Génie Logiciel 2 : Analyse (partie 1)

Christian Ingouff
Pierre-Alexandre Tyndal
Sonia Seddiki
Yann Charbonnier

EISTI

2013/2014
Semestre 2
Jalon 2

Table des matières

Partie 1

Présentation du jalon

Après avoir reformulé avec précision les contraintes et attentes définies par l'utilisateur, nous sommes maintenant en mesure de procéder à l'analyse. Cette phase est assez longue, d'autant plus que nous ne disposons pas encore de tous les outils théoriques nécessaires à son étude exhaustive. C'est pourquoi elle sera traitée en deux rapports successifs.

La phase d'analyse est importante. En effet, elle permet de modéliser de manière plus claire les attentes exprimées dans le cahier des charges, mais aussi de les rendre plus exploitables pour la suite du projet. Il est capital de distinguer clairement la phase d'analyse des phases de conception et réalisation. Ici, il s'agit de faire apparaître les fonctionnalités, contraintes, interactions que nous allons créer par la suite. Il n'est donc pas question de s'interroger sur les solutions techniques mises en oeuvre pour répondre au problème.

Le modèle d'analyse choisi pour ce projet est l'analyse orientée objet, réalisée grâce au langage de modélisation UML (Unified Modeling Language).

Dans ce premier compte-rendu, nous détaillerons :
- Le diagramme de cas d'utilisation
- Le diagramme de classes

Pour ce travail, la répartition des tâches s'est effectuée comme suit :
- Diagramme de cas d'utilisation : Christian
- Diagramme de classes : Yann, Pierre-Alexandre
- Rédaction du rapport : Sonia, Christian

Partie 2

Diagramme de cas d'utilisation

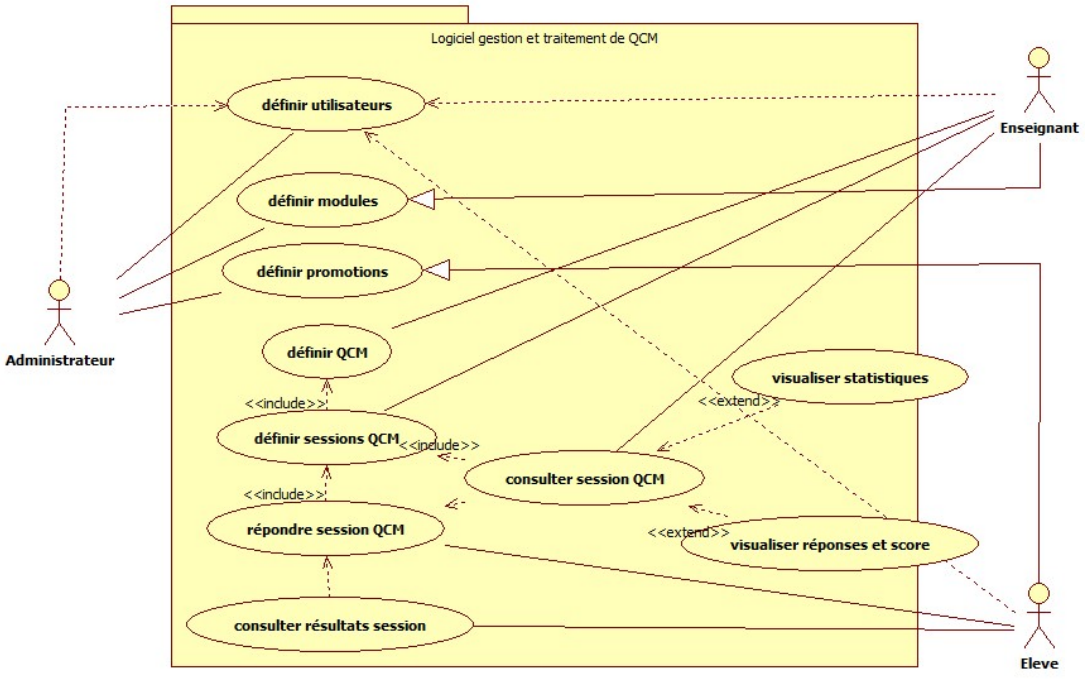

FIGURE 2.1 – Diagramme de cas d'utilisation

Ce diagramme décrit les différents cas d'utilisation de notre logiciel. Il le fait interagir avec des utilisateurs selon une liste d'opérations éventuellement liées entre elles. Ainsi, le diagramme présenté suit le raisonnement décrit ci-dessous.

Nous observons 3 types d'utilisateurs :
— L'administrateur
— L'enseignant
— L'élève

Un administrateur peut :
— Définir des utilisateurs (administrateurs, enseignants ou élèves)
— Définir des promotions (listes d'élèves)
— Définir des modules (que gèrent les enseignants)

Un professeur peut :
— Définir des QCM
— Définir des sessions de QCM, auparavant créés
— Consulter le statut d'une session de QCM
  — Visualiser les réponses et le score d'un élève à cette session
  — Visualiser les statistiques globales de la session

Un élève peut :
— Répondre à des sessions de QCM, auparavant créées
— Consulter les résultats de sa session de QCM

Partie 3

Diagramme de classes

3.1 Diagramme

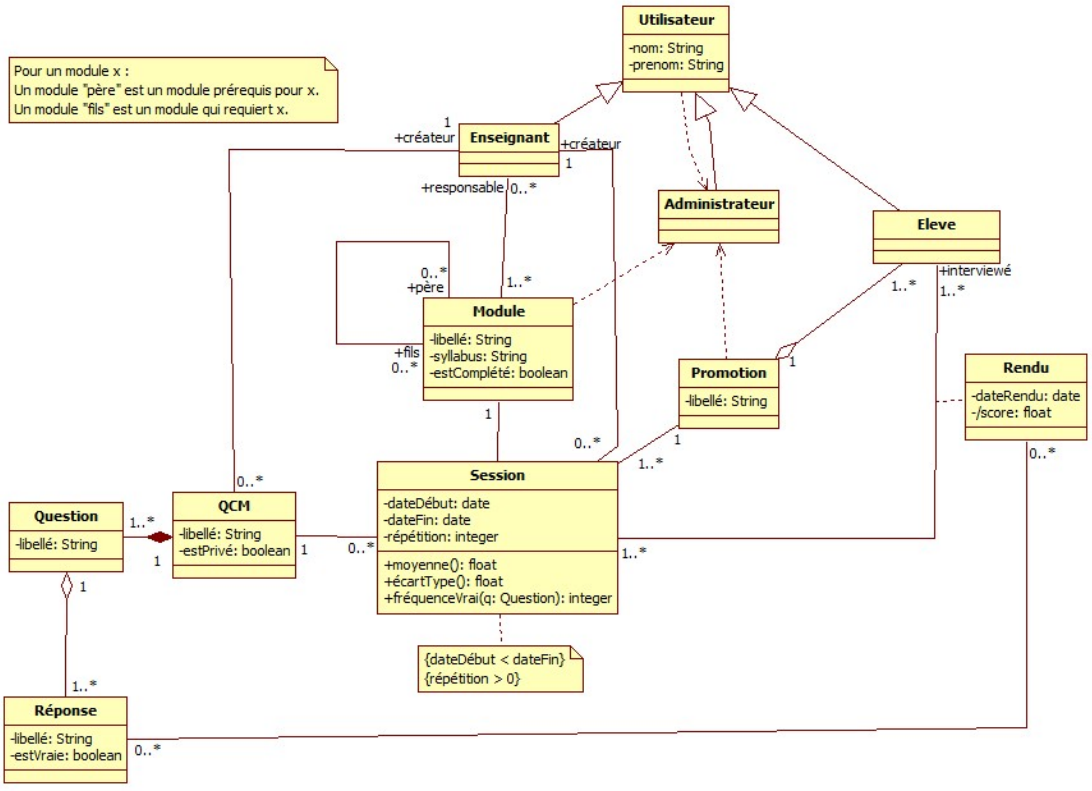

FIGURE 3.1 – Diagramme de cas d'utilisation

Ce diagramme décompose les besoins à l'aide de définitions de classes, ainsi que d'interactions entre ces classes. Le raisonnement ci-dessous explique le lien entre les classes et les interactions présentées et les besoins de notre projet.

Nous avons les classes suivantes :

- **Utilisateur** : défini par un nom et un prénom. Créé par un *Administrateur*.
- **Administrateur, Enseignant, Eleve** : les 3 types d'utilisateurs (généralisations de *Utilisateur*)
- **Module** : les enseignants doivent gérer un ou des modules (1..*). Un module peut être géré par un ou plusieurs enseignants (0..*). Défini par un libellé, un syllabus et un prédicat de complétion. Créé par un *Administrateur*.
- **Promotion** : liste d'élèves (agrégation de *Eleve*). Créé par un *Administrateur*.
- **QCM** : défini par un libellé et un prédicat de confidentialité. Il est créé par un *Enseignant*. Il est composé de :
  - **Question** : défini par un libellé. Contient un ensemble (agrégation) de :
    - **Réponse** : défini par un libellé et un prédicat de véracité.
- **Session** : définie par une date de début et de fin et par un nombre de répétition. Elle se passe dans le cadre d'un *Module* et est limitée à une *Promotion*. Elle est créée par un *Enseignant*.
- **Rendu** : un *Eleve* répond à une *Session* à l'aide d'un rendu. Il est défini par une date et enregistre les *Réponses* données par l'élève. Le score sera obtenu en fonction des réponses données.

Pour un *Module* x, un module "père" est un module prérequis (nécessitant qu'il soit "complété") pour x et un module "fils" est un module qui requiert x.

Les statistiques globales pour une session consultables par l'enseignant sont la moyenne, l'écart-type et la fréquence de bonnes réponses par question. Le contenu de ces fonctions, ainsi que le calcul du score d'un élève à un QCM sera développé dans la partie Conception de notre projet.

3.2 OCL

Certains attributs sont soumis à des conditions que nous insérons ici à l'aide d'OCL.

```
context Session::répétition: integer
init: 1
```

Le nombre indiquant la répétition dans une session de QCM vaut 1 par défaut.

```
context Session
inv: (self.qcm.estPrivé) implies
     (self.créateur = self.qcm.créateur)
```

Si un QCM est privé, ce QCM ne peut être utilisé que par son créateur. Nous traduisons ceci par l'égalité entre le créateur de la session et le créateur du QCM.

```
context Session
inv: (self.promotion = self.interviewé.promotion)
```

Une session de QCM ne peut être accessible qu'à une seule promotion. Il faut donc vérifier l'appartenance de chaque interviewé qui désire répondre à la promotion correspondante.

```
context Session
inv: self.interviewé->count(r : Rendu | r.session = self) <= self.répétition
```

On ne peut au maximum répondre qu'au nombre de fois spécifié par le nombre indiquant la répétition. Cette condition vérifie si le nombre de rendus est bien inférieur à ce nombre.

```
context Rendu
inv: (self.dateRendu > self.session.dateDébut)
     AND (self.dateRendu < self.session.dateFin)
```

Une session n'est accessible qu'après la date de début et avant la date de fin.

Conclusion

La première partie de cette phase d'analyse nous a donc permis de modéliser deux approches importantes de notre modèle :

— Les futurs utilisateurs du logiciel ainsi que leurs différentes interactions
— Les différentes classes prises en compte et les relations s'opérant entre elles

Ces deux diagrammes nous donnent une vision plus détaillée des doléances du cahier des charges, ce qui facilitera la conception, la rendant également plus complète. En effet, la précision de la description du modèle montre l'utilité de nouvelles fonctions qui n'étaient pas forcément explicitées par le client (par exemple, la création d'un système d'authentification pour savoir qui utilise le logiciel, et notamment son statut pour lui accorder les droits d'accès adéquats).

Par ailleurs, la décomposition sous forme de classes nous donne la base de notre code Java : création des classes, des accesseurs, prise en compte des contraintes, etc.

Toutefois, ces diagrammes ne sont pas figés. Ils sont destinés à être complétés, remaniés à mesure que l'analyse avance, voire à être modifiés dans la phase de conception. Par exemple, lors de l'élaboration de l'interface, il faudra changer le diagramme de classes en conséquence, pour choisir quels objets seront "visibles" par l'utilisateur.

Le suite de notre travail portera sur les diagrammes d'activité, de séquence et d'états-transitions. Après une analyse surtout structurelle du projet, ces diagrammes vont fournir une description fonctionnelle de notre programme, qui nous aidera, à la fin, à visualiser comment concevoir notre programme afin d'avancer dans notre projet.