Michel de Broux
Simon Lardinois
Victor Lecomte

Choix de conception

LSINF1225
Groupe V

Charles Momin
Valentin Rombouts
Harold Somers

Nous présentons ici les choix de conception réalisés pour ce second rapport. Nous nous concentrons sur la réalisation du schéma UML et des diagrammes de séquence, car les user stories et les cartes CRC représentent plus des phases préliminaires à la conception sur lesquelles il n'y a pas grand chose à ajouter.

1 Démarche de conception

Notre plus grande difficulté de modélisation était de trouver un juste milieu au niveau de la gestion des données, entre ces deux modèles :

- **Modèle 1 :**
  Toute l'information, y compris les données temporaires, est stockée au niveau de la base de donnée, et aucune information n'est stockée en local. L'application se charge uniquement de l'interface graphique et de l'envoi/réception de données. Toute opération dans l'application entraîne une ou plusieurs requêtes.
- **Modèle 2 :**
  Toute l'information de la base de donnée est stockée en local. Toutes les opérations de lecture et de modification se font en local. À chaque fois qu'une donnée est modifiée, une requête est envoyée pour l'actualiser sur la base de données, et dès qu'une modification par un autre utilisateur est détectée, elle est répercutée sur le modèle local. L'application se charge de tout et la base de données est en second plan pour mémoriser les données à long terme.

Le modèle 1 est celui qui demanderait le moins d'efforts parce que la base de données est déjà faite : il suffirait de rédiger des requêtes correctes pour toutes les opérations et la logique du programme serait faite.

Toutefois, ce modèle entraînerait un nombre important de requêtes pour des opérations qui pourraient être faites entièrement en local, comme la gestion du panier, qui contient les consommations que le client a l'intention de consommer mais n'a pas encore envoyées aux serveurs. Il ne permet pas non plus de mémoriser des données telles la carte qui ne sont pas susceptibles de changer régulièrement et pourraient être enregistrée pour limiter les échanges avec la base de données. De plus, nous admettons qu'il serait bien dommage de se passer totalement de modèle orienté objet dans un cours dont c'est l'énoncé principal.

Le modèle 2 demande plus d'efforts car il faut créer toute un structure orientée objet et toutes les fonctions permettant de synchroniser cette structure avec la base de données. Il a l'avantage de ne pas avoir besoin de communiquer avec la base de données pour des opérations qui peuvent se faire en local comme la gestion du panier.

Toutefois, pour d'autres, comme le décompte d'un panier dans le stock, ce modèle risque d'envoyer une requête pour chaque consommation ajoutée alors que dans le modèle 1, c'est faisable en une seule requête. En effet, ce modèle n'utilise pas les capacités de calcul efficaces de la base de donnée et ne l'utilise que pour la mémorisation.

Afin de limiter le nombre de requêtes tout en gardant une structure correcte et adaptable à n'importe quelle implémentation, il nous fallait isoler de la structure les accès à la base de données, pour s'assurer que les appels sont faits de la manière la plus efficace ; et de l'autre côté avoir une structure représentant les données de manière suffisamment complète pour pouvoir faire les

opérations les plus courantes, qui ne nécessitent pas d'actualiser la base de données, de manière tout à fait locale.

1.1 La solution : les DAO

Pour cela, nous avons utilisé le modèle des DAO (Data Access Objects) qui nous a été introduit en cours. De manière simple, il s'agit de séparer la couche de logique (déjà séparée de la couche de présentation) de la couche d'accès aux données.

Cette couche d'accès aux données se représente sous la forme d'une série de classes contenant des méthodes statiques qui fournissent des données ou modifient des valeurs dans la base de données, ou n'importe quelle structure qui pourrait être utilisée pour mémoriser les données. Leur intérêt est de donner accès aux données sans que le modèle lui-même n'ait à se soucier d'où elles viennent.

Les seules garanties sont que ces opérations donnent le bon résultat et qu'elles sont réalisées de la manière la plus efficace possible. Dans notre implémentation cela signifie que chaque fonction sera exécutée en une requête, ou du moins un nombre fixe, et que les classes DAO mémoriseront les données fixes pour qu'elles ne doivent être chargées qu'une fois même si les méthodes sont appelées plusieurs fois.

2 Schéma UML

Précisons maintenant la structure en nous appuyant sur le schéma UML fourni. Pour cause de manque de place, nous n'allons pas tout expliquer en détail mais pointer les détails plus délicats.

— Toutes les méthodes et tous les champs des classes DAO sont statiques (c'est la signification du soulignement proposé par Dia).

— Nous d'utiliser à la fois les liens d'associations et les champs, en faisant démarrer le lien au niveau du champ. En effet, cela met en évidence de manière plus claire quel objet a accès à quel objet, et cela permet de donner une indication sur la visibilité du lien.

— En général, à une classe du modèle correspond une classe DAO, qui va notamment créer les instances de la classe du modèle et modifier leurs valeurs dans la base de donnée.

— N'ont été ajoutés dans les classes du modèle que les champs qui sont nécessaire aux fonctionnalités locales de l'application. Si ils ne sont utilisé que dans la base de donnée via les classes DAO, ils ne sont pas ajoutés.

— Les types qui ne sont pas des classes du modèle, comme String, Image, Nombre, Quantité, Booléen sont des types conceptuels qui sont laissés à l'implémentation. En effet, leurs types effectifs dépendront du langage utilisé.

— La classe Utilisateur représente tout utilisateur de l'application. Elle est abstraite parce que tout utilisateur est au moins un client. Cela a du sens de séparer Utilisateur de Client pour séparer les opérations de gestion de compte des opérations liées à la commande de boissons. Il ne contient pas le mot de passe car cette information est considérée comme sensible et laissée à la base de données, et donc à DAO

— Serveur hérite de Client parce qu'il peut aussi commander des boissons en tant que client et qu'il aussi le système de panier de Client. Manager hérite de Serveur parce que le manager d'un bar peut aussi servir.

— Dans Manager, la méthode changerGrade vérifie que le grade changé n'appartient pas au manager qui l'exécute, avant d'appeler la méthode setGrade de DAOUtilisateur.

— Le panier dans la classe Client contient les consommations que le client souhaite consommer mais n'a pas encore envoyées au serveur. Il n'a pas d'existence dans la base de données. Une commande peut être ajoutée plusieurs fois, et cette structure pourra par exemple être implémentée comme un multiset, ou comme une map. Le serveur peut utiliser le panier pour ensuite ajouter les boissons à la commande d'une table avec confirmerPanierPour si le client n'a pas l'application.
— Lorsqu'une consommation est ajoutée ou enlevée du panier, le stock interne à l'application est actualisé pour que le client sache exactement ce qu'il peut commander comme boisson, du moins à condition que le stock n'ait pas été modifié par un autre utilisateur entretemps. Dans tous les cas, tout sera vérifié à la confirmation du panier.
— La classe Consommation regroupe des données qui dans la base de données n'étaient pas seulement dans la table Consommation mais aussi dans la table Description et la table Utilisation. En effet, pour la description, cela permet d'éviter de créer une classe en plus sans réelle fonctionnalité, et éviter la redondance n'est pas une priorité absolue comme elle l'est dans une base de données. Les utilisations permettent d'avoir un lien direct vers les ingrédients et ainsi de facilement actualiser les stocks.
— La carte comme le stock seront chargées au lancement de l'application dans leur entièreté. Ils sont gérés par DAOConsommation et DAOIngrédient.
— Dans DAOIngrédient, la méthode getInsuffisants renvoie une liste des ingrédients qui sont en quantité insuffisante pour décompter le panier du stock. Cette liste est vide s'il y a assez en stock.

3 Diagrammes de séquence

Quelques remarques générales :

— La classe Affichage dans les diagrammes est une classe conceptuelle que nous avons utilisé par nécessité pour illustrer des procédures liées à l'interface graphique sans faire croire qu'elles font partie de la business logic. Elle n'existera pas probablement pas en pratique, ses détails sont laissés à l'implémentation.
— Les classes DAO et la classe Affichage ne portent pas de deux-points parce que c'est la classe qui est appelée et pas un de ses objets.

Et maintenant, quelques remarques plus précises sur deux des diagrammes.

3.1 Confirmation du panier

Pour confirmer le panier, après avoir vérifié qu'il y a assez dans le stock, il faut s'assurer que le client a une commande ouverte associée. Dès lors, si commandeCourante n'existe pas ou que cette commande a été payée, il faut en créer une nouvelle, comme c'est illustré. Ensuite il faut créer les détails dans le modèle et décompter les ingrédients utilisés dans le stock.

3.2 Ajout au panier

Lorsqu'on ajoute une consommation au panier, elle est décomptée du stock local grâce à une simple boucle à travers les ingrédients qu'elle utilise. Aucune mesure n'est encore prise au niveau de la commande avant la confirmation.