Projet CSC4102 : Simulation de programmes pour la détection d'interblocage

Nom Prénom Étudiant1 et Nom Prénom Étudiants2

Année 2018–2019 — 3 janvier 2019

Table des matières

1 Spécification

1.1 Diagrammes de cas d'utilisation

Le diagramme suivant est à compléter.

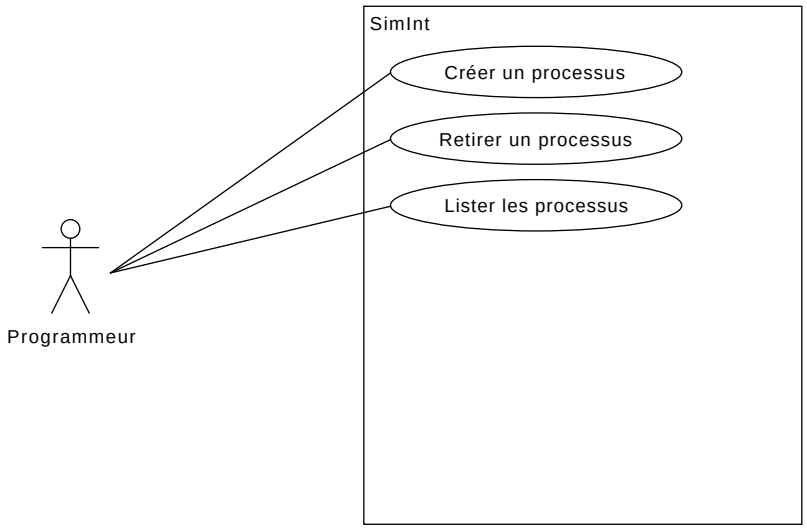

FIGURE 1 – Diagramme de cas d'utilisation

1.2 Priorités, préconditions et postconditions des cas d'utilisation

Les priorités des cas d'utilisation pour le sprint 1 sont choisies avec les règles de bon sens suivantes :

- pour retirer une entité du système, elle doit y être. La priorité de l'ajout est donc supérieure ou égale à la priorité du retrait ;
- pour lister les entités d'un type donné, elles doivent y être. La priorité de l'ajout est donc supérieure ou égale à la priorité du listage ;
- il est *a priori* possible, c.-à-d. sans raison contraire, de démontrer la mise en œuvre d'un sous-ensemble des fonctionnalités du système, et plus particulièrement la prise en compte des principales règles de gestion, sans les retraits ou les listages.
- la possibilité de lister aide au déverminage de l'application pendant les activités d'exécution des tests de validation.

Par conséquent, les cas d'utilisation d'ajout sont *a priori* de priorité « haute », ceux de listage de priorité « moyenne », et ceux de retrait de priorité « basse ».

Dans la suite, nous donnons les préconditions et postconditions pour les cas d'utilisation de priorité « Haute ». Pour les autres, nous indiquons uniquement leur niveau de priorité.

La précondition suivante est à compléter.

HAUTE n° 1
- Créer un processus
  - précondition : nom de processus bien formé (non `null` et non vide) $\wedge$ processus avec ce nom inexistant $\wedge$ exécution non débutée
  - postcondition : processus avec ce nom

basse
- Retirer un processus

Moyenne
- Lister les processus

2 Préparation des tests de validation

2.1 Tables de décision des tests de validation

La fiche programme du module CSC4102 ne permettant pas de développer des tests de validation couvrant l'ensemble des cas d'utilisation de l'application, les cas d'utilisation choisis sont de priorité HAUTE.

La table de décision suivante est à compléter.

Numéro de test	1	2	3	4
Nom processus bien formé ($\neq$ null $\wedge \neq$ vide)	F	T	T	T
Exécution non débutée		F	T	T
Processus inexistant avec ce nom			F	T
Création acceptée	F	F	F	T
Nombre de jeux de test	2	1	1	1

TABLE 1 – Cas d'utilisation « créer un processus »

3 Conception

3.1 Liste des classes

La liste des classes suivante est à compléter.

À la suite d'un parcours des diagrammes de cas d'utilisation et d'une relecture de l'étude de cas, voici une première liste de classes avec quelques attributs :

— SimInt (la façade),
— Processus — nom,
— ÉtatGlobal (l'état du système) — ,
— ÉtatProcessus (la partie de l'état concernant les processus) — .

3.2 Diagramme de classes

Le diagramme de classes suivant est à compléter.

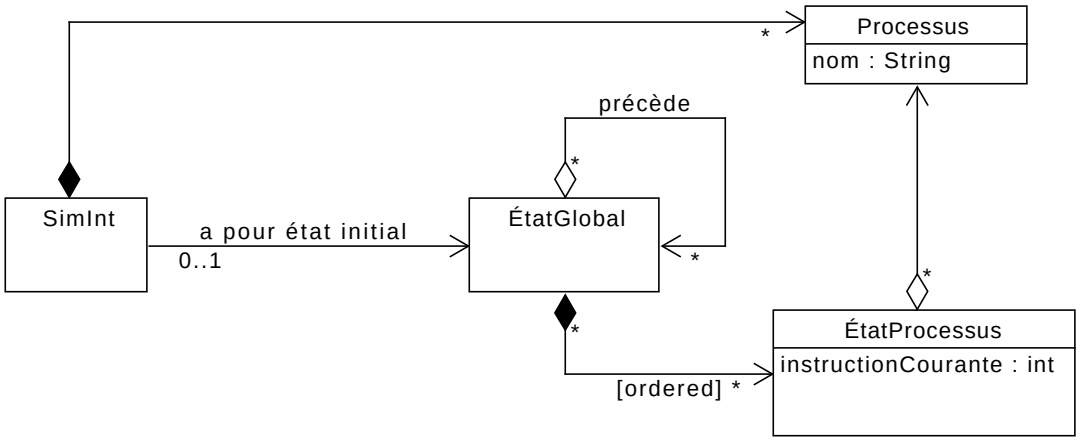

FIGURE 2 – Diagramme de classes

3.3 Diagramme d'objets

Comme l'une des difficultés de l'étude de cas est de comprendre, pour l'utiliser, la copie légère et la copie profonde, nous dessinons un diagramme d'objets de deux états globaux.

Le diagramme d'objets suivant est à compléter.

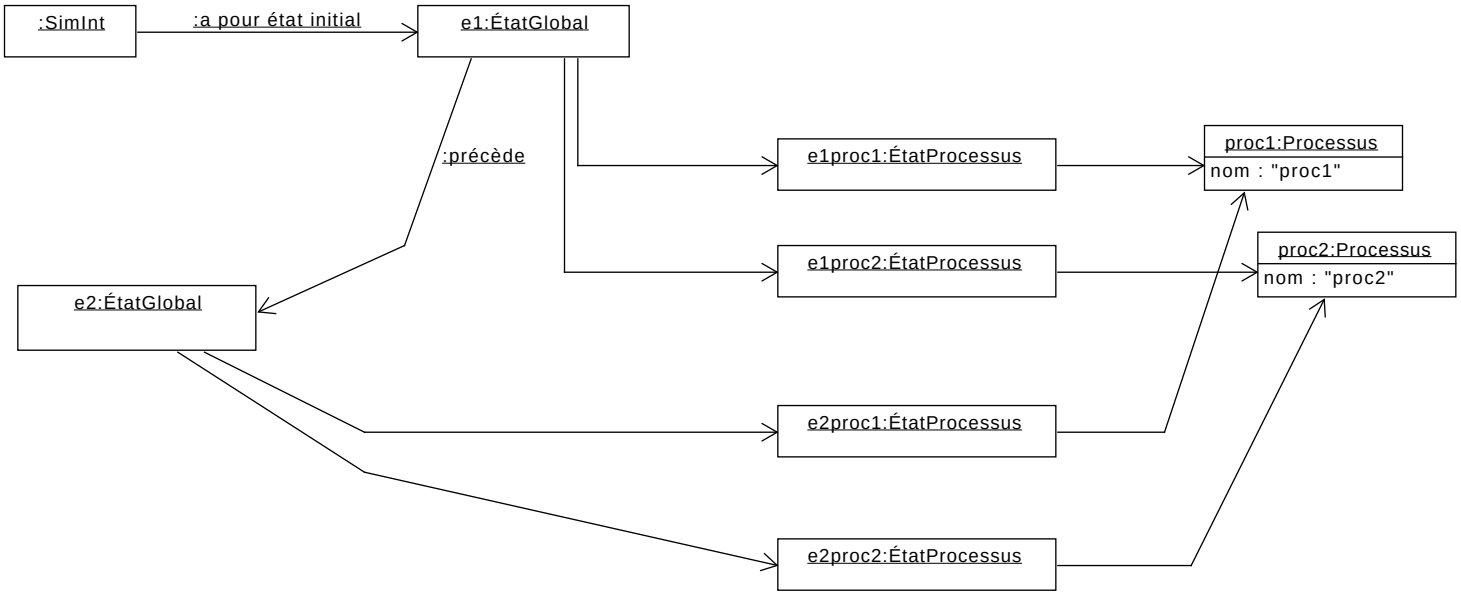

FIGURE 3 – Diagramme d'objets à partir de l'état initial

3.4 Diagrammes de séquence

La description et le diagramme de séquence suivants sont à compléter.

Voici la description textuelle du cas d'utilisation « créer un processus » :

— arguments en entrée : nom du processus, nom du programme

— rappel de la précondition : nom du processus bien formé (non null et non vide) ∧ processus avec ce nom existant ∧ exécution non débutée

— algorithme :

1. vérifier les arguments
2. si en outre l'exécution du système n'a pas débuté
   (a) vérifier que le processus n'existe pas
   (b) créer un processus avec ce nom
   (c) ajouter le processus à la collection des processus
   (d) ajouter un état dans l'état global initial pour ce processus
      i. créer un état pour ce processus
      ii. ajouter l'état de processus à l'état global initial

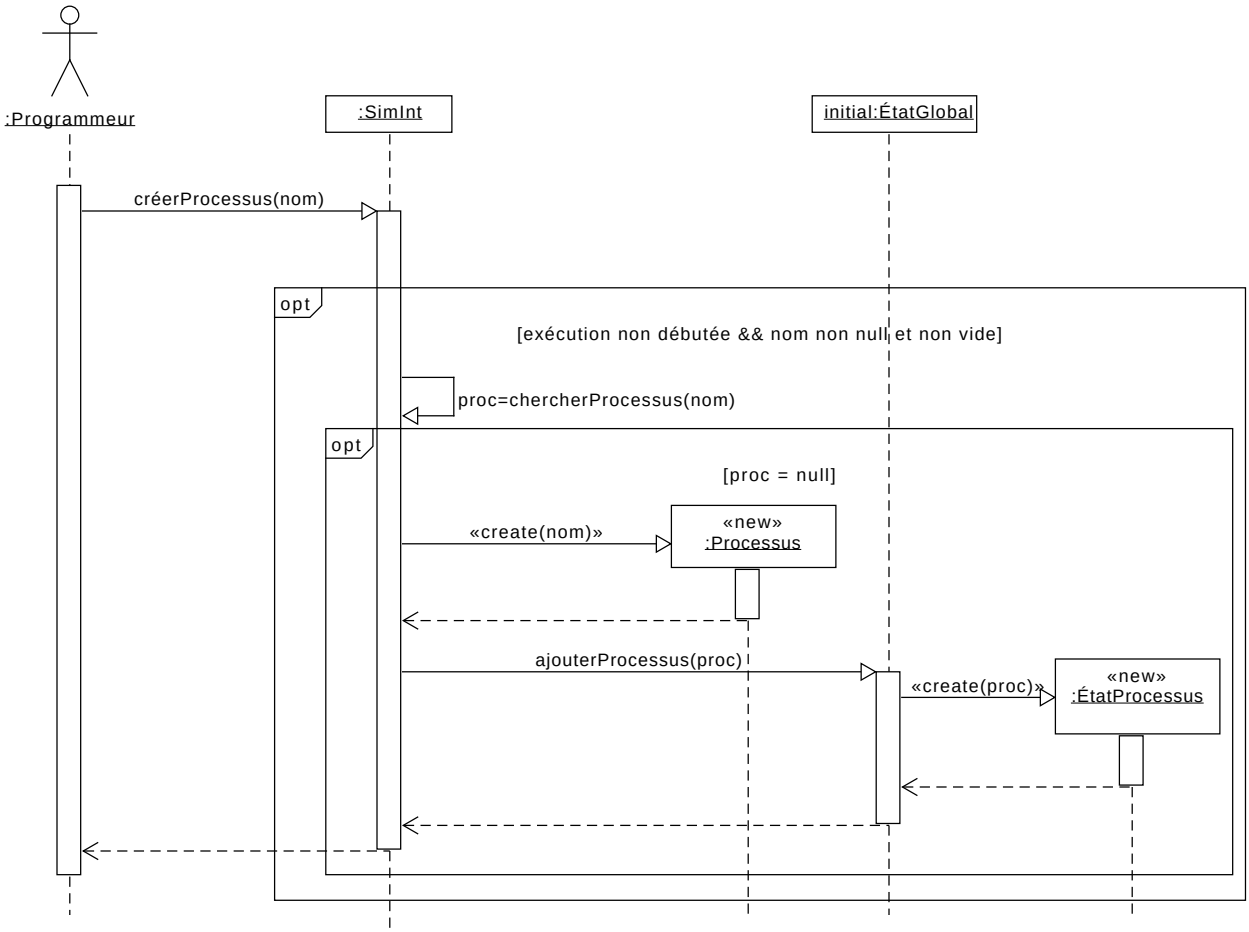

FIGURE 4 – Diagramme de séquence du cas d'utilisation « créer un processus »

4 Fiche des classes

Les fiches des classes suivantes sont à compléter.

4.1 Classe SimInt

Simint
<– **attributs « association »** –> – processus : collection de @Processus – étatGlobalInitial : ÉtatGlobal – exécutionDébutée : booléen
<– **constructeur** –> + SimInt() <– **operations « cas d'utilisation »** –> + créerProcessus(String nom) + chercherProcessus(String nom) : Processus

4.2 Classe Processus

Processus
<– **attributs** –> – nom : String
<– **constructeur** –> + constructeurProcessus(String nom)

4.3 Classe ÉtatProcessus

ÉtatProcessus
<– **attributs** –> – processus : Processus – compteurInstanciation : entier = 0 – compteurInstance : entier
<– **constructeur** –> + constructeurProcessus(Processus processus) + constructeurProcessus(ÉtatProcessus étatProcessus) // constructeur par copie

4.4 Classe ÉtatGlobal

ÉtatGlobal
<– **attributs** –> – estÉtatGlobalInitial : booléen – étatsGlobauxAtteignables : collection de @ÉtatGlobal – compteurInstanciation : entier = 0 – compteurInstance : entier <– **attributs « association »** –> – étatsProcessus : collection ordonnée de @ÉtatProcessus
<– **constructeurs** –> + constructeurÉtatGlobal() + constructeurÉtatGlobal(ÉtatGlobal origine) // constructeur par copie <– **operations « cas d'utilisation »** –> + ajouteÉtatProcessus(Processus processus) + chercherÉtatProcessus(String nom) : @ÉtatProcessus

5 Diagrammes de machine à états et invariants

Les invariants suivants sont à compléter.

5.1 Classes Processus

L'invariant de la classe Processus est le suivant :

nom $\neq$ null $\wedge$ nom $\neq$ “”

5.2 Classes ÉtatProcessus

L'invariant de la classe ÉtatProcessus est le suivant :

processus $\neq$ null

5.3 Classes ÉtatGlobal

L'invariant de la classe ÉtatGlobal est le suivant :

etatsProcessus $\neq$ null

6 Préparation des tests unitaires

6.1 Classe Processus

Numéro de test	1	2
nom $\neq$ null $\wedge \neg$vide	F	T
nom$'$ $\neq$ null $\wedge \neg$vide		T
invariant		T
Levée d'une exception	OUI	NON
Objet créé	F	T
Nombre de jeux de test	2	1

TABLE 6 – Méthode `constructeurProcessus` de la classe `Processus` »

A Algorithmes des classes

Les algorithmes des classes suivantes sont à compléter.

A.1 Classe SimInt

---

constructeurSimint()

processus = nouvelle collection vide
étatGlobalInitial = constructeurÉtatGlobal()
exécutionDébutée = false

---

créerProcessus(String nom)

si (nom null OU nom chaîne vide)
    lever exception
si exécution déjà débutée
    lever exception
Processus proc = chercherProcessus(nom)
si (proc = null)
    proc = constructeurProcessus(nom, p)
    étatInitial.ajouterProcessus(proc)

A.2 Classe Processus

---

constructeurProcessus(String nom)

```
si (nom null OU nom chaîne vide) // programmation défensive
        lever exception
this.nom = nom
assert invariant()
```

A.3 Classe ÉtatProcessus

---

constructeurProcessus(Processus processus)

```
si (processus = null)
        lever exception
this.processus = processus
compteurInstanciation++
compteurInstance = compteurInstanciation
assert invariant()
```

---

constructeurProcessus(ÉtatProcessus étatProcessus)

```
si (étatProcessus = null)
        lever exception
this.processus = proc.processus
compteurInstanciation++
compteurInstance = compteurInstanciation
assert invariant()
```

A.4 Classe ÉtatGlobal

---

constructeurÉtatGlobal()

étatGlobalInitial = true
étatsProcessus = création d'une collection vide
étatsGlobauxAtteignables = création d'une collection vide
compteurInstanciation++
compteurInstance = compteurInstanciation
assert invariant()

---

constructeurÉtatGlobal(ÉtatGlobal origine)

étatGlobalInitial = false
étatsProcessus = copie légère de la collection origine.étatsProcessus
étatsGlobauxAtteignables = création d'une collection vide
compteurInstanciation++
compteurInstance = compteurInstanciation
assert invariant()

---

ajouteProcessus(Processus processus)

si (!étatInitial) // programmation défensive
    lever exception
si (processus = null) // programmation défensive
    lever exception
étatProcessus = constructeurÉtatProcessus(processus)
si (étatsProcessus contient déjà étatProcessus)
    lever exception
ajouter étatProcessus à la collection étatsProcessus
assert invariant()
