Devoir de conception UML

Antonin CELESTIN
Adam FACI

14 novembre 2016

Table des matières

1 Introduction

Suite au projet de création d'un cahier des charges dans le cadre du cours de MOCI avec l'équipe MOCIcabing et son super nom d'équipe, il nous a été demandé d'effectuer un travail plus précis, par groupes de 2, sur les diagrammes de séquences de trois cas : saisie d'une fiche de renseignements, prise de décision concernant une fiche de renseignements et initialisation d'une nouvelle année.

Il est exigé que ces diagrammes soient assez précis afin que les fonctionnalités correspondantes puissent être directement implantées. Aussi, il fallait parallèlement fournir le diagramme de classe et le diagramme d'état.

Dans ce document, nous expliciterons nos choix de conceptions et tenterons d'éclaircir tout point pouvant poser souci. Il nous est paru important de travailler en anglais afin de s'habituer à son utilisation dans un cadre professionnel et dans un souci d'internationalisation de notre application.

2 Diagramme de séquence

Ici nous donnerons les choix de conception inhérents à chaque diagramme juste avant ceux-ci.

Dans notre application, nous avons choisi de disposer de deux bases de données que sont **Student_DB** et **Main_DB**. La première stocke toutes les informations liées aux étudiants de l'année en cours alors que la seconde s'occupe de stocker les données des étudiants des autres années (à des fins statistiques) ainsi que les informations de l'administration. Ce choix a été motivé par le fait que les informations stockées dans la **Student_DB** seront sollicitées (accédées, modifiées) très régulièrement au cours de l'année alors que celles de la **Main_DB** ne le seront qu'une fois par an lors de l'initialisation, et exceptionnellement lorsqu'un membre de l'administration change de poste ou quitte l'établissement par exemple.

Aussi, il a été décidé que toute la sécurité de l'application au niveau de l'accès aux et la modification des données, ne seront pas gérées par des méthodes. En réalité, ce sera l'instance de la classe de l'utilisateur qui définira les droits. Selon la classe, l'utilisateur aura accès à différentes méthodes : c'est ainsi l'architecture de l'application elle-même qui définit sa propre sécurité. Par exemple, le classe **Supervisor** aura accès à la méthode **openValidationInterface(studentID)** qui lui permet d'ouvrir le menu de validation d'une fiche de renseignements, menu auquel un étudiant n'aura donc pas accès et ne pourra donc pas valider une fiche. Aussi, en suivant cet exemple, pour permettre de gérer les droits d'accès entre responsables, qui n'ont pas tous le droit de valider la fiche de suivi de tous les élèves (le directeur le peut par exemple, mais en général un responsable ne peut valider qu'un élève dont il est le référent), eh bien l'application ne leur proposera que les élèves dont ils ont la charge dans le menu correspondant (via la méthode **displayValidationInterface**).

De plus, si l'on suit les diagrammes, l'on peut remarquer qu'aucune méthode ne récupère apparemment de valeur (il n'y a pas de **infosField = loadIS(studentID)** dans le premier diagramme par exemple). Ce sont en effet des requêtes qui attendent une requête en réponse leur fournissant ainsi l'information demandée. Cela permet de grandement réduire le couplage, permettant par exemple de modifier le comportement de requêtes de retour sans avoir à prendre en compte les requêtes d'arrivée.

Nous ne l'avons pas géré dans ces diagrammes, mais en cas d'erreur sur la chaîne de requêtes, l'application affiche un message d'erreur à l'utilisateur

et l'invite à réitérer sa requête ou bien à contacter l'administrateur système si le problème persiste.

Par ailleurs, afin d'être clair sur notre conception, il faut préciser que chaque utilisateur via sa classe interagira avec l'application par l'implantation de **Interface** correspondant à sa classe, elle même interagira avec la classe correspondant au cas d'utilisation et elle même avec les bases de données. Les classes implantant **Interface** sont un intermédiaire entre les utilisateurs et l'application en elle-même. Les classes finissant par "Manager" comme **InformationSheetManager** par exemple sont l'intermédiaire entre les "interfaces" qui envoient les requêtes utilisateurs et les bases de données qui renvoient les données demandées ou modifient leurs données en conséquence.

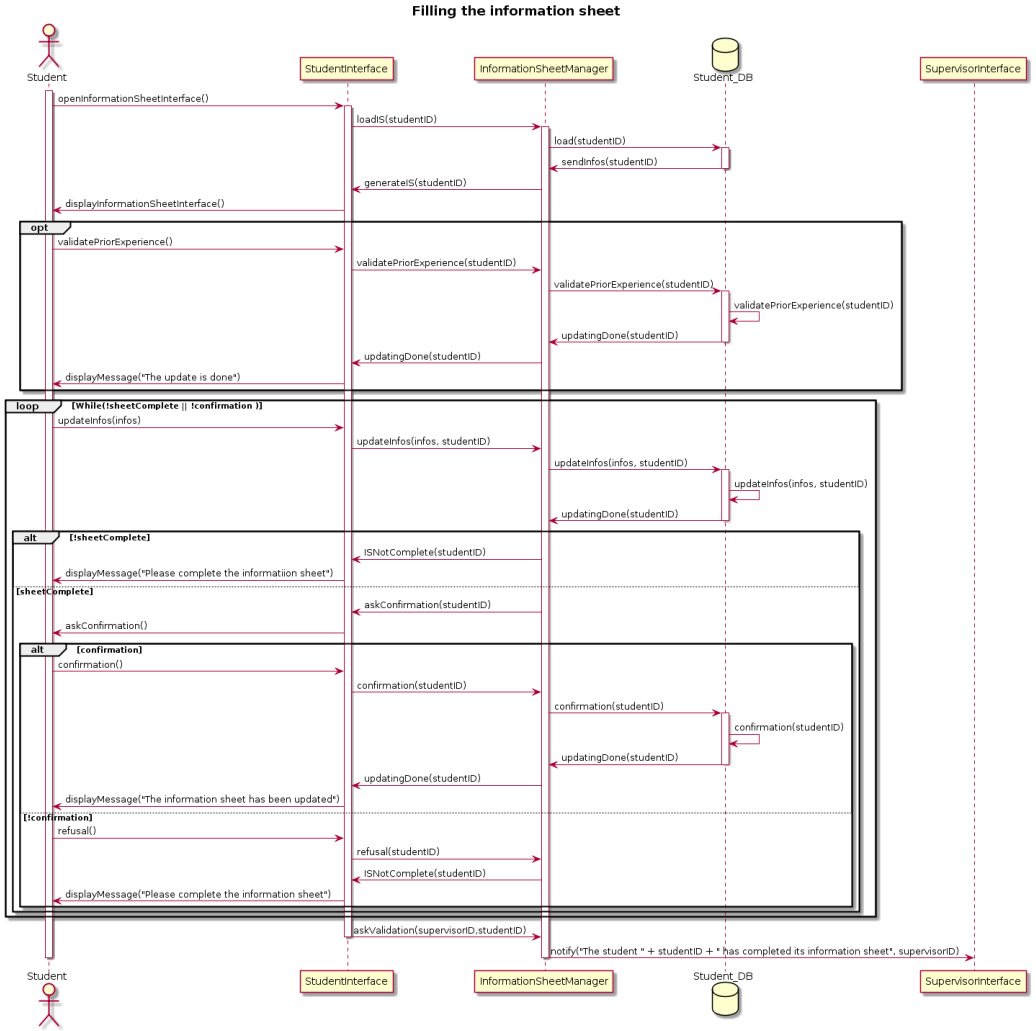
Par exemple, le directeur sera représenté par la classe **Director** qui hérite de **Supervisor**. **Director** enverra ses requêtes à **DirectorInterface** (héritant de **SupervisorInterface** qui implémente **Interface**) qui , selon le cas d'utilisation, interagira avec **InformationSheetManager** ou une autre classe. Cette dernière enverra sa requête à la base de donnée et attendra une requête de retour pour répondre à **DirectorInterface** qui transmettra les données ou la confirmation de modification de la base de données à **Director**.

Enfin, nous n'avons pas explicité les connexions et déconnexion dans nos cas d'utilisation, jugeant que l'utilisateur était déjà connecté au début de chaque cas, et que sa déconnexion le remettait au même endroit lors de sa prochaine reconnexion. Ainsi si un élève quitte sa session alors que l'application lui demandait de confirmer sa fiche de renseignements, elle lui redemandera directement suite à sa prochaine authentification.

2.1 Saisie d'une fiche de renseignements

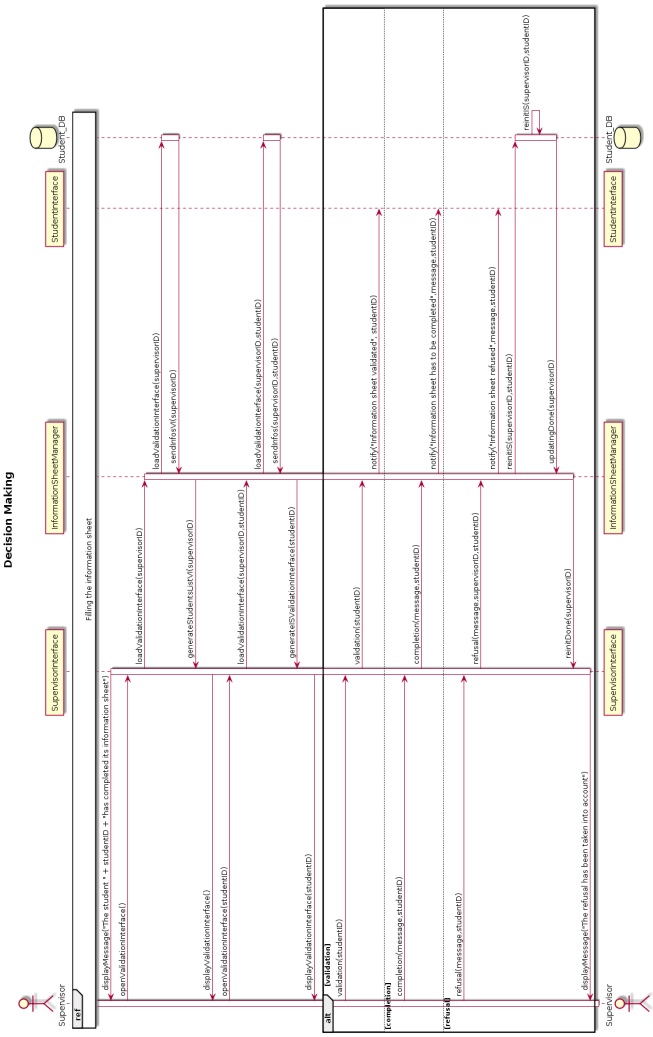
Voici ci-après le premier diagramme de séquence qui présente le cas de saisie d'une fiche de renseignements. L'élève suite à sa connexion demande l'affichage du menu de saisie, puis valide ou non une expérience précédente. Ensuite, il envoie les informations relative à sa fiche (via un formulaire par exemple). Si elles sont complètes, c'est-à-dire que les champs obligatoires sont renseignés et sous le bon format, l'application lui demande confirmation, et s'il dit oui, elle lui fait quitter le menu et notifie le superviseur concerné afin qu'il la valide avec la méthode **notify(String,int)**. Dans le cas contraire, s'il

a refusé de confirmer ou si les information ne sont pas complètes, il revient au menu lui demandant de saisir ses informations.


Filling the information sheet
Student
StudentInterface
InformationSheetManager
Student_DB
SupervisorInterface
openInformationSheetInterface()
loadIS(studentID)
load(studentID)
sendInfos(studentID)
generateIS(studentID)
displayInformationSheetInterface()
opt
validatePriorExperience()
validatePriorExperience(studentID)
validatePriorExperience(studentID)
validatePriorExperience(studentID)
updatingDone(studentID)
updatingDone(studentID)
displayMessage("The update is done")
loop
[While(!sheetComplete || !confirmation)]
updateInfos(infos)
updateInfos(infos, studentID)
updateInfos(infos, studentID)
updateInfos(infos, studentID)
updatingDone(studentID)
alt
[!sheetComplete]
ISNotComplete(studentID)
displayMessage("Please complete the informatiion sheet")
[sheetComplete]
askConfirmation(studentID)
askConfirmation()
alt
[confirmation]
confirmation()
confirmation(studentID)
confirmation(studentID)
confirmation(studentID)
updatingDone(studentID)
updatingDone(studentID)
displayMessage("The information sheet has been updated")
[!confirmation]
refusal()
refusal(studentID)
ISNotComplete(studentID)
displayMessage("Please complete the information sheet")
askValidation(supervisorID,studentID)
notify("The student " + studentID + " has completed its information sheet", supervisorID)

2.2 Prise de décision

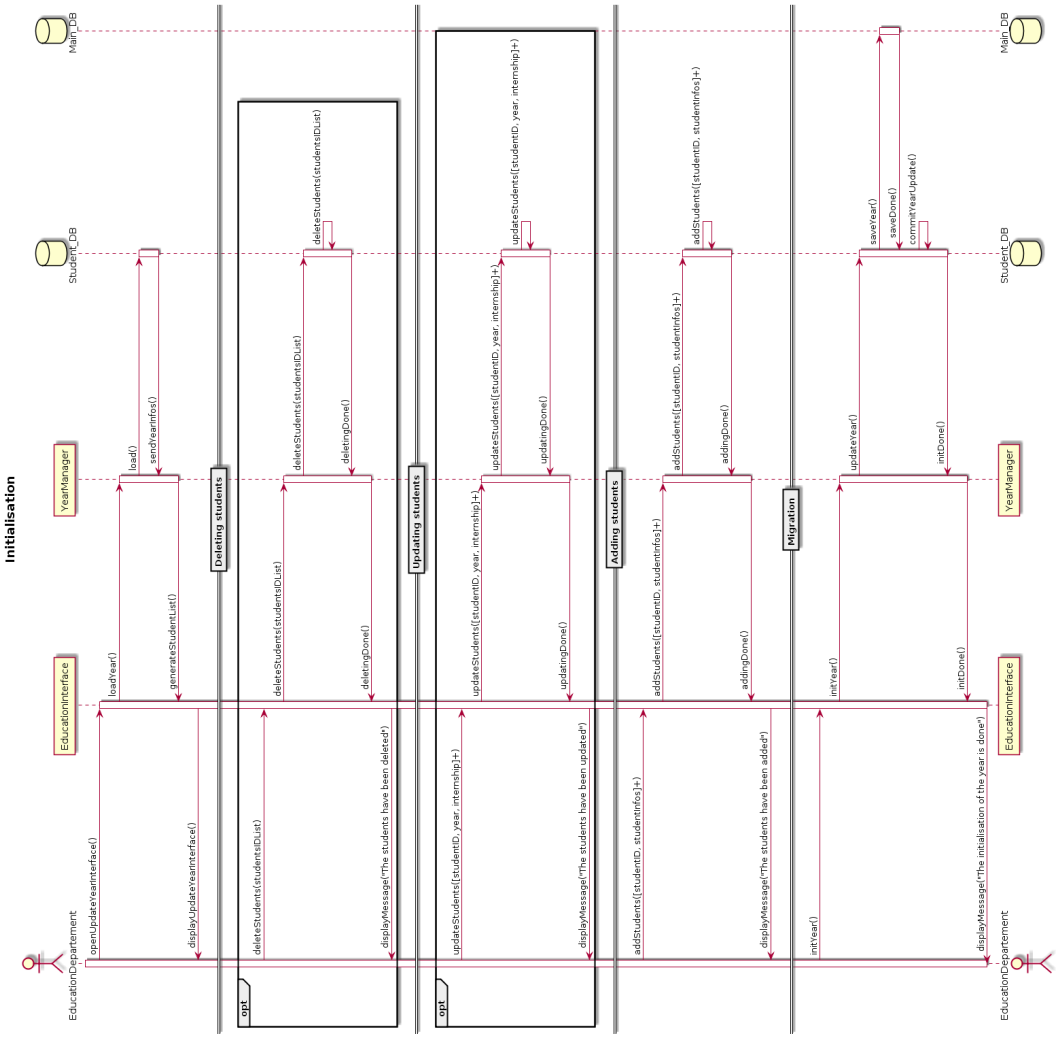
Dans ce diagramme, suite à la complétion de la fiche de renseignement par un élève, un responsable (**Supervisor**) reçoit un message en se connectant. Il va ensuit afficher le menu de validation qui lui liste tous les élèves dont il s'occupe, et éventuellement dispose d'une barre de recherche. Il lui est indiqué lesquels ont confirmé leur fiche et en cliquant sur l'un d'eux, il accède au menu de validation de la fiche de cet élève. L'élève est alors notifié de la réponse, et dans le cas d'un refus , sa fiche est réinitialisée.


Decision Making
Supervisor
SupervisorInterface
InformationSheetManager
StudentInterface
Student_DB
ref
Filling the information sheet
displayMessage("The student " + studentID + "has completed its information sheet")
openValidationInterface()
loadValidationInterface(supervisorID)
sendInfos(supervisorID)
generateStudentsList(supervisorID)
displayValidationInterface()
openValidationInterface(studentID)
loadValidationInterface(supervisorID,studentID)
sendInfos(supervisorID,studentID)
generateIDsValidationInterface(studentID)
displayValidationInterface(studentID)
alt
[validation]
validation(studentID)
validation(studentID)
notify("information sheet validated", studentID)
[completion]
completion(message,studentID)
completion(message,studentID)
notify("information sheet has to be completed",message,studentID)
[refusal]
refusal(message,studentID)
refusal(message,supervisorID,studentID)
notify("information sheet refused",message,studentID)
rentIDS(supervisorID,studentID)
rentIDS(supervisorID,studentID)
updatingDone(supervisorID)
rentIDone(supervisorID)
displayMessage("the refusal has been taken into account")

2.3 Initialisation

Enfin, dans ce diagramme, il est question de l'initialisation d'une nouvelle année. Chaque étudiant à un champ **newYearStatus** qui précisera son statut pour l'année suivante, et le responsable de l'initialisation devra tous les vérifier. A priori, une personne et une seule, représentera pour l'école la scolarité et sera chargée de supprimer les élèves ayant quitté l'établissement, mettre à jour le fait qu'un élève n'a ou n'a pas validé son année et son stage et ajouter les nouveaux élèves. Il sera possible dans un souci de réduction de la quantité de travail de fixer la valeur par défaut des booléens **year** et **internship** sur True.

Par la suite, elle devra valider les modifications en demandant la migration des statuts. Les données de l'année en cours sont alors sauvées dans la base de données **Main_DB** et les données de l'année suivante sont enregistrées dans la base **Student_DB**.

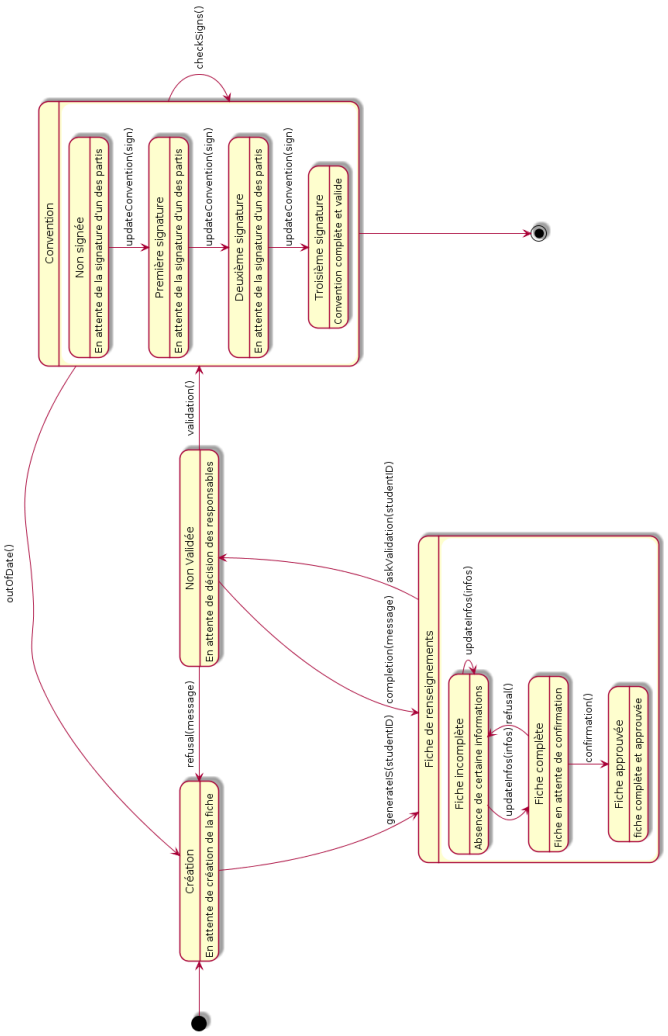

Initialisation
Main_DB
Student_DB
YearManager
EducationInterface
EducationDepartement
openUpdateYearInterface()
displayUpdateYearInterface()
loadYear()
load()
sendYearInfos()
generateStudentList()
Deleting students
opt
deleteStudents(studentsIDList)
deleteStudents(studentsIDList)
deletingDone()
deletingDone()
displayMessage("The students have been deleted")
Updating students
opt
updateStudents([studentID, year, internship]+)
updateStudents([studentID, year, internship]+)
updatingDone()
updatingDone()
displayMessage("The students have been updated")
Adding students
addStudents([studentID, studentInfos]+)
addStudents([studentID, studentInfos]+)
addingDone()
addingDone()
displayMessage("The students have been added")
Migration
initYear()
updateYear()
saveYear()
saveDone()
commitYearUpdate()
initDone()
initDone()
displayMessage("The initialisation of the year is done")

3 Diagramme d'état

Dans ce diagramme, nous avons explicité les cas d'utilisations précédents. L'application a pour état initial l'attente de la création d'une fiche de renseignements par un élève.

Le seul élément qu'il nous paraît nécessaire de préciser est la signature de la convention de stage. Ce qu'il se passe est que les trois signatures sont nécessaires mais que l'ordre n'importe pas. La méthode **updateConvention(sign)** enregistre une nouvelle signature et fait avancer l'état de l'application seulement si la signature provient d'un parti valide et que ce parti n'a pas encore signé. La méthode **checkSigns()** vérifiera à intervalle réguliers (1 fois par semaine par exemple) si la convention est signée ou pas, et dans le cas contraire en informera l'élève et les partis n'ayant pas signé. La méthode **outOfDate()** quant à elle va réinitialiser la fiche et notifier les responsables et l'élève après une date butoir déterminée par l'école.

State diagram of internship procedure



4 Diagramme de classes

Nous n'avons pas explicité les getters et les setters afin d'éviter un graphique trop lourd. aussi, en réalité, étant donné que notre application n'est basée que sur un système de requêtes et d'attente de réponse sous forme de requête, les données transmises (getters) ne le sont que par les arguments des requêtes et les données modifiées (setters) ne le sont que dans les bases de données.