TRAVAIL PERSONNEL

MODÈLE-VUE-CONTRÔLEUR ET MODÈLE-VUE-PRÉSENTATEUR

ALEXANDRE CLARK
MATRICULE 1803508
DÉPARTEMENT DE GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

TRAVAIL PRÉSENTÉ DANS LE CADRE DU COURS
LOG2410 : CONCEPTION LOGICIELLE

28 NOVEMBRE 2016

INTRODUCTION

> *Il y existe deux manières de concevoir un logiciel. La première, c'est de le faire si simple qu'il est évident qu'il ne présente aucun problème. La seconde, c'est de le faire si compliqué qu'il ne présente aucun problème évident. La première méthode est de loin la plus complexe* - ***C.A.R. Hoare (s.d)***

Depuis les tous débuts de l'informatique, l'intérêt principal de tout développeur est de trouver une façon de simplifier, faciliter et accélérer la conception de son produit. Une des meilleures solutions, pour ne pas dire *la meilleure*, est le *structured programming* ou la programmation structurée. Cette idée, avancée pour la première fois par les deux mathématiciens Bohm et Jacopini(1), mentionne qu'il est possible de matérialiser tout programme informatique grâce à trois structures de contrôle distinctes, la séquence, la sélection et l'itération. Bien que leur théorème fut réfuté par certains, l'idée de diviser le programme en diverses structures de contrôle est restée et a permis à plusieurs architectures de voir le jour et de simplifier grandement le développement d'application de toute sorte.

Dans le cadre du développement d'une application comportant une interface interactive sur le web, il est donc pertinent de déterminer quel architecture serait la plus appropriée pour concevoir un tel projet. Les deux architectures les plus propices à remplir la tâche sont le MVP (modèle-vue-présentateur) et le MVC (modèle-vue-contrôleur), toute deux divisant à leur façon le système de stockage des données de l'unité de logique et de l'interface utilisateur. Il sera donc question de départager les deux architectures avec leurs points positifs et négatifs pour ainsi déterminer le plus convenable.

MODÈLE-VUE-CONTRÔLEUR

L'architecture modèle-vue-contrôleur est une architecture d'interface utilisateur qui a pour but principal de séparer la logique applicative de la logique de l'interface utilisateur. L'idée derrière cela est que la logique de l'interface utilisateur change beaucoup plus souvent que sa contrepartie surtout dans les applications web. En effet, des nouvelles pages web peuvent être ajoutées ou modifiées chaque jours, il serait donc absurde de devoir modifié et redistribué toute l'application chaque fois. Ainsi, le MVC sépare une application en trois structures soit le **m**odèle, la **v**ue et le **c**ontrôleur.

Le *modèle* représente la structure gérant les données et les états de l'application. Principalement, il répond aux requêtes du contrôleur pour changer d'état ou aux requêtes de la vue pour obtenir de l'information sur son état actuel. Il communique avec la base de données pour récupérer les informations brutes puis les ordonne pour faciliter le traitement avec les autres parties de l'architecture.

La *vue* gère ce que l'utilisateur voit; lorsque l'utilisateur demandera une page ou effectuera une action la vue s'occupera de prendre l'information envoyée par le contrôleur pour savoir ce qu'elle doit afficher.

Le contrôleur est la structure qui prend les décision et l'intermédiaire dans le système. Il s'occupe de capter la souris, le clavier, bref tous les entrées possibles de l'utilisateur et va informer le modèle de changer d'état ou à la vue de changer.

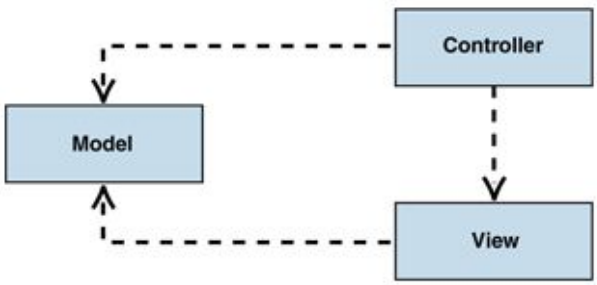

Figure 1 : Illustration des relations entre les composantes du MVC (2)

Comme le démontre la figure 1, la vue ne communique pas avec le contrôleur; celle-ci prend aucune décision et fait seulement réagir aux demandes du contrôleur qui va décider à quel moment elle doit changer. En effet, le but ultime du MVC est de rendre la vue la plus simple possible et ainsi conférer toute la tâche de prise décision et d'accès aux données au contrôleur et au modèle. De cette façon, une vue peut être interchangée beaucoup plus facilement avec une autre puisque seule la logique d'interface utilisateur est modifiée.

De surcroît, bien qu'il existe un seul et unique modèle contenant les données et les états, il peut exister plus qu'une seule vue et plus qu'un seul contrôleur. En effet, on entend plutôt parler dans ce cas d'un couple vue-contrôleur pour chaque élément dans une page et un pour l'écran en entier. Dans cette perspective, chaque contrôleur travaille à l'unisson pour voir lequel a été modifié ou reçu une entrée, puis une fois le contrôleur trouvé, c'est à lui que revient la responsabilité de gérer le modèle et la vue entière de l'application.

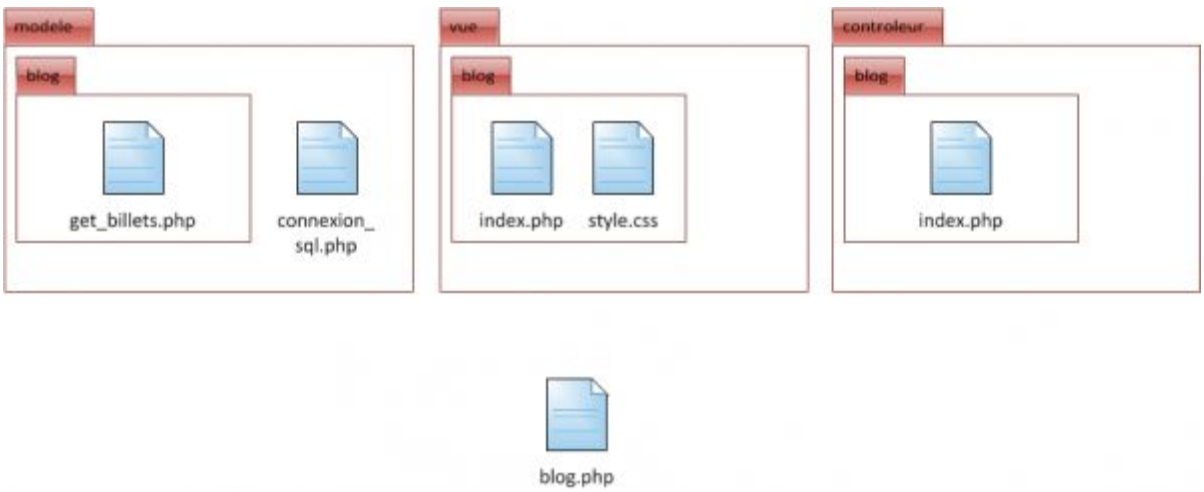

Figure 2 : Exemple d'architecture de fichiers pour un blog en MVC (3)

Bien qu'une telle architecture peut sembler complexe, posséder une telle structure apportent un nombre inquantifiable d'avantages à une application que ce soit au niveau de la conception que des performances d'une application de grande envergure.

AVANTAGES ET INCONVÉNIENTS DU MVC

Dans un premier temps, l'architecture modèle-vue-contrôleur permet de séparer clairement la logique applicative de la logique d'interface utilisateur. Non seulement, cela permet d'interchanger les vues à la disposition de l'administrateur, mais aussi permet de minimiser les erreurs de codes introduites lors de la conception. En effet, puisque l'application est séparée en noeuds ayant des responsabilités distinctes, lorsque l'on modifie l'affichage de notre application, on s'assure qu'aucunes erreurs ne sont induites malencontreusement dans la gestion des données et la logique de décision. Ainsi, le code est facile à maintenir puisqu'on minimise les erreurs et facilite les tests puisque chaque élément du MVC sont indépendants les uns des autres.

Dans un deuxième temps, une application conçue en suivant l'architecture MVC permet une réutilisation du code puisqu'étant donné que chacune des parties est divisée en fonctionnalité et que l'interface n'est pas mélangé avec les données et la prise de décision et vice versa, de grande partie du code reste les mêmes.

De l'autre côté, le MVC amène sa part de méfaits aussi. En effet, utiliser une architecture aussi complexe peut causer problème lorsque d'autres patrons de conception sont utilisés parallèlement puisque l'on ajoute plusieurs niveaux d'indirection entre une entrée et le résultat à l'écran. On augmente ainsi la complexité de notre application puisque l'on doit gérer et adapter les différents patrons de conception ensemble. De plus, la conception d'une application suivant l'architecture MVC demande un développement en parallèle de la part de plusieurs développeurs puisque chaque composante est interdépendante et doit être développé conjointement pour son bon fonctionnement. De surcroît, l'architecture MVC ne convient pas aux applications de petite ampleur et elle a plutôt un effet inverse sur les performances et la conception de l'application.

MODÈLE-VUE-PRÉSENTATEUR

L'architecture modèle-vue-présentateur est une architecture d'interface utilisateur développée par IBM dans les années 1990 s'inspirant de l'architecture modèle-vue-contrôleur. Son but principal, tout comme l'architecture dont il est inspiré, est de séparer la logique applicative de la logique de l'interface utilisateur. Cependant le MVP permet une meilleure séparation entre les données et la vue. Dans ce but, le MVC sépare une application en trois structures soit le **m**odèle, la **v**ue et le **p**résentateur.

Le *modèle* représente la structure gérant les données, les états de l'application et la logique applicative. Sa tâche est de répondre aux requêtes du présentateur et de changer d'état ou de lui envoyer de l'information dépendamment la tâche lui étant demandé. Il organise lui aussi l'information brute venant d'une base de donnée pour la rendre disponible à l'envoi au présentateur.

La *vue* s'occupe de recevoir les événements provenant d'entrée de l'utilisateur et de les rediriger vers le présentateur. De plus, la vue s'occupe de l'affichage de l'application et est le plus simple possible.

Le présentateur est la structure qui prend les décision et est le seul lien entre la vue et le modèle dans le système. Premièrement, lorsqu'un événement survient et est capté par la vue, celle-ci l'envoie au présentateur qui décide si l'événement enclenche un changement d'état dans le modèle ou un appel à une information quelconque. Deuxièmement, le présentateur reçoit l'information du modèle et détermine ce que la vue devra afficher en communiquant grâce à une interface implémentée par la vue. La communication avec le modèle se fait donc exclusivement avec le présentateur.

Dans une telle architecture, le présentateur connaît aucunement la nature de la vue. En effet, il est nécessaire d'implémenter le présentateur en fonction d'une

interface implémentée par la vue qui possède les méthodes, attributs et évènements que le présentateur a besoin. De cette façon la vue peut prendre n'importe quelle forme et l'on pourrait même interchanger diverses formes d'interfaces et le tout restera fonctionnel sans même avoir à modifier l'implémentation du présentateur.

De plus, le modèle peut aussi implémenter un interface auquel le présentateur aurait accès plutôt qu'à lui même. De cette façon, on peut interchanger les façons d'entreposer les données dans le modèle comme on le veut, le présentateur sera implémenté en fonction de l'interface et cela fonctionnera.

Étant donné que chacune des composantes interagissent l'une entre l'autre par le biais d'interface cela renforce l'encapsulation de la logique de chaque élément du MVP ce qui permet une certaine généricité dans la structure de l'application. Cette plus grande structure dans le code permet une meilleure compréhension de celui-ci, mais aussi permet une expansion future en facilitant la transition entre divers interfaces utilisateur ou différent modèle.

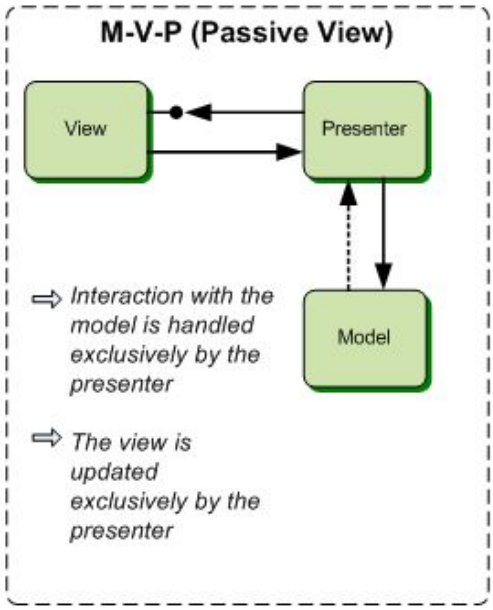

Figure 3 : Illustration des relations entre les composantes du MVP (4)

AVANTAGES ET INCONVÉNIENTS MVP

L'architecture MVP facilite grandement les tests grâce à sa séparation entre la vue et le modèle et l'encapsulation de ses objets. En effet, il est possible de tester le présentateur sans avoir à se soucier de la vue et du modèle puisqu'ils communiquent grâce à des interfaces. Il est donc possible de remplacer l'interface utilisateur et la base de donnée par des objets factices implémentés pour des tests et observer le comportement du présentateur et tester la logique entière de l'application.

L'architecture permet ainsi un très faible couplage entre le modèle et la vue, qui spécialement réduit les impacts sur la vue lorsque le modèle change. En effet, lors du développement d'une application contenant une interface interactive, la conception de la base de donnée se fait parallèlement à la conception de la vue et étant donné que la vue n'échange qu'avec le présentateur, celle-ci peut rester pratiquement intacte, ce qui réduit la quantité de code à modifier si des changements adviennent en cours de route. De l'autre côté, l'architecture MVP permet de supporter plusieurs interfaces utilisateur pour un même modèle et de bénéficier de plusieurs technologies d'accès aux données.

Par contre, tout comme l'architecture modèle-vue-contrôleur, la complexité de l'architecture MVP est un de ses défauts; non-seulement une gestion beaucoup plus grande est nécessaire si elle est couplée à d'autres patrons de conception, mais aussi elle demande des développeurs possédant beaucoup d'expérience dans le domaine pour ainsi identifier les requis avant même que la conception soit débutée. En effet, pour une application ayant très peu d'événements à gérer, l'architecture MVP se trouve être d'une très grande complexité; il est nécessaire d'implémenter des interfaces pour chaque composant et par la suite créé des objets factices pour les tests, ce qui représente beaucoup d'heures de conception pour un travail de faible envergure.

PROPOSITION

Le modèle que je recommande est l'architecture MVC. Effectivement, dans le cadre d'un projet de développement d'une application comportant une interface interactive sur le web comportant des étudiants en génie logiciel le modèle-vue-contrôleur est beaucoup plus approprié que sa contrepartie puisque les programmeurs de l'équipe ne sont pas nécessairement à maturité dans leur connaissances sur l'architecture MVP et il serait dur de prévoir les requis à l'avance. Si l'équipe choisissait une architecture MVP, nous serions enclin à devoir replanifier la structure de l'architecture sans cesse pour subvenir au besoins grandissants de l'application.

De son côté, l'architecture MVC permettrait non seulement une bonne structure pour l'application, mais aussi de faciliter son maintien tout en permettant de l'expansion future. De plus, le MVC propose trois noeuds de travail bien distinct, qui permettra ainsi une définition des tâches très facile, ce qui serait bénéfique pour une petite équipe de travail composée d'étudiants dans un cadre d'apprentissage.

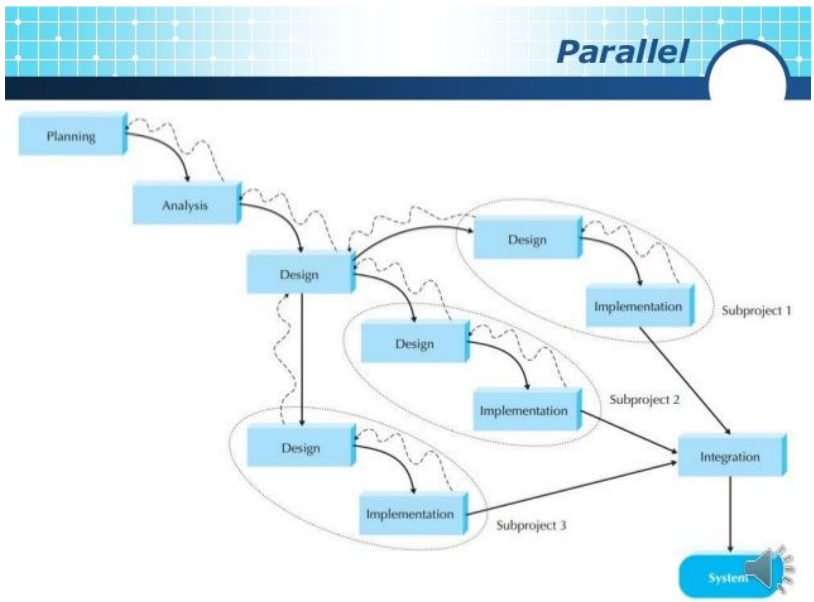

Figure 4 : Illustration du développement en parallèle d'une application

Tiré de :http://image.slidesharecdn.com/u7hathaosoftwaredevelopment-150402090622-conversion-gate01/95/u7-ha-thao-software-development-13-638.jpg?cb=1427983636

CONCLUSION

En conclusion, bien que l'architecture modèle-vue-présentateur permet une expansion future plus facile et simplifie grandement le processus pour effectuer des tests unitaires, une architecture modèle-vue-contrôleur est de loin suffisante et apporte beaucoup dans un projet de conception de longue haleine. Il serait maintenant intéressant de laisser des groupes d'élèves de la Polytechnique implémenter une telle architecture dans un projet semblable pour ainsi développer une certaine maturité de programmation qui leur permettrait d'atteindre un niveau où ils pourraient bel et bien utilisé à pleine capacité l'architecture MVP et ce sans prendre de retard dû à une mauvaise identification des requis.

BIBLIOGRAPHIE

Fowler, M. (2006). GUI Architectures. Tiré de http://martinfowler.com/eaaDev/uiArchs.html

Harrisson. N. (2008). The Model View Presenter / Passive view Benefits. Tiré de http://geekswithblogs.net/nharrison/archive/2008/09/22/125373.aspx

Microsoft. (s.d). The Model-View-Presenter (MVP) pattern. Tiré de https://msdn.microsoft.com/en-us/library/ff649571.aspx

(4) Microsoft. (s.d). The Model-View-Presenter. Tiré de https://msdn.microsoft.com/en-us/library/ff709839.aspx

(2) Microsoft. (s.d). The Model-View-Controller. Tiré de https://msdn.microsoft.com/en-us/library/ff649643.aspx

(3) Nebra, M. (2016) Concevoir votre site web avec PHP et MySQL. Tiré de https://openclassrooms.com/courses/concevez-votre-site-web-avec-php-et-mysql/organiser-son-code-selon-l-architecture-mvc

(1) Rouse.M. (2005) Structured Programming (Modular Programming). Tiré de http://searchsoftwarequality.techtarget.com/definition/structured-programming-modular-programming

Snyder, T. (2010). MVC or MVP pattern, What's the Difference. Tiré de http://www.infragistics.com/community/blogs/todd_snyder/archive/2007/10/17/mvc-or-mvp-pattern-whats-the-difference.aspx