Méthode B : Développement d'un système de suivi de l'évalutation d'étudiants

Formal Software

FAGNIEZ Florian, RULLIER Noémie et GOHIER Brian
9 décembre 2013

Table des matières

Table des figures

1 Introduction

Ce rapport contient l'ensemble des travaux effectués sur le projet de Formal Software : **Développement d'un système de suivi de l'évalutation d'étudiants**.

Nous expliquerons justifierons les choix faits concernant l'implantation de notre machine et le raisonnement qui a suivi.

2 Réflexion et Implantation

A la lecture de notre sujet, nous en avons ressorti plusieurs ensembles :

- **MODULES**
- **ETUDIANTS**
- **GROUPES**
- **PROJETS**

Suite à celà, nous avons également définis plusieurs variables :
- **lesModules :** sous-ensemble de MODULES
- **lesEtudiants :** sous-ensemble de ETUDIANTS
- **lesGroupes :** sous-ensemble de GROUPES
- **lesProjets :** sous-ensemble de PROJETS
- **appartientAuxGroupes :** définira les étudiants appartenant à un ou plusieurs groupes (**Figure 1 – Ensemble appartientAuxGroupes**)
- **specifiqueAuModule :** définira le projet qui appartiendra au module (chaque projet ne doit appartenir qu'a un seul module, **Figure 2 – Ensemble specifiqueAuModule**)
- **:** définira les étudiants travaillant sur un projet ou plusieurs projets (**Figure 3 – Ensemble participeAuxProjets**)
- **participeAuxModules :** définira les étudiants participant à un ou plusieurs modules (**Figure 4 – Ensemble participeAuxModules**)
- **participeAuxGroupesModules :** tous les couples Groupes/Modules auquel l'étudiant appartient (**Figure 5 – Ensemble participeAuxGroupesModules**)
- **possedeNote :** définira la note possédé par un étudiant par module

Une fois ces concepts définis, nous pouvions nous mettre à définir les différents invariants :

```
1  lesModules <: MODULES
2  & lesEtudiants <: ETUDIANTS
3  & lesGroupes <: GROUPES
4  & lesProjets <: PROJETS
```

permet de dire que lesModules et autres variables sont des sous-ensembles de MODULES et autres Sets

```
1  // Les etudiants peuvent appartenir a un ou plusieurs groupes
2   & appartientAuxGroupes : lesEtudiants <-> lesGroupes
3
4   // Chaque projet appartient a un module
```

```
5     & specifiqueAuModule : lesProjets +-> lesModules
6
7     // Les peuvent participer a des projets
8     & participeAuxProjets : lesEtudiants <-> lesProjets
9
10    // Les etudiants peuvent appartenir a des modules
11    & participeAuxModules : lesEtudiants <-> lesModules
12
13    // Les etudiants peuvent appartenir a un seul groupe par module
14    & participeAuxGroupesModules : lesEtudiants +-> lesModules * lesGroupes
15
16    // Un etudiant possede une note par module
17    & possedeNote : lesEtudiants * lesModules --> NAT
18
19    // Pour chaque etudiant le nombre de projet auxquels il participe
20    // doit etre egal au nombre de module auxquels il est inscrit
21    & ! ee.((ee : ETUDIANTS &
22            ee : dom(participeAuxProjets) &
23            ee : dom(participeAuxModules))
24            => card(participeAuxProjets[{ee}]) = card(participeAuxModules[{ee}]))
```

Le principe de relation entre les différentes variables nous semblait la plus correcte pour spécifier notre machine. En effet, un etudiant peut avoir plusieurs images (projets) ; comme un étudiant peut avoir plusieurs images (modules).

Cependant un étudiant ne peut avoir qu'un seul couple module,groupe comme image, la fonction partielle s'imposait ainsi d'elle même.

De même, le couple etudiants,modules ne peut posséder qu'une note qui est définie par un ensemble naturel.

3 Les Opérations

3.1 Ajouter/Supprimer un étudiant à un groupe/un projet/un module

Afin d'ajouter un étudiant à un groupe/projet/module, on vérifie d'abord que l'étudiant n'appartient pas à l'ensemble dans lequel on désire l'insérer et que les variables correspondent bien aux types attendus.

Une fois les préconditions respectées, on redéfinit notre ensemble comme étant cet ensemble et l'union de l'élément à insérer.

Concernant la suppression, on effectue les mêmes vérifications et le nouvel ensemble devient l'ensemble auquel on à retiré l'élément mis en paramètre.

```
/************************
* QUESTION a *
 ************************/

    addEtuModule(ee,mm,pp) =
        PRE
            ee : ETUDIANTS
            & ee : lesEtudiants
            & mm : MODULES
            & mm : lesModules
            & pp : PROJETS
            & pp :lesProjets
            & ee|->mm /: participeAuxModules
        THEN
            participeAuxModules := participeAuxModules \/ {(ee|->mm)}
            || participeAuxProjets := participeAuxProjets \/ {(ee|->pp)}
        END;

    addEtuProjet(ee,pp) =
        PRE
            ee : ETUDIANTS
            & ee : lesEtudiants
            & pp : PROJETS
            & pp : lesProjets
            & ee|->pp /: participeAuxProjets
        THEN
            ANY mm WHERE
                mm : MODULES
                & mm = specifiqueAuModule(pp)
            THEN
            participeAuxProjets := participeAuxProjets \/ {(ee|->pp)}
            || participeAuxModules := participeAuxModules \/ {(ee|->mm)}
            END
        END;

    addEtuGroupe(ee,gg) =
        PRE
            ee : ETUDIANTS
            & ee : lesEtudiants
            & gg : GROUPES
            & gg : lesGroupes
            & ee /: dom(appartientAuxGroupes)
        THEN
            appartientAuxGroupes := appartientAuxGroupes \/ {(ee|->gg)}
```

```
45          END;
46
47      delEtuModule(ee,mm,pp) =
48          PRE
49              ee : ETUDIANTS
50              & ee : lesEtudiants
51              & mm : MODULES
52              & mm : lesModules
53              & pp : PROJETS
54              & pp : lesProjets
55              & ee : dom(participeAuxModules)
56          THEN
57              participeAuxModules := participeAuxModules - {(ee|->mm)}
58              || participeAuxProjets := participeAuxProjets - {(ee|->pp)}
59          END;
60
61      delEtuProjet(ee,pp) =
62          PRE
63              ee : ETUDIANTS
64              & ee : lesEtudiants
65              & pp : PROJETS
66              & pp : lesProjets
67              & ee : dom(participeAuxProjets)
68              & ee : dom(participeAuxModules)
69          THEN
70              ANY mm WHERE
71                  mm : MODULES
72                  & mm : lesModules
73                  & mm = specifiqueAuModule(pp)
74              THEN
75                  participeAuxProjets := participeAuxProjets - {(ee|->pp)}
76                  || participeAuxModules := participeAuxModules - {(ee|->mm)}
77              END
78          END;
79
80      delEtuGroupe(ee,gg) =
81          PRE
82              ee : ETUDIANTS
83              & ee : lesEtudiants
84              & gg : GROUPES
85              & gg : lesGroupes
86              & ee : dom(appartientAuxGroupes)
87          THEN
88              appartientAuxGroupes := appartientAuxGroupes - {(ee|->gg)}
89          END;
```

3.2 Ajouter un module, un projet, un groupe

```
/************************
 * QUESTION b *
 ************************/

    // Ajouter date si on a le temps
    addModule(mm) =
        PRE
            mm : MODULES
            & mm /: lesModules
        THEN
            lesModules := lesModules \/ {mm}
        END;

    addProjet(pp) =
        PRE
            pp : PROJETS
            & pp /: lesProjets
        THEN
            lesProjets := lesProjets \/ {pp}
        END;

    addGroupe(gg) =
        PRE
            gg : GROUPES
            & gg /: lesGroupes
        THEN
            lesGroupes := lesGroupes \/ {gg}
        END;
```

On fait comme précédemment :
- On vérifie que le paramètre correspond bien au type d'élément que l'on désire insérer dans l'ensemble
- Le nouvel ensemble devient l'union de l'ensemble et de l'élement à insérer

3.3 Obtenir la liste des groupes auquel un étudiant appartient

On récupère notre liste qui contient l'ensemble des résultats obtenus

```
/************************
 * QUESTION c *
 ************************/

    list <-- listeGroupeParEtu(ee) =
        PRE
            ee : ETUDIANTS
            & ee : lesEtudiants
        THEN
// list := appartientAuxGroupes[{ee}]
         list := participeAuxGroupesModules[{ee}]
        END;
```

3.4 Obtenir la liste des projets auxquels un étudiant participe et le nombre de projets

Outre les préconditions, cette fonction est assez simple : on retourne dans une liste l'ensemble des résultats des projets auxquels participe l'étudiant et on retourne son cardinal dans une variable.

```
/************************
 * QUESTION d *
 ************************/

   list, nb <-- ProjetsEtNombreParEtu(ee) =
       PRE
           ee : ETUDIANTS
           & ee : lesEtudiants
       THEN
           list := participeAuxProjets[{ee}]
           || nb := card(list)
       END;
```

3.5 Les étudiants qui participent à plus de n groupes différents

Afin d'obtenir la liste de tous les étudiants qui participent à plus de n groupes différents, on vérifie d'abord que le paramètre passé est bien un $\mathbb{N}$. On va ensuite chercher pour chaque étudiant le nombre de groupe différents auxquel il appartient grâce à la variable *appartientAuxGroupes*. Pour chacun des couples (Etudiant, Nombre Groupe) retourné on ajoute dans la liste résultat les etudiant tel que la deuxième partie du couple soit supérieur à n.

```
/************************
 *      QUESTION e      *
 ************************/

   list <-- EssaiEtuParticipePlusNGroupesDiff(nbGroup) =
       PRE
           nbGroup : NAT
       THEN
           ANY coupleEtuNbpj WHERE
               coupleEtuNbpj : lesEtudiants --> NAT
               & coupleEtuNbpj = {ee,nn | ee : lesEtudiants & nn : NAT & nn = (
                   card(appartientAuxGroupes[{ee}]))} // Comment on calcul pour
                   chaque etudiant
           THEN
               list := coupleEtuNbpj |> {ii | ii : NAT & ii > nbGroup}
           END
       END;
```

3.6 Ajouter une note à un étudiant pour un module

Pour ajouter une note à un etudiant pour un module donné, on vérifie d'abord que le paramètre *ee* est bien du type *ETUDIANTS* et que celui-ci appartient bien aux etudiants. De même, on vérifie que le paramètre *mm* est bien du type *MODULES*, que celui-ci appartient bien aux modules et que le paramètre *nn* est bien du type $\mathbb{N}$. On stocke ensuite dans la variable *possedeNote* l'union entre la variable *possedeNote* et la note *nn* pour le couple (ee, mm).

```
/************************
 *    QUESTION Bonus   *
 ************************/

    addNote(ee,mm,nn) =
        PRE
            ee : ETUDIANTS
            & ee : lesEtudiants
            & mm : MODULES
            & mm : lesModules
            & nn : NAT
        THEN
            possedeNote(ee,mm) := nn
        END
```

4 Les preuves

La preuve permet d'apporter l'assurance de la validité de notre machine abstraite. Afin de prouver notre machine, **Atelier B** met à notre disposition différents outils. Dans un premier temps, nous exécutons un *Automatic Proof (Force 1)*, le résultat de cette commande nous retourne toutes les Proofs Obligation non prouvées. Nous exécutons ensuite un *Automatic Proof (Force 2)*, mais nous constatons aucune différence sur le nombre de Proofs Obligation non prouvée. Afin de voir quelles sont ces preuves nous exécutons ensuite *Interactive Proof*. Nous avons 85% de preuves prouvées, ce qui nous fait 5 operations non prouvées. Les 4 premières (*addEtuModule*, *addEtuProjet*, *delEtuModule* et *delEtuProjet*) ne sont pas prouvées car elle ne vérifient pas que le nombre de projets auquel est inscrit l'étudiant est égal au nombre de module auquel il est aussi inscrit. Cependant, nous n'avons pas réussi à résoudre ce problème car lors de l'ajout respectivement la suppression d'un module, nous ajoutons respectivement supprimons le projet associé. De même lors de l'ajout ou suppression d'un projet.

Nous avons une dernière operation non prouvée *addModule*, qui ne vérifie pas que *dom(possedeNote) = lesEtudiants*(lesModules\/mm)*.

5 Le raffinement

Afin d'affiner notre modèle, nous avons utilisé le raffinement. Ceci permet d'apporter des détails supplémentaires à chaque étape.

Nous n'avons pas réussi à raffiner notre modèle à l'aide du memento. Cependant nous avons effectué plusieurs essais non concluants. Nous avons décidé de rester sur le modèle initial afin d'avoir quelque chose de fonctionnel. Pour arriver à nos fins il aurait fallu décomposer le raffinement en plusieurs étapes.

FIGURE 1 – Ensemble appartientAuxGroupes

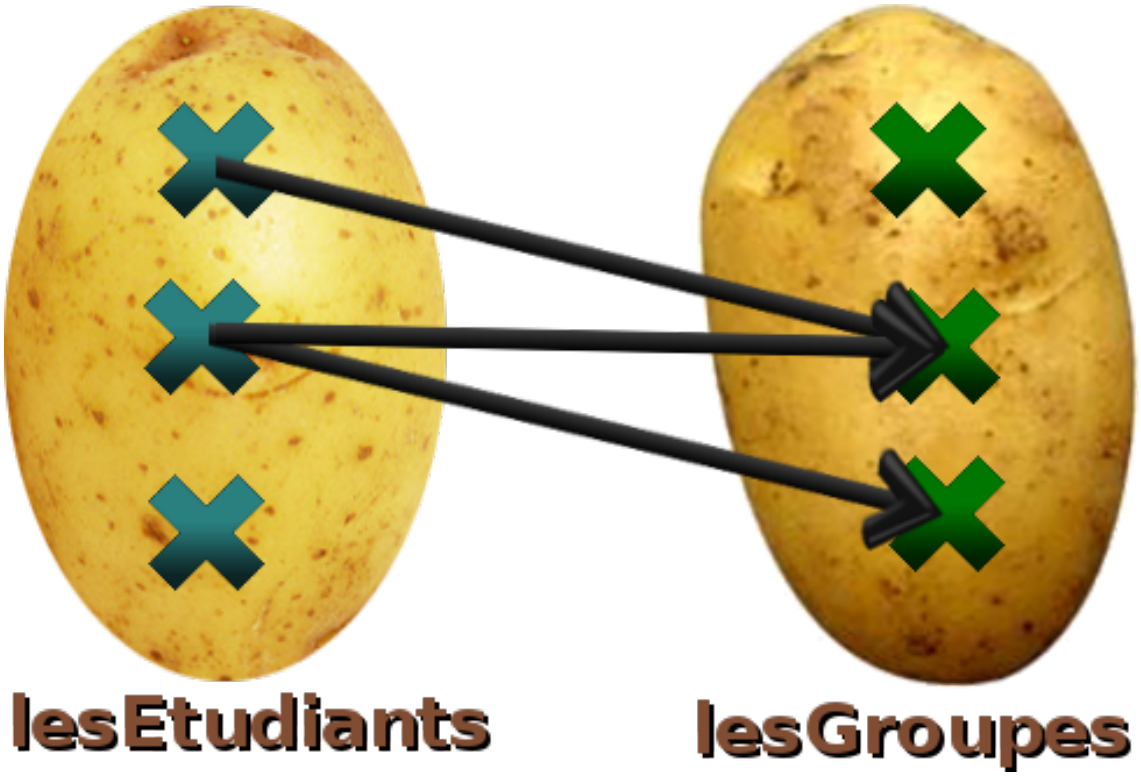

FIGURE 2 – Ensemble specifiqueAuModule

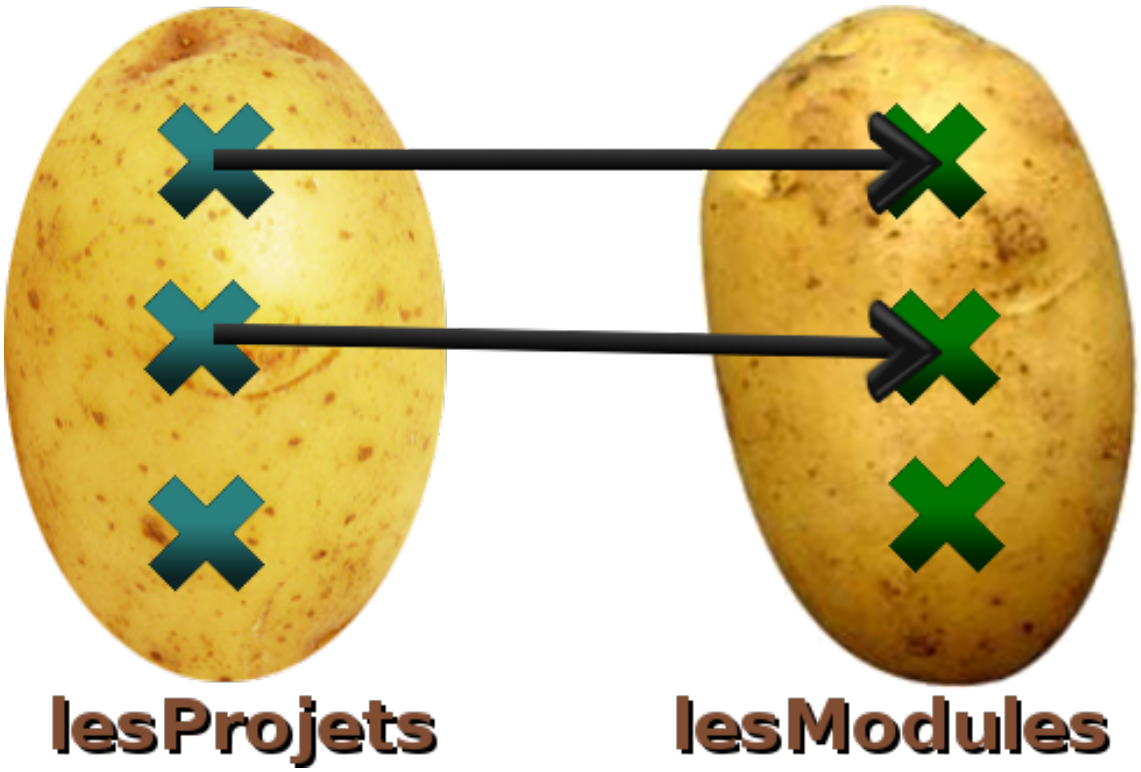

FIGURE 3 – Ensemble participeAuxProjets

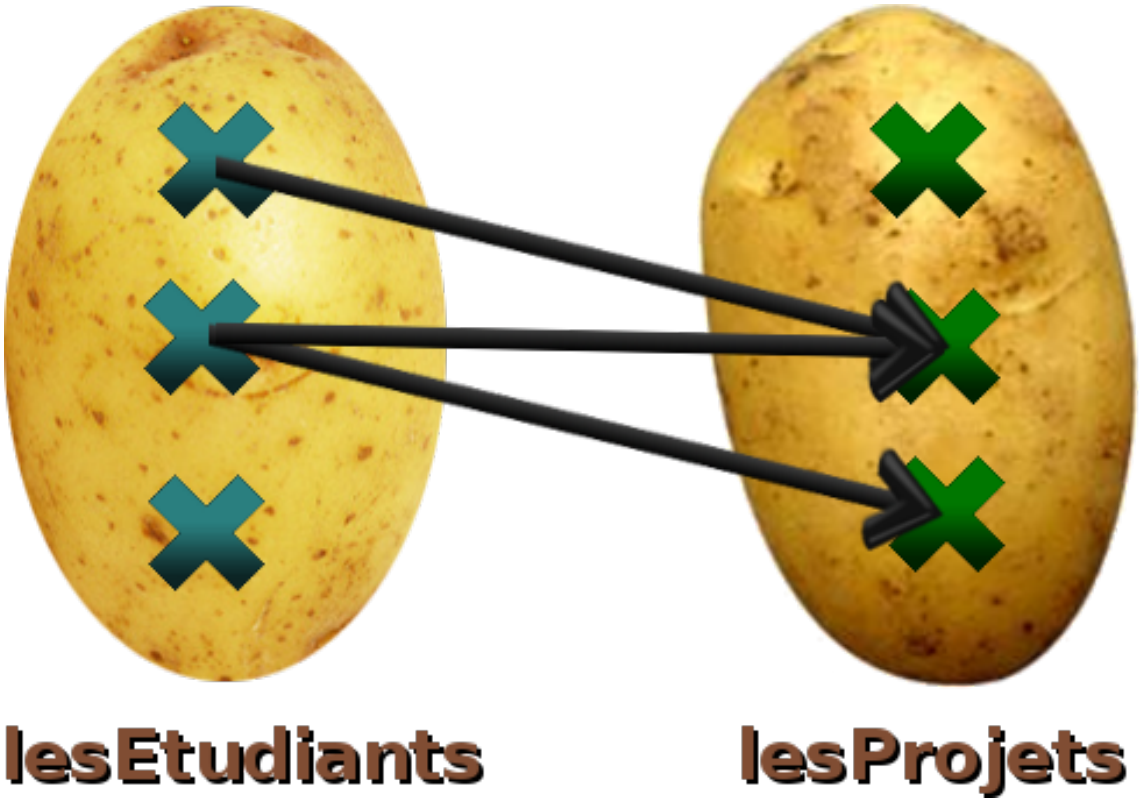

FIGURE 4 – Ensemble participeAuxModules

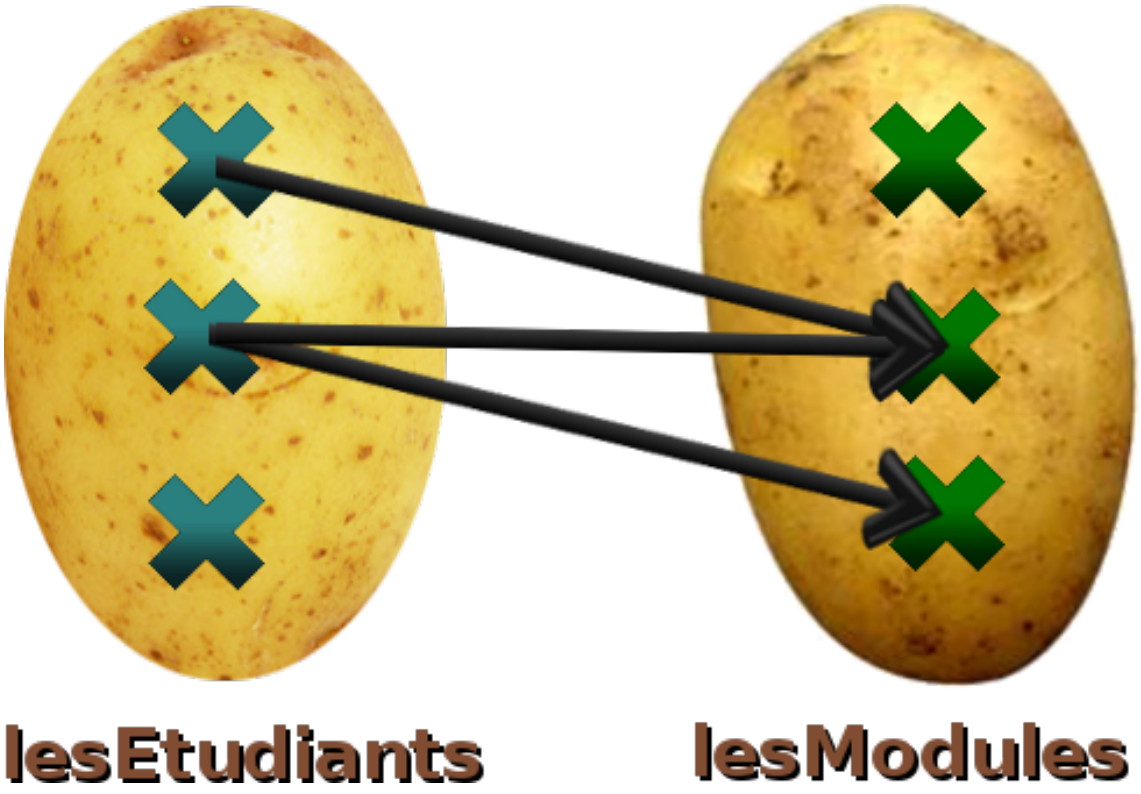

FIGURE 5 – Ensemble participeAuxGroupesModules