# Visual Studio 2010 et la modélisation

**Avec Visual Studio 2010 les portes de l'architecture et de la modélisation se sont ouvertes à un large public. L'outil n'a pas prétention de rivaliser avec les cadors du genre, mais de démocratiser ces disciplines qui sont aujourd'hui très importantes et d'apporter une valeur ajoutée indéniable à l'ensemble de l'équipe.**

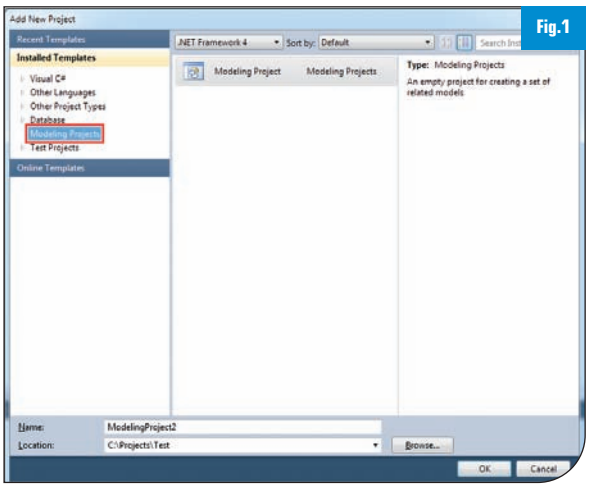


Fig.1

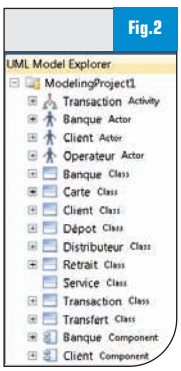


Fig.2

La modélisation est pour beaucoup de développeurs une étape obscure voire inutile. Cette réputation est malheureusement en partie fondée : de nombreux outils de modélisation UML existent sur le marché, mais quasiment tous ont pour effet d'isoler la phase de modélisation du reste de la réalisation d'un projet. Cela est dû au fait que ces logiciels n'offrent pas d'intégration (du moins assez efficace) avec les disciplines adjacentes comme le code ou les tests. Microsoft joue de nouveau les cartes de l'intégration et du pragmatisme avec la brique Modélisation de Visual Studio 2010. Les experts UML ne trouveront certainement pas cette version à la hauteur (malgré la sortie du Feature Pack qui comble bon nombre de lacunes), mais pour ceux qui approchent cette discipline comme un outil permettant de travailler mieux et plus vite il n'y a pas de doute : Microsoft est sur la bonne voie !

## Projet de type modélisation

Pour la modélisation, Visual Studio 2010 prévoit un nouveau type de projet spécifique qui regroupe tous les diagrammes. Ce type de projet est disponible dans la catégorie « Modeling Projects » de la fenêtre de dialogue « Add New Project » [Fig.1]. Un projet est nécessaire afin de pouvoir accéder aux différents types de diagrammes, chaque diagramme est constitué d'un ou plusieurs fichiers. En procédant de la sorte, Microsoft vous garantit le même comportement par rapport aux projets classiques (WinForms, Class Library, …) vis-à-vis de votre solution et de la gestion de configuration.

Deux points d'entrés sont alors disponibles : le premier vous est familier puisqu'il s'agit de l'explorateur de solution. Celui-ci vous permet de créer et d'organiser les différents diagrammes comme vous avez l'habitude de le faire et ainsi d'intégrer la modélisation directement au même niveau que les autres projets de la solution : les développeurs ont enfin un accès simple à ces données. Le second est une fenêtre inédite intitulée « Explorateur de modèle UML » qui centralise,comme son nom l'indique, l'ensemble des éléments de type UML.

Chaque élément créé dans un diagramme est référencé dans l'explorateur de modèle UML, cela permet de pouvoir constituer une bibliothèque d'éléments ayant un sens fonctionnel qui peuvent être par la suite réutilisés au travers des différents diagrammes [Fig.2]. Une petite astuce : la centralisation de ces éléments se fait dans un unique fichier XML, ce qui peut parfois aboutir à des fusions difficiles. Un contournement existe en utilisant l'élément UML de type Package : celui-ci et son contenu sont stockés dans un fichier XML à part. Si vous être amené à travailler en simultané sur un gros projet de modélisation, le regroupement des éléments UML par Package vous simplifiera la vie ! (même si ce n'est pas l'utilisation normale des packages). Dans sa version actuelle, Visual Studio 2010 gère 5 diagrammes UML :

- Le diagramme de cas d'utilisation.
- Le diagramme d'activités.
- Le diagramme de séquence.
- Le diagramme de classes.
- Le diagramme de composants.

Deux autres types de diagrammes non UML sont aussi proposés : le diagramme de couches qui vous permet de modéliser et cartographier sous forme de couches logiques votre code et le graphe direct qui, lui, est un modèle générique permettant de représenter des diagrammes tels que le graphe de dépendance. Nous reviendrons en détail sur le premier type de diagrammes par la suite.

Comme nous le disions au début de l'article, l'intégration est un atout majeur de la suite de développement de Microsoft.

Conjuguée avec Team Foundation Server 2010, comme pour les différents éléments qui composent notre développement (documentation, fragment de code source, résultat de test, …), nous avons la possibilité d'associer un ou plusieurs éléments de travail à un élément UML ou directement à un diagramme. Cette relation est très précieuse car nous pouvons de ce fait obtenir une traçabilité qui ira jusqu'au code et aux tests !

Pour les développeurs, c'est la relation inverse qui leur simplifie la vie : pouvoir, à partir d'une tâche, remonter aux différents diagrammes qui décrivent le code à réaliser.

Malheureusement, dans leurs versions initiales, Visual Studio et TFS ne le permettent pas ! Un élément UML peut référencer un élément de travail mais on ne retrouve pas ce lien sur l'élément de travail.

Plus d'une personne ayant trouvé ce problème très gênant, Microsoft n'a pas attendu une nouvelle version ou

# 1&1 HÉBERGEMENT

## TOUS LES PACKS AU PRIX DU MOINS CHER !

**TOUS LES PACKS HÉBERGEMENT À SEULEMENT : 1,99 € HT/mois (2,38 € TTC/mois) pendant les 3 premiers mois***


**Vous voudriez choisir votre solution d'hébergement sans vous soucier du prix ? Chez 1&1, tous les packs d'hébergement vous sont actuellement proposés au prix du moins cher : seulement 1,99 € HT/mois* durant 3 mois quelque soit le pack !**

**Faites votre choix sur 1and1.fr, où vous découvrirez également nos serveurs, e-boutiques et noms de domaines à petits prix !**

## DOMAINES À PRIX CASSÉS :

**le .fr à 3,99 € HT/an (4,77 € TTC/an),**
**le .info à 0,99 € HT/an (1,18 € TTC/an)* !**

* Tous les Packs Hébergement sont au prix du Pack Initial (1,99 € HT/mois, soit 2,38 € TTC/mois). A l'issue des 3 premiers mois, les produits concernés sont aux prix habituels (Pack Confort à 5,97 € TTC/mois, Pack Pro à 11,95 € TTC/mois, Pack Premium à 23,91 € TTC/mois). Frais de mise en service : 5,97 € TTC (Pack Confort) ou 11,95 € TTC (Pack Pro, Pack Premium). Offre soumise à un engagement de 12 mois. Offre domaines applicable la première année au lieu du prix habituel de 6,99 € HT/an (8,36 € TTC). Conditions détaillées sur www.1and1.fr. Offres sans engagement également disponibles.

MEMBRE DE united internet

Appel non surtaxé
0970 808 911

www.1and1.fr

1&1

un service pack pour y remédier. Avec le « Visualization & Modeling Feature Pack », il est maintenant possible via un nouveau type de lien d'élément de travail de référencer un élément UML.
Après cette présentation du projet de type modélisation, nous allons voir, au travers d'un exemple de distributeur de banque, chaque diagramme supporté par Visual Studio 2010 et nous apprécierons comment ils peuvent simplifier le développement.

## Diagramme de cas d'utilisation

La première phase de tout développement est la définition des besoins. A partir des interviews des utilisateurs, on pourra créer des diagrammes de cas d'utilisation afin de formaliser leurs besoins de manière globale, compréhensible, concise et claire.
On va d'abord définir les acteurs qui vont interagir avec le système. Dans notre exemple, l'opérateur assure la maintenance, la Banque exécute les transactions et le Client utilise le système. La prochaine étape consiste en la définition des cas d'utilisation. La meilleure approche est de trouver les besoins que chaque acteur a envers le système.
On donne ensuite un nom unique et une brève description qui décrit clairement les objectifs pour chaque cas d'utilisation. Appliqué à notre exemple, on peut facilement trouver les cas d'utilisation suivants :

- L'opérateur démarre et arrête le système.
- Le client veut effectuer des transactions telles que retrait, dépôt ou transfert.
- La banque agit comme système extérieur avec lequel les transactions sont exécutées.

Enfin, les associations permettent de définir les liens entre les éléments du diagramme.
Dans Visual Studio 2010, tous ces éléments sont disponibles dans la boîte à outils. Après création d'un nouveau diagramme, il suffit d'utiliser le glisser/déposer pour constituer le diagramme avec les acteurs, les cas d'utilisation et leurs associations [Fig.3]. Jusque-là nous sommes dans le classique. Là où Visual Studio 2010 va apporter un plus, c'est grâce à son intégration avec TFS 2010 en permettant de lier les différents cas d'utilisation aux exigences stockées sous forme d'éléments de travail. Un développeur pourra alors, en partant de ce qu'il connaît le mieux (les éléments de travail), retrouver facilement via ces liens les diagrammes correspondant aux exigences sur lesquelles il va devoir travailler. Les exigences dans TFS pouvant être liées à des cas de tests, on aura donc aussi automatiquement un lien logique entre cas d'utilisation et cas de tests !

## Diagramme d'activités

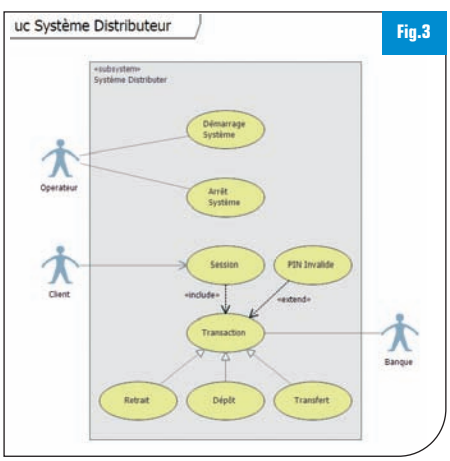
Après la première phase d'analyse et de définition de besoins, on commence à détailler les cas d'utilisation en diagrammes d'activités. Le diagramme d'activités décrit le comportement d'un processus d'un système. Une activité est un déroulement d'étapes séquentielles.
Dans notre exemple, on va étendre le cas d'utilisation « Transaction » par un diagramme d'activités qui va expliquer le comportement attendu :

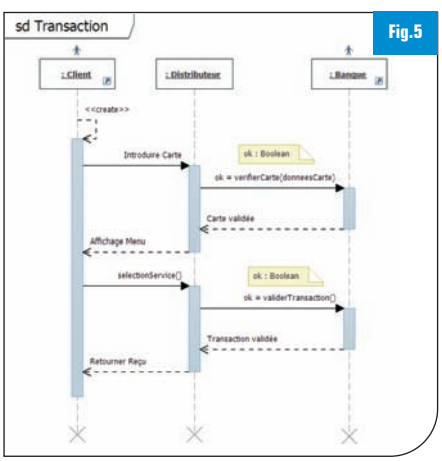
- le distributeur attend l'introduction d'une carte valide par le client.
- Le client sélectionne un service.
- Le distributeur délivre un reçu de la transaction.

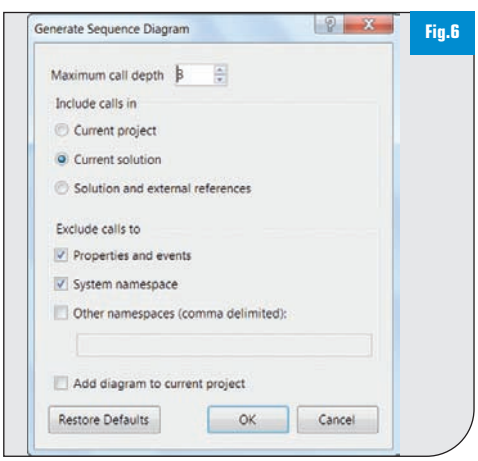
La modélisation de ce diagramme dans Visual Studio 2010 se fait de la même manière que pour la création du diagramme de cas d'utilisation, par glisser/déposer. La boîte à outils contenant maintenant les éléments adaptés tels que les actions, les décisions, les connecteurs … En quelques clics, le diagramme correspondant à notre exemple est créé [Fig.4].
Le diagramme d'activités sert à illustrer et consolider la description textuelle des cas d'utilisation. Il est également utile dans la phase de réalisation, car il donne les lignes directrices pour la programmation et pourra donc être lié aux tâches dans TFS afin que le développeur puisse se simplifier le travail en ayant accès directement au diagramme décrivant l'algorithme à mettre en place.

## Diagramme de séquence

Une autre façon de détailler une partie ou l'intégralité des cas d'utilisation est le diagramme de séquence. Ce diagramme a pour but de montrer la séquence des interactions entre objets ou acteurs sur un axe de temps donné. Le diagramme est constitué de deux dimensions. La dimension verticale montre la séquence des messages dans le temps. La dimension horizontale montre les instances des objets auxquelles sont envoyés les messages. Les messages peuvent être synchrones ou asynchrones.
Un diagramme de séquence est très



simple à élaborer dans Visual Studio 2010. En haut du diagramme sont identifiées les instances de classe. Le temps est représenté du haut vers le bas le long des lignes de vie. Les messages sont représentés par des flèches d'un acteur vers un autre. Tout ceci s'effectue rapidement dans Visual Studio 2010 encore une fois grâce au glisser/déposer des objets de la boîte d'outils.

Dans notre exemple de la « Transaction », on identifie les interactions entre Client, Distributeur et Banque. On définit les messages entre les entités comme l'introduction et la vérification de la carte, l'affichage du menu et sélection du service par le client, la validation et l'exécution de la transaction et enfin le retour du reçu [Fig.5]. Comme pour le diagramme d'activités, on pourra lier le diagramme de séquence aux différents éléments de travail.

Une fonctionnalité très intéressante de Visual Studio 2010 est la possibilité de créer un diagramme de séquence à partir de code existant ! Dans le cas d'une application à reprendre ne possédant pas ou peu de documentation, cette fonctionnalité sera très utile afin de se donner une idée de ce que font certaines méthodes. Pour cela, il suffit de faire un clic droit sur la méthode concernée et on peut générer automatiquement le diagramme en choisissant les paramètres voulus [Fig.6].

## Diagramme de classes

Le diagramme de classe permet de décrire les informations et les données utilisées par les utilisateurs et le système afin de répondre aux besoins exprimés. Un développeur est généralement plus habitué à ce type de diagramme car il est très proche du code. Visual Studio 2008 possédait déjà des diagrammes de classes mais ceux-ci n'étaient que des représentations visuelles des objets .Net du projet. Dans Visual Studio 2010, le diagramme de classe est un vrai diagramme de classe UML dont le contenu n'est pas lié à une implémentation particulière.

Une fois un nouveau diagramme de classe créé dans notre projet de modélisation, on va pouvoir le remplir avec des éléments classiques tel que :

- Une classe.
- Une interface.
- Une énumération.
- Un package.
- Des liens entre éléments (dépendance, héritage, ...).

Appliqué à notre exemple, on obtiendrait les éléments suivants [Fig.7] :

- Client : une classe représentant le client.
- Distributeur : une classe représentant le distributeur.
- Transaction : une classe abstraite qui sera la classe parente des classes représentant les différentes transactions (retrait, dépôt, transfert).
- Banque : une classe représentant la banque.

A partir des différents diagrammes de classe un développeur a une meilleure vision de ce qu'il faut réaliser en termes d'objet. Ces classes et interfaces peuvent bien entendu être liées, comme pour tous les autres éléments de modélisation, à des éléments de travail dans TFS. Ces éléments sont généralement liés aux tâches de développement et permettent aux développeurs de rapidement trouver les informations nécessaires pour la réalisation de ces tâches. Mieux encore, avec le « Visualization and Modeling Feature Pack » le code peut être généré automatiquement via des template T4 ! L'inverse est aussi possible, générer les diagrammes de classes à partir de code déjà écrit.

Enfin afin de se simplifier la vie, les nouvelles fonctionnalités d'extension de Visual Studio 2010 permettent d'ajouter ses propres éléments dans la boîte à outils. Cela permet, par exemple, de définir un élément représentant un design pattern couramment utilisé afin de ne pas avoir à le re-modéliser à chaque fois.

## Diagramme de composants

Une fois que l'on a une vue sur les classes et interfaces nécessaires afin de réaliser les besoins des utilisateurs, on peut commencer à le regrouper sous forme de composants. Ces composants représentent les parties centrales de l'application et les diagrammes de composants permettent de les représenter en spécifiant comment ils dépendent les uns aux autres. On obtient alors une représentation macro de la conception de l'application. Dans le cas de composants complexes, on peut créer des sous-diagrammes de composants pour décrire leurs structures internes. Le principe étant d'apporter une vision globale sur l'architecture de l'application.

Comme pour les autres diagrammes avec Visual Studio, la création se fait par simple glisser/déposer depuis la boîte à outils. Dans notre exemple cela donnerait les composants suivants [Fig.8] :

- Client : composant représente l'utilisateur.
- Distributeur : composant représentant le distributeur.
- Transaction : composant représentant une transaction de manière générale.
- Banque : composant représentant la banque.

Pour l'ajout des interfaces exposées

Fig.4

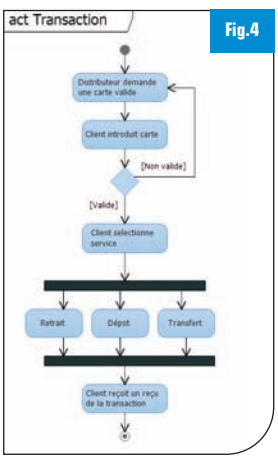
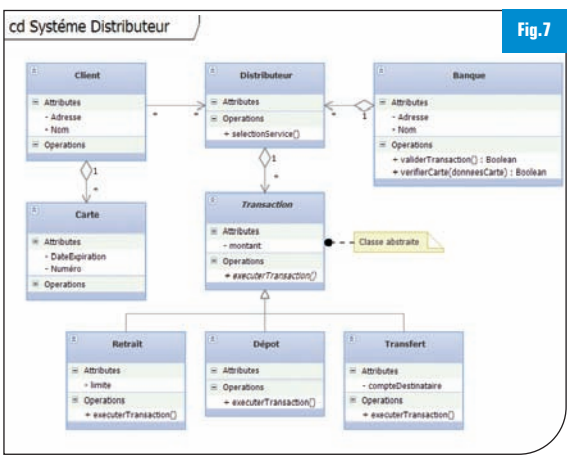



Fig.7



et attendues sur les composants, rien de plus simple, il suffit d'ouvrir l'explorateur de modèle et de glisser/déposer les interfaces qui ont été définies dans les diagrammes de classes ! Bien entendu l'inverse est aussi possible, vous pouvez définir l'interface dans le diagramme de composant et ensuite la glisser/déposer depuis l'explorateur vers un diagramme de classe.

Une fois les diagrammes de composants finalisés, on pourra vérifier que l'on n'a pas d'erreur comme des dépendances circulaires et il sera normalement simple de pouvoir regrouper ces composants en couches (IHM, service, métier et accès aux données par exemple).

## Diagramme de couches

Nous quittons l'univers de l'UML pour aborder le dernier (et non des moindres) type de diagramme : celui des couches logiques. Pour beaucoup de personnes qui ont eu la lourde tâche de faire respecter le cloisonnement de l'architecture aux équipes de développement, cette fonctionnalité vaut de l'or !

Que l'on soit architecte, urbaniste ou développeur, avec l'expérience accumulée on parvient à se rendre compte qu'un des plus gros ennemis auquel nous ayons à faire face est le couplage entre les différentes briques de notre application. L'absence de maîtrise dans ce domaine fait que nous perdons totalement le contrôle sur les évolutions ou corrections que nous allons apporter à l'application : il est impossible de déterminer les répercussions de ce que nous faisons !

Combattre ce fléau à mains nues n'est pas une simple tâche et bon nombre d'entre nous finissent par rendre les armes très rapidement, laissant l'application livrée à elle-même, ce qui garantit un résultat accablant !

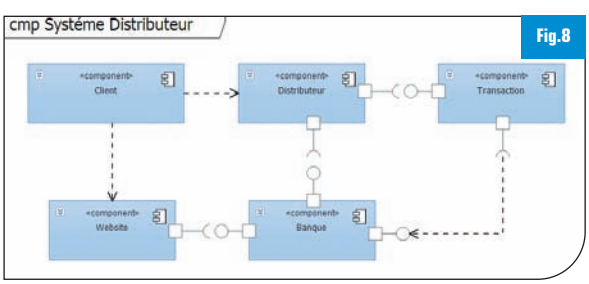


Fig.8

Le diagramme de couche apporte une solution aussi simple que radicale. Dans un premier temps on va représenter les différentes briques de son application sous forme de couches pouvant s'imbriquer puis définir les relations possibles entres chacune d'entre elles [Fig.9].

A ce stade nous avons un diagramme purement logique, nous avons alors la possibilité d'associer des éléments de notre solution aux différentes couches. Selon la granularité recherchée nous pourrons ainsi associer méthodes, classes, espaces de nom ou assemblages.

Là où ce diagramme prend tout son sens, c' est dans la phase dite de Validation. Visual Studio pourra valider pour vous que votre code est toujours conforme aux règles établies dans le diagramme ! Cette Validation peut être effectuée à la demande et lors de l'intégration continue.

Par exemple, si un nouveau développeur vient à appeler une méthode de la couche d'accès aux données à partir de l'interface graphique, la validation du diagramme générera une erreur car aucune relation n'existe entre la couche « UI » et « Accès aux données ».

Cette fonctionnalité permet de gagner un temps considérable, voire de rendre possible ce qui n'était auparavant que grossièrement faisable et de manière relativement inefficace.

## Conclusion

Utilisée à bonne escient, la modélisation apporte une réelle simplification au développement et aux développeurs. Elle permet de représenter visuellement et simplement les besoins utilisateurs et comment ceux-ci doivent être réalisés. En intégrant la modélisation UML, Visual Studio 2010 rattrape son retard vis-à-vis de la concurrence mais Microsoft a su ajouter des fonctionnalités très utiles pour apporter de la simplification là où il en manque cruellement :

- Le projet de type modélisation, intégré directement à la solution, permet aux développeurs de s'approprier les diagrammes et de ne plus voir la modélisation comme une discipline obscure et sans intérêt.
- L'intégration avec Team Foundation Server va quant à elle permettre à la fois une meilleure traçabilité entre la définition des besoins et les diagrammes UML et un accès simplifié au diagramme à partir des éléments de travail, grâce aux liens que l'on va pouvoir créer.
- Enfin, en plus de la modélisation UML, Microsoft est allé plus loin et a ajouté le diagramme de couches qui permet non seulement d'architecturer son application mais en plus de la valider vis-à-vis du code !

■ Guillaume **Rouchon**
*Architecte .Net / Expert ALM, .Net Rangers by Sogeti*
*Blog :* http://blog.qetza.net

■ Loïc **Baumann**
*Architecte .Net / Expert ALM, .Net Rangers by Sogeti*
*Blog :* http://loicbaumann.fr

■ Jason **De Oliveira**
*Solutions Architect chez Winwise, 13 ans d'expérience dans le développement logiciel.*
*Blog :* http://jasondeoliveira.com

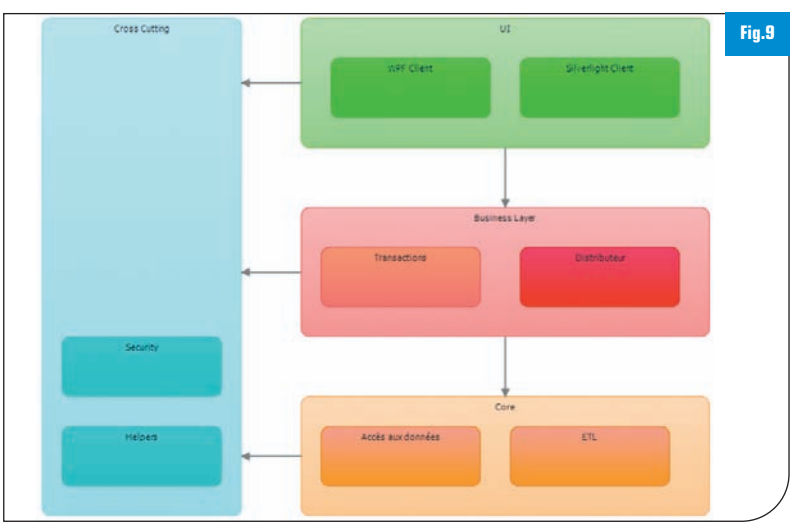


Fig.9