L’architecture 3 tiers
(pour référence)

Le présent article décrit l’architecture 3 tiers. Non pas pour en faire la promotion car, à bien des égards, on peut considérer qu’elle est dépassée. Mais parce que c’est l’une des premières architectures à s’être répandues dans l’industrie du logiciel de gestion, dans les années 1990. À ce titre, elle a servit de référence, y compris pour en faisant l’objet de critiques, quand il s’est agit de mettre au point d’autres architectures (notamment pour lui succéder).

Je la classe dans les **architectures impures** pour ce qui est de la gestion des dépendances qu’elle implique.

1. Objectif

L’objectif premier de l’architecture 3 tiers est, à défaut de doter le logiciel à développer d’une architecture optimale, de le doter d’une architecture tout court.

L’architecture 3 tiers s’est apparue dans les années 1990, à la jonction entre les logiciels dits « client / serveur » et l’apparition des premières applications web (alors rudimentaires, à base de simples formulaires HTML).

L’approche client / serveur à laquelle elle succède peut être qualifiée d’architecture à deux niveaux : le client d’un côté et le serveur de l’autre. Elle est apparue avec la mise en place des réseaux d’entreprise, permettant à plusieurs postes de travail d’être reliés et donc d’exploiter et d’alimenter des données dans une même base centralisée.

Cette architecture à deux niveaux préconise donc de réaliser d’une part un logiciel serveur, qui sera déployé sur le serveur seul et qui s’occupera de gérer les opérations centralisées, et un logiciel client d’autre part, qui sera déployé sur chacun des postes de travail des utilisateur·ices du logiciel. Mais elle ne dit rien de la façon d’organiser le code de chacune de ces deux composantes.

L’architecture 3 tiers va un peu plus loin, en ordonnant un peu plus la façon dont la partie serveur pourra être organisée.

2. Principe

L’architecture 3 tiers tire son nom du fait qu’elle organise le logiciel en **trois couches distinctes**, ayant chacune un **domaine de responsabilités** :

- Une couche **présentation**, qui a la charge de présenter les données à l’utilisateur·ice du logiciel, que ce soit à l’écran ou sous d’autres formes de sorties, comme une imprimante. En plus de ces sorties, elle doit également gérer des entrées, typiquement des saisies de données par l’utilisateur·ice sur des formulaires dans le cadre d’applications de gestion.

  Cette couche correspond à la partie client de l’approche client / serveur.

- Une couche **logique métier**, qui a la charge d'implémenter les règles de gestion, propres au(x) métier(s) ciblé(s) par l'application.

  Un apport essentiel de cette couche, à l'époque où ce type d'architectures a été proposé, a été d'inciter les développeur·euses à déporter leur code métier en dehors des écrans, c'est à dire de la couche présentation. Cela a été particulièrement important dans le domaine des clients « lourds » (sous forme d'applications natives plutôt que web).

  Les outils de programmation de ces écrans (par exemple des IDE comme Visual Basic ou Delphi) permettaient en effet d'associer directement à des éléments d'interface graphique, comme des boutons, des procédures automatiquement exécutées lors de leur activation. Nombreux étaient les développeur·euses qui avaient alors tendance à intégrer leur code, y compris métier et souvent même de persistance, dans ces procédures, ce qui pouvait donner lieu à de nombreuses occasions manquées de partager du code commun entre différents éléments de l'application.

- Une couche **persistance**, qui a la charge de récupérer les données et de les inscrire dans des supports persistants, tels que des bases de données ou éventuellement des fichiers.

Le principe clé de l'architecture 3 tiers est que les dépendances directes entre composants logiciels vont de la couche présentation vers la couche logique métier et de cette dernière vers la couche persistance.

En revanche :

- La couche présentation ne peut pas s'adresser directement à la couche persistance. Elle passe toujours par l'intermédiaire de la couche logique métier.

- Il ne saurait y avoir aucune dépendance à rebours du sens présentation → logique métier → persistance.

  Donc pas de dépendance de la couche logique métier vers la couche présentation, ni de la couche persistance vers la couche métier et évidemment encore moins de la couche persistance vers la couche présentation.

Une opération de la part de l'utilisateur·ice prendra donc typiquement la forme suivante :

1. La couche présentation prend en compte l'action effectuée par l'utilisateur·ice sur l'interface graphique.

   Cela se traduit par une demande qu'elle adresse à la couche logique métier. Cette demande peut s'accompagner de données (montantes), typiquement passées en paramètres aux opérations mises à disposition par la couche logique metier.

2. La couche logique métier prend en compte cette demande, avec les données passées en paramètres, et y applique ses règles de gestion pour élaborer sa réponse, typiquement sous forme de nouvelles données (descendantes, cette fois).

Si cette élaboration nécessite de collecter ou de pousser des données vers un support persistant, comme une base de données, alors la couche logique métier va à son tour adresser une demande à la couche persistance, pour y pousser ou collecter ces données.

3. La couche persistance va prendre en compte cette demande et retourner les données qui en résultent.

4. La couche logique métier va réceptionner ces données et les intégrer dans l'élaboration de sa réponse.

5. La couche présentation va enfin réceptionner les données retournées par la couche logique métier, les mettre en forme et les afficher à l'utilisateur·ice.

NB. - Aucune hypothèse n'est faite a priori sur les formats de communication entre ces couches. Il peut s'agir d'appels de procédures, locales ou à distance, via la technologie COM/DCOM dans le monde Microsoft, la technologie RMI dans le monde Java ou la norme aujourd'hui quasiment disparue CORBA, ou encore de requêtes HTTP…

3. Critiques

L'architecture 3 tiers a eu le mérite, du fait de sa simplicité, de populariser un modèle d'architecture dans des entreprises qui n'en avaient tout simplement pas auparavant.

Pour filer la traditionnelle métaphore de la cuisine italienne, elle a permis à des logiciels de prendre la forme d'un plat de lasagne (architecture en couche) au lieu d'un plat de spaghetti (gros désordre) et de maîtriser un temps soit peu les dépendances entre composants logiciels et ainsi éviter, ou au moins contenir, le *dependency hell* que peuvent devenir des projets logiciels d'une certaine importance.

Mais elle n'est évidemment pas parfaite et on peut lui adresser un certain nombre de critiques. Parmi lesquelles :

Des ambigüités sur les formats de communication entre couches

Considérons la notion d'entités du domaine.

C'est une notion fondamentale de la conception de base de données. Mais c'est une notion également directement issu du métier, car c'est bien du domaine métier qu'il s'agit quand on parle de « domaine ».

S'il s'agit d'entités métier, alors on imagine que leur siège doit être la couche logique métier. D'autant plus qu'elle pourront donner les classes métier qu'on utilisera avec des langages orientés objet.

Mais quand sont apparus les ORM (correspondances objet / relationnel), dont le siège est indiscutablement au niveau de la couche persistence, on a commencé à se faire des nœuds au cerveau (en supposant que l'on veuille absoluement rester dans le strict cadre d'une architecture 3 tiers), car les classes d'entités du domaine devaient désormais également servir de paramètres et de valeurs de retour des procédures proposées par ces ORM. Or, si le siège naturel des classes entités du domaine est la couche logique métier, alors celles-ci sont innaccessible à la couche persistance, car toute dépendance persistance → logique metier est prohibée.

Alors que faire ?

- Les déplacer de la couche logique métier vers la couche persistance et compter sur la dépendance logique metier → persistance pour les utiliser dans ces deux couches ?

  Mais, plus tard, avec l'avènement d'échanges en XML, puis en JSON, donc également (plus ou moins) orientés objets et potentiellement à base de sérialisations et désérialisation automatiques d'objets natifs, entre la couche présentation et la couche logique métier, la question se pose à nouveau. Car la couche présentation n'aura pas accès, pas directement, à la couche persistance siège de ces entités.

- Les dupliquer, en dotant chaque couche en ayant besoin sa propre définition des mêmes entités du domaine ?

  Cela est tout à fait possible, mais va nécessiter du travail supplémentaire :

  - Pour maintenir en double ou en triple ces différentes entités dans le cas ou des changements devaient survenir dans leur définition.

  - Pour convertir (*mapper* en bon franglais) les entités de la couche présentation en celles de la couche logique applicative, puis celles de la couche logique applicative en celles de la couche persistance, puis celles de la couche persistance en celles de la couche logique applicative à nouveau et enfin celles de la couche logique applicative en celles de la couche présentation à nouveau.

- Transgresser les règles de l'architecture 3 tiers en isolant ces entités du domaine dans un composant logiciel distinct qui sera une dépendance commune aux composants des trois couches ?

L'impureté des dépendances logique métier → persistance

Parmi les trois couches proposées par l'architectures 3 tiers, deux sont nécessairement destinées à effectuer des opérations d'entrées et de sorties :

- La couche présentation va devoir gérer les entrées de données effectuées par les utilisateur·ices et les sorties faites vers l'écran ou d'autres périphériques comme des imprimantes.

- La couche persistance va devoir gérer les entrées et sorties de données vers des supports persistants, comme des bases de données ou des fichiers.

Au mileu de ces couches, seules la couche logique métier aurait le potentiel pour contenir du code pur. C'est à dire, notamment, n'effectuant pas d'opérations d'entrées et de sorties.

Mais son utilisation de la couche persistance va se traduire par des dépendances vers des composants impurs. Les composants de la couche logique métier vont donc se retrouver eux-mêmes impurs par « contamination ».