Développement avec UML

Objectifs

- Présenter les principes d'un support méthodologique
- Proposer une méthodologie simple de support au développement avec UML
- Illustrer l'intérêt d'UML pour le développement

Analyse et conception

Nous avons déjà indiqué au chapitre 1 que la finalité de l'activité de développement était de fournir une solution informatique à un problème posé par un utilisateur, aussi appelé client. Nous avons précisé que le code n'était que la matérialisation de la solution, tandis que le modèle contenait toutes les informations facilitant d'une manière ou d'une autre la construction de la solution.

Après cette introduction aux vues essentielles d'un modèle UML (structurelle, comportementale et fonctionnelle) d'une application, il nous reste à présenter la façon dont nous devons utiliser chacune de ces parties afin de réaliser notre objectif : la réalisation de la solution à partir du problème.

Pour atteindre cet objectif, l'ingénierie logicielle a identifié depuis plusieurs années deux phases principales à réaliser : l'analyse du problème et la conception de la solution.

Analyse du problème

La phase d'analyse sert à modéliser la *compréhension du problème* posé par le client. Cette phase sert aussi à bien définir les contours de l'application.

Grâce à la phase d'analyse, nous savons ce qui doit être intégré dans la solution, mais aussi ce qui ne doit pas l'être, puisque la solution ne doit prendre en compte que ce qui a été identifié lors de l'analyse. Idéalement, une analyse doit être réalisée par l'équipe de développement et validée par le client, lequel certifie ainsi que l'équipe de développement a bien compris son problème.

Dans ce cours, nous considérons que le problème du client est spécifié dans un cahier des charges. Le cahier des charges est un document textuel fourni par le client, mais qui n'est pas intégré dans le modèle d'une application. Dans notre contexte, la phase d'analyse consiste à modéliser tous les besoins présents dans le cahier des charges.

Une analyse est complète lorsque l'intégralité du problème est modélisée de manière non ambiguë.

Pour modéliser un cahier des charges avec UML, nous considérons que seules deux parties du modèle UML sont intéressantes, les parties fonctionnelle et structurelle :

- La partie fonctionnelle permet de spécifier les fonctionnalités réalisées par l'application (cas d'utilisation) ainsi que les contours de l'application (acteurs).
- La partie structurelle permet de spécifier sous forme d'objets les données que doit manipuler l'application.

Ces deux parties du modèle UML sont suffisantes pour modéliser les besoins du client exprimés dans le cahier des charges. Cependant, afin de bien préciser les liens de cohérence entre ces deux parties, nous utilisons aussi la partie comportementale, comme nous l'avons montré au chapitre précédent. Cette partie permet en effet de lier les cas d'utilisation aux interactions; elles-mêmes liées aux classes.

Dans le cadre de notre vision schématique du modèle UML d'une application, nous considérons que la phase d'analyse correspond à un niveau d'abstraction que nous appelons le niveau besoin.

Conception de la solution

La phase de conception consiste à modéliser une solution qui résout le problème modélisé dans la phase d'analyse.

Contrairement à ce que nous pourrions croire, fournir une solution informatique n'est pas ce qu'il y a de plus difficile : c'est juste un problème algorithmique. Par contre, il est bien plus compliqué de fournir la meilleure solution au problème, car, à un problème donné, correspondent bien souvent plusieurs solutions.

Prenons l'exemple du tri. Il existe plusieurs façons de trier des éléments (tri itératif, tri à bulles, tri rapide, etc.). Toutes ces solutions résolvent le problème du tri d'un point de vue

algorithmique, mais elles ne sont pas toutes équivalentes, et nous savons très bien que certaines sont meilleures que d'autres.

Pour différencier les solutions, deux critères sont bien connus : la complexité en espace et la complexité en temps. Ces critères permettent d'établir un classement des solutions en fonction de la place mémoire qu'elles occupent et du temps qu'elles mettent à réaliser le traitement.

D'autres critères, plus adaptés au monde industriel et au paradigme objet, permettent d'effectuer d'autres classements des solutions. Citons notamment la maintenabilité, la portabilité, la robustesse, la rapidité de développement, le coût de développement, etc. Cette liste non exhaustive de critères montre que la construction d'une solution optimale est loin d'être triviale.

Pour être pragmatique, mais aussi pour simplifier la difficulté de la phase de conception, nous considérons dans le cadre de ce cours qu'une conception optimale est une conception qui maximise l'indépendance à l'égard des plates-formes techniques et minimise les dépendances entre les différents objets de l'application.

À ces deux objectifs, nous faisons naturellement correspondre les deux niveaux d'abstraction que nous avons introduits au chapitre 8. En effet, nous avons vu que le niveau conceptuel permettait de définir une conception indépendante des plates-formes d'exécution. Nous avons vu en outre au chapitre 4 qu'il était possible de minimiser les relations de dépendance entre les packages du niveau physique.

Comment passer du quoi au comment ?

De manière intrinsèque, l'analyse et la conception sont fondamentalement différentes, la première correspondant à la modélisation du problème, et la seconde à la modélisation de la solution. Il existe entre ces deux niveaux une relation de résolution, puisque la conception résout l'analyse.

Il est important de souligner qu'il ne s'agit pas là d'une relation d'abstraction telle qu'elle existe entre les niveaux d'abstraction conceptuel et physique. La solution (définie par la conception) n'est pas la concrétisation d'un problème (défini par l'analyse) sur une plate-forme particulière. Il existe une réelle différence entre le problème et la solution. C'est d'ailleurs là où le travail du développeur prend tout son sens : fournir la meilleure solution susceptible de réponde au problème.

Pour autant, le fait que l'analyse et la conception tirent parti du paradigme objet et que la solution soit, par définition, « la solution du problème » rendent quelque peut confuse cette différence pourtant fondamentale. En effet, le modèle d'analyse contient des classes, lesquelles définissent la structure et le comportement des objets du problème. Or, il n'est pas rare de trouver dans la solution des classes aux structures et aux comportements relativement voisins.

Prenons l'exemple d'une application de gestion de prêts bancaires. Le modèle d'analyse définit la classe `Prêt`. Cette classe définit la structure et le comportement des prêts tels

que perçus dans le problème. Cette classe permet, par exemple, de renseigner le montant du prêt ainsi que son taux. Or, il est à parier que la solution exploite elle aussi la classe `Prêt` et que cette classe ressemble « fortement » à la classe appartenant au problème. Pour autant, cette relation étroite établie entre les classes d'analyse et les classes de conception ne doit pas faire oublier que ces deux phases ont des objectifs fondamentalement différents.

Afin de faciliter le passage de la phase d'analyse à la phase de conception, nous préconisons d'identifier au niveau conceptuel les *gros composants* de l'application.

Un gros composant, représenté à l'aide d'un package, est une entité relativement autonome, responsable d'une partie des traitements nécessaires au bon déroulement de l'application. À titre d'exemple, nous pouvons mentionner, pour l'application de gestion de prêts bancaires, un composant responsable des traitements graphiques de l'application (affichage des résultats, présentation des formulaires de saisie, etc.), un composant responsable de la sauvegarde des prêts sur disque dur (sauvegarde sur fichier, chargement à partir d'un fichier, etc.) et un composant responsable du calcul des taux et de la validation de l'acceptation du prêt.

Chaque composant joue un rôle spécifique dans l'application, et l'ensemble des composants est responsable de toutes les fonctionnalités de l'application (exprimées dans la phase d'analyse). La découpe en composants d'une application est une opération délicate, qui reste à la charge du développeur. C'est là que réside la relation de « résolution » entre le niveau analyse et le niveau conception.

Dans le modèle UML de l'application, les composants sont spécifiés au niveau conceptuel. Chaque composant est spécifié par une partie fonctionnelle, qui représente les fonctionnalités du composant offertes à son environnement, une partie comportementale, qui représente des exemples de réalisation des fonctionnalités du composant, et une partie structurelle, qui représente les classes du composant.

De plus, toujours au niveau conceptuel, mais de façon transverse à tous les composants, la partie structurelle du modèle spécifie les relations de dépendance entre les composants. Celles-ci sont exprimées sous forme de dépendance entre les packages représentant les composants.

UML ne proposant aucun concept pour spécifier une relation de « résolution » entre les niveaux besoin et conceptuel, nous préconisons d'utiliser des notes de commentaires intégrées dans le niveau conceptuel.

Chaque note de commentaires, attachée à un élément de n'importe quel type (classe, association, objet, message, cas d'utilisation, acteur, etc.), doit identifier les éléments du niveau besoin spécifiant le problème en partie résolu.

Nous préconisons de préciser les relations de « résolution » pour chaque composant du niveau conceptuel.

Plus précisément, nous préconisons, d'une part, d'attacher aux cas d'utilisation de chaque composant une note ciblant les cas d'utilisation du niveau besoin que le compo-

sant résout et, d'autre part, d'attacher aux classes ou aux packages de chaque composant une note ciblant les classes du niveau besoin que le composant résout.

La figure 10.1 illustre ces relations de résolution entre le niveau besoin et le niveau conceptuel. La figure fait apparaître deux gros composants dans le niveau conceptuel. Chacun de ces gros composants est spécifié à l'aide d'un diagramme de cas d'utilisation, d'un ensemble de diagrammes de séquence et d'un diagramme de classes. Les relations de résolution sont schématisées à l'aide de flèches entre les niveaux besoin et conceptuel.

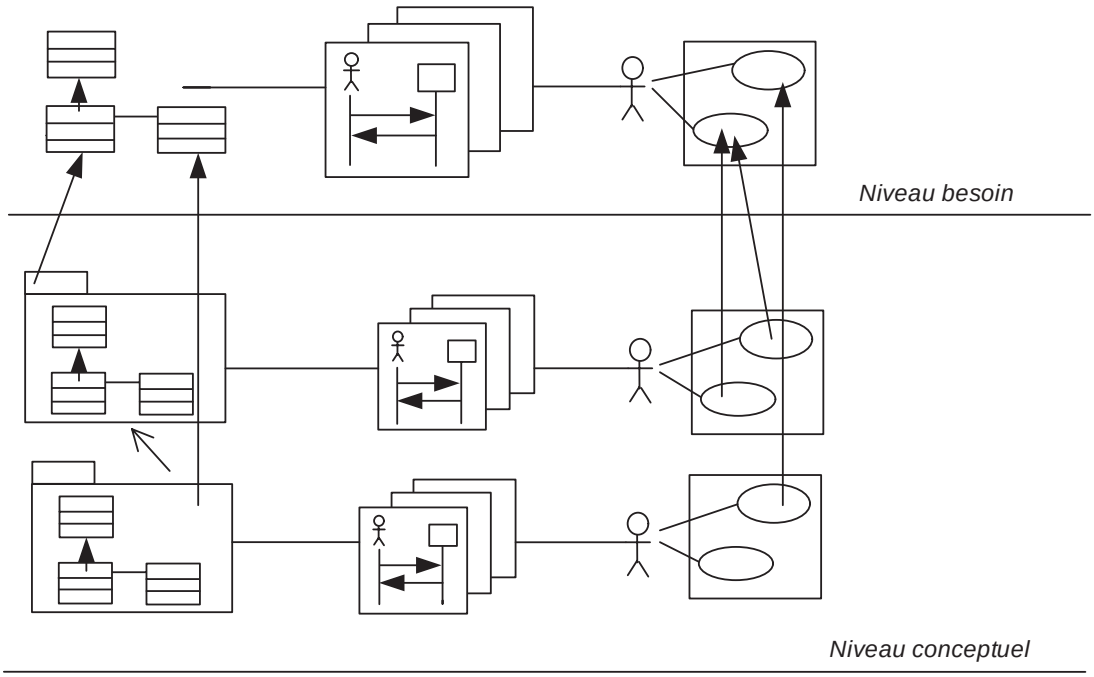

Figure 10.1
Relation de résolution entre le niveau besoin et le niveau conceptuel

En résumé, selon notre vision schématique du modèle UML d'une application, nous considérons que :

- L'analyse correspond à un niveau d'abstraction « besoin ».
- La conception correspond à deux niveaux d'abstraction : le niveau conceptuel et le niveau physique. Le niveau conceptuel spécifie les différents composants de l'application. Le niveau physique spécifie la façon dont ces composants sont réalisés sur la plate-forme technique (Java dans notre cas).
- Il existe des relations de résolution entre le niveau besoin et le niveau conceptuel. Ces relations sont exprimées à l'aide de commentaires dans le niveau conceptuel.

- Il existe des relations d'abstraction entre le niveau conceptuel et le niveau physique. Ces relations sont exprimées à l'aide de la relation d'abstraction entre les classes *(voir le chapitre 8).*

Grâce à ces relations entre toutes les parties du modèle UML et les phases d'analyse et de conception, nous pouvons compléter notre vision schématique d'un cycle de développement avec UML comme illustré à la figure 10.2.

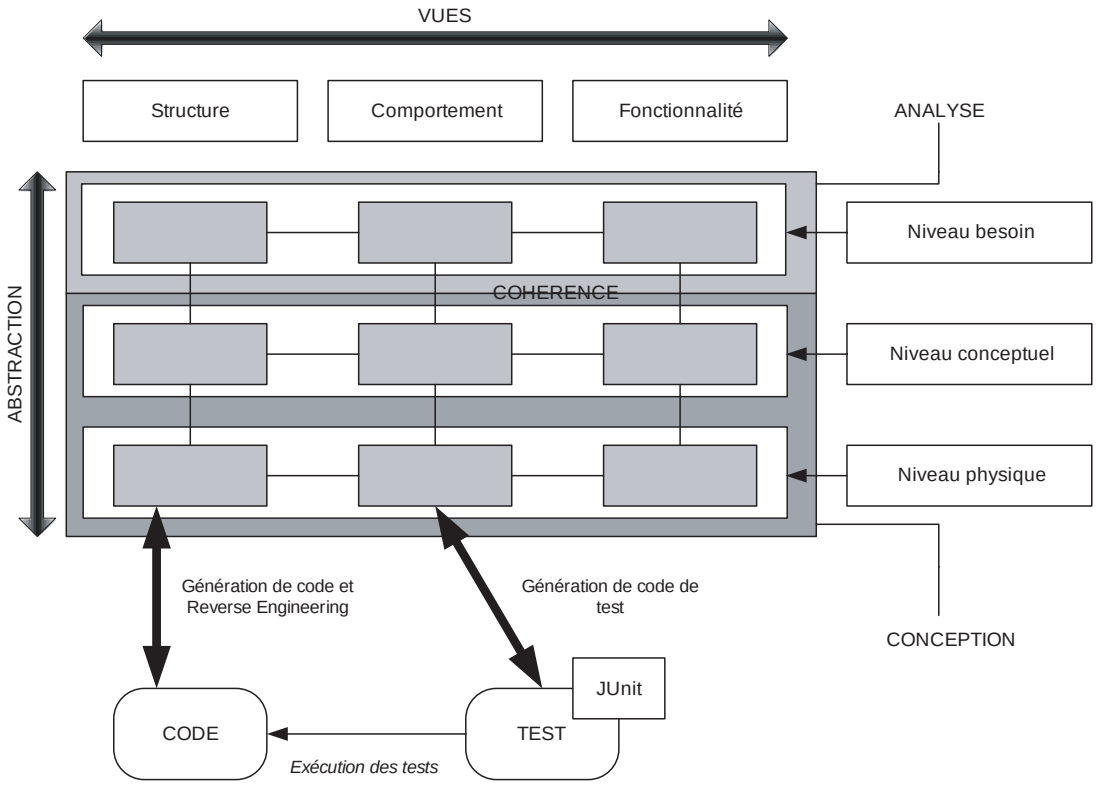

Figure 10.2

Cycle de développement avec UML (version complète)

Méthode de développement

Comme expliqué au chapitre 1, une méthode de développement doit répondre aux questions suivantes :

- *Quand* réaliser une activité ?
- *Qui* doit réaliser une activité ?
- *Quoi* faire dans une activité ?
- *Comment* réaliser une activité ?

Maintenant que nous avons présenté toutes les parties du modèle UML d'une application, nous savons ce que propose UML pour répondre aux questions du *quoi* et du *comment*.

Nous pouvons dès lors proposer une réponse relativement minimaliste aux questions du *quand* et du *qui* et, ainsi, proposer une méthode de développement avec UML.

Notre objectif est de finir notre présentation d'UML en présentant les principes de base d'une méthode de développement. Notre méthode ne doit donc pas être considérée comme une réelle méthode, applicable en milieu industriel, mais plutôt comme une méthode pédagogique.

Pour répondre à la question du *qui*, nous considérons qu'il existe essentiellement deux personnes qui participent au développement d'une application. Le client est la personne qui a besoin de l'application, et le développeur la personne qui réalise l'application. Dans le cadre de ce cours, nous considérons que l'équipe de développement n'est composée que de développeurs. Ainsi, la rédaction du cahier des charges, qui spécifie tous les besoins, est à la charge du client, et toutes les activités de développement relatives à l'analyse et à la conception sont à la charge de l'équipe de développement.

Pour répondre à la question du *quand*, nous considérons qu'il suffit de proposer un ordre de réalisation de chacune des neuf parties du modèle UML. Nous faisons le choix de proposer un ordre partant du cahier des charges et finissant par la production de l'application finale. Ainsi, notre méthode, qui se borne à préconiser un parcours dans la réalisation des parties du modèle, est une méthode dite « top-down ». Cela signifie qu'elle n'est utilisable que pour la construction de nouvelles applications.

Les gains de notre méthode de développement avec UML sont, comme nous l'avons souligné tout au long de ce cours, de profiter des avantages des opérations réalisables sur les modèles (génération de documentation, génération de tests, identification et correction des dépendances), de profiter des avantages des opérations réalisables sur le code (rédaction du code, compilation et exécution), de permettre une conception indépendante des plates-formes d'exécution et minimisant les dépendances entre les composants.

La méthode « UML pour le développeur »

Nous récapitulons dans cette section toutes les étapes de notre méthode de développement avec UML. Soulignons que les étapes 1 à 9 concernent l'élaboration des neuf parties du modèle UML.

0. Rédaction du cahier des charges :

Objectif : spécifier tous les besoins du client sur l'application.

Quand : cette étape doit être réalisée avant de commencer le développement. Elle n'appartient donc pas réellement à la méthode.

Qui : le client.

Quoi : un document textuel recensant tous les besoins. L'idéal serait de pouvoir identifier chacun des besoins (en leur donnant un numéro, par exemple).

Comment : nous ne préconisons aucune façon de rédiger un cahier des charges, cela sortant du contexte de ce cours.

1. **Analyse (niveau besoin) – cas d'utilisation :**

   Objectif : modéliser les fonctionnalités de l'application et l'environnement de l'application de manière non ambiguë.

   Quand : cette étape est la première de notre méthode.

   Qui : l'équipe de développement.

   Quoi : l'unique diagramme de cas d'utilisation du niveau d'abstraction « besoin ».

   Comment : à l'aide d'une lecture soigneuse du cahier des charges, il faut, d'une part, identifier les fonctionnalités offertes par l'application afin de les modéliser sous forme de cas d'utilisation et, d'autre part, identifier les entités externes à l'application bénéficiant des fonctionnalités de l'application afin de les modéliser sous forme d'acteurs. Rappelons que nous déconseillons l'utilisation de relations d'inclusion, d'extension et d'héritage entre les cas d'utilisation ainsi que les relations d'héritage entre acteurs.

2. **Analyse (niveau besoin) – interaction :**

   Objectif : modéliser les exemples de réalisation des fonctionnalités de l'application de manière non ambiguë.

   Quand : cette étape doit être réalisée après l'étape 1. Il est possible de réaliser l'étape 3 en même temps que cette étape. Analyse classe et analyse interaction peuvent être faites ensemble.

   Qui : l'équipe de développement.

   Quoi : un diagramme de séquence, par exemple nominal de réalisation d'une fonctionnalité et un diagramme de séquence, par exemple de réalisation d'une fonctionnalité soulevant une erreur.

   Comment : à l'aide d'une lecture soigneuse du cahier des charges, il faut modéliser des exemples de réalisation des fonctionnalités. Nous conseillons de réutiliser autant que possible de mêmes objets dans les différentes interactions. Afin d'établir un lien de cohérence entre les parties fonctionnelle, comportementale et structurelle, nous conseillons de typer tous les objets participant aux interactions, soit par des acteurs (partie fonctionnelle), soit par des classes (partie structurelle).

3. **Analyse (niveau besoin) – classes :**

   Objectif : modéliser les classes représentant les données spécifiées dans le cahier des charges.

   Quand : cette étape doit être réalisée après l'étape 1 et peut être faite en même temps que l'étape 2. Analyse classe et analyse interaction peuvent être faites ensemble.

   Qui : l'équipe de développement.

Quoi : autant de diagrammes de classes que nécessaire afin de faciliter la lecture des classes ainsi que leurs relations.

Comment : à l'aide d'une lecture soigneuse du cahier des charges, il faut modéliser les données spécifiées par le cahier des charges. Nous conseillons de ne pas rendre les associations navigables, car les informations que nous pourrions en retirer (dépendances entre objets) ne sont pas du ressort de la phase d'analyse.

4. **Conception (niveau conceptuel) – cas d'utilisation :**

   Objectif : modéliser les composants de la conception de l'application.

   Quand : cette étape, la quatrième de notre méthode, est la première de conception. Elle doit être réalisée après toutes les étapes de la phase d'analyse.

   Qui : l'équipe de développement.

   Quoi : liste des composants et un diagramme de cas d'utilisation par composant.

   Comment : à l'aide d'une lecture soigneuse de toutes les parties de la phase d'analyse, il faut identifier les différents composants de l'application puis élaborer le diagramme de cas d'utilisation de chacun des composants. Il faut ensuite spécifier les relations de résolution entre les diagrammes de cas d'utilisation des composants et le diagramme de cas d'utilisation de la phase d'analyse.

5. **Conception (niveau conceptuel) – interaction :**

   Objectif : modéliser les exemples de réalisation des fonctionnalités de chacun des composants de l'application.

   Quand : cette étape est la deuxième de la phase de conception. Il est possible de réaliser l'étape 7 en même temps que cette étape.

   Qui : l'équipe de développement.

   Quoi : un diagramme de séquence, par exemple nominal de réalisation d'une fonctionnalité. Un diagramme de séquence, par exemple de réalisation d'une fonctionnalité soulevant une erreur.

   Comment : à partir de la définition des composants, il faut élaborer des exemples de réalisation des fonctionnalités qu'ils proposent. Nous conseillons de réutiliser autant que possible de mêmes objets dans les différentes interactions. Afin d'établir un lien de cohérence entre les parties fonctionnelle, comportementale et structurelle, nous conseillons de typer tous les objets participant aux interactions, soit par des acteurs (partie fonctionnelle), soit par des classes (partie structurelle).

6. **Conception (niveau conceptuel) – classes :**

   Objectif : modéliser les classes des composants. Toutes les classes d'un même composant doivent appartenir à un même package.

   Quand : cette étape, la troisième de conception de notre méthode, doit être réalisée après ou en même temps que l'étape 5.

   Qui : l'équipe de développement.

Quoi : autant de diagrammes de classes que nécessaire afin de faciliter la lecture des classes ainsi que leurs relations.

Comment : à partir de la définition des composants, il faut modéliser les données qu'ils manipulent. Il faut préciser les relations de dépendance entre les classes (intra-composants et inter-composants).

Objectif : modéliser les classes des composants en intégrant les classes de la plate-forme d'exécution.

7. **Conception (niveau physique) – classes :**

   Quand : cette étape est la première de la phase de conception du niveau physique de notre méthode.

   Qui : l'équipe de développement.

   Quoi : autant de diagrammes de classes que nécessaire afin de faciliter la lecture des classes ainsi que leurs relations.

   Comment : à partir de la spécification des composants (partie structurelle du niveau conceptuel), il faut identifier les classes de la plate-forme d'exécution permettant la concrétisation des composants. Il faut aussi intégrer les traitements associés aux opérations sous forme de note de code. Pour finir, il faut préciser les relations d'abstraction avec le niveau conceptuel.

8. **Conception (niveau physique) – interaction :**

   Objectif : modéliser les cas de test abstraits.

   Quand : cette étape est la deuxième de la phase de conception du niveau physique.

   Qui : l'équipe de développement.

   Quoi : un diagramme de séquence de test par cas de test abstrait.

   Comment : pour chaque classe du niveau conceptuel, il faut identifier plusieurs tests abstraits à réaliser et modéliser ces cas de test à l'aide de diagrammes de séquence de test. Comme indiqué au chapitre 7, notre méthode ne donne aucun moyen d'identifier les bons cas de test.

9. **Conception (niveau physique) – cas d'utilisation :**

   Objectif : modéliser les fonctionnalités offertes par les composants, mais au niveau physique.

   Quand : cette étape est la troisième de la phase de conception du niveau physique.

   Qui : l'équipe de développement.

   Quoi : un diagramme de cas d'utilisation par package du niveau physique.

   Comment : en principe, tous les composants du niveau conceptuel apparaissent dans le niveau physique sous forme de packages. Certaines fonctionnalités des composants du niveau conceptuel peuvent toutefois être offertes directement par la plate-forme d'exécution. La modélisation des fonctionnalités au niveau physique permet

ainsi de différencier les fonctionnalités directement réalisées par l'application de celles offertes par la plate-forme.

10. Génération du code et des tests :

Objectif : générer le code de l'application et celui des tests.

Quand : cette étape est la première de la phase de codage.

Qui : l'équipe de développement.

Quoi : le code de l'application et celui des tests.

Comment : en exécutant les opérations de génération de code et de tests.

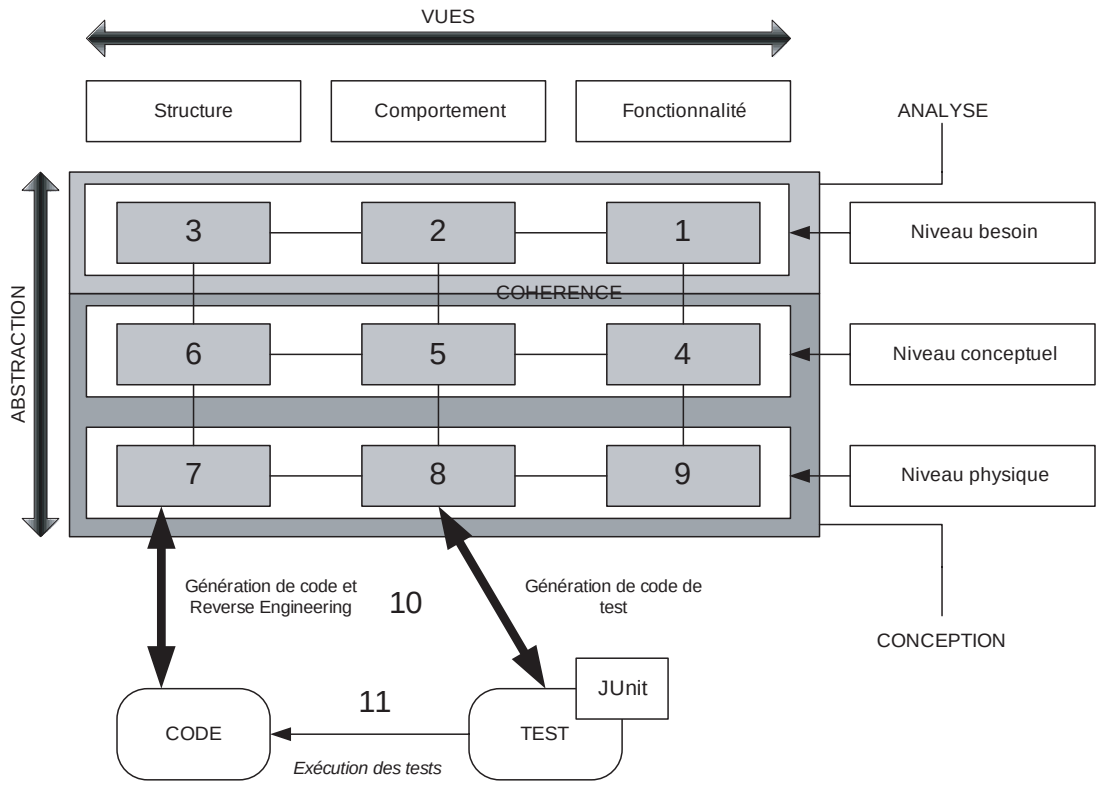

Figure 10.3
Étapes de la méthode et parties du modèle UML

11. Compilation et exécution du code et des tests :

Objectif : compiler et exécuter le code de l'application et celui des tests.

Quand : cette étape est la deuxième de la phase de codage.

Qui : l'équipe de développement.

Quoi : l'exécutable.

Comment : en exécutant les opérations de compilation et d'exécution fournies par la plate-forme d'exécution.

12. **Modification de l'application (correction de bogues ou réalisation d'évolutions) :**

Objectif : mettre à jour l'application.

Quand : à tout moment après l'étape 11.

Qui : l'équipe de développement.

Quoi : modification du modèle ou du code.

Comment : en modifiant n'importe quelle partie du modèle ou en modifiant le code. Les opérations de génération de code et de Reverse Engineering doivent être utilisées pour assurer la synchronisation entre le code et le modèle.

Toutes ces étapes, après la rédaction du cahier des charges, se retrouvent dans notre vision schématique du modèle UML d'une application, telle qu'illustrée à la figure 10.3.

Synthèse

Dans ce chapitre, qui clôt ce cours, nous avons présenté la façon dont s'articulent toutes les parties du modèle UML d'une application lors de la réalisation d'un développement avec UML.

Nous avons en particulier souligné la distinction entre les phases d'analyse et de conception, qui permettent respectivement de spécifier le problème posé par le client et la solution proposée par l'équipe de développement.

Nous avons en outre proposé une méthode pédagogique permettant de suivre un cycle de développement avec UML lors de la construction d'une nouvelle application. Ce cycle préconise un chemin dans l'élaboration de toutes les parties du modèle UML et se termine par la génération de code.

Grâce à cette méthode mais surtout grâce aux différentes parties du modèle élaborées en UML, il est possible de cumuler les avantages de la modélisation avec ceux du codage.

Insistons à nouveau sur le fait que la méthode que nous avons présentée n'utilise pas de manière exhaustive toutes les possibilités d'UML ni toutes les opérations applicables sur les modèles (simulation de modèles, vérification de propriétés, etc.), qui permettent d'obtenir d'autres gains. Cependant, notre méthode présente les parties d'un modèle UML qui offrent le plus d'avantages en terme de développement et qu'il est nécessaire de maîtriser pour pouvoir aller plus loin dans la modélisation des systèmes.

Travaux dirigés

TD10. Développement avec UML

Une association d'ornithologie vous confie la réalisation du système logiciel de recueil et de gestion des observations réalisées par ses adhérents (le logiciel DataBirds). L'objectif est de centraliser toutes les données d'observation arrivant par différents canaux au sein d'une même base de données, qui permettra ensuite d'établir des cartes de présence des différentes espèces sur le territoire géré par l'association.

Les données à renseigner pour chaque observation sont les suivantes :

- Nom de l'espèce concernée. Il y a environ trois cents espèces possibles sur le territoire en question. Si l'observation concerne plusieurs espèces, renseigner plusieurs observations
- Nombre d'individus.
- Lieu de l'observation.
- Date de l'observation.
- Heure de l'observation.
- Conditions météo lors de l'observation.
- Nom de chaque observateur.

Quelle que soit la façon dont sont collectées les données, celles-ci sont saisies dans la base dans un état dit « à valider ». Tant que les données ne sont pas validées par les salariés de l'association, des modifications peuvent être faites sur les données.

La validation des données se fait uniquement par les salariés de l'association qui ont le droit de modifier la base de DataBirds. Ils doivent vérifier que les données saisies sont cohérentes. Plus précisément, ils doivent valider les noms des observateurs (les noms doivent correspondre à des noms d'adhérents) et l'espèce (celle-ci doit correspondre à une espèce connue sur le territoire).

Après validation, une saisie se trouve soit dans l'état dit « validé », soit dans l'état dit « non validé ». Les saisies dans l'état « non validé » sont automatiquement purgées de la base une fois par semaine.

Grâce aux données saisies et validées, l'association souhaite pouvoir établir différents types de cartes de présence des différentes espèces :

- Cartes géographiques par espèce présentant un cumul historique des populations. Ce traitement peut être demandé par un adhérent.
- Cartes des observations réalisées par chaque observateur. Ce traitement peut être demandé par un salarié uniquement.

Ces cartes de présence des oiseaux sont générées par DataBirds et accessibles soit par le Web, soit par demande *via* un courrier électronique ou postal.

Question 83 *Effectuez la première étape de la méthode.*

Question 84 *Effectuez la deuxième étape de la méthode (niveau besoin - comportement).*

Question 85 *Effectuez la troisième étape de la méthode.*

Question 86 *Effectuez la quatrième étape de la méthode.*

Question 87 *Effectuez la cinquième étape de la méthode.*

Question 88 *Effectuez la sixième étape de la méthode.*

Question 89 *Effectuez la septième étape de la méthode.*

Question 90 *Effectuez la huitième étape de la méthode sur une seule classe.*

Question 91 *Effectuez la neuvième étape de la méthode.*

Ce TD aura atteint son objectif pédagogique si et seulement si :

- Vous savez appliquer chaque étape de la méthode.
- Vous comprenez l'importance des relations de cohérence entre les parties du modèle.
- Vous êtes capable de mesurer les gains offerts grâce à l'élaboration du modèle UML d'une application.