TD/TP 2

Analyse de CDC et conception de modèles objets : diagrammes de classes UML

Objectifs, méthodes

L'objectif que nous nous donnons lors des 2 séances de ce TD/TP (3H en tout) est de mettre en pratique les méthodes et outils conceptuels présentés au cours 2 : en partant d'une lecture attentive d'un Cahier Des Charges on doit parvenir à un « modèle objet » de l'application, c'est à dire la description précise de toutes les classes (Noms, attributs, méthodes) et de leurs relations (associations, rôles, multiplicités, navigabilité, composition éventuelle, héritage). Cette description prendra la forme d'un diagramme de classes normalisé en notation UML. Un ou des diagrammes d'objets pourront servir d'intermédiaires d'étude entre le CDC et les cas concrets qu'il présente et le modèle général « abstrait » du diagramme de classes. C'est en particulier le cas pour les exos correspondants au fil conducteur « maillage triangulé » puisque nous avons étudié des diagrammes d'objets de ce CDC lors du TD/TP 1. Nous introduirons 2 autres sujets respectivement thématique « réseau bancaire » et thématique « flotte de véhicules ». Le fil conducteur sur le maillage reviendra régulièrement au cours des TPs successifs : rassurez-vous, nous allons bien coder une partie de ces études papiers préliminaires. Gardez précieusement les archives de tout le travail fait en amont ! Nous entrerons dans le vif du code C++ la semaine prochaine.

Les diagrammes de classes demandés sont à réaliser **selon les conventions UML normalisées** vues au cours 2. Dans le domaine de la modélisation il est fréquent que de nombreuses variantes soient acceptables : il n'y a pas forcément une réponse juste, les modèles restent ouverts à la discussion. De ce fait lors des évaluations d'exercices UML (suivi de TP, DS...) nous faisons un effort de souplesse, nous essayons de voir en quoi votre modèle objet peut être correct. Cependant **toute violation manifeste des concepts objets et/ou de leur représentation UML est sanctionnée**

Fil conducteur : Sujet maillage 2D triangulé

L'équipe de développement en charge du projet est heureuse d'apprendre que le Cahier Des Charges initial n'a pas évolué. Sans doute parce que c'est un CDC rédigé par des informaticiens qui veulent une bibliothèque utilitaire de bas niveau : les informaticiens savent spécifier des exigences informatiquement précises ! Pour l'exercice qui suit veuillez vous référer au CDC complet sur l'énoncé du TD/TP 1 : https://fercoq.bitbucket.io/cpp/tdtp/OOP_C++_tdtp1.pdf

Et pour les plus courageux : voir complément sur composition/association, 2 dernières pages de ce TD/TP 2

1. Faire le diagramme de classes pour le sujet maillage 2D triangulé

En vous aidant des analyses et des diagrammes d'objets du TD/TP1, appliquez les concepts et outils du cours 2 pour proposer un diagramme de classes du modèle objet qui vous semble le meilleur. Si vous hésitez entre plusieurs solutions qui vous semblent également correctes, dessinez ces différents diagrammes de classes en commentant vos doutes par écrit.

Sujet réseau bancaire

CDC

On souhaite réaliser un simulateur de réseau bancaire. L'application sera monoposte et mono-utilisateur, aucun aspect réseau (transmission de paquets Internet) ne sera étudié, ni aucun aspect base de données (SQL etc...). La persistance des données passera par sauvegarde/restauration d'une simulation en cours (toutes données sauvées / toutes données restaurées, voir cours 1 page 22). Les différentes opérations sur le réseau simulé se feront par la console avec un menu interactif. Ce menu interactif permettra de visualiser tout ou partie des données qui seront décrites ci-après.

L'utilisateur de l'application pourra, au cours d'une même cession, s'identifier successivement comme étant soit un employé du réseau bien précis, soit un client en particulier. Suite à quoi l'application ne devra proposer que les actions correspondantes : par exemple, une fois identifié en tant que client du réseau bancaire, l'utilisateur du simulateur ne pourra plus voir que le(s) compte(s) de ce client. Cependant à tout moment une option du menu permettra de changer d'identification : l'utilisateur du simulateur peut redevenir qui il veut, un autre client, un directeur d'agence... Dans une 2ème phase du développement il est prévu qu'un moteur d'événements vienne interagir avec le système en injectant des opérations aléatoires. L'utilisateur du simulateur pourra modifier l'horloge de la simulation et faire passer plusieurs semaines en quelques secondes réelles.

Le réseau correspond à une seule banque (comme on dirait que BNP est une seule banque). Le réseau est constitué de guichets (agences physiques) et il est destiné à la « banque de détail ». On se contentera de simuler des comptes courants pour particuliers. Un particulier pourra disposer dans ce réseau bancaire d'autant de comptes qu'il le souhaite. Chaque compte sera rattaché à une agence. Un client pourra changer un compte d'agence de rattachement mais un compte ne peut pas changer de titulaire. Naturellement il sera possible d'enregistrer de nouveaux clients, d'ouvrir un nouveau compte pour un client connu ou de clôturer un compte. Les comptes clôturés ne seront plus utilisables mais on conservera la trace des opérations qui ont été effectuées dessus pendant au moins 1 an. Un client pourra évidemment consulter le relevé d'opérations d'un compte dont il est titulaire.

Les opérations à enregistrer sont l'ouverture, les débits, crédits, retraits liquide, dépôts liquide, clôture. Pour chaque opération sur un compte il faudra enregistrer date(année/mois/jour), heure (heure/minute/seconde), montant et type (débit, crédit...). Pour simplifier on ne considère pas

le problème de débits à découvert : les opérations de débits ou retraits liquide sont supposées immédiates, soit la somme est présente sur le compte et l'opération est effectuée, soit la somme n'est pas présente et l'opération est refusée. Dans ce dernier cas l'opération n'est pas ajoutée au relevé d'opérations du compte. Seul le client titulaire d'un compte a le droit de faire des opérations de débit et retraits. Lors d'un débit il peut verser la somme débitée sur un autre compte du réseau : Mme Giroud "BNP Caen Sud" transfert 300000€ à Mme Hernandez "BNP agence du Stade Auvers"

Les employés de la banque n'ont pas le droit de faire des opérations (débiter ou créditer) sur les comptes, sauf sur leur propre compte si ils ont un compte client (en tant que client). Comme pour les sorties, un titulaire de compte déclare lui même ses rentrées d'argent (crédits ou dépôts liquides). La simulation ne tenant pas compte d'une économie réelle (l'argent vient de nul part et repart nul part) on tiendra une comptabilité séparée de toutes les sommes qui entrent et sortent du système sous forme de 2 caisses globales : totalEntrant et totalSortant. Il devra être à tout moment possible de vérifier la cohérence des flux financiers en faisant un bilan :
totalEntrant – totalSortant = Σ (soldes comptes).

Les clients seront identifiés par nom(s) et prénom(s) (séparément) adresse (en 3 champs, rue, ville, code postal) numéro de téléphone et date de naissance. Les homonymes (même nom/prénom) sont possibles, ils seront distingués par date de naissance (on supposera que 2 Paul Durand nés le même jour sont une seule et même personne). Un client a un droit de consultation et de correction sur toutes ses données (changements d'adresse, de numéro de téléphone...) sauf ses nom/prénom/date naissance que seuls des employés de son agence pourront modifier. Chaque client a un mot de passe unique pour accéder à tous ses comptes, un menu lui permet de choisir quel compte il souhaite consulter/débiter/créditer.

Chaque agence portera un nom (par exemple « agence du Stade ») et aura une adresse et un numéro de téléphone. Le staff de chaque agence sera suivi individuellement : chaque employé aura ses informations personnelles nom(s) et prénom(s), adresse, numéro de téléphone, date de naissance et dans quelle agence il travaille ainsi que son mot de passe pour faire des opérations en tant qu'employé. Un employé aura aussi un grade parmi Stagiaire / Caissier / Directeur / DG. Naturellement un employé peut aussi être un client de la banque et disposer de compte(s). Pour gérer ses comptes il se connectera en tant que client et comme un client devra donner un mot de passe, ce mot de passe client sera distinct de son mot de passe employé : tout se passe comme si Lucie Griezmann en tant qu'employée de banque était une personne différente de Lucie Griezmann en tant que cliente, mais avec même nom et même adresse (et téléphone différent bien sûr). Tous les mois chaque agence verse un salaire sur les comptes de ses employés : ceci correspond à une somme de 1€ prélevée sur chaque compte client géré (hors comptes des employés, pour lesquels les frais de gestion sont offerts) le total est divisé en 3 puis réparti équitablement à chaque échelon.

Un directeur ou un DG a le droit d'embaucher/licencier un employé de grade inférieur ou égal. Un directeur est rattaché à une agence (c'est le directeur de l'agence) et n'a le droit de recruter/licencier que dans son agence, il peut aussi opérer des mutations au départ de son agence (transférer un employé de son agence dans une autre agence). Le DG recrute/licencie/mute des directeurs au niveau national. Le DG est le seul employé qui n'est rattaché à aucune agence. Seul le DG a le pouvoir de créer/fermer/modifier une agence. Au démarrage de l'application on a juste un DG et aucune agence, aucun employé, aucun client, aucun compte.

2. Défricher les classes, les attributs, les objets ou collections d'objets

À partir de la lecture du CDC repérez ce qui vous semble devoir être des classes et leurs attributs. Organisez vos classes avec un cadre en haut indiquant le nom que vous donnerez à la classe (1ère lettre en majuscule) et un cadre en dessous avec les attributs des objets de cette classe (1ère lettre en minuscule). A ce stade vous pouvez encore douter, c'est une phase préliminaire, ajoutez des points d'interrogation à coté des éléments dont vous n'êtes pas sûr en précisant par écrit.

3. Diagramme de classes

Mettez en place le diagramme de classes en précisant les relations entre classes. Dans la mesure ou le CDC est encore un peu vague à ce sujet, vous pouvez omettre des méthodes.

Sujet flotte de véhicules

CDC incomplet

On souhaite réaliser une étude préliminaire pour la gestion d'une flotte de véhicules d'entreprise. La société « Apogeo logistics », transporteur dans toute la France et au Luxembourg des articles "sport, aventure et culture" de la SARL Culturisma et des produits coquins de la marque Climaxeo dispose d'une importante flotte de véhicules : environ 30 camions pour les livraisons en grosses quantités ou formats spécifiques (kayaks et catamarans vendus par Culturisma...), plus de 25 voitures pour les commerciaux, et une centaine de scooters (modèles 2-roues et 3-roues) principalement pour la livraison urbaine et urgente des produits Climaxeo. Les usages ne sont pas exclusifs, et certains employés ont négocié l'usage de scooters pour leur trajets quotidien depuis et vers leur domicile. Bien sûr chaque véhicule est immatriculé et référencé par l'entreprise.

Lors de la phase de contact préliminaire avec Apogeo, les commerciaux de la société SS2I Culmineo dans laquelle nous travaillons comme architecte SI ont interviewé un directeur de Apogeo. Le manager de la branche développement de Culmineo a besoin d'avoir notre avis sur le modèle objet qui découle de ces besoins pour faire une estimation de sa complexité et donc de son coût et des délais de livraison, afin de proposer un devis à Apogeo logistics.

Charles Deschamps, directeur exécutif logistique Apogeo :

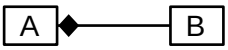
« *Nous avons tous ces véhicules et un parking avec des places à attribuer. On veut savoir quel véhicule est garé où pour que les gars de la maintenance puissent le retrouver par rapport aux signalements des utilisateurs. Les places sont numérotées, E15, B10 etc... On a des places de parking adaptées aux camions, aux voitures et aux scooters. Pour chaque véhicule on veut pouvoir dire à tout moment qui le conduit ou si il est au parking et à quelle place il est. Ce sont les employés qui renseigneront directement les infos sur la borne (console!) à l'entrée du Parking. On a un fichier avec tous les employés. Chaque employé peut avoir tout où partie des permis Voiture/Camion/Moto. Les scooters à 2-roues nécessitent un permis moto et une assurance spéciale en plus (pour le conducteur et pour le véhicule), pas les scooters à 3-roues qui peuvent se conduire avec un simple permis voiture. On a des modèles différents de véhicules mais on ne veut pas ressaisir à chaque fois les même infos (nom de modèle, consommation...) quand on a plusieurs fois un même modèle, en particulier pour les camions on veut connaître hauteur/largeur/profondeur. D'ailleurs les camions auront une info de cargaison (poids, description) quand ils ont un chargement, pas les autre véhicules. Ah oui et on veut planifier : un employé pourra réserver l'utilisation d'un véhicule pour telle date de telle heure à telle heure.* »

4. Classes et diagramme de classes

Identifiez les classes, attributs, méthodes, et mettez en place le diagramme de classes en précisant les relations entre classes. Dans la mesure ou le CDC est encore très vague à ce sujet, vous pouvez omettre des méthodes et même des attributs en mettant « etcetera » ou « … »
Précisez par écrit les infos manquantes, de quoi auriez vous besoin pour affiner le modèle ?

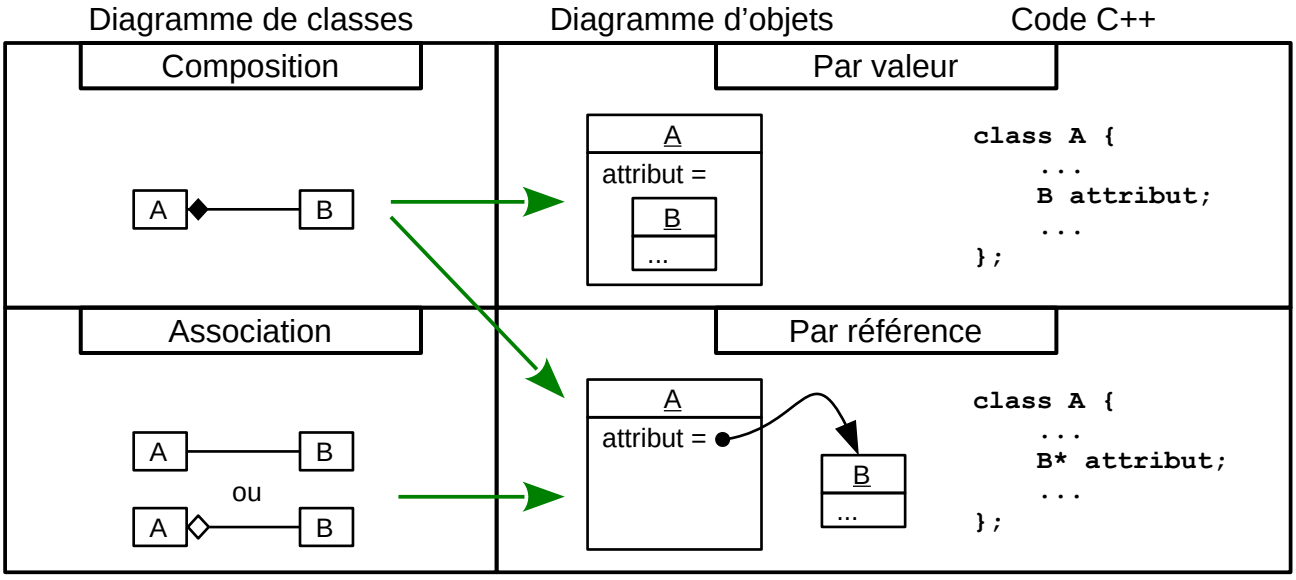
Complément d'explication sur la distinction entre valeur/référence et composition/association

Lors du TD/TP 1 nous avons vu la distinction extrêmement importante en programmation C++ entre la sémantique par valeur et la sémantique par référence. Cette distinction est importante spécifiquement en C++ parce que le C++ (comme le C#) fait la différence et que cette différence se comprend par rapport à la façon dont les données s'organisent en mémoire. Avec une sémantique par valeur les données d'un attribut de type classe se retrouvent dans le même bloc de données contiguës que les données des autres attributs de l'objet. Avec une sémantique par référence les données d'un attribut de type classe se retrouvent dans un bloc séparé, et ce que contient le bloc de données de l'objet référent c'est juste un pointeur (l'adresse) de ce bloc séparé.

Donc les données d'un attribut de type classe gérés par valeur ne peuvent pas être partagées par valeur par plusieurs objets : ils ne peuvent pas être à la fois dans un bloc et dans un autre ! Par contre les même valeurs de données peuvent être copiées facilement : les données d'un attribut de type classe sont automatiquement clonés quand on clone l'objet englobant. Ces caractéristiques font d'un attribut de type classe géré par valeur une traduction directe en C++ du sens qu'on donne à la relation de composition en modélisation objet. Un attribut par valeur de type B dans une classe A indique nécessairement une relation de composition entre A et B. 

Les données d'un attribut de type classe gérés par référence peuvent être référencés par plusieurs objets qui pointent sur ce même bloc séparé. Ils peuvent mais ce n'est pas obligé : on a le choix, on peut décider que les données pointées sont la propriété principale d'un référent et que seul lui est responsable de leur destruction, et dans ce cas on retrouve une relation de composition ! Un attribut de type classe géré par référence (concrètement : par pointeur) n'implique pas automatiquement association simple (ou agrégation) ou composition, tout dépendra de l'usage qu'on fera de ces données associées par rapport à la classe référente.

En conclusion : partant d'un diagramme de classes, une association simple (ou une agrégation) entre classes doit nécessairement se traduire par un attribut par référence (concrètement pointeur). Une composition pourra elle se traduire soit par un attribut par valeur soit par un attribut par référence. Ça veut dire qu'on peut tout faire avec des attributs par référence ! Et les langages « haut niveau » comme Java ne se gênent pas : un attribut de type classe est automatiquement géré par référence. En C++ on a le choix...



Pourquoi souhaite-t-on gérer par valeur les attributs de type classe quand le modèle indique une composition ? C'est généralement plus performant : on évite le poids non négligeable d'un pointeur, on enlève un niveau d'indirection pour accéder à la donnée et on augmente la localité des données (3 choses qui arrangent le processeur), et on permet à l'objet A d'être cloné facilement et d'emmener avec lui un clone des données de l'attribut (ce qui est le genre de copie attendue avec une composition).

Pourquoi ne pourrait-on pas systématiquement gérer par valeur les attributs de type classe quand le modèle indique une composition ? Il y a souvent des contraintes qui l'empêchent ou le découragent... Nous en reparlerons bientôt, mais en particulier les données qui sont stockées dans les « vecteurs » qui sont des sortes de tableaux améliorés du C++ et qui peuvent augmenter de taille, et bien ces vecteurs pour faire leur magie d'augmenter de taille doivent bouger leurs données en mémoire. Or si on utilise un vecteur pour stocker une collection d'objets par valeur parce que ces objets sont des composants d'une classe composite, alors ces objets composants vont changer de place en mémoire et des objets tiers qui les pointeraient verront des adresses cassées, ce qui conduit à un plantage complet du logiciel.

Scénario tiré par les cheveux et trop compliqué ? Et bien il se trouve que c'est justement ce qui va arriver avec les classes Maillage, Triangle et Sommet. Les triangles partagent les objets sommet : il est naturel de les associer et non de les composer. Un Triangle est associé à 3 Sommet. D'un autre côté un Maillage est lui littéralement composé de ses Triangles et ses Sommets, et rien dans le CDC ne laisse penser qu'il puisse les partager. Le modèle naturel qui vient est donc une composition. Mais si on implémente la collection de Sommets d'un Maillage sous forme d'un vecteur (ce qui est la démarche « normale » en C++) et que ce vecteur est un vecteur de Sommet (par valeur) et non pas un vecteur de Sommet* (par référence) alors des Sommets déjà existants et déjà reliés à des Triangles vont bouger en mémoire quand on en ajoute : ceci va casser les pointeurs des Triangles sur ces Sommets (anciens emplacements) et planter.

Finalement pour cette raison (et d'autres) on optera plutôt pour une implémentation par référence (probablement avec un vecteur de Sommet*) pour traduire en C++ la relation de composition que le modèle donne entre la classe Maillage et la classe Sommet. On fera probablement de même pour la relation de composition entre Maillage et Triangle, ne serait-ce que pour assurer un minimum de cohérence et de lisibilité.