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

CHAFFARDON Pierre et DENIZE Julien

Année 2018–2019—21 janvier 2019

Table of Contents

1 Spécification

1.1 Diagrammes de cas d'utilisation

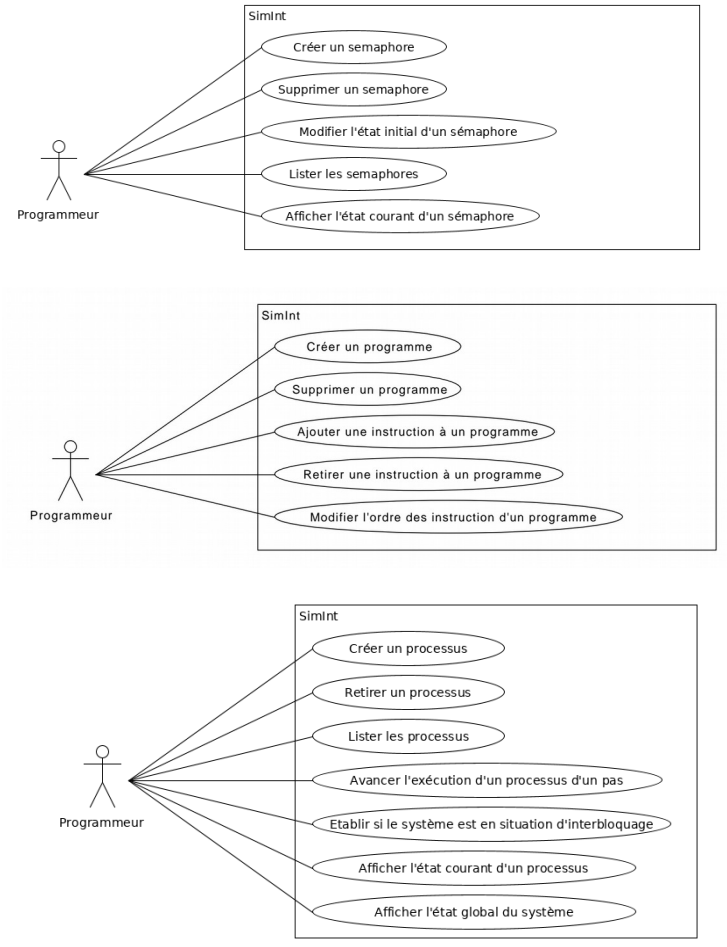

FIG. 1: Diagrammes de cas d'utilisation pour dans l'ordre: semaphore, programme, processus

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.

- Créer un semaphore : HAUTE n°1
    - précondition :nom de semaphore bien formé (non null $\wedge$ non vide) $\wedge$ semaphore avec ce nom inexistant $\wedge$ valeur initiale du compteur bien formée (non null et non vide)
    - postcondition : semaphore initialisé avec ce nom existant
- Supprimer un sémaphore : basse
- Modifier l'état initial d'un semaphore : basse
- Lister les semaphores : moyenne
- Afficher l'état courant d'un semaphore : moyenne
- Créer un programme : HAUTE n°2
    - précondition : nom de programme bien formé (non null $\wedge$ non vide) $\wedge$ programme avec ce nom inexistant
    - postcondition : programme avec ce nom existant
- Supprimmer un programme : basse
- Ajouter une instruction à un programme : HAUTE n°3
    - précondition : instruction bien formée (non null $\wedge$ non vide) $\wedge$ nom semaphore manipulé bien formé (non null et non vide) $\wedge$ semaphore manipulé avec ce nom existant $\wedge$ nom programme bien formé (non null $\wedge$ non vide) $\wedge$ programme avec ce nom existant
    - postcondition : instruction ajoutée au programme

- Retirer une instruction à un programme : basse
- Modifier l'ordre des instruction d'un programme : basse
- Créer un processus : HAUTE n°4
    - précondition : nom de processus bien formé (non null ∧ non vide) ∧ processus avec ce nom inexistant ∧ exécution non débutée ^ nom programme bien formé (non null ∧ non vide) ∧ programme avec ce nom existant
    - postcondition : processus avec ce nom exécutant le programme
- Retirer un processus : basse
- Lister les processus : moyenne
- Avancer l'exécution d'un processus d'un pas : HAUTE n°5
    - précondition : nom processus bien formé (non null ∧ non vide) ∧ processus avec ce nom existant ∧ exécution en cours ∧ processus non bloqué
    - postcondition : exécution du processus avancée d'un pas
- Etablir si le système est en situation d'interbloquage : HAUTE n°6
    - précondition : noms processus en cours d'exécution ou terminés bien formés (non null ∧ non vide) ∧ processus en cours d'exécution ou terminés existants
    - postcondition : vraie (pas de modification de l'état du système)
- Afficher l'état courant d'un processus : moyenne
- Afficher l'état global du système : moyenne

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.

Numéro de test	1	2	3	4
Nom semaphore bien formé (≠ null∧ ≠ vide)	F	T	T	T
Semaphore inexistant avec ce nom		F	T	T
Valeur initiale du compteur bien formé (≠ null∧ ≠ vide)			F	T
Création acceptée	F	F	F	T
Nombre de jeux de test	2	1	2	1

TAB. 1: Cas d'utilisation «créer un semaphore»

Numéro de test	1	2	3
Nom programme bien formé (≠ null∧ ≠ vide)	F	T	T
Programme inexistant avec ce nom		F	T
Création acceptée	F	F	T
Nombre de jeux de test	2	1	1

TAB. 2: Cas d'utilisation «créer un programme»

Numéro de test	1	2	3	4	5	6
Instruction bien formée (≠ null∧ ≠ vide)	F	T	T	T	T	T
Nom semaphore manipulé bien formé (≠ null∧ ≠ vide)		F	T	T	T	T
Semaphore manipulé avec ce nom existant			F	T	T	T
Nom programme bien formé (≠ null∧ ≠ vide)				F	T	T
Programme avec ce nom existant					F	T
Instruction ajoutée au programme	F	F	F	F	F	T
Nombre de jeux de test	2	2	1	2	1	1

TAB. 3: Cas d'utilisation «ajouter une instruction à un programme»

Numéro de test	1	2	3	4	5	6
Nom processus bien formé (≠ null∧ ≠ vide)	F	T	T	T	T	T
Exécution non débutée		F	T	T	T	T
Processus inexistant avec ce nom			F	T	T	T
Nom programme bien formé (≠ null∧ ≠ vide)				F	T	T
Programme existant avec ce nom					F	T
Création acceptée	F	F	F	F	F	T
Nombre de jeux de test	2	1	1	2	1	1

TAB. 4: Cas d'utilisation «créer un processus»

Numéro de test	1	2	3	4	5
Nom processus bien formé (≠ null∧ ≠ vide)	F	T	T	T	T
Processus avec ce nom existant		F	T	T	T
Exécution processus en cours			F	T	T
Processus non bloqué				F	T
Exécution du processus avancée d'un pas	F	F	F	F	T
Nombre de jeux de test	2	1	1	1	1

TAB. 5: Cas d'utilisation «avancer l'exécution d'un processus d'un pas»

Numéro de test	1	2	3
Nom processus en cours d'exécution ou terminés bien formés (≠ null∧ ≠ vide)	F	T	T
Processus en cours d'exécution ou terminés existants		F	T
Situation d'interbloquage ou non établie	F	F	T
Nombre de jeux de test	2	1	1

TAB. 6: Cas d'utilisation «établir si le système est en situation d'interbloquage»

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.

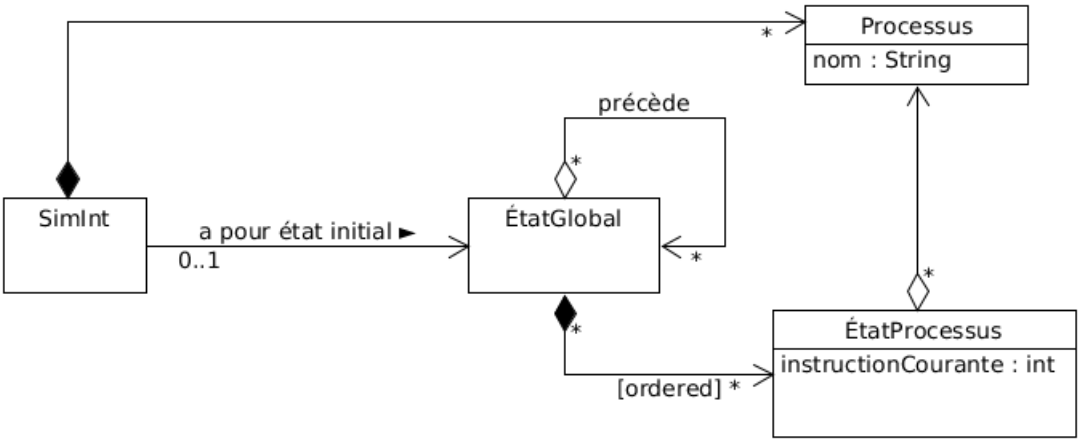

FIG. 2: Diagramme de classes

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.

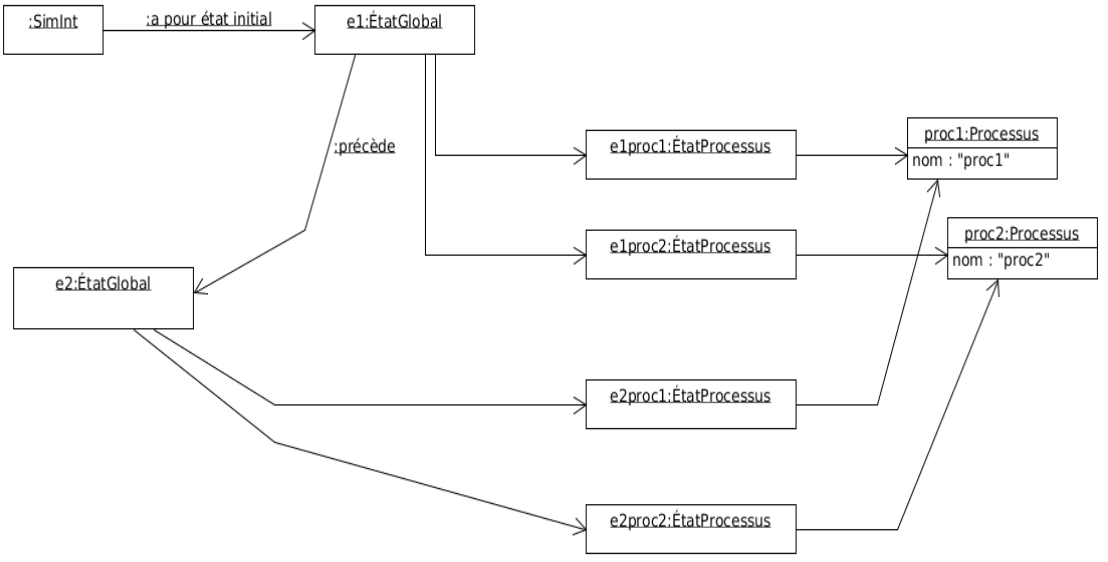

FIG. : Diagramme d’objets à partir de l’état initial

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 nullnon 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

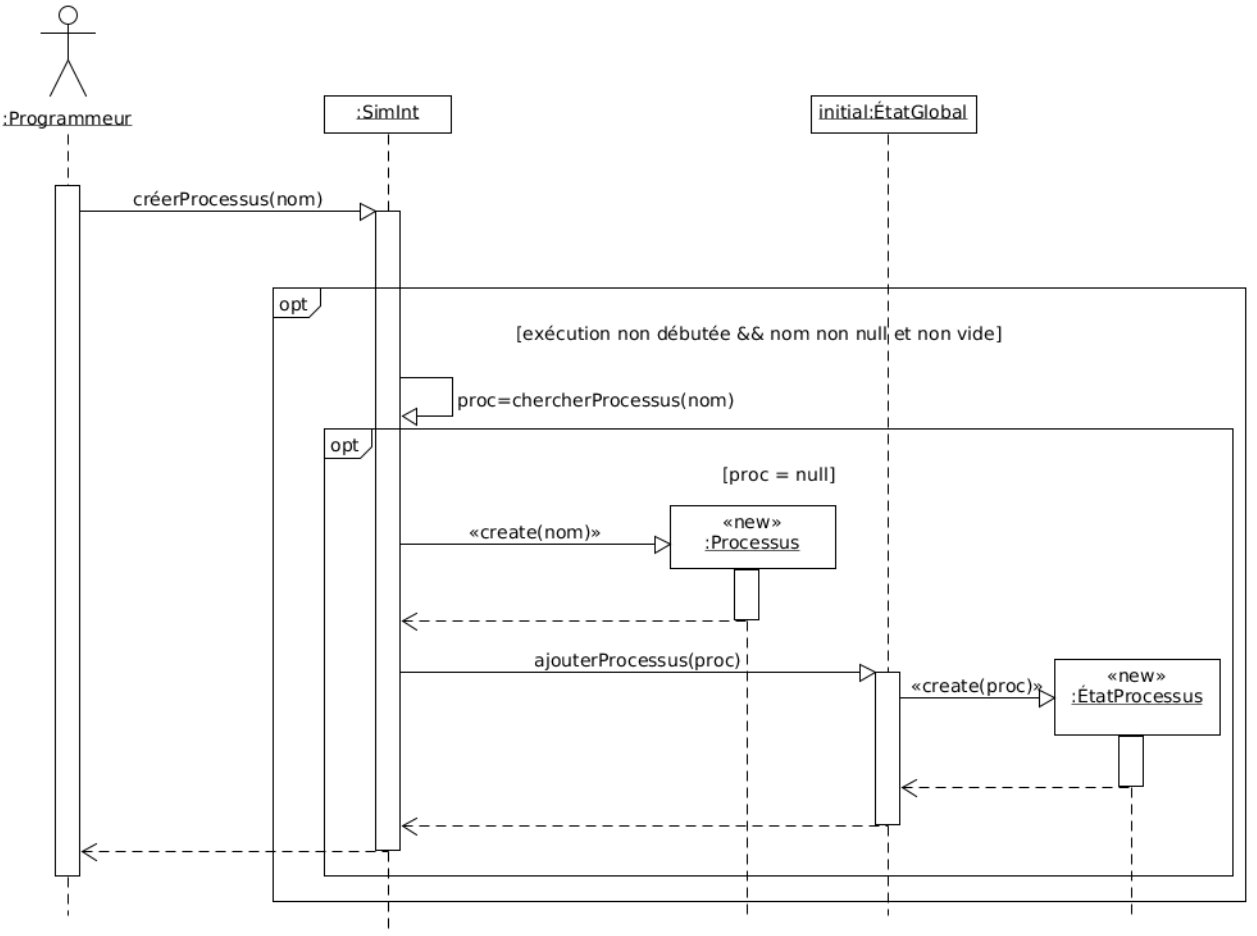

FIG. 4: Diagramme de séquence du cas d'utilisation «créer un processus»

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` ≠*null*∧≠*vide*	F	T
nom'≠*null*∧≠*vide*		T
invariant		T
Levée d'une exception	OUI	NON
Objet créé	F	T
Nombre de jeux de test	2	1

TAB. 2: Méthode `constructeurProcessus` de la classe `Processus`»

A Algorithmes des classes

Les fiches 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()