M1 : Ingénierie du Logiciel – SORBONNE UNIVERSITE

Examen Réparti 2eme partie

15 Janvier 2020 (2 heures avec documents : tous SAUF ANNALES CORRIGEES). Barème indicatif sur 21 points (donne le poids relatif des questions)

Questions de cours [4 Pts]

Répondez de façon précise et concise aux questions.

Question Cours (QC):

QC1) (1 point) Citer deux défauts du cycle en V présenté dans l'UE.

QC2) (1 point) Expliquer le rapport entre le cahier des charges et les tests de Validation, d'Intégration et Unitaires.

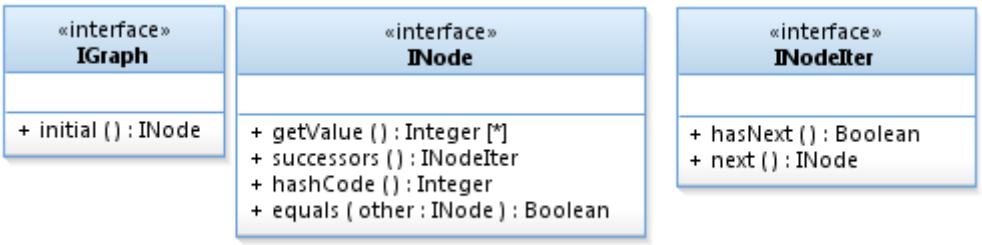
QC3) (1 point) Pourquoi demandes-t-on dans l'UE de ne pas placer les interfaces dans les packages qui réalisent les composants ?

QC4) (1 point) Pourquoi les diagrammes de structure interne sont-ils nécessaires en plus des diagrammes de composant ?

2. Problème: Conception DMC [Barème sur 17 Pts]

L'objectif de ce problème est de traiter la conception d'un outil permettant l'exploration de grands graphes.

I. Un graphe construit à la volée (4,5 points)

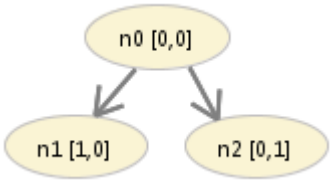


On considère la notion d'état ou nœud du graphe. Chaque nœud du graphe porte une valeur composée d'un vecteur d'entiers ; cette valeur sert aussi d'identifiant pour le nœud, pour une valeur donnée il n'y a qu'un seul nœud qui la porte. Les arcs orientés du graphe lient les nœuds entre eux : chaque nœud a des successeurs. L'opération *successors()* permet d'obtenir un itérateur positionné au début des successeurs. Les utilitaires standard *hashCode()* et *equals()* seront utilisés plus tard dans cet énoncé.

Les successeurs peuvent être obtenus en itérant sur un nœud ; on applique donc ici un DP *Iterator* aligné avec la syntaxe Java, *hasNext()* rend vrai s'il reste des successeurs et *next()* rend le successeur suivant et décale l'itérateur.

Un graphe est défini par son nœud initial, à partir duquel tous les nœuds du graphe sont accessibles en suivant des arcs.

On considère par exemple le début du graphe suivant, où n0 (portant la valeur [0,0]) est le nœud initial, qui a deux successeurs n1 (valeur [1,0]) et n2 (valeur [0,1]).



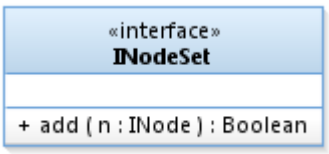
1. (1 point) On considère un composant qui doit explorer un graphe donné. Son objectif est de visiter tous les nœuds du graphe. Représentez ce composant **CExplorer** sur un diagramme de composant.

2. (2,5 points) Sur un diagramme de séquence, où chaque ligne de vie représente une instance d'une des trois interfaces définies, modélisez l'exploration par une instance de **CExplorer** des successeurs du nœud initial *n0* sur le graphe donné en exemple. On ne modélisera pas l'exploration des successeurs de ces successeurs.
3. (1 point) Expliquez en quoi cette définition du graphe permet potentiellement d'explorer des très grands graphes (voire des graphes infinis), où la capacité mémoire de la machine est insuffisante pour pré-calculer et stocker tous les noeuds.

II. Distributed Hash Table DHT (7,5 points)

Dans cette partie, on va développer une table de hash distribuée sur plusieurs machines. L'objectif est de permettre le stockage de l'ensemble des nœuds d'un grand graphe en répartissant le stockage sur N machines. Chaque machine du réseau va héberger un sous-ensemble des nœuds. Le principe d'une DHT est d'avoir une partition des nœuds en fonction de leur valeur de hash : la machine d'indice k hébergera les nœuds dont le hash modulo N vaut k.

On considère donc des composants permettant de stocker un ensemble de nœuds du graphe. Ils répondront à l'interface *INodeSet* suivante. L'opération *add()* permet d'ajouter un élément à l'ensemble, elle rend vrai si l'élément n'existait pas encore dans l'ensemble et faux s'il était déjà présent. On va construire une DHT implantant cette interface au fil des questions qui suivent.



1. (1 point) Sur un diagramme de composant, définir un composant basique **CHashNodeSet** qui permet le stockage des nœuds dans une table de hash.

2. (1 point) Sur un diagramme de séquence, modéliser l'ajout d'un nœud *n0* dans un **CHashNodeSet** initialement vide.
3. (1 point) On considère un composant **CSetMulti** qui réalise *INodeSet* et prépare la réalisation de la DHT. Ce composant s'appuie sur N occurrences de *INodeSet* et joue un rôle d'aiguillage : quand on ajoute un nœud au **CSetMulti**, il calcule d'abord sa valeur de hash modulo N, puis il ajoute le nœud à l'instance de *INodeSet* désignée par cet indice. Représenter ce composant sur un diagramme de composant.
4. (1 point) Modéliser sur un diagramme de séquence l'ajout d'un nœud *n0* à une occurrence de **CSetMulti** configurée pour s'appuyer sur deux occurrences (s0 et s1) de *INodeSet*.

Ajout du Réseau. On considère un couple de composants **CSetProxy** et **CSetStub** réalisant le pattern *Proxy Distant* pour un *INodeSet* arbitraire. On instancie un **CSetStub** pour chaque instance de *INodeSet* qu'on souhaite exposer au réseau, et un **CSetProxy** pour chaque *INodeSet* distant auquel on souhaite accéder.

On fait abstraction des échanges réseau à proprement parler (sockets, sérialisation) qui seront simplement représentés par une interface *INetwork* dont on ne précisera pas les opérations ; cette interface *INetwork* permet à un **CSetProxy** hébergé sur une machine A de transmettre une demande d'ajout d'un nœud à un

CSetStub hébergé sur une machine B. Le **CSetStub** transmet les demandes reçues du réseau vers un *INodeSet* local à la machine B.

5. (1 point) Modéliser ces composants **CSetProxy** et **CSetStub** sur un diagramme de composants.

Assemblage et configuration. On souhaite assembler tous ces éléments dans une configuration qui permet d'implanter une table de hash distribuée sur N machines. Chaque machine d'indice *k* va héberger une (petite) table de hash **CHashNodeSet** qui contient les éléments dont la clé de hash modulo N vaut k et qui est exposée au réseau. Sur la machine d'indice 0, on instancie aussi un **CSetMulti** configuré de manière à ce que son *INodeSet* d'indice 0 soit simplement le **CHashNodeSet** local à la machine, mais que les autres indices correspondent à des *INodeSet* qui sont des proxy permettant d'accéder aux données hébergées sur les autres machines.

6. (2,5 points) Sur un diagramme de structure interne, modéliser cette configuration en considérant que l'on a une table de hash distribuée sur deux machines d'indices 0 et 1. On précisera par des commentaires sur quelle machine chaque composant est instancié.

III. Une File de noeuds à explorer (5 points)

On considère le problème consistant à visiter l'ensemble des nœuds d'un graphe fini.

1. (1,5 point) Définir un composant **CQueue** capable de stocker des nœuds du graphe dans une file de nœuds à traiter : on doit pouvoir insérer un nouveau nœud dans la file et extraire le plus ancien. Sur un diagramme de composant, représenter ce composant et préciser la signature des opérations offertes par son interface offerte *IQueue*.
2. (1 point) On considère à présent un composant **CSetQueue**, il réalise l'interface *INodeSet* en appui sur une occurrence de **CHashNodeSet** (cf. partie II) et sur une occurrence de **CQueue**. Sa sémantique est la suivante : si le nœud inséré est nouveau, il est également ajouté dans la queue, sinon *add()* rend faux. Représenter ce composant sur un diagramme de composant.
3. (1,5 point) Sur un diagramme de séquence, modéliser un scenario où l'on insère deux fois le nœud *n0* dans un **CSetQueue** initialement vide. On se limitera aux messages dont la source ou la destination sont le **CSetQueue**, sans chercher à modéliser les interactions entre les autres composants.
4. (1 point) Modéliser une instanciation nominale de ces composants **CSetQueue**, **CHashNodeSet** et **CQueue** permettant d'offrir un *INodeSet* ayant le comportement décrit en question III.2.