Dossier d'analyse et de conception
Système de gestion de parking

Table des matières

I Introduction

a) Rappel du sujet

Une entreprise de gestion de parkings souterrains a besoin d'une base de données pour gérer l'occupation et le paiement de ses parkings.

Hypothèses :

- Un parking est composé de plusieurs étages, contenant des places.
- Les places du premier étage peuvent être réservées par un client suivant un abonnement mensuel ou annuel.
- Un ticket est distribué à l'entrée du parking à une voiture, et récupéré lors du paiement à la sortie du véhicule.
- Un parking est associé à plusieurs employés, responsables de la sécurité, de l'entretien et de la maintenance.

Besoins :

- Créer des parkings et leur configuration (étages, places par étage, places pouvant être réservées, etc.)
- Gérer et garder en mémoire les réservations de place et les abonnements.
- Garder en mémoire les informations concernant les tickets de paiement (date d'entrée, date de sortie, paiement en espèce ou par carte, etc.)
- Gérer la liste des employés travaillant actuellement sur les parkings, avec leur occupation.
- Connaître le taux d'utilisation d'un parking à une date précise, la somme totale gagnée par an pour un parking.

b) Analyse préliminaire du besoin

Nous regroupons ici les différentes questions / réponses du forum d'analyse fonctionnelle (Moodle).

Ces différents éléments de réponses sont à prendre en compte dans la définition puis dans la réalisation du projet, avec de satisfaire au mieux les attentes du client.

i) Mercredi 7 mars, 08h00 – « Gestion de parkings »

- Le prix d'une place de parking peut-il varier en fonction de la date ou de l'heure ? (par exemple tarif réduit la nuit ou plus cher le samedi)

-> A priori pas, mais vous pouvez considérer cette option.
- Les places du premier étage peuvent être réservées, cependant sont-elles strictement réservées aux abonnements ?

-> Pas forcément, elle peuvent être rendues disponibles si elles ne sont pas réservées.
- Le nombre de places par étage dans un même parking est-il fixe ?

-> Non, il peut varier (travaux...)
- Les réservations ont-elles lieu au même moment que l'achat des places ? (à l'entrée dans le parking), ou en dehors ?

-> Je ne suis pas sûr d'avoir compris la question, les réservations peuvent se faire via Internet.

- Quels sont tous les moyens de règlement ? carte bancaire et espèces uniquement ?

-> Vous pensez à quoi d'autre ?
- La numérotation des places doit-elle se faire d'une manière particulière ? (exemple : 1-1-001 pour la place 001 de l'étage -1 du parking n°1)

-> Vous pouvez le décider.

ii) Mardi 13 mars, 10h06 – « les employés »

- Un employé peut-il être associé à plusieurs parkings ?
- Un employé peut-il avoir plusieurs fonctions distinctes ? (ex. : responsable de l'entretien ET de la maintenance)
- Faut-il garder un historique des fonctions et des affectations des employés (qui peuvent changer au cours du temps) ? Ou seulement connaître les occupations et les affectations actuelles ?

> Oui à vos deux premières questions, l'historique des affectations ne paraît pas indispensable.

iii) Mardi 13 mars, 10h07 – « les abonnements »

- Les abonnements sont-il réservés à des personnes ? Par exemple, est-ce qu'une société ou une association peut souscrire à un abonnement ?
- Doit-on gérer le paiement des abonnements ?

> OK pour les abonnements à des structures morales, et oui vous devez les gérer.

iv) Mardi 13 mars, 10h08 – « les tickets »

- Le prix des tickets varie-t-il d'un parking à l'autre ?
- Peut-il exister différents types de tickets ? (ex. : on peut considérer des tickets donnant accès à des types de place différents)

> Oui pour les prix.
Plutôt non pour les places différentes sauf si vous pensez à quelque chose en particulier.

v) Mardi 13 mars, 10h09 – « la configuration des parkings »

- Etant donné que la configuration des parkings peut évoluer au cours du temps (ex. : travaux), faut-il enregistrer ces évolutions ? (notamment pour pouvoir donner des taux d'occupation exacts, comme cela est demandé)
- Dans le même esprit, doit-il être possible d'indiquer à l'avance quand une place de parking est disponible ou indisponible ? Doit-on pouvoir indiquer la cause d'une indisponibilité ? (ex. : travaux, réservation, etc.)
- Par ailleurs, un parking dans sa globalité peut-il être fermé ?
- Même question pour un étage.

> A priori oui à toutes vos questions, sinon ça ne peut pas fonctionner.

c) Définition du projet : cf. « Note de clarification »

➔ Cliquez-ici pour ouvrir la Note de clarification.

II Diagramme de classe / MCD

a) UML

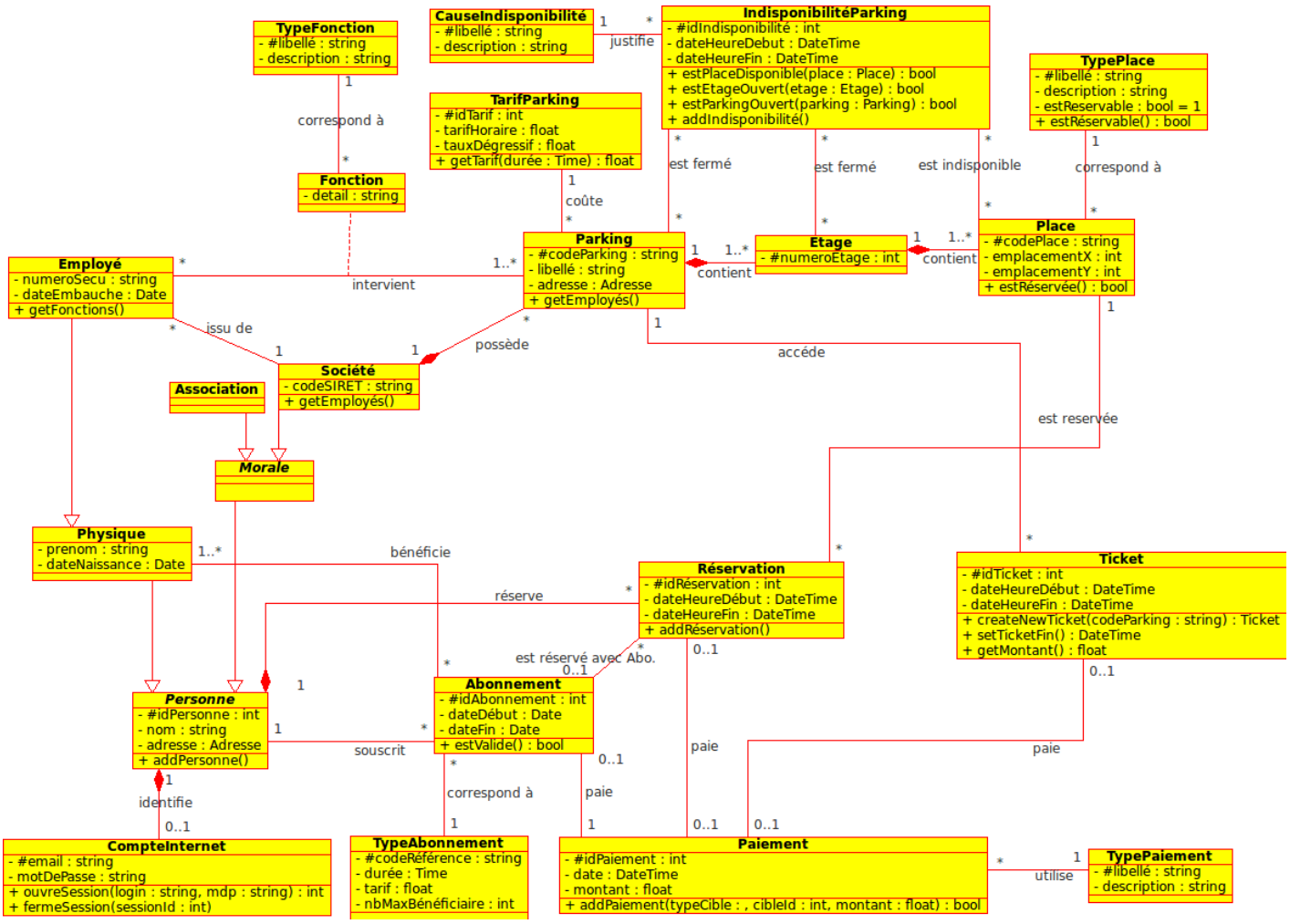

Figure 1 : Diagramme de classe

➔ Cliquez-ici pour voir le diagramme de classe en plein écran.

b) Explication du modèle et des choix de conception

i) Configuration du parking

« Un parking est composé de plusieurs étages, contenant des places. »

Nous avons choisi de rendre les parkings totalement configurables et de ne pas intégrer de contraintes spécifiques dans notre modèle. En effet, nous aurions pu intégrer directement dans le modèle certaines contraintes évoquées dans le sujet, comme la réservation possible uniquement sur le premier étage du parking (« Les places du premier étage peuvent être réservées par un client »). Nous avons voulu éviter d'ajouter ce type de

contraintes pour garder un modèle le plus évolutif possible : intégrer des contraintes spécifiques, c'est prendre le risque que le modèle devienne caduc en cas de changement des besoins et des attentes du client, ce qui est très courant et naturel dans un cas réel.

Nous avons donc choisi une solution générique, qui permet au client d'ajouter des contraintes facilement s'il le souhaite, sans avoir à refondre complètement le modèle.

En conséquence de cela, le parking est totalement configurable : à la création d'un parking, le client doit insérer dans la base de données, d'une part les données générales concernant le parking (code, libellé, adresse, etc.), d'autre part renseigner complètement sa configuration. C'est-à-dire que le client doit créer chacun des étages du parking, et chacune des places de ces différents étages. Il peut ensuite procéder à l'ajout de contrainte, à travers la classe TypePlace. Cela permet d'indiquer notamment quelles places sont réservables et quelles places ne le sont pas ; mais on peut également prévoir et ajouter facilement d'autres types de contraintes (par exemple : places réservées aux handicapés, places réservées au personnel, etc.). Cette phase de configuration peut s'avérer assez longue, mais elle n'intervient qu'à la création du parking, et permet de couvrir de nombreux cas en étant facilement évolutive.

Par ailleurs, à la création du parking, le client doit indiquer les tarifs qui sont appliqués au parking. Nous avons choisi dans le diagramme un système de tarif dégressif selon le temps passé par le client dans le parking ; mais selon l'attente du client, d'autres solutions peuvent facilement être mises en place pour calculer les tarifs du parking (on peut par exemple penser à une grille des tarifs, plus précise et plus souple). Toutefois, nous avons gardé un système simple afin de ne pas trop complexifier le modèle. Notez que les tarifs appliqués à un (ou plusieurs) parking doivent avoir un intervalle de temps de validité (ce qui a été oublié sur le diagramme), avec une date de début de validité et une date de fin de validité des tarifs. Cela permet en effet de garder l'historique des évolutions des tarifs, ce qui est demandé dans le sujet (notamment pour pouvoir calculer les statistiques d'exploitation des parkings). Le système futur devra donc prendre en compte cet aspect.

ii) Disponibilité du parking et historique

En plus de l'historique des tarifs, la création de statistiques d'exploitation exige qu'un historique des disponibilités des parkings/étages/places soit présent dans la base de données.

Pour intégrer cette fonctionnalité, nous avons choisi d'enregistrer dans la base de données les indisponibilités pour les parkings, les étages et les places. Une indisponibilité est simplement un intervalle de temps (date et heure de début d'indisponibilité et date et heure de fin d'indisponibilité), auquel nous avons rajouté une énumération CauseIndisponibilité afin de renseigner la cause d'une indisponibilité de façon normalisée (par exemple : travaux, fermeture la nuit, maintenance périodique, maintenance ponctuelle, fermeture exceptionnelle, etc.). Une indisponibilité peut être créée par le client puis ajoutée indifféremment à un ou plusieurs parkings, un ou plusieurs étages et à une ou plusieurs places. Pour déterminer si une place est disponible, il faut vérifier qu'aucune indisponibilité n'est actuellement en cours sur le parking, sur l'étage, et enfin sur la place (en absence d'indisponibilité, le parking/étage/place est par défaut disponible).

Ce système, en outre de permettre d'avoir des statistiques d'exploitation exactes (le taux de remplissage d'un parking doit notamment prendre en compte le nombre de places disponibles à un instant T), permettra au client d'avoir des statistiques supplémentaires

concernant le taux de disponibilité de ses parkings et les différentes causes sur lesquelles il peut agir pour améliorer la disponibilité.

Par ailleurs, afin de rendre ce système plus facile d'utilisation pour le client, un système de récurrence peut être envisagé aisément. Il suffirait de remplacer les 2 champs de date-heure de début et de fin d'indisponibilité par 4 champs : une heure de début et une heure de fin d'indisponibilité ; et une date de début et une date de fin de récurrence de l'indisponibilité. On peut également rajouter d'autres informations pour rendre la récurrence plus performante, comme une liste de jours de la semaine (lundi, mardi, etc.) sur lesquels s'applique la récurrence.

iii) Arborescence Personne

Notre système implique l'intervention de plusieurs types de personne (au sens abstrait du terme). Le sujet fait ressortir les occurrences « client », « employé » et « entreprise ». Nous avons donc choisi de créer une arborescence Personne, qui se divise en deux pans : les personnes physiques et les personnes morales (au sens légal). Une personne physique peut ensuite être éventuellement un employé ; alors qu'une personne morale doit être soit une association, soit une société.

Toute Personne est un client potentiel, qui pourra souscrire à un abonnement ou faire des réservations via le biais de son compte internet (que la Personne ait un abonnement ou non). L'arborescence n'est donc pas réellement utile pour le cœur du système, puisqu'une Personne abstraite a la possibilité de faire toutes les opérations majeures (souscription à un abonnement, inscription/authentification sur le site internet). Toutefois, l'arborescence est utile pour ajouter plus de détail et rendre possible certaines contraintes. Par exemple, seul un employé peut intervenir sur un parking, et cet employé ne peut être issu que d'une entre entreprise. Par ailleurs, seule une société peut posséder et exploiter des parkings (dans notre cas, il s'agira uniquement de la société cliente du système, mais on peut très bien imaginer l'existence de plusieurs filiales et donc de plusieurs sociétés légales).

iv) Information sur le personnel

L'arborescence Personne nous permet alors facilement d'enregistrer les données relatives au personnel, comme cela est demandé dans le sujet.

Un employé, qui possède certaines informations de base de plus qu'une personne physique (son numéro de sécurité sociale, sa date d'embauche, etc.), est issu d'une et d'une seule société (son employeur). Il peut alors intervenir sur un ou plusieurs parkings, avec pour chacune des interventions une fonction précise (qui peut être identique ou différente selon les interventions). Chaque fonction d'employé sur un parking est précisée par une énumération générique TypeFonction, ce qui permettra notamment de faire des recherches par type de fonction (par exemple : rechercher tous les responsables de la sécurité, ou recherche le responsable de la maintenance sur un parking donné).

v) Compte internet

Le compte internet permet d'authentifier tous les utilisateurs du site internet. La procédure d'inscription et les différentes contraintes qui pourront y être liées seront à déterminer.

vi) Abonnement

N'importe quelle Personne, c'est-à-dire une personne physique (employé ou non), une association ou une société, peut être client du système et souscrire à un abonnement.

Un abonnement possède une date de début de validité et une date de fin de validité et correspond à un type d'abonnement (catalogue des différents abonnements disponibles). Un type d'abonnement comprend au minimum les informations sur la durée de l'abonnement, son tarif et le nombre maximum de bénéficiaires. Toutefois, d'autres données et contraintes pourront être ajoutées. Notamment, on peut prévoir des abonnements réservés à certains types de Personne (abonnement spéciaux pour les sociétés, les associations ou bien pour les employés de l'entreprise ou des entreprises partenaires, etc.). Mais également, limiter et indiquer le nombre de parking auquel donne accès chaque types d'abonnement. De même, il peut être possible d'indiquer précisément les différents bénéficiaires physiques d'un abonnement (nous avons fais figuré cette contrainte à titre d'exemple). Bref, les contraintes qu'il est possible d'ajouter peuvent être très nombreuses et dépendront des attentes du client. Nous avons choisi de limiter le nombre de contraintes pour garder une bonne visibilité sur le modèle et ne pas complexifier plus que nécessaire ce travail pédagogique. Néanmoins, le modèle doit pouvoir accepter ces types de contraintes sans devoir être refondu entièrement (évolutivité).

vii) Réservation

Une réservation peut être réalisée par n'importe quelle Personne. Il s'agit d'un choix que nous avons fait, le sujet n'étant pas très précis sur ce point. On pourrait tout aussi bien estimer que les réservations sont destinées uniquement aux Personnes ayant un abonnement en cours.

Dans notre hypothèse, les réservations se font via le site internet (et donc le compte internet). Une réservation permet de réserver une place sur un intervalle de temps. Pour qu'une place puisse être réservée, il faut d'une part qu'elle ait été indiquée réservable à la configuration du parking (TypePlace) et qu'elle ne soit pas réservée sur une partie de l'intervalle de temps. Des méthodes appropriés (que nous avons placé sur les classes TypePlace et Place, mais qui pourraient par exemple être regroupées sur la classe Place) permettront de vérifier aisément ces conditions.

Par ailleurs, une réservation peut se faire avec un abonnement (en considérant que l'abonnement inclus ce service, ce qui n'est pas forcément le cas). Dans ce cas, notre modèle doit indiquer que la réservation a été faite en utilisant l'abonnement, et qu'il n'y aura donc pas de paiement directement relatif à la réservation (d'où la cardinalité 0..1 de la relation paie vers Paiement).

viii) Accès au parking et tickets

Le système futur doit posséder une interface de programmation (on peut penser à ces webs services) permettant aux automates du parking d'enregistrer les informations liées aux clients ponctuels (non-abonnés, sans réservation). Les automates devront pouvoir créer un nouveau ticket à l'entrée d'un utilisateur ponctuel ; puis indiquer la sortie de l'utilisateur et récupérer le montant à facturer ; enfin, l'automate pourra spécifier que l'utilisateur a réglé le paiement. Pour spécifier ces comportements, nous avons fait figurer sur le diagramme les méthodes respectives createNewTichet(), setTicketFin(), getMontant() dans la classe Ticket et addPaiement() dans la classe Paiement.

Cette partie du système sera particulièrement soumise à des contraintes de performance. C'est-à-dire que le système devra pouvoir répondre à l'automate dans une durée appropriée. Ceci est évident, mais il est important de le souligner pour le prendre en compte dans le choix de la solution technique : les clients ne doivent pas par exemple

patienter 2 min à la sortie du parking avant de pouvoir régler leur paiement et sortir (la durée sera plutôt de l'ordre de la milliseconde ou de la seconde).

ix) Paiement / facturation des services

Le système doit permettre la facturation des différents services et enregistrer l'historique des paiements.

Un paiement comprend des informations de base (montant, date) et indique le type de paiement utilisé (carte bancaire, espèce, Moneo, PayPal, etc.). De plus, il doit être possible de savoir à quel service correspond un paiement ; un paiement ne pouvant correspondre qu'à un seul est unique service. Il y a donc des contraintes XOR (que nous n'avons pas réussi à faire figurer sur le diagramme) entre les 3 relations « paie », respectivement pour Abonnement, Réservation et Ticket.

Il serait également envisageable et intéressant pour les statistiques d'indiquer en plus du type du paiement, sa source ; c'est-à-dire savoir si le paiement provient par exemple d'un paiement par internet, d'un automate ou d'un paiement en guichet, etc.

x) Réflexion sur l'opportunité de créer une classe IntervalleDeTemps

Dans de nombreuses classes de notre diagramme, nous utilisons des intervalles de temps (notamment dans les classes IndisponibilitéParking, Réservation, TarifParking, Ticket). Etant donné que cette information est identique quelque soit son utilisation, il est légitime de se poser la question de l'opportunité de la créer une classe IntervalleDeTemps pour enregistrer ce type d'information.

Nous pensons qu'il s'agirait d'une bonne idée (même si nous ne l'avons pas indiqué sur le diagramme pour ne pas le rentre trop fouillis) et d'une bonne pratique de conception orientée objet. Cela permettrait notamment de factoriser l'information d'intervalles de temps complexes, notamment comme on a pu l'évoquer dans la partie « Disponibilité parking et historique », avec des systèmes de récurrence plus ou moins évolué. Par ailleurs, on pourrait imaginer différents types d'intervalle de temps, plus ou moins complexes et complets, à utiliser selon la situation. L'intervalle de temps de base comprendrait uniquement une date / heure de début et une date / heure de fin ; puis on pourrait imaginer héritant de cette classe de base des intervalles de temps plus complets, comprenant des fonctionnalités de récurrence. IntervalleDeTemps deviendrait alors un type/classe modulaire et réutilisable dans différents contextes, grâce à des sous-types/classes filles permettant de choisir un comportement particulier, plus ou moins spécifique.

III Schéma relationnel / MLD

a) Le modèle relationnel textuel

Configuration parking :

Parking (#codeParking : string, libellé : string, adresse : Adresse, coûte =>TarifParking)

Etage (#numeroEtage : integer, #estContenu =>Parking)

Place (#codePlace : string, #estContenuEtage =>Etage , #estContenuEtage =>Etage, emplacementX : integer, emplacementY : integer, correspondA =>TypePlace)
TypePlace (#libellé : string, description : string, estReservable : bool)
TarifParking (#idTarif : integer, tarifHoraire : float, tauxDégressif : float)

Disponibilité parking :

IndisponibilitéParking (#idIndisponibilité : integer, dateHeureDebut : DateTime, dateHeureFin : DateTime, estJustifié =>CauseIndisponibilité)
CauseIndisponibilité (#libellé : string, description : string)
EstFerméParking (#idferméParking => Parking, #idIndisponibilité => IndisponibilitéParking)
EstFerméEtage (#estFerméEtage => Etage, #duParking => Etage, #idIndisponibilité => IndisponibilitéParking)
EstIndisponiblePlace (#estFerméPlace => Place, #deLEtage => Place, #duParking => Place , #idIndisponibilité => IndisponibilitéParking)

Arborescence Personne :

Personne (#idPersonne : integer, nom : string, adresse : Adresse)
Physique (#idPersonne => Personne, prenom : string, dateNaissance : Date)
vPhysique = jointure (Personne, Physique, idPersonne = idPersonne)
Association (#idPersonne => Personne)
vAssociation = jointure (Personne, Association, idPersonne = idPersonne)
Société (#idPersonne => Personne, codeSIRET : string)
vSociété = jointure (Personne, Société, idPersonne = idPersonne)

Fonction des employés :

Employé (#idPersonne => Personne, numeroSecu : string, dateEmbauche : Date, issuDe => Société)
vEmployé = jointure (Physique, Employé, idPersonne = idPersonne)
FonctionEmployé (#intervient => Employé, #surParking => Parking, #enTantQue => TypeFonction, detail : string)

TypeFonctionEmployé (#libellé : string, description : string)

Compte internet :

CompteInternet (#email : string, motDePasse : string, identifie => Personne)

Abonnement :

Abonnement (#idAbonnement : integer, dateDébut : Date, dateFin : Date, correspondA => TypeAbonnement, souscrit => Personne)
TypeAbonnement (#codeRéférence : string, durée : Time, tarif : float, nbBénéficiairesMax : integer)
AbonnementBénéficie (#idPersonne => Physique, #idAbonnement => Abonnement)

Réservation :

Réservation (#idRéservation : integer, dateHeureDébut : DateTime, dateHeureFin : DateTime, estRéservéePlace => Place, estRéservéeDansEtage => Place, estRéservéeDansParking => Place, réserve => Personne, estRéservéAvec => Abonnement)

Ticket :

Ticket (#idTicket : integer, dateHeureDébut : DateTime, dateHeureFin : DateTime, accéde => Parking)

Paiement :

Paiement (#idPaiement : integer, date : DateTime, montant : float, utilise => TypePaiement, abonnementEstPayé => Abonnement, réservationEstPayée => Réservation, ticketEstPayé => Ticket) ; XOR (Abonnement, Réservation, Ticket)
TypePaiement (#libellé : string, description : string)

b) Explication du schéma et des choix de transformation

i) Choix des clés primaires

Les clés primaires ont été choisies en priorité de code référence venant (supposément) de l'existant, comme le code d'un parking ou d'une place, le code de référence d'un type d'abonnement dans le catalogue, etc.

Dans les énumérations, nous avons choisi d'utiliser le libellé (supposé unique) comme clé primaire.

Ensuite, nous avons essayé d'utiliser les autres informations contenant dans les classes pour former des clés primaires, lorsque cela était raisonnablement possible. Par exemple, dans CompteInternet, nous avons choisi d'utiliser l'email comme clé primaire. Toutefois, nous avons choisi de ne pas utiliser certaines informations qui, bien qu'uniques, n'étaient pas adaptés à la formation de clés primaires convenablement utilisables. C'est notamment le cas des champs dateDébut ou dateHeureDébut que nous aurions pu utiliser (éventuellement en combinaison avec des clés étrangères) pour former des clés primaires, mais que nous avons préférer ne pas utiliser car il s'agit de champs peu adaptés à ce genre d'utilisation.

Dans tous les autres cas, nous avons inséré des clés artificielles, c'est-à-dire ne provenant ni de l'existant ni des besoins exprimés du client. Ces clés primaires artificielles ont l'avantage d'être simples à mettre en place et à utiliser (champ interger).

Enfin, dès que cela était possible et avait un sens vis-à-vis du modèle, nous avons utilisé des clés primaires composées de clés étrangères. Il s'agit du cas des classes d'association ou de certaines classes en relation forte (composition), notamment Place, qui récupére la clé primaire d'Etage et Etage qui récupère la clé primaire de Parking pour composer une clé primaire.

ii) Choix de transformation de la relation d'héritage Personne

La seule réelle difficulté de la transformation du diagramme de classe en modèle relationnel a été de choisir quel type d'héritage relationnel utiliser pour l'arborescence Personne.

Dans notre cas, nous avons dans notre modèle UML un héritage exclusif (une Société ou une Association ne peut pas être une personne Physique) à plusieurs niveaux, dont la classe mère principale (Personne) est abstraite et dont l'une des sous-classes (Morale) est abstraite. Dans un cas tel que celui-ci, le polycopié de cours recommande fortement d'utiliser héritage relationnel par les classes filles, en créant uniquement les classes filles concrètes.

Cependant, cette méthode d'héritage relationnel n'est pas convenable dans notre situation, qui s'avère plus complexe. En effet, il est primordial dans notre modèle que la classe mère Personne offre un identifiant et surtout une interface commune (à toutes les classes filles) pour les classes clientes CompteInternet, Abonnement et Réservation. En effet, s'il on ne garde pas un point d'entrée commun (Personne, avec un identifiant unique commun à toutes les sous-classes et des informations communes), la communication avec les classes clientes se complexifie de manière très importante. Ainsi, au lieu d'avoir seulement besoin d'un identifiant de Personne, les classes clientes devront également savoir à quel sous-type de Personne s'adresser. Or, ce n'est pas ce que spécifie notre modèle, et ce n'est pas non plus un comportement souhaitable.

Nous avons donc choisi d'utiliser la méthode d'héritage relationnel par référence, qui permet de conserver l'interface de la classe mère Personne, puis de créer les classes filles à l'aide d'une référence sur la classe mère et d'une vue.

Cette méthode toutefois ne permet pas d'avoir dans le modèle relationnel toutes les contraintes du modèle de classe. Notamment, rien n'indique dans le modèle relationnel que la classe Personne est une classe abstraite qui ne pourra pas être 'instanciée' seule (seuls les sous-classes Physique, Employé, Association et Société pourront être instanciées). Par ailleurs, la relation d'exclusivité entre les différentes branches de l'arborescence n'est pas présente dans le modèle relationnel. Il faudra donc veiller à assurer ces contraintes soit par des contraintes logiques si cela est possible, soit en faisant les vérifications nécessaires dans la couche métier via le langage procédural.

IV Normalisation

a) Fermeture transitive

Parking :
codeParking → libellé
codeParking → adresse
codeParking → idTarif(TarifParking)

Etage :
numeroEtage, codeParking(Parking)

Place :
codePlace, numeroEtage(Etage), codeParking(Parking) → emplacementX
codePlace, numeroEtage(Etage), codeParking(Parking) → emplacement
codePlace, numeroEtage(Etage), codeParking(Parking) → libellé(TypePlace)

TypePlace :
libellé → description
libellé → estReservable

TarifParking :
idTarif → tarifHoraire
idTarif → tauxDégressif

IndisponibilitéParking :
idIndisponibilité → dateHeureDebut
idIndisponibilité → dateHeureFin
idIndisponibilité → libellé(CauseIndisponibilité)

CauseIndisponibilité :
libellé → description

EstFerméParking :
codeParking(Parking), idIndisponibilité(IndisponibilitéParking)

EstFerméEtage :
numeroEtage(Etage), codeParking(Parking),
idIndisponibilité(IndisponibilitéParking)

EstIndisponiblePlace :
codePlace(Place), numeroEtage(Etage), codeParking(Parking),
idIndisponibilité(IndisponibilitéParking)

Personne :
idPersonne → string
idPersonne → adresse

Physique :
idPersonne(Personne) → prenom
idPersonne(Personne) → dateNaissance

Association :
idPersonne(Personne)

Société :
idPersonne(Personne) → codeSIRET

Employé :
idPersonne(Physique) → numeroSecu
idPersonne(Physique) → dateEmbauche
idPersonne(Physique) → idPersonne(Société)

FonctionEmployé :
idPersonne(Employé), codeParking(Parking),
libellé(TypeFonctionEmployé) → detail

TypeFonctionEmployé :
libellé → string

CompteInternet :

email → motDePasse
email → idPersonne(Personne)

Abonnement :
idAbonnement → dateDébut
idAbonnement → dateFin
idAbonnement → codeRéférence(TypeAbonnement)
idAbonnement → idPersonne(Personne)

TypeAbonnement :
codeRéférence → durée
codeRéférence → tariff
codeRéférence → nbBénéficiairesMax

AbonnementBénéficie :
idPersonne(Physique), idAbonnement(Abonnement)

Réservation :
idRéservation → dateHeureDébut
idRéservation → dateHeureFin
idRéservation → codePlace(Place)
idRéservation → numeroEtage(Place)
idRéservation → codeParking(Place)
idRéservation → idPersonne(Personne)
idRéservation → idAbonnement(Abonnement)

Ticket :
idTicket → dateHeureDébut
idTicket → dateHeureFin
idTicket → codeParking(Parking)

Paiement :
idPaiement → date
idPaiement → montant
idPaiement → libellé(TypePaiement)
idPaiement → idAbonnement(Abonnement)
idPaiement → idRéservation(Réservation)
idPaiement → idTicket(Ticket)

TypePaiement :
Libellé → description

b) Preuve pour 1NF

Le schéma relationnel est en 1ère forme normale car toutes les relations possèdent au moins une clé et ont tous leurs attributs atomiques

c) Preuve pour 2NF

Le schéma relationnel est en 2nd forme normale car toutes les relations sont en 1NF et tous leurs attributs qui ne sont pas dans une clé ne dépendent pas d'une partie seulement d'une clé, mais de la clé entière.

d) Preuve pour 3NF

Le schéma relationnel est en 3ème forme normale car toutes les relations sont en 2NF et tous leurs attributs qui ne sont pas dans une clé ne dépendent pas d'autres attributs n'appartenant pas à une clé.

V Conclusion

Nous avons volontairement choisi de prendre un grand nombre de cas de figure en compte dans cette phase de conception, afin de réaliser un modèle le plus évolutif et le plus modulaire possible. Cela a rendu ce travail de compréhension et de spécification du besoin très intéressant.

Toutefois, étant donné que le temps dédié à la seconde partie du projet reste assez limité, il faudra lors de la phase de réalisation que l'équipe pose des limites à l'étendue du travail et choisisse quels éléments de la conception devront être implémentés, partiellement ou complètement, dans la solution logicielle à produire

Nous souhaitons bon courage à l'équipe réalisatrice et espérons que ce document aura répondu à son principal objectif : expliquer et clarifier le problème posé afin de rendre le travail de développement le moins problématique possible et la solution finale la plus élégante et appropriée possible.