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

CHAFFARDON Pierre et DENIZE Julien

Année 2018–2019—12 février 2019

Table of Contents

1 Spécification....3
1.1 Diagrammes de cas d'utilisation....3
1.2 Priorités, préconditions et postconditions des cas d'utilisation....4

2 Préparation des tests de validation....6
2.1 Tables de décision des tests de validation....6

3 Conception....8
3.1 Liste des classes....8
3.2 Diagramme de classes....9
3.3 Diagramme d'objets....10
3.4 Diagrammes de séquence....11

4 Fiche des classes....17
4.1 Classe SimInt....17
4.2 Classe Processus....18
4.3 Classe ÉtatProcessus....19
4.4 Classe ÉtatGlobal....20

5 Diagrammes de machine à états et invariants....21
5.1 Classes Processus....21
5.2 Classes ÉtatProcessus....21
5.3 Classes ÉtatGlobal....21

6 Préparation des tests unitaires....22
6.1 Classe Processus....22

A Algorithmes des classes....23
A.1 Classe SimInt....23
A.2 Classe Processus....24
A.3 Classe ÉtatProcessus....25
A.4 Classe ÉtatGlobal....26

1 Spécification

1.1 Diagrammes de cas d'utilisation

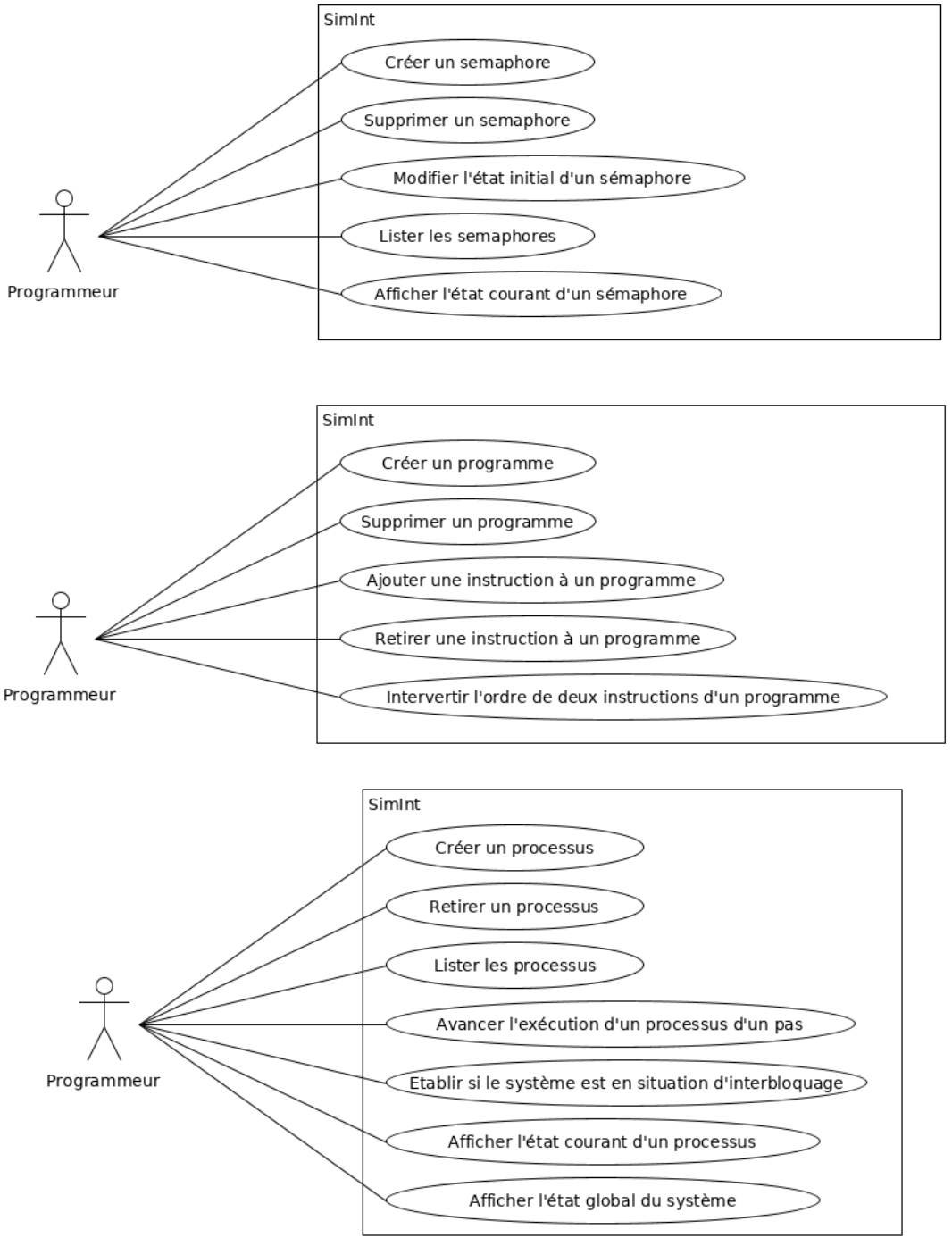

FIG. 1: Diagrammes de cas d'utilisation pour dans l'ordre: semaphore, programme, processus

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 ∧ non vide) ∧ semaphore avec ce nom inexistant ∧ valeur initiale du compteur bien formée (supérieur ou égale à 0) ∧ exécution non débutée
    - 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 ∧ non vide) ∧ programme avec ce nom inexistant ∧ exécution non débutée
    - 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 ∧ non vide) ∧ Type d'instruction P ou V ∧ nom semaphore manipulé bien formé (non null et non vide) ∧ semaphore manipulé avec ce nom existant ∧ nom programme bien formé (non null ∧ non vide) ∧ programme avec ce nom existant

    - postcondition : instruction ajoutée au programme
- Retirer une instruction à un programme : basse
- Intervertir l'ordre de deux instructions 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 ∧ processus vivant
    - postcondition : exécution du processus avancée d'un pas
- Etablir si le système est en situation d'interbloquage : HAUTE n°6
    - précondition : processus non terminés 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	4
Nom semaphore bien formé (≠ null∧ ≠ vide)	F	T	T	T	T
Semaphore inexistant avec ce nom		F	T	T	T
Valeur initiale du compteur bien formée (supérieur ou égale à 0)			F	T	T
Exécution non débutée				F	T
Création acceptée	F	F	F	F	T
Nombre de jeux de test	2	1	1	1	1

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

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

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

Numéro de test	1	2	3	4	5	6	7
Instruction bien formée (≠ null∧ ≠ vide)	F	T	T	T	T	T	T
Type d'instruction P ou V		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	1	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
Nom processus bien formé (≠ null∧ ≠ vide)	F	T	T	T
Processus avec ce nom existant		F	T	T
Processus vivant			F	T
Exécution du processus avancée d'un pas	F	F	F	T
Nombre de jeux de test	2	1	1	1

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

Numéro de test	1	3
Processus en cours d'exécution ou terminés existants	F	T
Situation d'interbloquage ou non établie	F	T
Nombre de jeux de test	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) — , situationInterbloquage
- ÉtatProcessus (la partie de l'état concernant les processus) — terminé, bloqué, instructionCourante
- Sémaphore – nom, valeurInitiale
- EtatSémaphore – valeurCompteur
- Programme – nom
- Instruction – numéro
- Instruction P
- Instruction V

3.2 Diagramme de classes

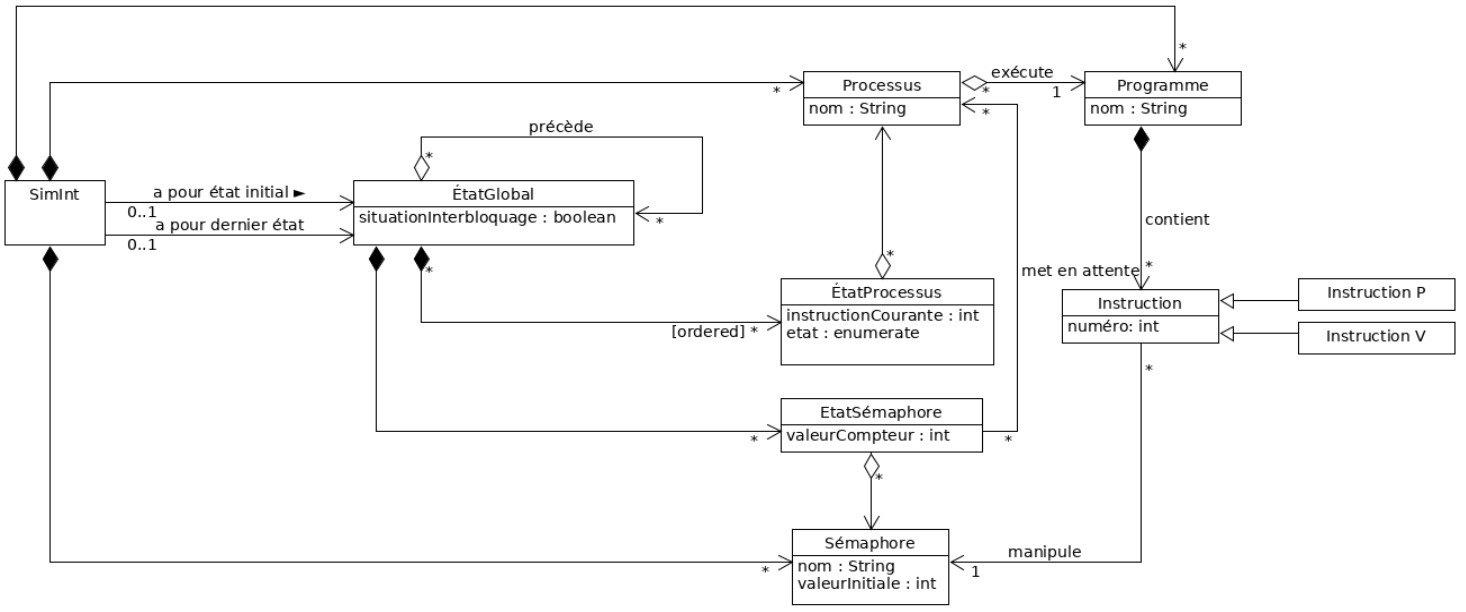

FIG. 2: Diagramme de classes

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.

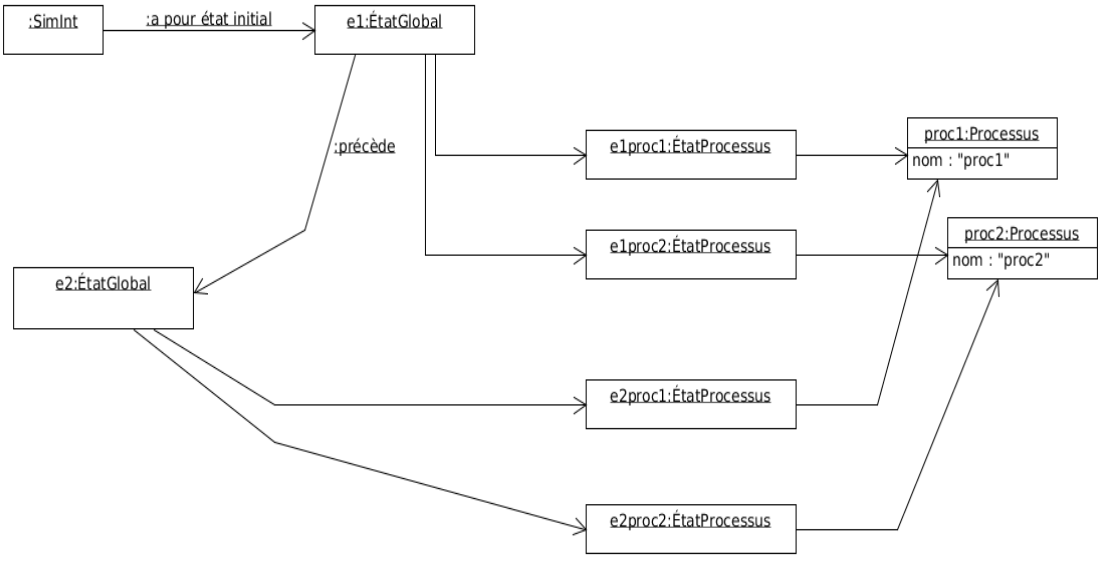

FIG. : Diagramme d’objets à partir de l’état initial: non mis à jour

3.4 Diagrammes de séquence

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

- arguments en entrée : nom du semaphore, valeur initiale du compteur
- rappel de la précondition : nom de semaphore bien formé (non null ∧ non vide) ∧ semaphore avec ce nom inexistant ∧ valeur initiale du compteur bien formée (supérieur ou égale à 0) ∧ exécution non débutée
- algorithme :
    1. vérifier les arguments
    2. si en outre l'exécution du système n'a pas débuté
        (a) vérifier que le semaphore n'existe pas
        (c) créer un sémaphore en initialisant son compteur avec la valeur initiale
        (d) ajouter le semaphore à la collection des semaphores

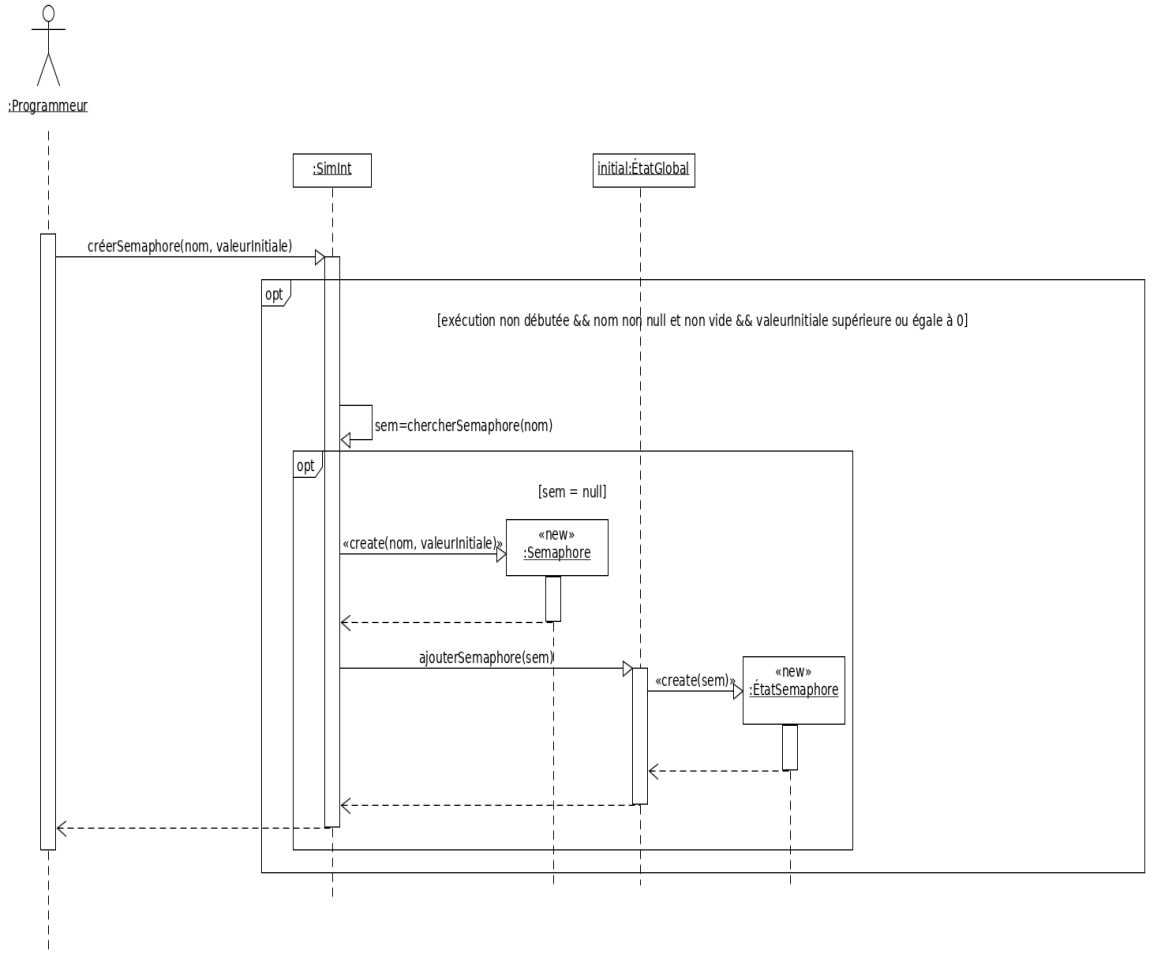

FIG. 4: Diagramme de séquence du cas d'utilisation «créer un semaphore»

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

- arguments en entrée : nom du programme
- rappel de la précondition : nom de programme bien formé (non null $\wedge$ non vide) $\wedge$ programme avec ce nom inexistant $\wedge$ exécution non débutée
- algorithme :
  1. vérifier les arguments
  2. si en outre l'exécution du système n'a pas débuté
     (a) vérifier que le programme n'existe pas
     (c) créer un programme
     (d) ajouter le programme à la collection des programmes

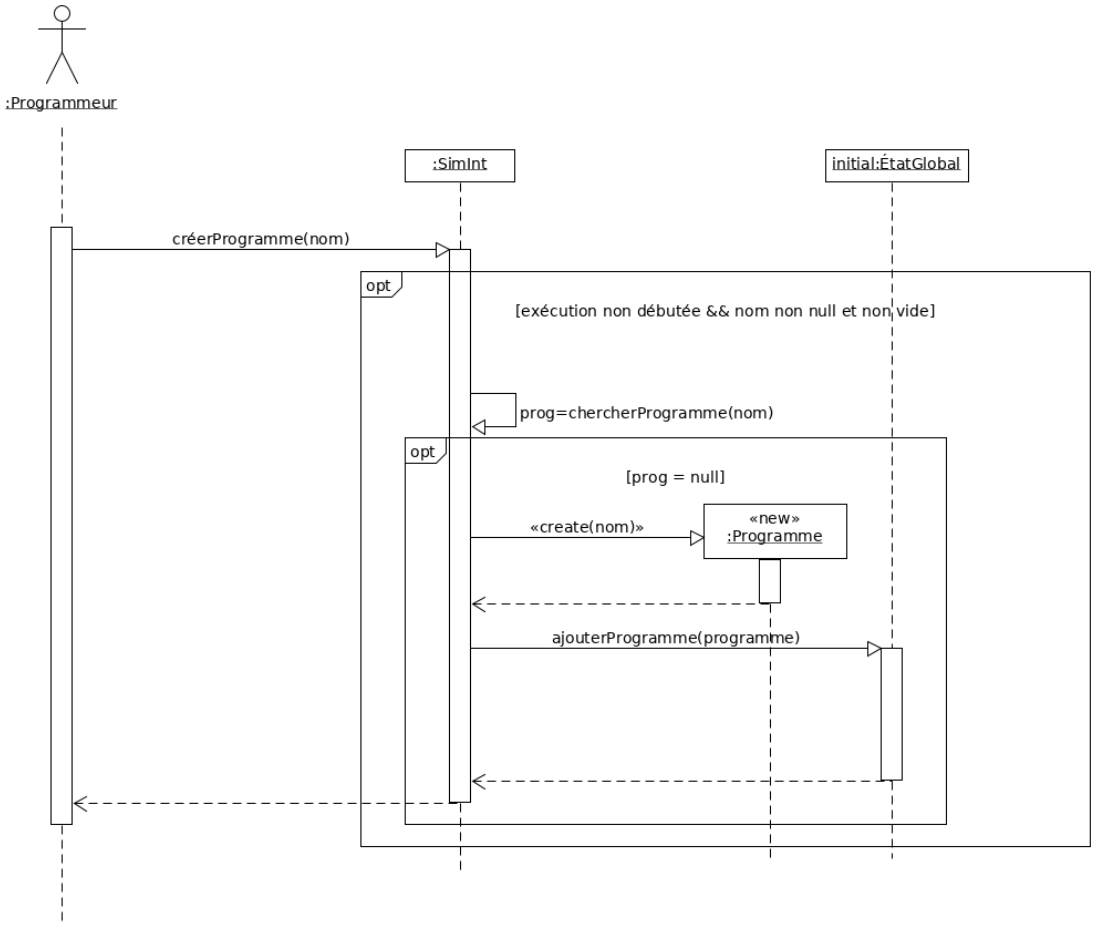

FIG. 5: Diagramme de séquence du cas d'utilisation «créer un programme»

Voici la description textuelle du cas d'utilisation « ajouter une instruction à un programme » :

– arguments en entrée : instruction, nom du semaphore, nom du programme

– rappel de la précondition : instruction bien formée (non null ∧ non vide) ∧ Type d'instruction P ou V ∧ nom semaphore manipulé bien formé (non null et non vide) ∧ semaphore manipulé avec ce nom existant ∧ nom programme bien formé (non null ∧ non vide) ∧ programme avec ce nom existant

– algorithme :

1. vérifier les arguments
2. si le semaphore existe et le programme existe

   a) compter le nombre d'instructions du programme

   b) ajouter l'instruction au programme en fonction de son type avec comme numéro d'instruction : nombre d'instructions + 1

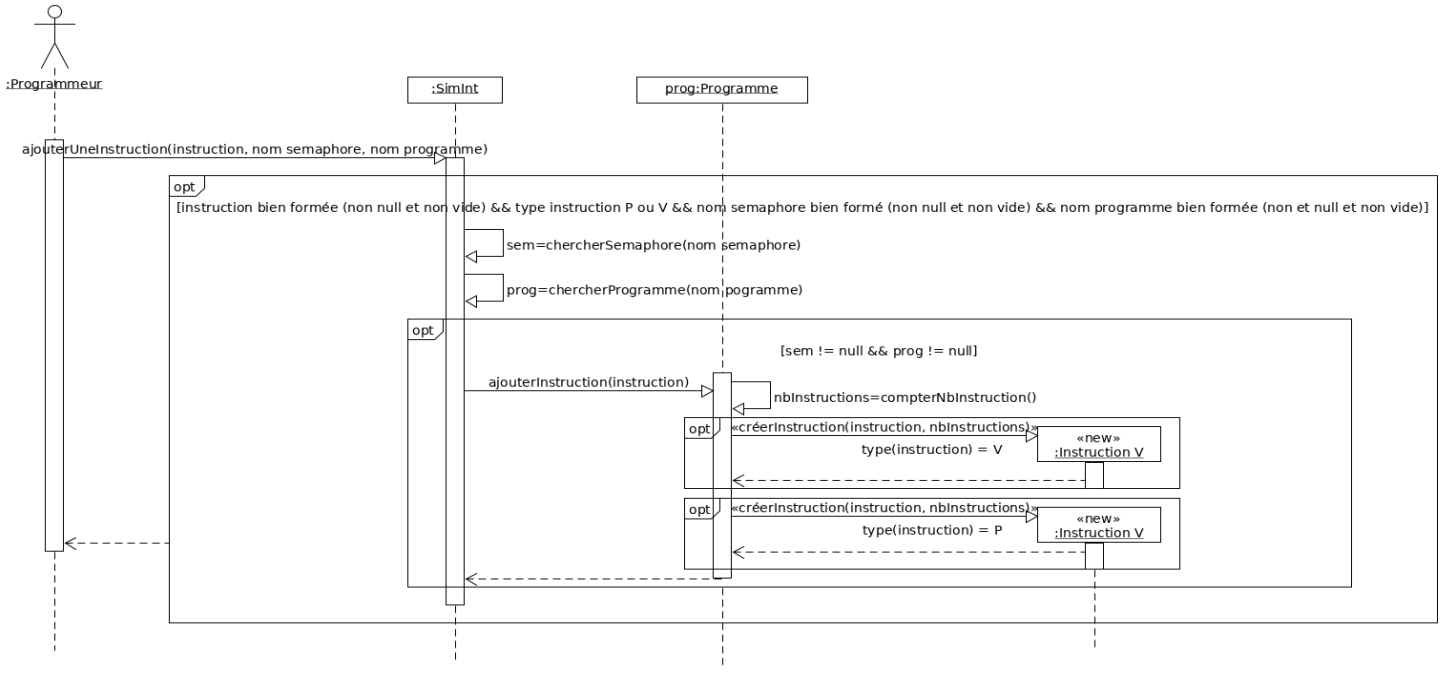

FIG. 6: Diagramme de séquence du cas d'utilisation « ajouter une instruction à un programme»

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 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
- 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) vérifier que le programme existe
        (c) créer un processus avec ce nom et la référence du programme
        (d) ajouter le processus à la collection des processus
        (e) 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

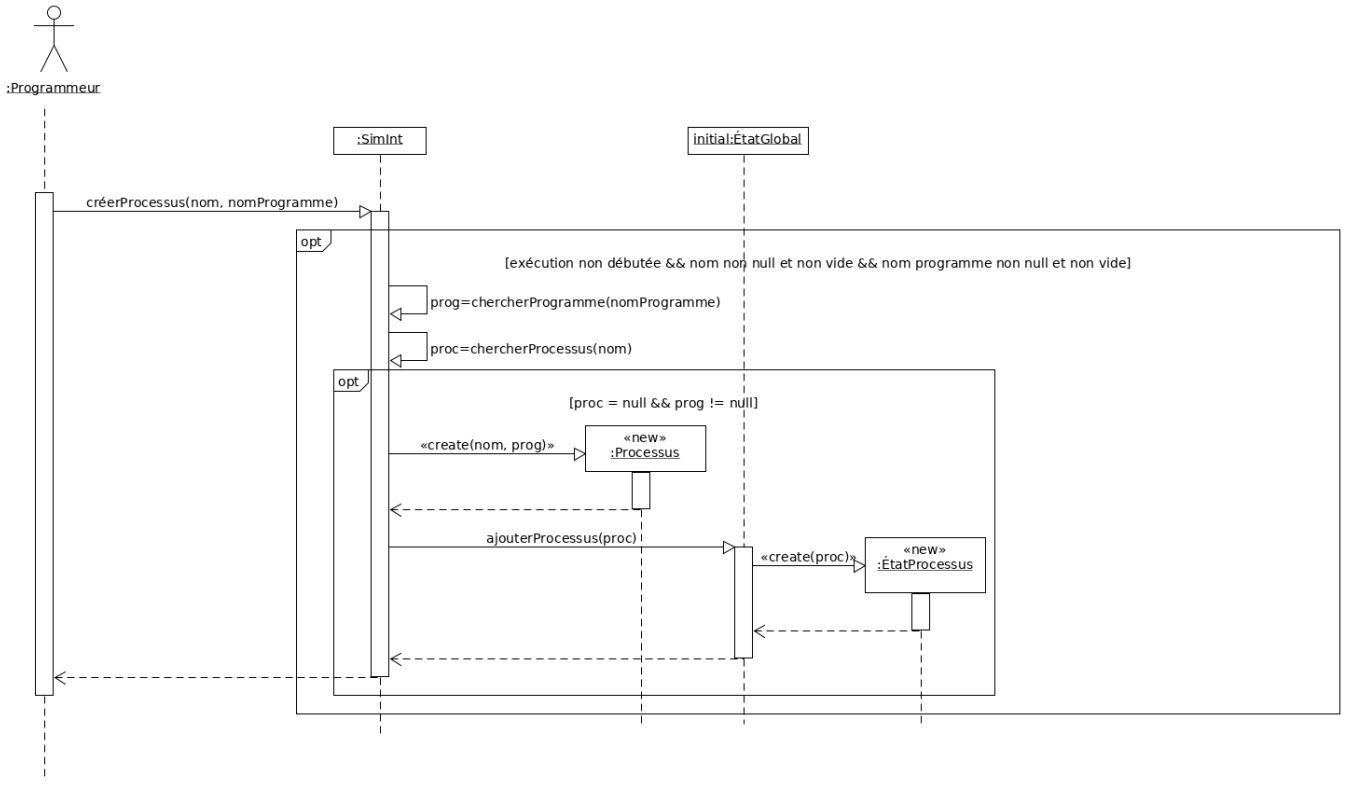

FIG. 7: Diagramme de séquence du cas d'utilisation «créer un processus»

Voici la description textuelle du cas d'utilisation « avancer l'exécution d'un processus d'un pas » :

– arguments en entrée : nom processus

– rappel de la précondition : nom processus bien formé (non null ∧ non vide) ∧ processus avec ce nom existant ∧ processus non bloqué ∧ processus non terminé

– algorithme :

1. vérifier les arguments
2. si le processus existe, cherche l'état de ce processus
3. si le processus esr vivant

   a) copie de l'état courant vers un nouvel état

   b) cherche l'état du processus copié

   c) cherche la prochaine instruction du programme exécuté par le processus

   d) exécute cette instruction

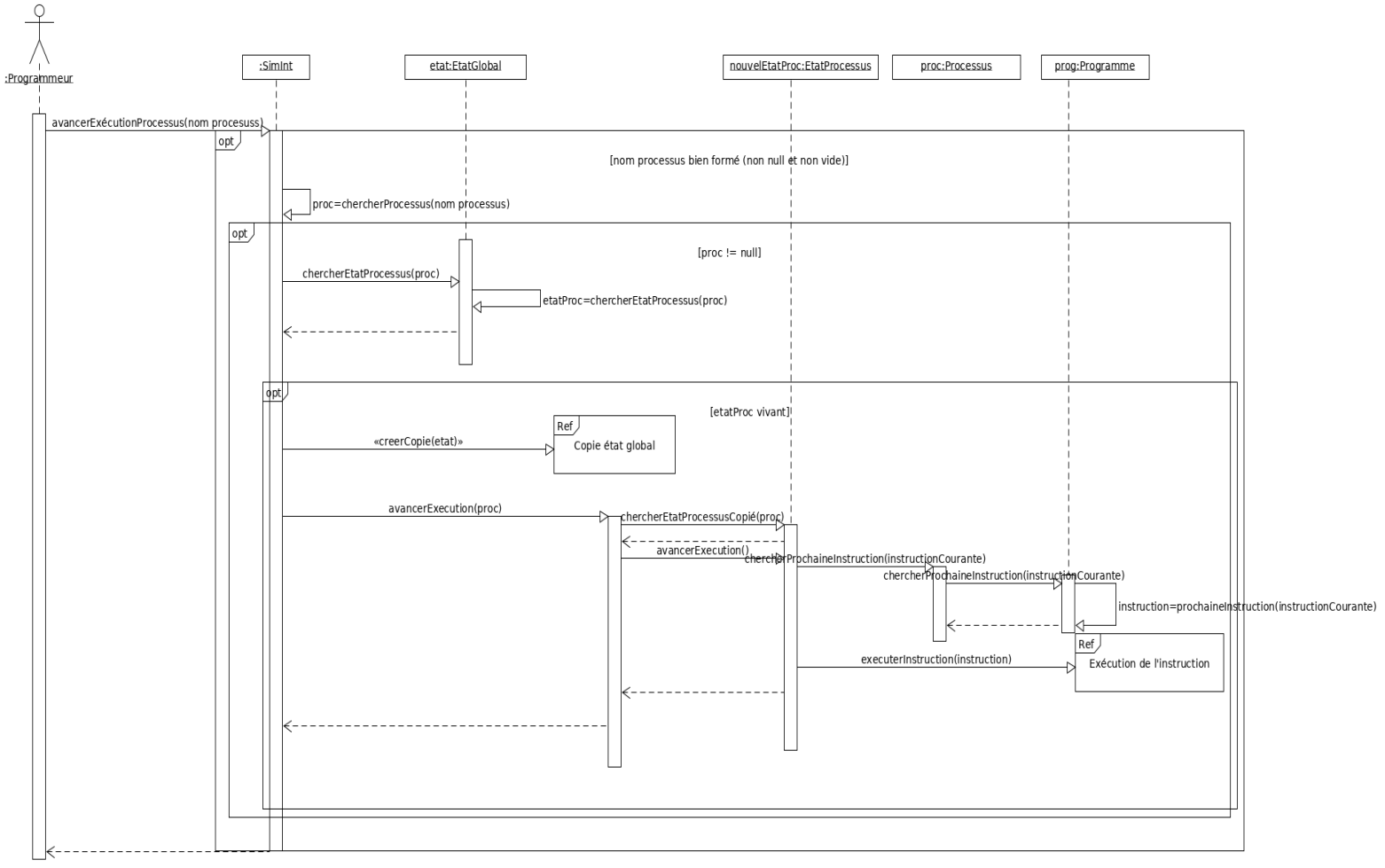

FIG. 8: Diagramme de séquence du cas d'utilisation «avancer l'exécution d'un processus d'un pas»

Voici la description textuelle du cas d'utilisation « établir si le système est en situation d'interbloquage » :

- rappel de la précondition : processus non terminés ou terminés existants
- algorithme :

1. chercher des processus non terminés ou terminés
2. si des processus non terminés ou terminés existent

a) établir si le système est en interbloquage

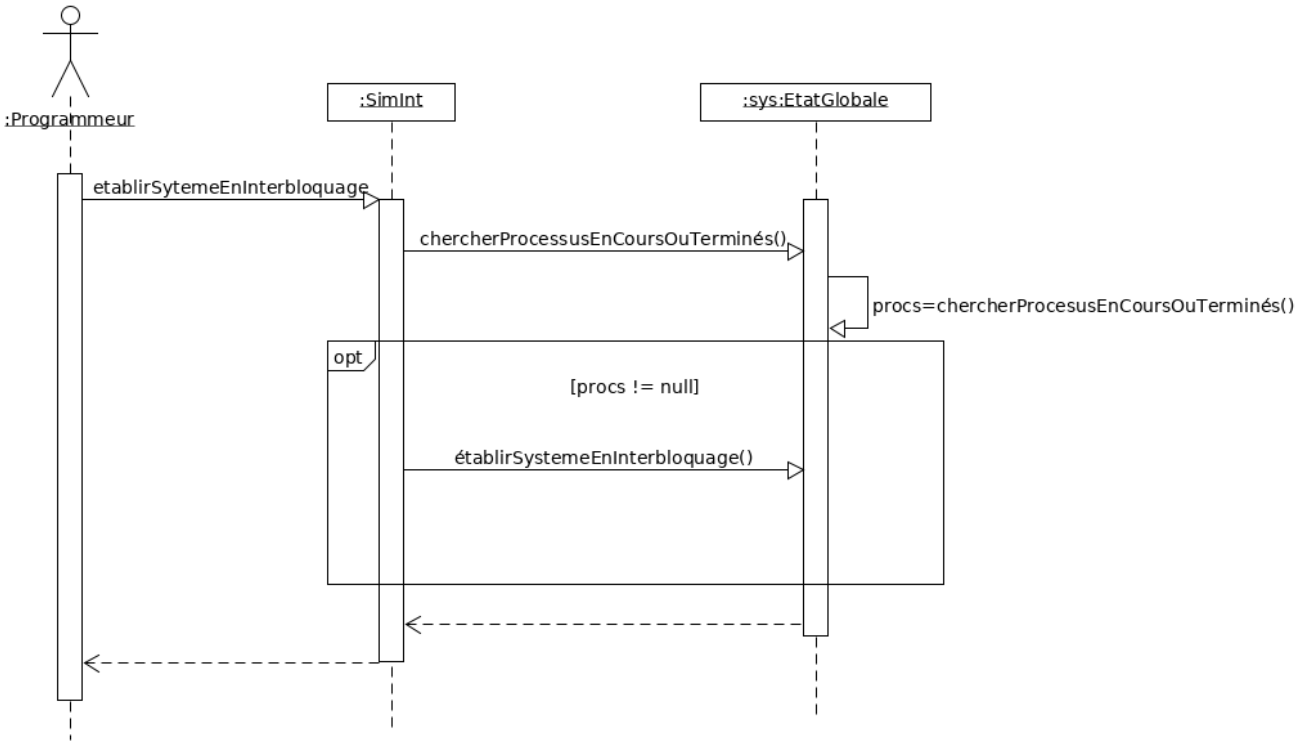

FIG. 10: Diagramme de séquence du cas d'utilisation «établir si le système est en situation d'interbloquage»

4 Fiche des classes

4.1 Classe SimInt

Simint
<– attributs « association » –> − processus : collection de @Processus − semaphores : collection de @Semaphore − programmes : collection de @Programme − étatGlobalInitial : @ÉtatGlobal − dernierEtatGlobal : @EtatGlobal − exécutionDébutée : booléen
<– constructeur –> + SimInt() <– operations « cas d'utilisation » –> + créerProcessus(String nom) : boolean + chercherProcessus(String nom) : @Processus + créerSemaphore(String nom, Integer valeurInitiale) : boolean + chercherSemaphore(String nom) : @Semaphore + créerProgramme(String nom) : boolean + chercherProgramme(String nom) : @Programme + établirSystemeInterbloquage(EtatGlobal etatGlobal) : boolean <– operations « nouveau cas d'utilisation »–> + listerProcessus() : void + listerSemaphore() : void + listerProgramme() : void + supprimerProcessus(String nom) : boolean + supprimerProgramme(String nom) : boolean + supprimerSemaphore(String nom) : boolean - existeProcessus(String nom) : boolean - existeProgramme(String nom) : boolean - existeSemaphore(String nom) : boolean

4.2 Classe Processus

Processus
<– attributs –> − nom : String <− attributs « associations » −> − programme : @Programme
<– constructeur –> + constructeurProcessus(String nom, String nomProgramme) + destructeur() <− opérations « cas d'utilisation » −> + chercherProchaineInstruction(Integer instructionAcutelle) : @Instruction + executerInstruction(Instruction instruction)

4.3 Classe ÉtatProcessus

ÉtatProcessus
<– attributs –> − processus : @Processus – etat : @Enumerate – instructionCourante : integer − compteurInstanciation : integer = 0 − compteurInstance : integer <− attributs « associations » −> – processus : @Processus
<– constructeur –> + constructeurProcessus(Processus processus) + constructeurProcessus(ÉtatProcessus étatProcessus) // constructeur par copie <– opérations « cas d'utilisation » –> + avancerExecution(instructionCourante) : boolean <– opérations « nouveau cas d'utilisation » –> + afficherEtatProcessus() : void

4.3 Classe Semaphore

Semaphore
<– attributs –> − nom : String − valeurInitiale : integer
<– constructeur –> + constructeurProcessus(String nom, Integer valeurInitiale) + destructeur()

4.4 Classe ÉtatSemaphore

ÉtatSemaphore
<– attributs –> – valeurCompteur : integer − compteurInstanciation : integer = 0 − compteurInstance : integer <− attributs « associations » −> – fileAttente : Collection de @Processus – semaphore : @Semaphore
<– constructeur –> + constructeurSemaphore(Semaphore semaphore) + constructeurProcessus(Semaphore semaphore) // constructeur par copie <– opérations « cas d'utilisation » –> + mettreEnAttenteProcessus(Processus processus) : void + libérerProcessus(Processus processus) : void <– opérations « nouveau cas d'utilisation » –> + afficherEtatSemaphore() : void

4.5 Classe Programme

Programme
<– attributs –> − nom : String <− attributs « associations » −> − instructions : Collection de @Instruction
<– constructeur –> + constructeurProcessus(String nom) + destructeur() <− opérations « cas d'utilisation » −> + ajouterInstruction(Instruction) : @Instruction + chercherInstruction(Integer numeroInstruction) : @Instruction <− opérations « nouveau cas d'utilisation » −> + supprimerInstruction(Integer numeroInstruction) : boolean + modifierOrdreInstruction(Instruction instruction1, Instruction instruction2) : boolean

4.6 Classe Instruction

Instruction
<– attributs –> − numero : Integer <− attributs « associations » −> − semaphore : @Semaphore
<– constructeur –> + constructeurProcessus(Semaphore semaphore, String typeInstruction) + destructeur()

4.7 Classe ÉtatGlobal

ÉtatGlobal
<– attributs –> − estÉtatGlobalInitial : boolean − estDernierEtat : boolean − étatsGlobauxAtteignables : collection de @ÉtatGlobal − situationInterbloquage : boolean − compteurInstanciation : integer = 0 − compteurInstance : integer <– attributs « association » –> − étatsProcessus : collection ordonnée de @ÉtatProcessus − étatsSemaphore : collection ordonnée de @ÉtatSemaphore
<– constructeurs –> + constructeurÉtatGlobal() + constructeurÉtatGlobal(ÉtatGlobal origine) // constructeur par copie <– operations « cas d'utilisation » –> + ajouterÉtatProcessus(Processus processus) : boolean + chercherÉtatProcessus(String nom) : @ÉtatProcessus + ajouterEtatSemaphore(Semaphore semaphore) : boolean + chercherEtatSemaphore(String nom) : @EtatSemaphore + etablirSituationInterbloquage() : boolean <– operations « nouveau cas d'utilisation » –> + listerEtatsProcessus() : void + listerEtatsSemaphore() : void

5 Diagrammes de machine à états et invariants

5.1 Classes Processus

L'invariant de la classe Processus est le suivant :

*nom≠null∧nom≠“"∧*programme≠null

5.2 Classes ÉtatProcessus

L'invariant de la classe ÉtatProcessus est le suivant :

processus≠null∧instructionCourante>0∧(etat=vivant V etat=bloqué V etat=terminé)

5.3 Classes ÉtatGlobal

L'invariant de la classe ÉtatGlobal est le suivant :

etatsProcessus≠null∧etatsSemaphore≠null∧(situationInterbloquage V ¬situationInterbloquage)

5.4 Classes ÉtatSemaphore

L'invariant de la classe ÉtatSemaphore est le suivant :

semaphore≠null∧valeurCompteur>=0

5.5 Classes Semaphore

L'invariant de la classe Semaphore est le suivant :

nom≠null∧nom≠“"∧valeurInitiale>=0

5.6 Classes Programme

L'invariant de la classe Programme est le suivant :

nom≠null∧nom≠“"

5.7 Classes Instruction

L'invariant de la classe Instruction est le suivant :

NumeroInstruction > 0 ∧(typeV V typeP)

6 Préparation des tests unitaires

6.1 Classe Processus

Numéro de test	1	2	3
nom ≠*null*∧≠*vide*	F	F	T
programme ≠ null		F	T
invariant			T
Levée d'une exception	OUI	OUI	NON
Objet créé	F	F	T
Nombre de jeux de test	2	1	1

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

6.2 Classe EtatProcessus

Numéro de test	1	2	3	4
processus≠null	F	T	T	T
instructionCourante>0		F	T	T
vivant V bloqué V terminé			F	T
invariant				T
Levée d'une exception	OUI	OUI	OUI	NON
Objet créé	F	F	F	T
Nombre de jeux de test	2	1	3	1

TAB. 3: Méthode `constructeurEtatProcessus` de la classe Etat`Processus`»

Numéro de test	1	2	3
instructionCourante>0	F	T	T
vivant		F	T
invariant			T
Levée d'une exception	OUI	OUI	NON
Objet créé	F	F	T
Nombre de jeux de test	1	1	1

TAB. 4: Méthode `avancerExecution` de la classe Etat`Processus`»

6.3 Classe Semaphore

Numéro de test	1	2	3
nom ≠*null*∧≠*vide*	F	T	T
valeurInitiale>=0		F	T
invariant			T
Levée d'une exception	OUI	OUI	NON
Objet créé	F	F	T
Nombre de jeux de test	2	1	1

TAB. 5: Méthode `constructeurSemaphore` de la classe `Semaphore`»

6.4 Classe EtatSemaphore

Numéro de test	1	2	3
nom ≠*null*∧≠*vide*	F	T	T
valeurCompteur>=*0*		F	T
invariant			T
Levée d'une exception	OUI	OUI	NON
Objet créé	F	F	T
Nombre de jeux de test	2	1	1

TAB. 6: Méthode `constructeurEtatSemaphore` de la classe Etat`Semaphore`»

6.5 Classe Programme

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

TAB. 7: Méthode `constructeurProgramme` de la classe `programme`»

6.6 Classe Instruction

Numéro de test	1	2	3
numeroInstruction > 0	F	T	T
typeV V typeP		F	T
invariant			T
Levée d'une exception	OUI	OUI	NON
Objet créé	F	F	T
Nombre de jeux de test	1	2	1

TAB. 8: Méthode `constructeurInstruction` de la classe `Instruction`»

6.7 Classe EtatGlobal

Numéro de test	1	2	3	4
etatsProcessus≠null	F	T	T	T
etatsSemaphore≠null		F	T	T
situationInterbloquage V ¬situationInterbloquage			F	T
invariant				T
Levée d'une exception	OUI	OUI	OUI	NON
Objet créé	F	F	F	T
Nombre de jeux de test	1	1	2	1

TAB. 9: Méthode `constructeurEtatGlobal` de la classe `EtatGlobal`»

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()