Avant-propos

UML (Unified Modeling Language) est aujourd'hui le langage de modélisation d'applications informatiques le plus important du marché. Il est supporté par la quasi-totalité des outils de développement, lesquels permettent l'édition de modèles UML et offrent des capacités telles que la génération de code, de test et de documentation, le suivi d'exigences ou encore le Reverse Engineering.

Pour autant, ce langage reste très complexe et n'est pas facile à assimiler, surtout lorsque nous souhaitons obtenir rapidement un gain de productivité. La raison à cela est que l'approche classique d'utilisation d'UML, que nous nommons *UML pour l'architecte,* vise surtout à utiliser les modèles UML comme des moyens de réflexion, d'échange et de communication entre les membres d'une même équipe de développement. Cette approche suit toutes les phases du cycle de vie des applications. La génération de code n'arrive alors qu'à la fin et n'est rentable que si nous avons respecté scrupuleusement toutes les phases antérieures. La difficulté de cet exercice nous fait mieux comprendre pourquoi les gains de productivité ne sont que rarement obtenus.

Une autre approche UML, que nous nommons *UML pour le développeur,* est déjà identifiée par quelques outilleurs du marché. Davantage adaptée au développeur qu'au travail en équipe, cette approche vise à obtenir des gains de productivité très rapidement. L'idée principale à la base de cette approche consiste à effectuer des allers-retours entre modèles UML et code dans l'objectif d'utiliser conjointement les meilleurs avantages de chacun des deux mondes (modèle et code). Ainsi, l'écriture d'algorithmes, la compilation et l'exécution sont laissées au niveau des langages de programmation, tandis que la découpe en packages ou l'application de patrons de conception s'effectue au niveau des modèles UML. Des synchronisations sont effectuées entre les modèles et le code afin d'assurer une cohérence de l'ensemble. Cette approche très pragmatique offre rapidement de forts gains de productivité.

Ces deux approches opposées compliquent l'apprentissage d'UML pour toute personne désireuse de savoir comment utiliser ce langage dans son propre contexte. De plus, tous les ouvrages existants adressent principalement l'approche *UML pour l'architecte* et ne font que peu de cas des mécanismes liant UML au code des applications. L'étudiant, tout comme le développeur de métier, ne peuvent dès lors mesurer pleinement les avantages de ce langage pour leur contexte, qui porte essentiellement sur le développement du code des applications.

C'est pourquoi nous présentons dans cet ouvrage un cours exclusivement dédié à l'approche *UML pour le développeur.* Notre objectif est de montrer la complémentarité que peut offrir UML à n'importe quel langage de programmation. Nous présentons dans chaque cas les gains de productivité que nous pouvons en obtenir.

Une approche à contre-pied

Le déroulement pédagogique de ce cours est volontairement à contre-pied des cours UML classiques. Alors que ces derniers commencent invariablement par présenter les fameux diagrammes de cas d'utilisation et finissent par la génération de code, nous proposons l'inverse, en commençant par le code et en finissant par les diagrammes de cas d'utilisation. Notre objectif est de mettre au premier plan les mécanismes UML qui offrent directement des gains de productivité et d'en mesurer les impacts. Pour autant, les principales notions UML auront été introduites et commentées à la fin du cours.

Organisation de ce cours

Le plan de ce cours est le suivant :

1. *Un curieux besoin de modèles* : ce chapitre présente les principaux avantages des modèles UML afin de bien faire comprendre les relations entre modèle et code. Nous y définissons la notion de niveau d'abstraction qui permet de représenter une même application suivant différentes vues.
2. *Diagrammes de classes UML* : ce chapitre présente le plus employé des diagrammes UML. Ce chapitre n'est pas un guide de référence du diagramme de classes. Nous présentons les concepts nécessaires dans le contexte de ce cours.
3. *Reverse Engineering* : ce chapitre présente les principes du Reverse Engineering, qui consiste à construire automatiquement un modèle UML à partir de code. Nous définissons un ensemble de règles permettant de produire un diagramme de classes à partir du code d'une application.
4. *Rétroconception et patrons de conception* : ce chapitre présente les opérations de restructuration d'applications effectuables sur des modèles UML. Nous expliquons le rôle des patrons de conception et comment les appliquer sur un diagramme de classes.
5. *Génération de code* : ce chapitre présente les principes de la génération de code à partir de modèles UML. Nous définissons un ensemble de règles permettant de générer du code à partir d'un diagramme de classes.
6. *Diagrammes de séquences* : ce chapitre présente les diagrammes de séquence. Nous expliquons en quoi ces diagrammes sont nécessaires pour mieux comprendre le comportement d'une application. Nous insistons sur le fait qu'ils ne contiennent pas l'information nécessaire à la génération de code.

7. *Diagrammes de séquences et tests* : ce chapitre présente l'utilisation des diagrammes de séquence pour la génération de tests. Nous expliquons les principes du test d'application et sa mise en œuvre par l'intermédiaire de diagrammes de séquence de test.
8. *UML et les plates-formes d'exécution* : ce chapitre présente les relations entre UML et les plates-formes d'exécution afin de bien faire comprendre la capacité d'abstraction des modèles UML. Nous insistons sur le fait qu'il est important, pour une même application, d'avoir des modèles indépendants de la plate-forme d'exécution et d'autres plus étroitement liés à cette dernière.
9. *Diagrammes de cas d'utilisation* : ce chapitre présente les diagrammes de cas d'utilisation. Ils sont utilisés pour représenter les fonctionnalités d'une application quel que soit le niveau d'abstraction considéré.
10. *Développement avec UML* : ce chapitre présente une méthode de développement avec UML permettant d'obtenir l'ensemble des diagrammes nécessaires à la représentation d'une application Nous partons cette fois de la description de l'application et nous expliquons l'ensemble des étapes à suivre pour obtenir le code de l'application tout en ayant construit l'ensemble des diagrammes nécessaires pour faire le lien entre tous les niveaux d'abstraction.

Pour rendre plus concrète les relations entre code et modèle, nous avons choisi de baser ce cours sur le langage Java. Tous les mécanismes de génération de code ou de Reverse Engineering que nous présentons s'appuient donc sur Java. Les principes que nous présentons dans ce cours sont cependant transposables vers d'autres langages de programmation.

Chaque cours est suivi d'un ensemble d'exercices complémentaires du cours. Nous soulignons que la lecture de la partie cours uniquement ne permet pas d'accéder à l'ensemble des informations présentées dans ce livre.

À qui s'adresse ce cours ?

Cet ouvrage s'adresse principalement aux étudiants et aux développeurs de métier ayant des connaissances en programmation par objets et désireux de découvrir les bénéfices du langage UML pour le développement d'applications. Il ne s'agit pas d'un guide de référence sur UML.

Chaque notion importante dans le contexte du développement avec UML est introduite par un exemple, et chaque chapitre se clôt par une série d'exercices (91 au total) avec corrigés, qui permettront au lecteur de tester ses connaissances.

L'ouvrage s'adresse aussi aux enseignants désireux de transmettre les principes de base des langages de modélisation selon une approche pragmatique, en liaison avec les techniques classiques de développement d'applications.