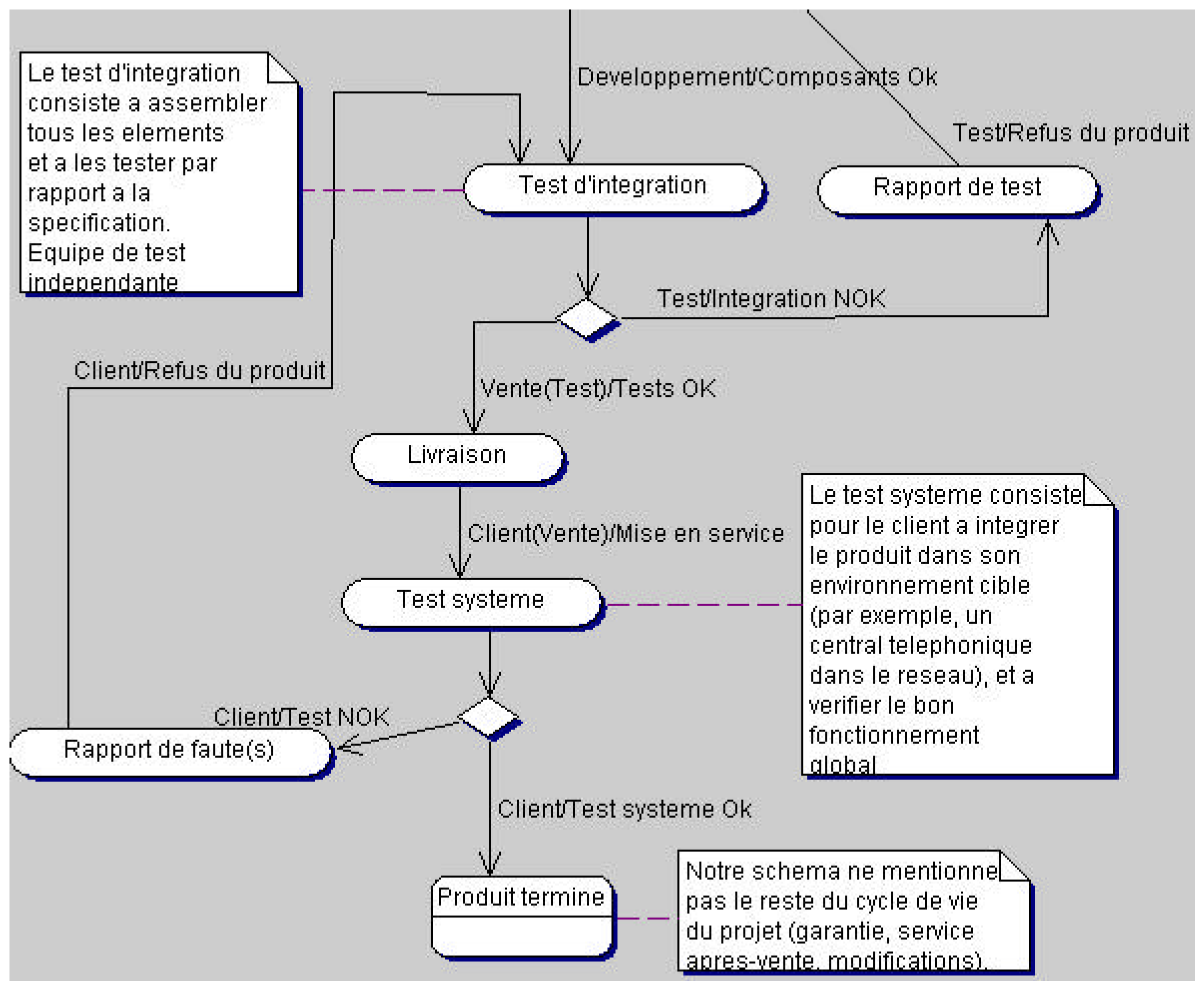


Dans la deuxième phase, le développeur est l'acteur principal. La première tâche est la rédaction de la spécification du produit à réaliser, tâche des plus délicates, puisque cette spécification est juridiquement contraignante : si le produit fini ne correspond pas à la spécification, fût-ce sur l'un ou l'autre point de détail, le client peut refuser le paiement du travail. Bien qu'il soit rare que l'on en arrive là, cette procédure est possible, surtout entre partenaires commerciaux ne se connaissant pas très bien.

Ensuite, on débute le développement proprement dit. La phase de modélisation est celle qui nous préoccupe principalement ici. Avec la spécification, il s'agit de la phase la plus importante du processus de développement; bien que dans le processus de fabrication du produit, les deux phases sont représentées de manière consécutive, il existe un certain paral-

lélisme entre ces deux phases. Il est en effet difficile de réaliser une spécification digne de ce nom sans avoir au moins une légère idée de l'architecture de la solution[1]. En particulier, les cas d'utilisation (Use Case, Voir *diagrammes de cas d'utilisation*, page82.) devraient constituer l'une des bases les plus importantes pour l'établissement de toute spécification. La modélisation permet de décomposer le problème en éléments plus petits, donc plus facilement maîtrisables. Le fin du fin consiste à effectuer une décomposition de telle manière que des éléments déjà développés auparavant puissent être identifiés et intégrés dans le modèle. L'implémentation et le test de composants sont ensuite menées souvent de front par des développeurs individuels.

FIGURE 7.3 Fin du cycle de développement



Lorsque les divers développeurs ont terminé l'implémentation des éléments du modèle, il faut encore les assembler. Souvent, des assemblages ont déjà été faits au cours de l'implé-

1. Certains auteurs pensent qu'il est au contraire indispensable de bien séparer ces deux phases; d'ailleurs, certaines grandes firmes utilisent des équipes différentes pour la spécification et la réalisation, et ne favorisent pas les contacts entre les deux équipes. L'avantage réside en une plus grande neutralité de la spécification; le risque est de spécifier des caractéristiques difficiles (donc coûteuses) à implémenter.

mentation, mais la phase de test d'intégration permet de comparer le produit avec la spécification. C'est une équipe indépendente du développement, n'ayant pas participé à la réalisation jusque là, qui idéalement effectue ce test (test d'intégration), sur les seules bases de la spécification et du mode d'emploi.

Ensuite, le client va prendre livraison du produit, et mener à bien ses propres tests, dans l'environnement d'exploitation. Cette phase est appelée test système. Ce n'est que lorsque cette dernière phase a été passée avec succès que le produit est considéré comme terminé. En réalité, on devrait encore tenir compte du service après-vente, de la maintenance, du suivi de produit, des mises à jour, etc... Nous ne le ferons pas dans le cadre de ce chapitre, car cela dépasse le problème du développement d'un produit proprement dit.

7.1.2 *Comment minimiser les risques ?*

Il est évident, dans ce modèle, que plus la détection d'une erreur de conception se fait tardivement, plus le coût nécessaire à la correction sera élevé. Le scénario catastrophe pourrait être le suivant :

- Le test système, chez le client, fait apparaître une faute grossière relativement à la spécification, faute liée à l'environnement du client, ce qui explique que le test d'intégration n' aie pas su la déceler. La société est contrainte de reprendre le produit.
- Sur la base des indications fournies par le client (si tant est qu'indications il y a !), l'équipe de test parvient à reproduire et à documenter, dans un rapport de test, la faute à l'intention de l'équipe de développement.
- L'équipe de développement, après analyse détaillée de l'erreur, constate que la cause est comprise dans le modèle: en d'autres termes, il n'est pas possible de corriger l'erreur sans se livrer à des modifications majeures, touchant la structure même de l'application (modification radicale d'un circuit intégré dans un développement hardware, ou redéveloppement de 30% du code d'une application logicielle).
- L'analyse de la spécification montre enfin l'origine du problème : la spécification est imprécise sur un point précis, et il est possible de l'interpréter de plusieurs manières. Visiblement, le client qui a signé la spécification, l'équipe ayant rédigé cette spécification, et l'équipe de développement ayant interprété la spécification dans le but de réaliser le produit n'ont pas compris la même chose.
- Le client refuse d'assumer le surcoût, et fort de son bon droit, réclame une indemnité pour livraison retardée. La société hésite à engager une querelle juridique sur la base des imprécisions constatées dans la spécification, car l'issue en est hasardeuse, et de toutes façons mauvaise pour l'image de marque de la société. La société va donc prendre en charge les rais de correction, et payer une indemnité pour livraison retardée.
- La société perd donc beaucoup d'argent dans cette affaire, et sa réputation est néanmoins ternie du fait du retard à la livraison.

Il est donc vital que la méthode de développement choisie permette de minimiser le nombre d'erreurs intervenant en fin de cycle de développement (lors des tests, par exemple). Cette méthode doit donc fournir des outils permettant d'obtenir très tôt des représentations dans un formalisme strict du produit, tel qu'il sera à la livraison. Le formalisme doit permet-

tre un meilleur échange d'informations entre le client, le développeur, et les rédacteurs de la spécification.

Une spécification est en majeure partie textuelle, donc sujette à interprétation. Il est donc important que l'outil de modélisation permette aussi de servir de support à la rédaction de la spécification. UML fournit des outils permettant de répondre au moins partiellement à ce besoin.

7.2 *Démarches*

La notion de démarche est importante dans le sens qu'il n'y a pas de règle préétablie dans le processus de modélisation : beaucoup d'étapes sont fortement conditionnées par l'intuition, le *feeling* de l'ingénieur. On peut d'ailleurs s'estimer heureux qu'il en soit ainsi ! A quoi pourrait bien servir un ingénieur si son travail pouvait être entièrement codifié, et de ce fait implémenté par le premier ordinateur venu?

7.2.1 *Une démarche itérative et incrémentale ?*

L'idée est simple : pour modéliser (comprendre et représenter) un système complexe, il vaut mieux s'y prendre en plusieurs fois, en affinant son analyse par étapes. Cette démarche devrait aussi s'appliquer au cycle de développement dans son ensemble, en favorisant le prototypage. Le but est de mieux maîtriser la part d'inconnu et d'incertitudes qui caractérisent les systèmes complexes.

7.2.2 *Une démarche pilotée par les besoins des utilisateurs ?*

Avec UML, ce sont les utilisateurs qui guident la définition des modèles (mais cela devrait en fait être le cas de toute démarche de modélisation) :

- Le périmètre du système à modéliser est défini par les besoins des utilisateurs (les utilisateurs définissent ce que doit être le système).
- Le but du système à modéliser est de répondre aux besoins de ses utilisateurs (les utilisateurs sont les clients du système).

Les besoins des utilisateurs servent aussi de fil rouge, tout au long du cycle de développement (itératif et incrémental) :

- A chaque itération de la phase d'analyse, on clarifie, affine et valide les besoins des utilisateurs.
- A chaque itération de la phase de conception et de réalisation, on veille à la prise en compte des besoins des utilisateurs.
- A chaque itération de la phase de test, on vérifie que les besoins des utilisateurs sont satisfaits.

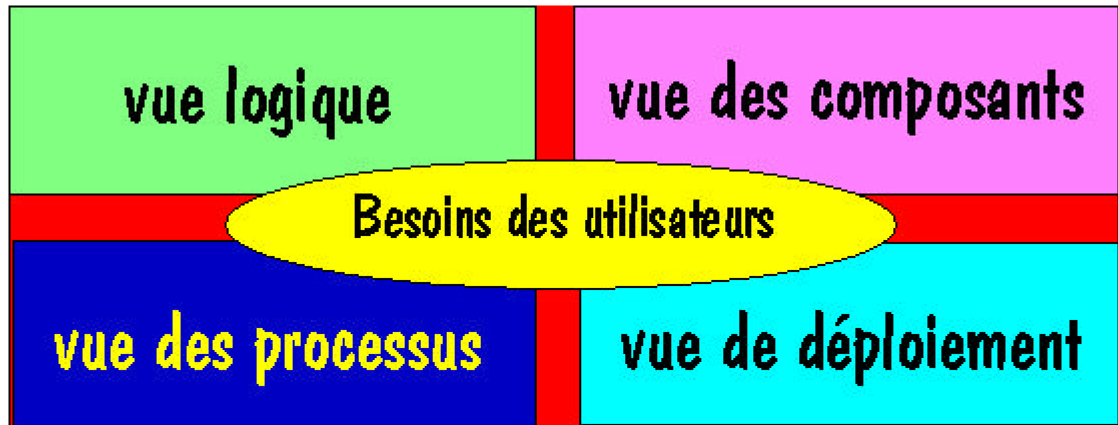
7.2.3 *Une démarche centrée sur l'architecture ?*

Une architecture adaptée est la clé de voûte du succès d'un développement. Elle décrit des choix stratégiques qui déterminent en grande partie les qualités du logiciel (adaptabilité, performances, fiabilité...).

Ph. Kruchten propose différentes perspectives, indépendantes et complémentaires, qui permettent de définir un modèle d'architecture (publication IEEE, 1995). Cette vue ("4+1") a fortement inspiré UML.

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

FIGURE 7.4 **La vue "4+1"**



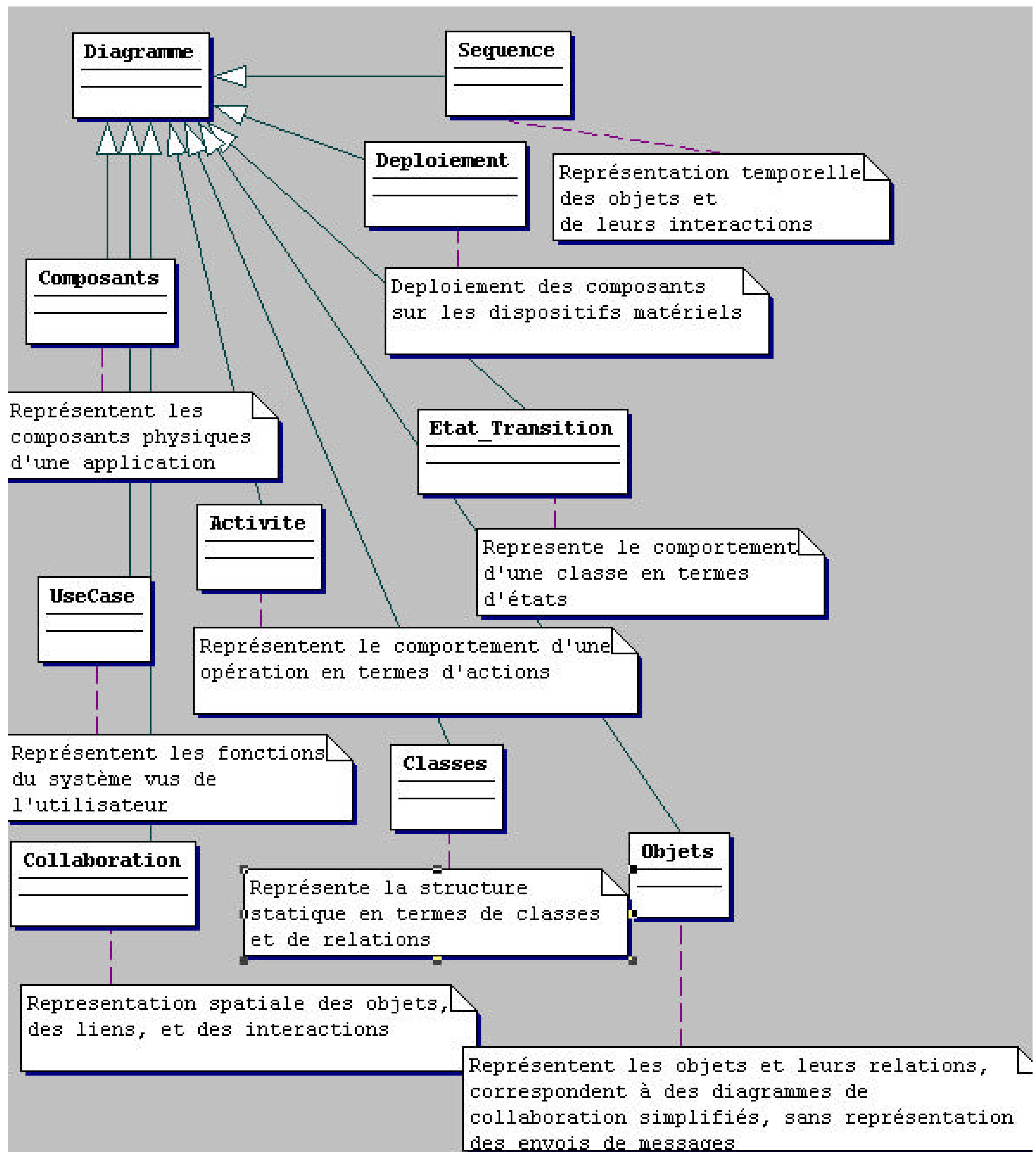
Il faut noter un fait important dans cette vue : les besoins des utilisateurs sont au centre des préoccupations, et tendent à influencer toute autre vue. Cette optique, à l'origine de laquelle se trouve surtout Ivar Jacobsson, a donné naissance à l'une des composantes les plus importantes d'UML : les cas d'utilisation.

7.3 *Vues*

On distingue deux catégories de vues du système : les vues statiques et les vues dynamiques. Chacune de ces catégories comprend plusieurs vues : toutes ont leur importance, mais l'importance relative des diverses vues peut varier en fonction du problème à résoudre : ainsi, la modélisation d'un protocole de communication accordera probablement plus d'importance aux vues dynamiques du système, alors que la modélisation d'un téléphone portable accordera plus d'importance au côté statique.

Une vue est la représentation d'un aspect du modèle, et tend à mettre en évidence certaines caractéristiques particulières, et à mettre en lumière certains problèmes particuliers qui vont se poser lors du développement, si possible avant que la phase de réalisation proprement dite ait commencé.

FIGURE 7.5 Les vues définies par UML



7.4 *Vues statiques du système*

7.4.1 *diagrammes de cas d'utilisation*

Le diagramme des cas d'utilisation est souvent la représentation directrice du système, celle qui permet de valider la modélisation. Il décrit le système sous forme d'une suite d'actions et de réactions du système à des stimuli, vu du point de vue de l'utilisateur.

7.4.2 *diagrammes d'objets*

Le diagramme d'objets permet de représenter les instances et leurs interactions. Ils décrivent des instantanés de l'état du système, en montrant dans une situation particulière (cas d'utilisation, souvent) les instances impliquées, ainsi que leurs attributs significatifs.

7.4.3 *diagrammes de classes*

Le diagramme de classes permet l'identification et la définition des types composant le système. Il exprime de manière générale la structure statique du système, et montre les relations entre les classes composant le système. Le diagramme de classes est très lié au code: on peut générer le code automatiquement à partir du diagramme de classes.

7.4.4 *diagrammes de composants*

Ils décrivent les éléments physiques du système et leurs relations dans l'environnement de réalisation.

7.4.5 *diagrammes de déploiement*

Ils montrent la disposition physique des différents matériels (les noeuds) qui entrent dans la composition d'un système et la répartition des programmes exécutables sur ces matériels.

7.5 *Vues dynamiques du système*

7.5.1 *diagrammes de collaboration*

Ce diagramme exprime les interactions entre les objets (quel objet a besoin de quel autre pour remplir son rôle). On peut considérer que le diagramme de collaboration est une extension du diagramme d'objets. Le diagramme de collaboration change au cours de la durée de vie du programme; il est difficile de définir un diagramme valable pour l'ensemble de la durée de vie du programme. Aussi se contente-t-on souvent de diagrammes décrivant les collaborations dans des circonstances particulièrement critiques ou difficiles.

7.5.2 *diagrammes de séquence*

Les diagrammes de séquence montrent des interactions entre objets selon un point de vue temporel.

7.5.3 *diagrammes d'états-transitions*

Ces diagrammes visualisent des automates d'états finis, du point de vue des états et des transitions.

7.5.4 *diagrammes d'activités*

Il s'agit d'une variante du diagramme d'états-transition, organisé par rapport aux actions, et destiné à représenter le comportement interne d'une méthode ou d'un cas d'utilisation.

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com