Perion Maxence

Pinon Alexandre *L3 informatique, groupe BG, TP 1*

# Rapport pour le premier jalon du projet de CDAA

*Introduction : Le but de ce projet est de produire un logiciel de gestion de relations clients prenant en compte des interactions et des taches à faire, extraites de ces interactions. Pour ce premier jalon, nous avons créé les classes modélisant les objets et allons expliquer dans ce rapport comment nous avons conçu ces objets et surtout les relations entre eux.*

## I. Organisation générale des classes

### a. Les objets

Pour modéliser nos objets, nous avons créés quelques classes : Contact représentant la fiche d'un client, Interaction et Todo représentant une tâche. Ces objets ayant besoin d'interagir avec le temps, nous avons également eu besoin d'une classe Date. Pour finir, la classe GestionContact a pour but d'agir comme une interface entre la multitude de contacts et l'utilisateur afin d'en faciliter la gestion (création, suppression, date de dernière suppression, …).

### b. Héritage

Nous avons décidé de ne pas mettre en place de hiérarchie d'héritage car cela ne nous aurait pas été utile : la classe GestionContact nous permet de gérer l'ensemble des contacts, sans avoir besoin d'inclure les caractéristiques d'un contact et une liste suffit à nos besoins.
Il en est de même pour la class Contact qui prend en compte une liste d'interaction.

Nous nous sommes posé la question d'un éventuel héritage d'Interaction et Todo car ces classes se ressemblaient beaucoup en termes d'attributs : une description et une date. Cependant, nous avons jugé que ce n'était pas la relation la mieux adaptée, puisqu'un Todo n'est pas « une sorte de » Interaction, mais bien un objet créé à partir d'une interaction. Inversement, une Interaction n'est pas « une sorte de » Todo puisqu'une Interaction peut contenir plusieurs Todo.

## II. Les relations entre les objets

### a. Interaction et Todo

Un todo est une tâche à faire trouvée dans une interaction selon les tags. Nous avons donc choisi de les générer directement depuis l'instance d'interaction en question, dans le constructeur ou depuis le setter du contenu de l'interaction (si jamais on souhaite modifier l'interaction, les todos sont donc regénérés automatiquement).

b. Les interactions des contacts

Pour la class Contact, nous avons une liste nommé interactions et chaque élément pointe vers une instance d'Interaction. C'est-à-dire que les éléments de la liste sont des pointeurs sur Interaction. Les pointeurs sont utilisés car nous avons voulu laisser la possibilité à plusieurs contacts de pointer sur une même interaction, dans le cas où plusieurs contacts auraient généré une même interaction (exemple : Une usine utilise le logiciel et veut ajouter une interaction du type « réunion avec les fournisseurs, représentés par les contacts A, B et C »).

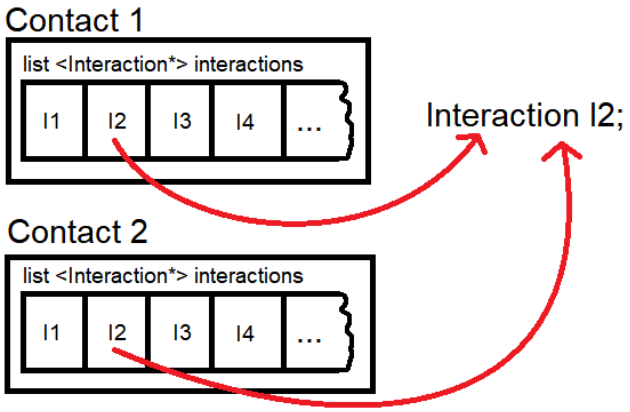


Schéma de plusieurs contacts ayant une même interaction

Cette configuration peut certes être pratique mais apporte un problème supplémentaire : il faut faire attention quand on veut supprimer en mémoire les contacts. En effet, le destructeur de Contact se charge de détruire toutes les instances d'interaction pointées par la liste d'interactions du contact. Or, une fois l'exécution terminée on détruit l'instance de GestionContact (et donc par extension toute la liste des contacts). Mais dans le cas où plusieurs contacts pointent vers la même instance d'interaction, on pourrait tenter de supprimer une interaction déjà supprimée.

Pour détruire correctement les données en mémoire, le destructeur de la classe GestionContact est un peu particulier : au lieu de supprimer aveuglement par rapport à l'ensemble des contacts les interactions dans le destructeur de Contact, il nous est utile de remonter au destructeur de la classe GestionContact car il permet d'avoir une vue d'ensemble de toutes les interactions de tous les contacts et il va pouvoir récupérer toutes les interactions de tous les contacts, supprimer tous les doublons et détruire toutes les instances d'interaction, puis tous les contacts de la liste.

Pour finir, nous pouvons indirectement accéder aux todos d'un contact par le biais de ses interactions (qui pointent vers leurs todos). Pour plus de facilité, nous avons donc ajouté un accesseur virtuel à Contact qui sort la liste de tous les todos de l'instance de Contact. Pour cela, nous avons juste à récupérer et à indiquer dans une liste les todos de chaque interaction.

c. Date

La date est utilisée par tous les autres objets décrits ici (date de création d'un contact ou d'une interaction, date d'échéance d'un Todo, …) et elle est donc encapsulée dans sa propre classe. Elle contient notamment des méthodes utilitaires permettant par exemple d'ajouter un délai à l'instance de date (pour reculer la date à faire d'un todo par exemple) ou des surcharges d'opérateurs tel que l'opérateur < de comparaison, permettant de savoir quelle date est la plus « petite » parmi 2, dans le sens où elle précède l'autre date ; nous permettant par la suite de comparer 2 todos ou 2 interactions, en surchargeant également leur opérateur < et en renvoyant le résultat de la comparaison de leurs dates respectives.

d. Diagramme de classe

Pour résumer les différents objets et leurs interactions, voici le diagramme des classes du projet.

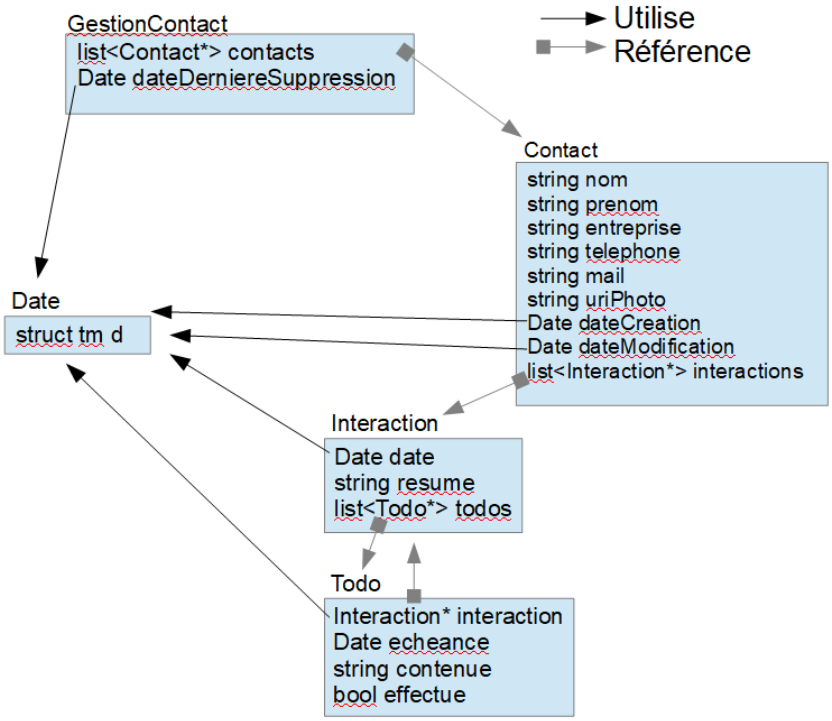


Diagramme des classes

*Conclusion : Nous avons donc créés différentes classes qui représenteront les objets à manipuler pour la suite du projet. Ces objets n'étant pas seuls dans leur monde, ils interagissent entre eux via des relations d'utilisation ou de référencement. Ces deux caractéristiques peuvent être résumés par le diagramme des classes présenté précédemment.*

*Nous allons désormais pouvoir continuer avec l'interface graphique, qui aura pour but de permettre à l'utilisateur de manipuler ces objets.*