IPST-CNAM
Intranet et Designs patterns
NSY 102
Vendredi 26 Avril 2013

Durée : **3 heures**
Enseignants : LAFORGUE Jacques

1ère Session NSY 102

1ère PARTIE – SANS DOCUMENT
CORRECTION

1. QCM (40 points) (1h)

Mode d'emploi :

Ce sujet est un QCM dont les questions sont de 3 natures :

- **les questions à 2 propositions**: dans ce cas une seule des 2 propositions est bonne.
  - +1 pour la réponse bonne
  - –1 pour la réponse fausse
- **les questions à 3 propositions** dont 1 seule proposition est bonne
  - + 1 pour la réponse bonne
  - –½ pour chaque réponse fausse
- **les questions à 3 propositions** dont 1 seule proposition est fausse
  - + ½ pour chaque réponse bonne
  - –1 pour la réponse fausse

Il s'agit de faire une croix dans les cases de droite en face des propositions.

On peut remarquer que cocher toutes les propositions d'une question revient à ne rien cocher du tout (égal à 0).

Si vous devez raturer une croix, faites-le correctement afin qu'il n'y ait aucune ambiguïté.

N'oubliez pas d'inscrire en en-tête du QCM, votre nom et prénom.

Vous avez droit à **4 points** négatifs sans pénalité.

NOM:	PRENOM:

Un Middleware est :		**1**
1	dans les architectures web, un framework, comme eclipse, d'aide au développement, à la mise au point et au déploiement des logiciels basés sur une architecture répartie	
2	dans une architecture client-serveur, une couche logicielle, utilisée par le client et le serveur pour communiquer par exemple par envoi/réception de message	X
3	dans une architecture répartie, un ORB (Object Request Broker) assurant la communication entre les différentes entités du réseau	X

Une application dite "distribuée" est une application logicielle dans lequel les données informatiques sont réparties sur le réseau et accessibles par tout logiciel qui utiliserait un ORB		**2**
1	OUI	X
2	NON	

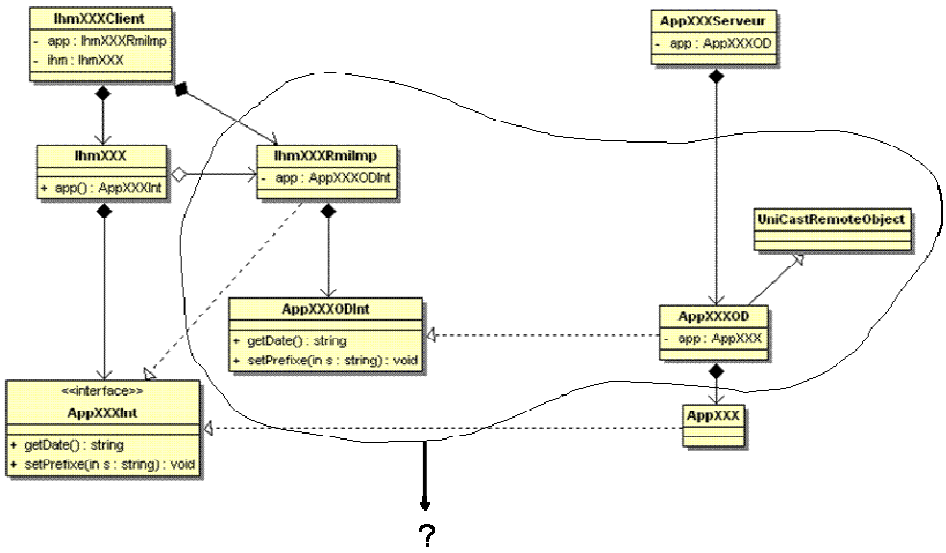
Pour la conception d'une architecture logicielle Intranet, la technologie CORBA n'est pas bien adaptée		**3**
1	OUI	
2	NON	X

L'IDL (Interface Definition Language) permet de créer les souches et les squelettes dans différents langages informatique assurant ainsi l'interopérabilité des services entre eux		**4**
1	OUI	X
2	NON	

Les composants d'un ORB (Object Request broker) sont :		**5**
1	Une interface Java, la classe UnicastRemoteObject, la classe LocateRegistry	X
2	Eclipse, JDK, Apache	
3	Une API (fonctions de base de l'ORB), un service de nommage, un compilateur IDL	X

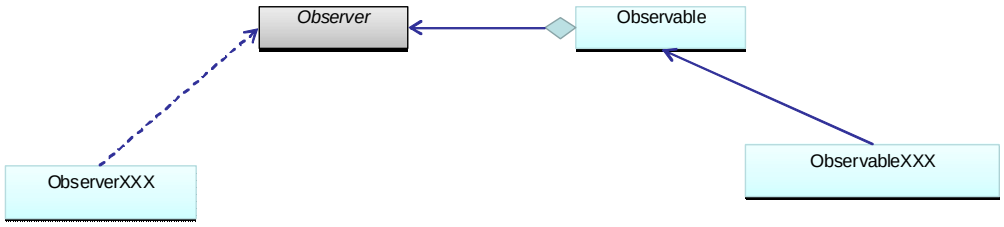
On appelle un objet "distribué" quand ce dernier est passé en paramètre d'une méthode distante (ou méthode Remote)		**6**
1	OUI	
2	NON	X

En RMI de Java,		**7**
1	la classe d'appartenance d'un objet distribué, hérite de UnicastRemoteObject et implémente une interface qui décrit les méthodes distantes	X
2	la classe d'appartenance d'un objet distribué, hérite de RemoteObject et implémente l'interface Remote	

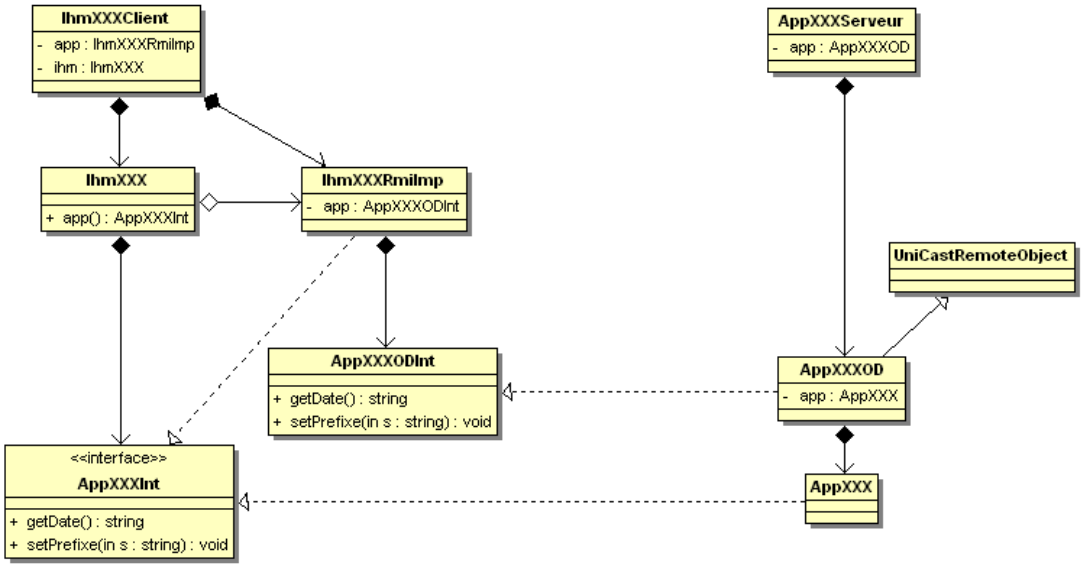
Ceci est le schéma d'architecture de l'atelier 16 (exemple 04) en RMI qui représente une application composé d'un client IHM et de son serveur.		**8**
		
La partie entourée constitue un pont RMI de communication entre l'IHM et l'applicatif		
1	OUI	X
2	NON	

Le rôle d'un factory est, entre autre, de créer à la demande de nouveaux objets distribués. Tous ces objets distribués doivent être enregistrés dans un adaptateur local à la machine sinon ils ne seraient pas accessibles.		**9**
1	OUI	
2	NON	X

CORBA (Common Object Request Broker Architecture) est une norme de Middleware		**10**
1	OUI	X
2	NON	

Soit le Design Pattern Observateur :		**11**
		
1	La classe ObservableXXX notifie les évènements à une instance de Observable	
2	La classe ObserverXXX implémente la méthode update de l'interface Observer qui est appelée par Observable	X
3	La classe Observable pousse (modèle du "push") les évènements à ObserverXXX	X

L'indépendance de la situation géographique d'un objet distribué passe par l'utilisation d'une table de correspondance physique/logique des objets distribués (appelé annuaire ou adaptateur) :		**12**
1	La partie physique est un pointeur sur l'objet distribué en mémoire de la JVM La partie logique est un nom unique	
2	La partie physique est constituée des éléments de connexion à l'objet distribué La partie logique est un nom unique dans un contexte donné	X



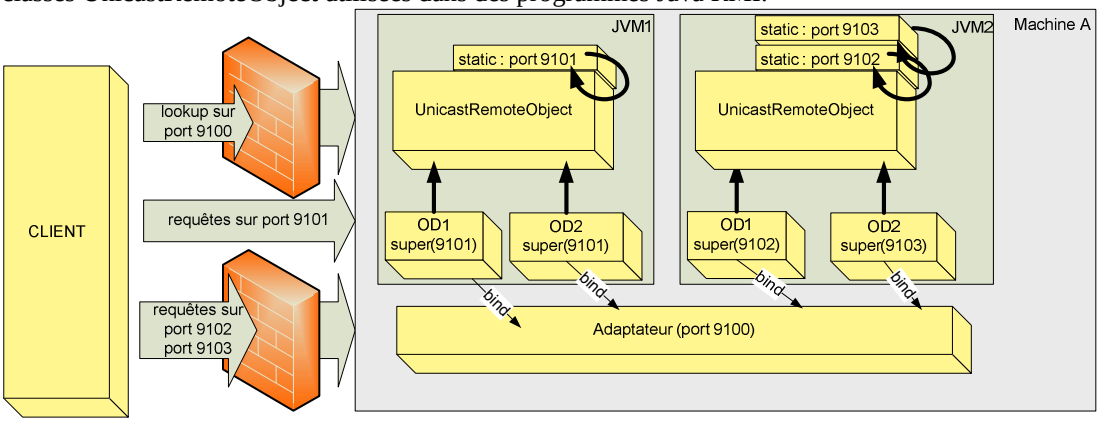
	Cette représentation de modèle de classes en UML est celle de l'atelier 16 en RMI. Le rôle de l'interface AppXXXODInt est essentiel car :	**13**
1	Elle décrit les méthodes que la classe IhmXXXRmiImp doit implémenter	
2	Elle est une interface RMI, une description des méthodes distantes de AppXXXOD	X
3	Elle décrit les méthodes distantes que l'objet distribué AppXXXOD doit implémenter	X

	En JAVA, avec RMI, plusieurs clients d'un objet distribué peuvent utiliser en parallèle une même méthode distante	**14**
1	OUI	X
2	NON	

	Le ClientProxy est un DP Proxy utilisé par un client, dans lequel les services réels ont été remplacés par un appel distant à ces services	**15**
1	OUI	X
2	NON	

	En RMI de Java, les paramètres des méthodes distantes peuvent être de n'importe quelle classe qui implémente sérializable	**16**
1	OUI	X
2	NON	

Soit le schéma suivant qui représente un fonctionnement possible de plusieurs serveurs de socket des classes UnicastRemoteObject utilisées dans des programmes Java RMI.		**17**



1	On peut créer un nouvel OD dans la JVM1 qui s'exécute sur le port 9102	
2	On peut créer une nouvelle JVM3 dans laquelle, on crée un nouvel OD qui s'exécute sur le port 9103	
3	Dans la JVM2, on peut créée un nouvel objet distribué RMI sur le port 9102	X

En RMI, l'appel d'une méthode distante, entre un client et un objet distribué RMI se fait de la manière suivante : 1/ l'appel de méthode est converti en une requête qui encode les paramètres 2/ la requête est envoyée à l'adaptateur RMI qui la renvoie au serveur dont il a les coordonnées réseau		**18**
1	OUI	
2	NON	X

A l'opposé de la communication synchrone, la communication asynchrone est un type de communication basé sur le modèle du pull, comme par exemple un thread d'un client qui tire régulièrement les événements d'un serveur		**19**
1	OUI	X
2	NON	

Les descriptions suivantes sont des modèles de communication asynchrones :		**20**
1	un serveur pousse ses évènements dans une file d'attente par client connecté (intermédiaire), et les clients tirent ses évènements à leur rythme	X
2	un serveur pousse son évènement dans un proxy de consommateur, et à son tour, le proxy de consommateur pousse l'évènement au consommateur	
3	un serveur appelle la méthode distante d'un client afin de lui transmettre l'évènement	

Il existe deux façons pour utiliser les méthodes distantes d'un objet distribué : 1/ demander le stub de connexion à un annuaire, 2/ demander le stub de connexion à un factory qui a créé l'objet distribué ; puis d'utiliser ce stub pour appeler les méthodes distantes		**21**
1	OUI	X
2	NON	

Dans CORBA, l'IOR (Interface Object Request) est une chaîne de caractère qui est :		**22**
1	une chaîne de caractère permettant de créer le stub de connexion qui permet à sont tour d'appeler les méthodes distantes	X
2	le nom de l'objet distribué qui permet d'obtenir, via l'annuaire, le stub de connexion qui permet à sont tour d'appeler les méthodes distantes	

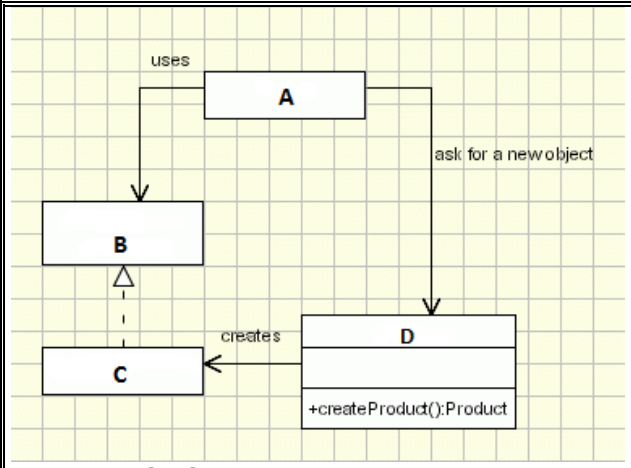
Dans le cadre de la communication entre un composant Java et un composant C++ sur un bus CORBA		**23**
1	on doit créer un socket de communication dans chacun des composants pour les faire communiquer	
2	on peut exécuter les deux composants sur la même machine	X
3	on définit un IDL qui réalise une projection Java et une projection C++ des composants logiciels utilisés pour faire communiquer les deux composants	X

En Java RMI, l'instruction rebind consiste à établir une connexion socket entre l'adaptateur et le serveur et l'instruction lookup consiste à établir une connexion socket entre le client et l'adaptateur. La communication entre le client et le serveur peut ainsi s'établir, l'adaptateur servant d'intermédiaire pour chaque requête.		**24**
1	OUI	
2	NON	X

Un Design Pattern (DP) ou Patron est une norme de description des interfaces entre les composants d'un architecture logicielle orientée objet		**25**
1	OUI	
2	NON	X

Un DP définit des principes de conception, et non des implémentations spécifiques de ces principes		**26**
1	OUI	X
2	NON	

Le DP Factory a pour fonction :		**27**
1	la création à distance d'objet distribué	X
2	la création d'objet tous décrit par la même interface	X
3	la création d'objet qui sont des singletons	



Ce DP est celui du Factory. La signification des lettres A, B, C et D est :		**28**
1	A=Factory; B = Concrete Product; C=Product (Interface); D=Client	
2	A=Client; B=Factory; C=Product (interface); D=Concrete Product	
3	A = Client; B=Product (interface); C=Concrete Product; D = Factory	X

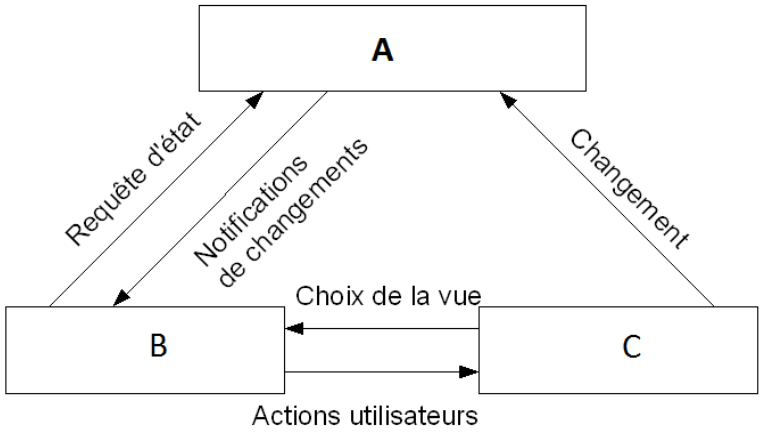
Dans la conception d'un factory, il est également envisagé que toutes les classes d'appartenance des produits créés par le factory, héritent toutes d'une même classe abstraite.		**29**
1	OUI	X
2	NON	

Le DP Builder est utilisé par le DP Factory afin de faciliter la production d'un objet		**30**
1	OUI	X
2	NON	

Le DP Adaptateur correspond à une classe qui sert d'intermédiaire entre un appelant et un appelé qui sont incompatibles entre eux		**31**
1	OUI	X
2	NON	

L'adaptateur et l'adapté implémente la même interface		**32**
1	OUI	
2	NON	X

33



Ce schéma représente le DP Modèle-Vue-Controleur.
Les lettres A, B et C sont définies ainsi :

1	A = Modèle; B=Vues ; C= Controleur	X
2	A=Controleur; B=Vues; C=Modèle	

Un Proxy est un DP dans lequel deux classes A et B implémentent la même interface, et A est composée de B		**34**
1	OUI	X
2	NON	

Un service est un comportement défini par un contrat, qui peut être implémenté et fourni par un composant afin d'être utilisé par un autre composant sur la base exclusive du contrat		**35**
1	OUI	X
2	NON	

Le DP Observateur est implémentée en Java via la classe Observable et l'interface Observer. Cette implémentation utilise par conception le modèle de communication synchrone suivant :		**36**
1	modèle du pull	
2	modèle du push	X
3	modèle du push-and-pull	

La communication synchrone entre un producteur et un consommateur par "Canal d'évènement" se fait :		**37**
1	via le modèle du "invoke" en passant par un intermédiaire	
2	via le modèle du "Push" en passant par un intermédiaire	X
3	via le modèle du "Pull" en passant par un intermédiaire	X

Dans la communication synchrone via un "canal d'évènement" entre un producteur et un consommateur, le producteur utilise un proxy de consommateur (et non le consommateur directement), afin de lui pousser un évènement		**38**
1	OUI	X
2	NON	

Le DynamicProxy est un DP qui permet d'exécuter dynamiquement une classe Java (Le ClassLoader de Java est implémenté suivant ce DP)		**39**
1	OUI	
2	NON	X

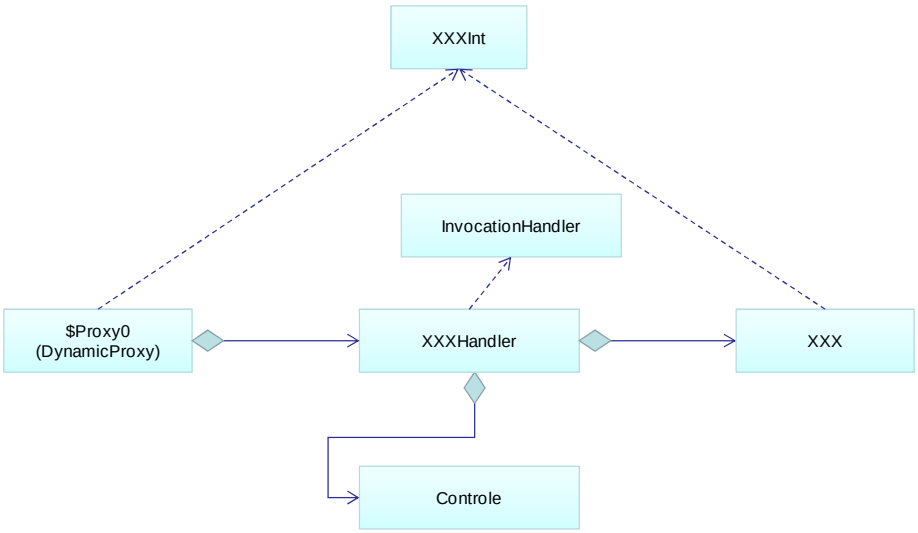
Le DynamicProxy est utilisé dans l'implémentation de RMI en Java 1.7, pour réaliser les appels distants d'un objet distribué		**40**
1	OUI	X
2	NON	

Suite (Tournez la page)

2. Questions libres (20 points) (30mn)

Chaque question est notée sur 5 points.
*Vous répondez à ces questions sur une **copie vierge double** en mettant bien le numéro de la question, sans oublier votre nom et prénom.*
Vous mettez le QCM dans la copie vierge double.

QUESTION NUMERO 1



Commentez le schéma ci-dessus (Définition, principes, rôles de chaque composant, ..)

Ce schéma représente le Design Pattern dit "Dynamic Proxy" dont le principe est la création de la classe $Proxy0 qui implémente toutes les méthodes de l'interface XXXInt. Cette classe est créée par invocation du langage Java et donc dynamiquement.
Dans le $Proxy0, le traitement réalisé pour chaque méthode est l'appel de la méthode "invoke" de la classe XXXHandler, en lui passant en paramètre, la méthode concernée.
A la charge du handler de réaliser l'exécution ou non de la méthode proprement dite en utilisant la classe XXX.
La classe Contrôle est utilisée pour utiliser des données nécessaires à l'exécution du handler (test, filtre, journalisation, trace, …).

La classe XXX est la classe réelle d'exécution des méthodes court-circuitées par le Proxy.
L'interface InvocationHandler contient la signature de la méthode "invoke" utilisée par le Proxy.

Dans quel cas rencontre-t-on l'utilisation d'un DynamicProxy. Expliquez.

On rencontre l'utilisation du DynamicProxy dans l'implémentation du middleware natif Java RMI : les méthodes distantes d'un objet distribué sont, côté client, implémentées par un DynamicProxy qui utilise un handler spécifique à RMI. Cet handler traduit chaque appel de méthode par une écriture socket et se charge de la sérialisation des paramètres.

QUESTION NUMERO 2

Nous avons sur le réseau un objet distribué RMI (Java), appelé XXOD, qui doit notifier des évènements à une IHM distante, appelée XXIhm. Pour cela nous faisons le choix d'utiliser le design pattern Observateur.

Expliquez ce qu'il est nécessaire de faire pour que la notification se fasse de manière distante entre XXOD et XXIhm.

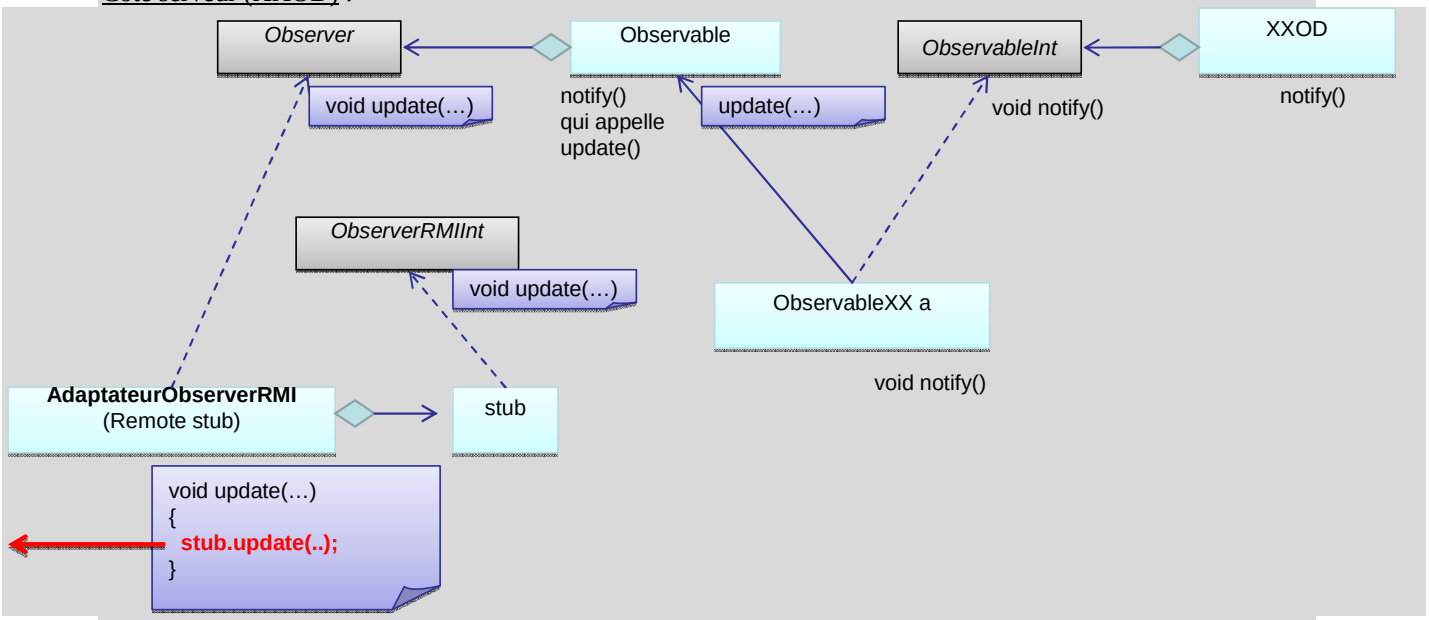
Il faut découpler le DP Observateur classique en un Observateur RMI dans lequel l'appel de la méthode « update » de l'observateur (« Observer ») se fait de manière distante par l'observé (« Observable).
Cela revient à mettre un Proxy RMI entre l'Observeur et l'Observé.

Pour cela, XXIhm crée un Objet Distribué (ProxyObserverIhmRMI) qui est un Proxy de l'observer naturel de l'IHM et qui implémente la méthode « update ». Ensuite l'IHM appelle la méthode distante « addObserver » de XXOD afin de s'abonner et lui envoyer le stub de ProxyObserverIhmRMI.

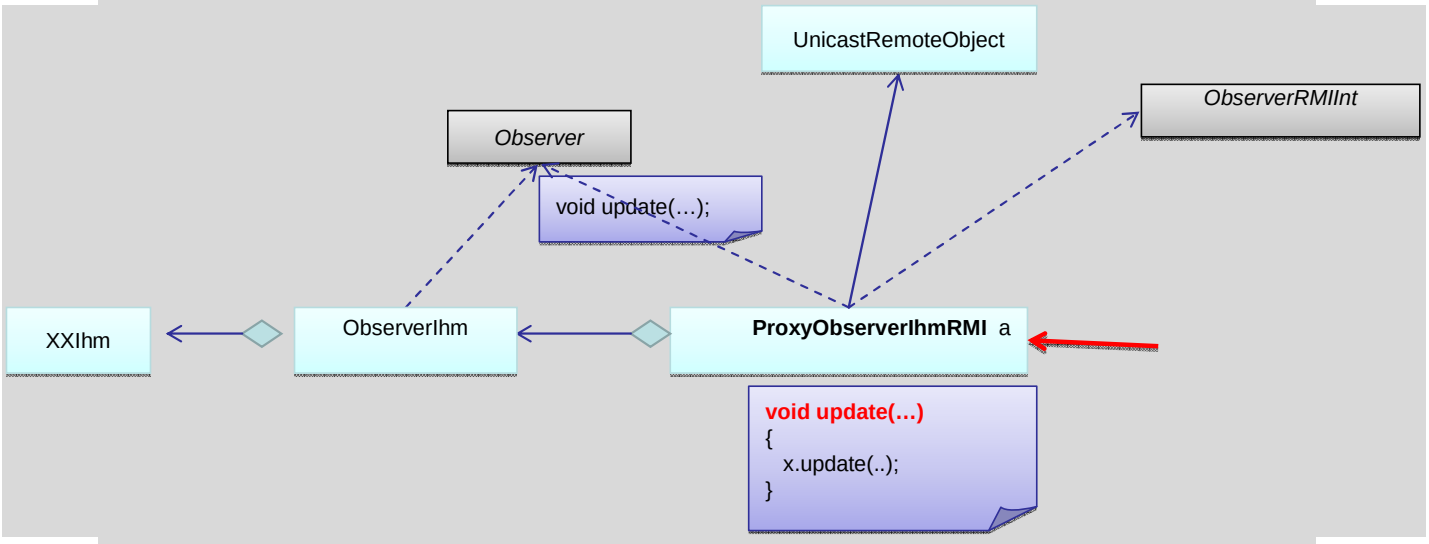
Ensuite, on crée une sorte d' « Adaptateur » (AdaptateurObserverRMI) qui se substitue à l'observer naturel afin qu'il soit notifié par l' « Observable » de XXOD. Cet observer appelle la méthode distante « update » du Proxy de l'IHM.

Décrivez sous la forme UML votre choix de conception et de Design Pattern.

Côté serveur (XXOD) :



Côté Client (XXIhm) :



QUESTION NUMERO 3

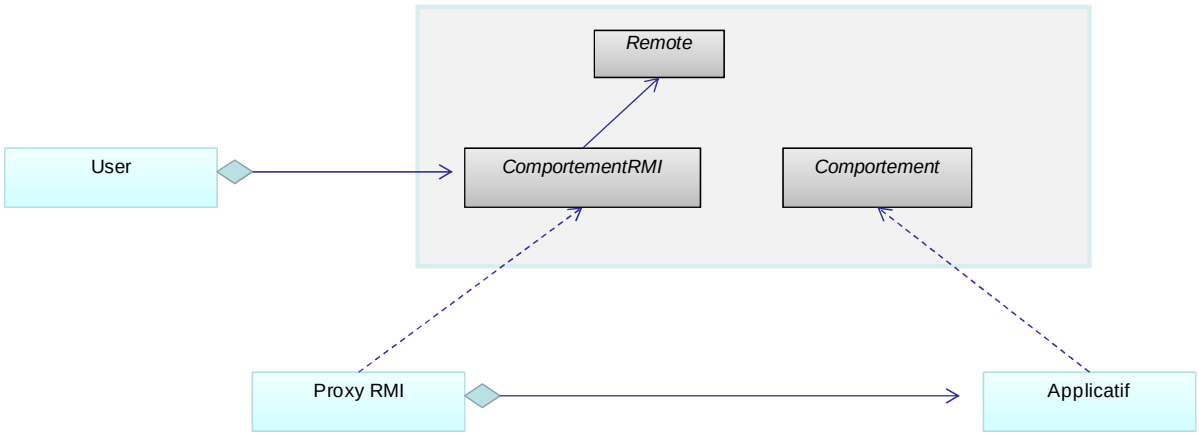
Nommez les avantages de l'utilisation d'un middleware basé sur la norme CORBA.

Les avantages de l'utilisation sont :

- la richesse des services déjà implémentées (service de nommage, service de réplication, service de persistance, service de notification synchrone et asynchrone, …)
- la compatibilité des interfaces de communication dans différents langages informatiques grâce à l'utilisation d'un IDL
- tous les services prédéfinis sont vus comme des services répartis sur le réseau (accessibles par tous (multi-plateforme, multi-language)
- l'utilisation du principe de l'IOR sous la forme d'une chaine de caractère qui contient les éléments de connexion et de localisation d'un objet distribué quelque soit le langage ou l'OS.

Suite (Tournez la page)

QUESTION NUMERO 4



Commentez ce schéma.

Ce schéma représente la conception de l'utilisation à distance d'un applicatif pour lequel les méthodes décrites par l'interface Comportement sont implémentée dans un objet distribué qui est représentée par la classe ProxyRMI. Cette classe implémente l'interface ComportementRMI qui contient les mêmes méthodes que Comportement.

L'utilisateur utilise l'interface ComportementRMI (via un lookup dont le stub implémente l'interface ComportementRMI) pour appeler les méthodes distantes.

(Fin de la 1ère partie sans document)

Si vous rendez la copie de cette 1ère partie, vous pourrez commencer la 2ème partie.

2ème PARTIE – AVEC DOCUMENT

3. PROBLEME (60 points) (1h 30mn)

Nous envisageons de réaliser un système d'information Intranet composé d'objets dit "objets de supervision" et d'une supervision unique appelée "supervision centralisée".

Les objets de supervision sont créés par un factory appelé "Factory d'instance de supervision". Il peut donc exister plusieurs Factory sur le réseau.

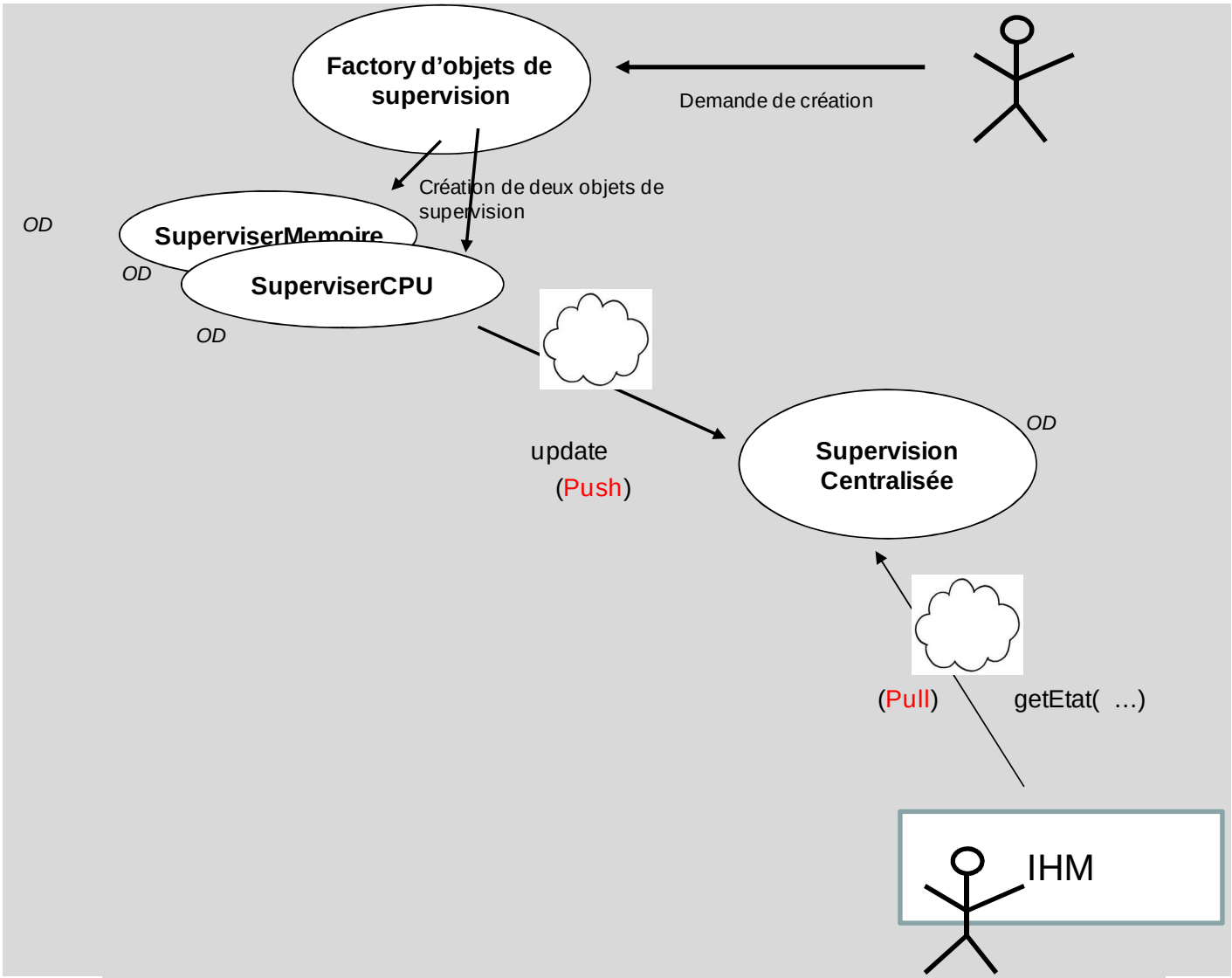
Un objet de supervision est un objet distribué RMI qui exécute un thread générique : la méthode run appelle cycliquement la méthode **public Etats superviser()** décrite dans l' interface ***Supervisable***.
Par exemple, les classes suivantes SuperviserMemoire, SuperviserCPU, SuperviserPort(int port) implémentent cette interface.

Après avoir appelé la méthode ***superviser***, l'objet de supervision notifie à la supervision centralisée, les états retournés par la méthode *superviser*.
Les états retournés sont un tableau d'état. Un état est défini par (DATE, HEURE, NOM, VALEUR).

La supervision centralisée est un objet distribué RMI qui se trouve sur une autre machine que le factory d'instance de supervision. Elle contient tout l'historique des états notifiés par toutes les objets de supervision.

Une IHM de visualisation se trouvant sur une autre machine, tire cycliquement certains états de la supervision centralisée afin de les afficher, en fonction de leurs NOM. L'IHM récupère cycliquement l'état le plus récent d'un NOM donné.

1/ Faite le schéma d'architecture logiciel de votre solution (composants, acteurs, fonctions)
Précisez le rôle de chacun des composants. Justifiez vos choix.



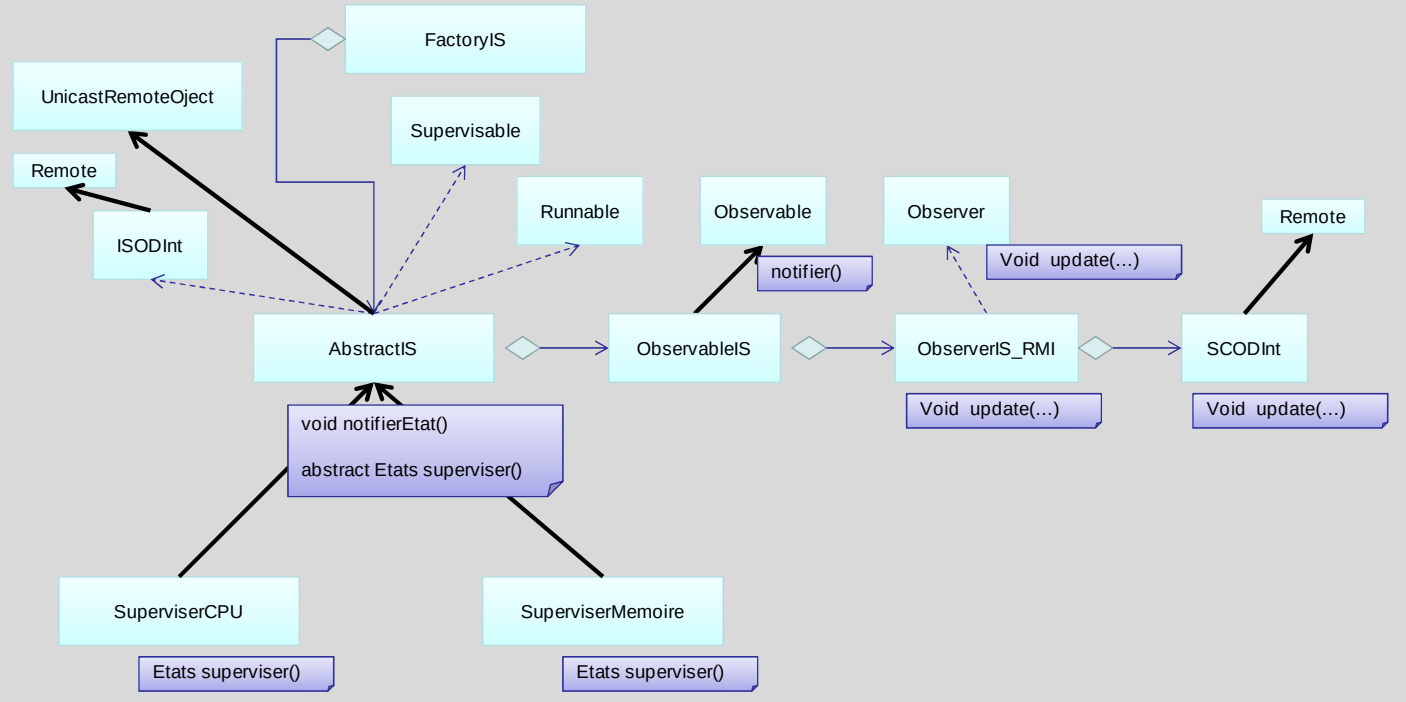
Le composant "factory d'objets de supervision" crée, sur demande, des objets de supervision (ou instance de supervision). Ces objets de supervision sont des objets distribués créés sur la même machine que le factory. Ils exécutent chacun un thread et notifient à la supervision centralisée les états de supervision calculés.
Il peut exister plusieurs factory répartis sur les différentes machines du réseau.

La supervision centralisée est un autre composant qui peut s'exécuter sur une autre machine. Ce composant est un objet distribué, unique sur le réseau. Il reçoit par notification les états de supervision calculés par tous les objets de supervision du réseau.

L'IHM affiche certains états de supervision connus de la supervision centralisée. Pour cela, l'ihm appelle, cycliquement, une méthode distante "getEtats" de la supervision centralisée afin d'obtenir ces états et les afficher.

2/ Faire le diagramme de classe UML de chacun des composants.
Mettez en évidence les méthodes et les attributs importants. Précisez en marge des diagrammes le rôle de chacune des méthodes.

Pour le composant " factory " :

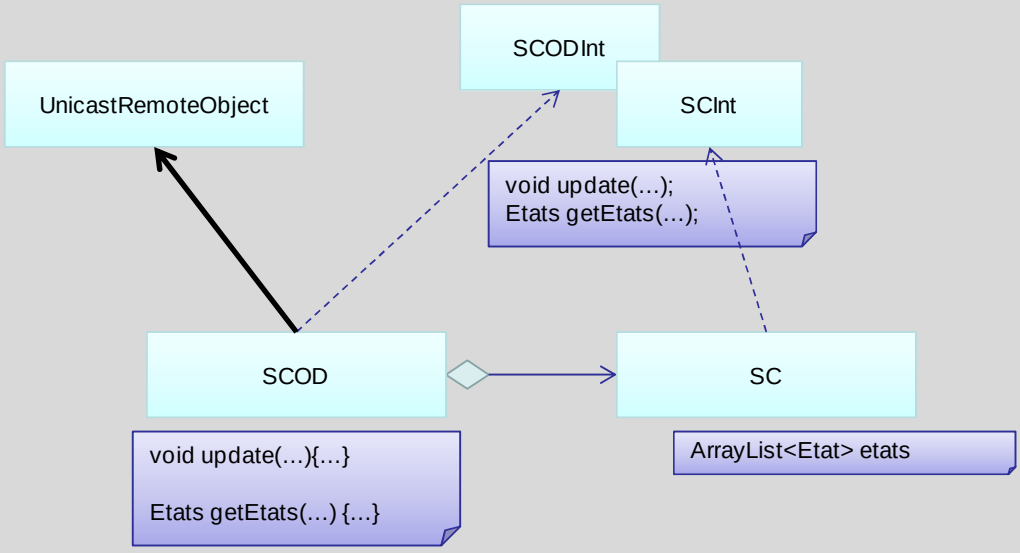


Toutes les classes d'instance de supervision (IS) héritent d'une classe abstraite (AbstractIS) qui crée un thread (implémente l'interface Runnable) dont le traitement générique est d'appeler la méthode abstraite, superviser. Cette méthode est implémentée par les classes d'IS. Le thread appelle la méthode notifierEtat, implémentée par la classe abstraite. Cette méthode utilise une instance de ObservableIS afin de notifier ces états à un observer, ObserverIS_RMI (appelle de la méthode update du Design Pattern Observateur).
Le role de ObserverIS_RMI est d'utiliser le stub d'un observer distant qui n'est autre que la supervision centralisée. L'interface SCODInt est l'interface Remote (RMI) de la supervision centralisée.

Tous les objets du factory sont des objets distribués : hérite de UnicastRemoteObject et implémente l'interface ISODInt.

Le factory est une instance de la classe FactoryIS et il contient des instances de AbstractIS..

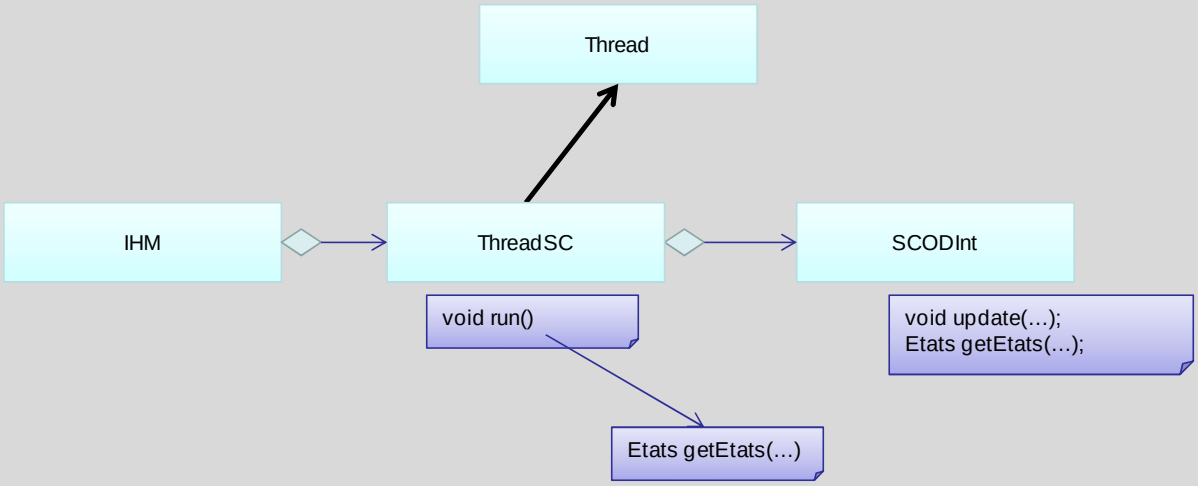
Pour le composant "supervision centralisée" :



La Supervision Centralisée est une classe SC qui est encapsulée par un objet distribué SCOD qui permet d'appeler les méthodes de la SC de manière distante.
Ces méthodes sont :
- la méthode update (utilisées par les IS) permettant la notification des états de supervision, et
- la méthode getEtats qui retourne les états de supervision (utilisée par l'IHM).
La classe SC contient l'historique de tous les états de supervision remontés par les instances de supervision.

La classe SCOD est donc un objet distribué qui hérite de UnicastRemoteObject et implémente l'interface Remote SCODInt.

Pour le composant "IHM" :



L'IHM utilise la classe ThreadSC pour instancier un thread qui demande cycliquement à la Supervision Centralisée certains états de supervision.
Pour cela elle utilise l'interface Remote SCODInt pour appeler la méthode distante getEtats.

3/ Mettez en évidence les Designs Patterns que nous avons vu en cours qui se retrouvent dans vos diagrammes de classe. Commentez.

Dans le composant "Factory", on a le DP Factory constitué des classes ISODInt, FactoryIS, AbstractIS, SuperviserCPU, SuperviserMemoire.
Ce DP contient une usine (FactoryIS) qui crée des objets réels (SuperviserCPU, …) vus par une même interface (ISODInt).

Dans le composant "Factory", on a le DP Observateur constitué des classes ObservableIS, ObserverIS_RMI, Observable, Observer.
Ce DP crée un observeur (ObserverIS_RMI) qui implémente la méthode update de l'interface Observer. Cet observateur est ajouté à l'observable ObservableIS.
L'observeur est notifié qui à son tour appelle la méthode distante update.

Dans le composant "Supervision Centralisée", on a le DP Proxy constitué des classes SCOD, SC, SCODInt et SCInt. Le proxy traduit l'appel de méthode distante en appel de méthode locale.

Dans le composant "IHM", on n'a pas de DP proprement dit car pour tirer les états de supervision, l'IHM appelle (via un thread) directement une méthode distante de la supervision centralisée.
Nous aurions pu aussi créer un DP proxy Fournisseur-Consommateurs suivant le principe du « pull » dans lequel le consommateur voit le fournisseur à travers un ProxyPullSupplier afin de lui tirer un évènement. Et le fournisseur voit les consommateurs à travers un ProxyPullConsummer.