IPST-CNAM
Intranet et Designs patterns
NSY 102
Vendredi 7 Avril 2017

Durée : **2 h 45**
Enseignants : LAFORGUE Jacques

1ère Session NSY 102

CORRECTION

1ère PARTIE – SANS DOCUMENT (durée: 1h15)

1. QCM (35 points)

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:

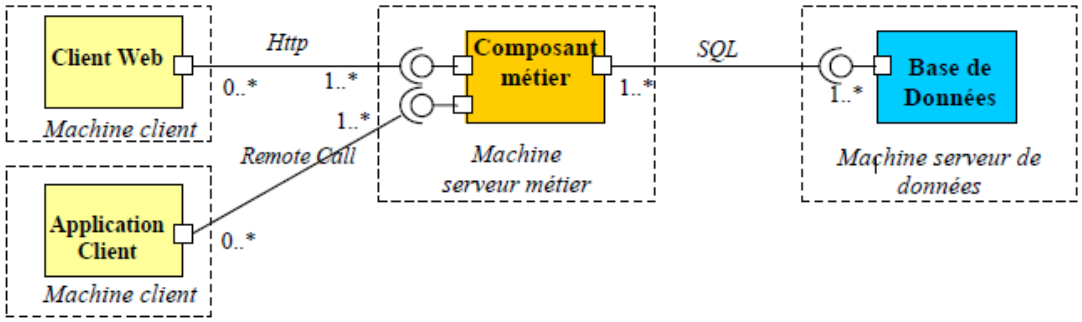
Réaliser l'architecture d'un Système d'Information (SI) permet de valider les choix technique pour la réalisation du SI.		Q 1.
1	OUI	**X**
2	NON	

La **Configuration Architecturale** d'un Système d'Information		Q 2.
1	est la description de tous les composants du SI qui sont utilisés pour configurer l'exécution de ce dernier	
2	représente l'organisation des composants logiciels et des sous-composants qui constituent un Système d'Information	**X**

Dans la description de l'architecture technique, un connecteur est un lien de dépendance entre deux composants qui peut être réalisé par le principe du design pattern de l'injection de dépendance.		Q 3.
1	OUI	**X**
2	NON	

Dans une Configuration Architecturale, un lien entre deux composants correspond :		Q 4.
1	toujours une dépendance distante (machine à machine) entre les composants	
2	souvent à la transmission d'information entre les composants	**X**

Dans la réalisation de l'architecture d'un futur SI, il n'est pas rare de réaliser une maquette ou un prototype qui représente au mieux l'ensemble du futur système permettant de s'assurer des choix réalisés tant sur la capacité que sur les performances.		Q 5.
1	OUI	**X**
2	NON	



Ce schéma représente une architecture 4-Tiers		Q 6.
1	OUI	
2	NON	**X**

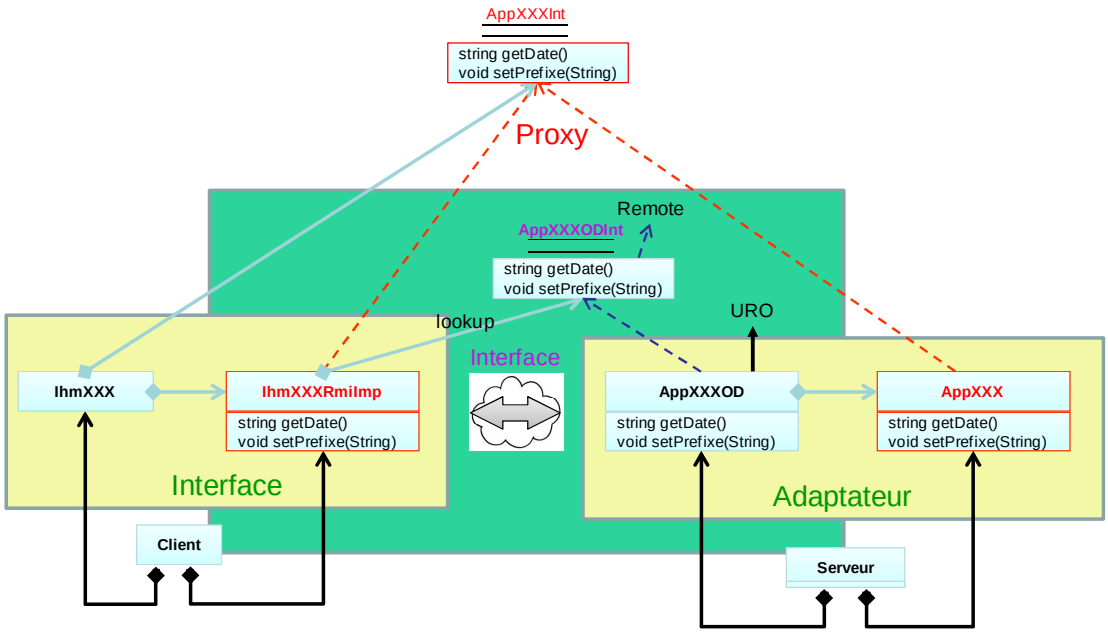
Une application dite "distribuée" est une application logicielle dans lequel les données informatiques sont		Q 7.
1	nécessairement centralisées dans un singleton (exemple un Factory)	
2	réparties sur le réseau et accessibles par tout logiciel qui utiliserait un ORB	**X**

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		Q 8.
1	OUI	**X**
2	NON	

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 celui qui a créé l'objet distribué (exemple un factory); puis d'utiliser ce stub pour appeler les méthodes distantes		Q 9.
1	OUI	**X**
2	NON	

Soit 2 clients (A et B) qui appellent en même temps la méthode distante m1 de l'objet distribué OD1.		Q 10.
1	A et B peuvent appeler en même temps la méthode m1 de OD1	**X**
2	A et B ne peuvent pas appeler en même temps la méthode m1 de OD1 et il faut donc que la méthode m1 soit "synchronized".	

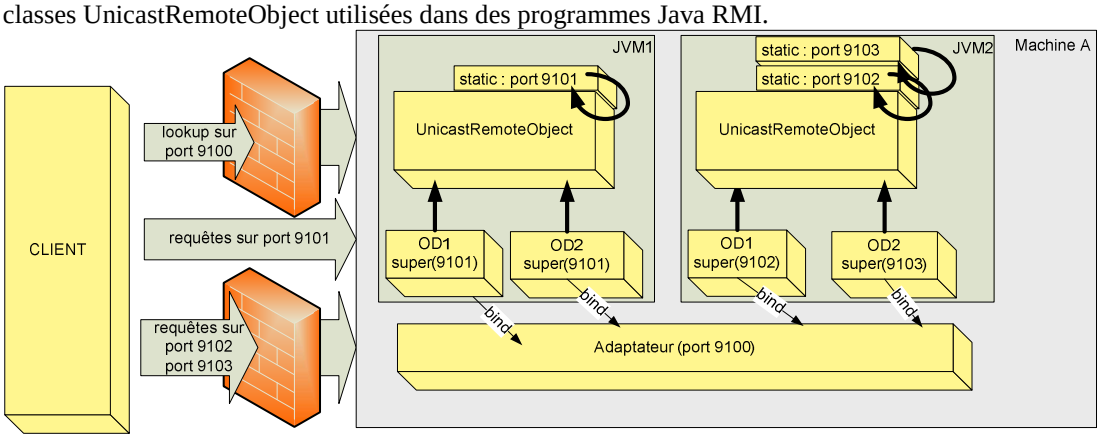
Soit un objet quelconque Obj (instance de la classe A qui n'hérite pas d'une autre classe). En Java RMI, il est très facile de transformer cet objet en un objet distribué. Pour cela il suffit de :		Q 11.
1	faire que la classe A implémente l'interface Remote	
2	faire que la classe A implémente l'interface Serializable, puis écrire cet objet dans un annuaire RMI	
3	créer un proxy de A . Ce proxy hérite de UnicastRemoteObject et implémente l'interface de A qui hérite de Remote	**X**



Ceci est un diagramme de classe d'un système composé d'un client IHM (classe IhmXXX) et de son applicatif (AppXXX) que l'on veut rendre distant.

IhmXXXRmiImp est un Proxy de AppXXX :

		Q 12.
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.		Q 13.
		
1	On peut créer un nouvel OD dans la JVM1 qui s'exécute sur le port 9101	**X**
2	On peut créer un nouvel OD dans la JVM1 qui s'exécute sur le port 9102	
3	On peut créer un nouvel OD dans la JVM2 qui s'exécute sur le port 9100	

En RMI, l'appel d'une méthode distante, entre un client et un objet distribué RMI se fait en utilisant un objet qui est un proxy de communication de l'objet distribué. Ce proxy implémente l'interface distante que l'objet distribué implémente.		Q 14.
1	OUI	**X**
2	NON	

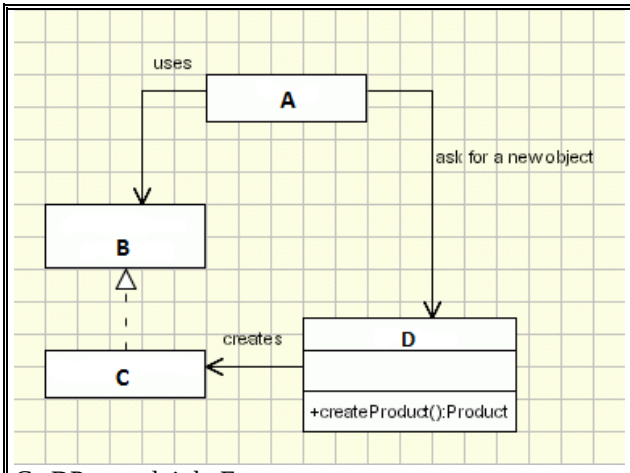
En RMI, l'amorce ou stub d'un Objet Distribué est un proxy de l'Objet Distribué qui est créé grâce à un Dynamic Proxy		Q 15.
1	OUI	**X**
2	NON	

Un Design Pattern (DP) ou Patron de Conception est une norme de description des interfaces entre les composants d'une architecture logicielle orientée objet		Q 16.
1	OUI	
2	NON	**X**

Dans un système réparti, le DP Singleton est utilisé pour créer un objet distribué unique sur le réseau		Q 17.
1	OUI	
2	NON	**X**

Le rôle d'un factory dans une architecture distribuée est 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 sont pas accessibles.		Q 18.
1	OUI	
2	NON	**X**

Le DP Factory a pour fonction la création d'objet dont les classes héritent d'une même classe abstraite ou implémentent la même interface		Q 19.
1	OUI	**X**
2	NON	



Q 20.

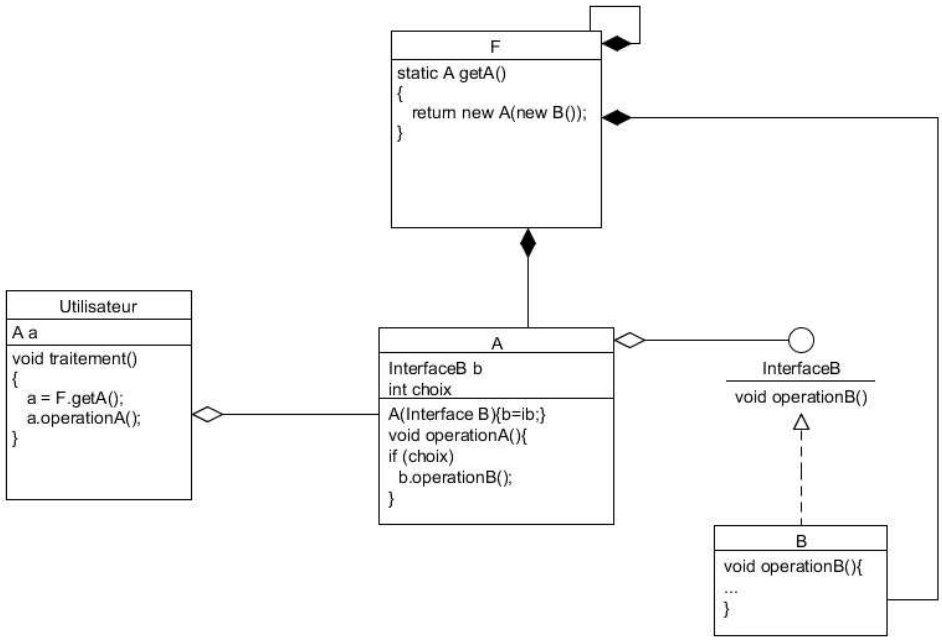
Ce DP est celui du Factory.
La signification des lettres A, B, C et D est :

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**

Le rôle du DP "Délégation" est de déléguer à une autre classe de réaliser des traitements qu'une classe aurait dû implémenter.		Q 21.
1	OUI	**X**
2	NON	

Le diagramme suivant :

Q 22.

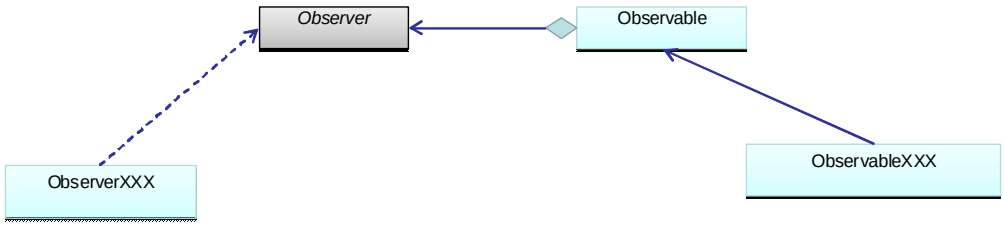


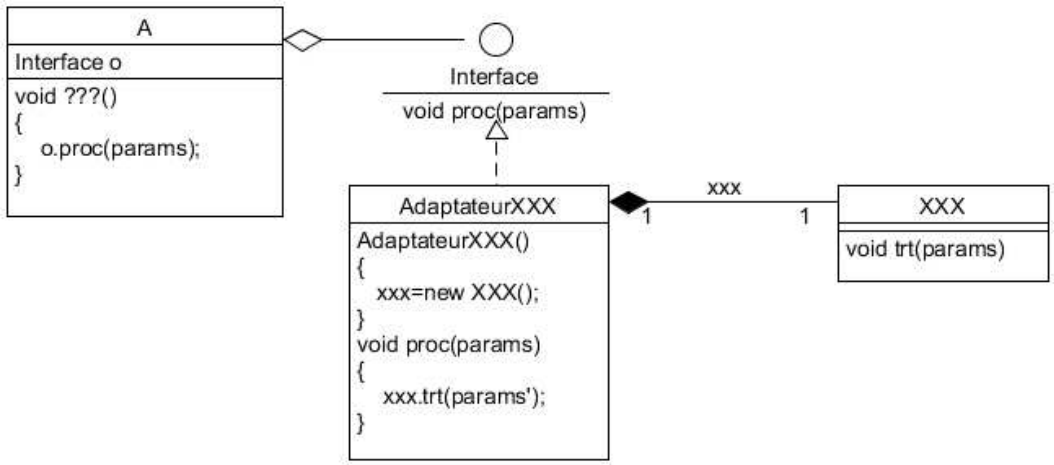
représente une injection de dépendance par l'utilisation d'un setteur

1	OUI	
2	NON	**X**

Dans le DP Observer, la communication entre l'Observable (producteur d'évènement) et l'Observer (consommateur d'évènement) est :		Q 23.
1	synchrone ou asynchrone (choix de conception)	X
2	toujours asynchrone	
3	toujours synchrone	

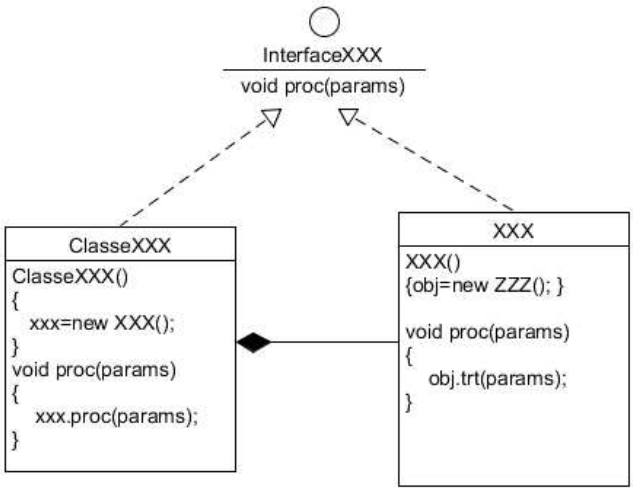
Le DP Observeur/Observable, peut être utilisé pour réaliser un connecteur Producteur/Consommateur		Q 24.
1	OUI	X
2	NON	

Soit le Design Pattern Observateur :		Q 25.
		
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

Soit le diagramme de classe suivant :		Q 26.
		
Ce diagramme de classe représente celui d'un DP Adaptateur		
1	OUI	X
2	NON	

Un Adaptateur est un DP constitué d'une classe A qui implémente une interface I à la place d'une autre classe B qui ne peut pas implémenter cette interface		Q 27.
1	OUI	X
2	NON	

Soit le diagramme de classe suivant :		Q 28.



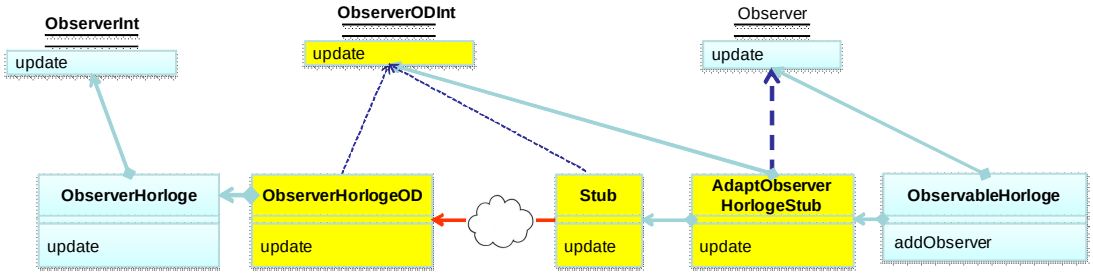
1	Ce diagramme de classe représente le DP Adaptateur	
2	Ce diagramme de classe représente le DP Proxy	**X**

A l'opposé de la communication synchrone, la communication asynchrone est un type de communication notamment basé sur le modèle du pull, comme par exemple un thread d'un client qui tire régulièrement les événements d'un serveur		Q 29.
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 :		Q 30.
1	modèle du pull	
2	modèle du push	**X**
3	modèle du push-and-pull	

Dans la communication synchrone via un "canal d'évènement" entre un producteur et un consommateur, le producteur utilise un proxy de consommateur (et non les consommateurs directement), afin de lui pousser un évènement		Q 31.
1	OUI	**X**
2	NON	

Le modèle de communication "Push asynchrone" est un DP dans lequel la classe qui implémente l'interface Observer crée un thread qui réalise l'appel à la notification de l'Observable.		Q 32.
1	OUI	
2	NON	**X**

Soit le schéma suivant :		Q 33.
		
Les classes et interfaces en gris représentent :		
1	un proxy client de communication entre ObservableHorloge et ObserverHorloge	
2	un pont de communication permettant à un Observable (ObservableHorloge) de notifier les évènements à un Observer (ObserverHorloge) se trouvant dans une autre JVM.	**X**

Le principe d'un MOM (Model Orienté Message) est d'utiliser un composant logiciel qui sert d'intermédiaire entre les producteurs et les consommateurs. Ce composant logiciel utilise :		Q 34.
1	un DP Factory de canaux d'évènement pour créer les canaux d'évènement	**X**
2	Un DP Observer/Observable pour notifier les Consommateurs des évènements produits par les Producteurs	**X**
3	Un DP ModelVueControler pour produire les évènements des Producteurs	

Dans le cadre de la communication entre un composant Java et un composant C++ sur un bus CORBA		Q 35.
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**

Fin du QCM

Suite (Tournez la page)

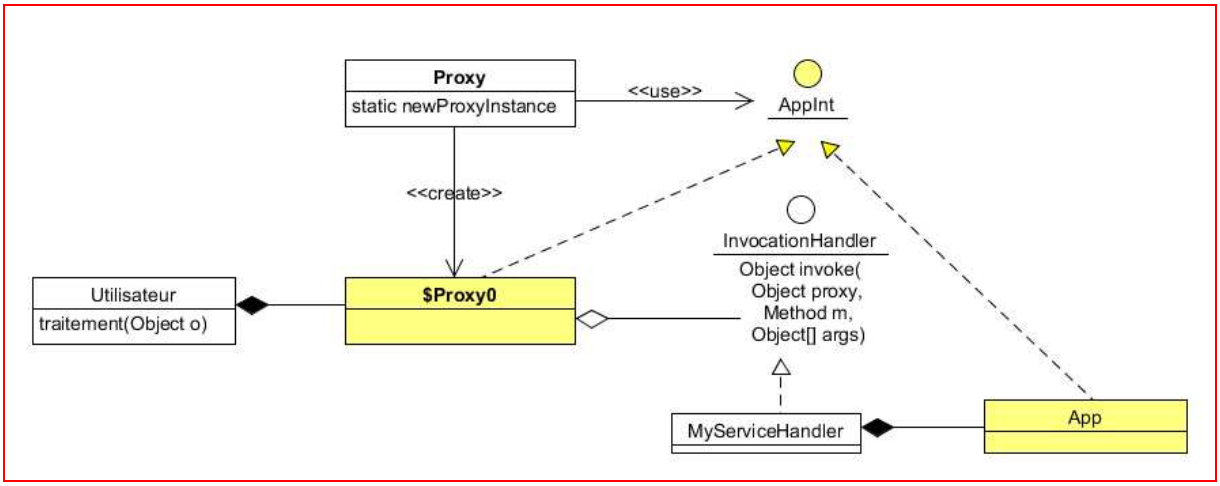
2. Questions libres (15 points)

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

Faire la description du Design Pattern du **Dynamic Proxy** sous la forme d'un diagramme de classe UML.
Quel est le rôle de ce Design Pattern ?

Correction :



Le rôle de ce Dp est de créer un proxy dynamiquement au lieu de coder ce proxy de toute pièce.
Le proxy créé implémente toutes les méthodes de l'interface AppInt qui réalise l'appel à la méthode invoke de l'handler qui lui doit être codé.

QUESTION NUMERO 2

Citez 3 exemples de l'utilisation du Design Pattern Proxy en précisant, pour chacun, le rôle fonctionnelle du proxy.

Correction :

Exemple 1 : Un proxy qui permet de réaliser une notification à un utilisateur client pour chaque utilisation des setteurs dans la couche serveur par exemple.
Dans ce cas le proxy implémente les méthodes de setteur en réalisant successivement les deux actions suivantes :

- la notification à travers un observable de l'appel au setteur (nom de l'attribut + la valeur de l'attribut)
- l'appel au setteur de la classe applicative dont il est le proxy.

Exemple 2 : Un proxy pour vérifier que à chaque appel d'un traitement métier, méthode d'un contrôleur d'un MVC, que l'utilisateur est bien connecté : vérification du user et mot de passe.
Dans ce cas le proxy implémente les méthodes du contrôleur en réalisant les deux actions suivantes :

- vérifier que un utilisateur est bien connecté (attribut du controleur)
- l'appel au traitement du contrôleur si correct sinon erreur.

Exemple 3 : Un proxy pour utiliser de manière distante les méthodes d'une classe (exemple RMI).
Dans ce cas le proxy est créé par le lookup dont le rôle est d'implémenter chaque méthode de l'interface distante en réalisant les actions suivantes :

- connexion au serveur de socket de UnicatRemoteObject
- écriture d'une requête sur le socket (encodage du nom de la méthode et des paramètres de la méthode)

QUESTION NUMERO 3

Dans un MOM (Message Oriented Middleware), il existe deux modes de communication entre les producteurs et les consommateurs. Expliquez ce que sont ces deux modes de communication.

Ces deux modes de communication sont appelés "Queue" et "Topic".
Le principe du mode "Queue" et que le producteur dépose son évènement dans une file de message (intermédiaire). Il est alors à la charge des consommateurs de consommer ces évènements.
Le principe du mode "Topic" est que le producteur donne son évènement à un composant intermédiaire qui connait les consommateurs (ils se sont abonnés au préalable) et leurs envoie l'évènement.

Fin de la 1ère partie sans document

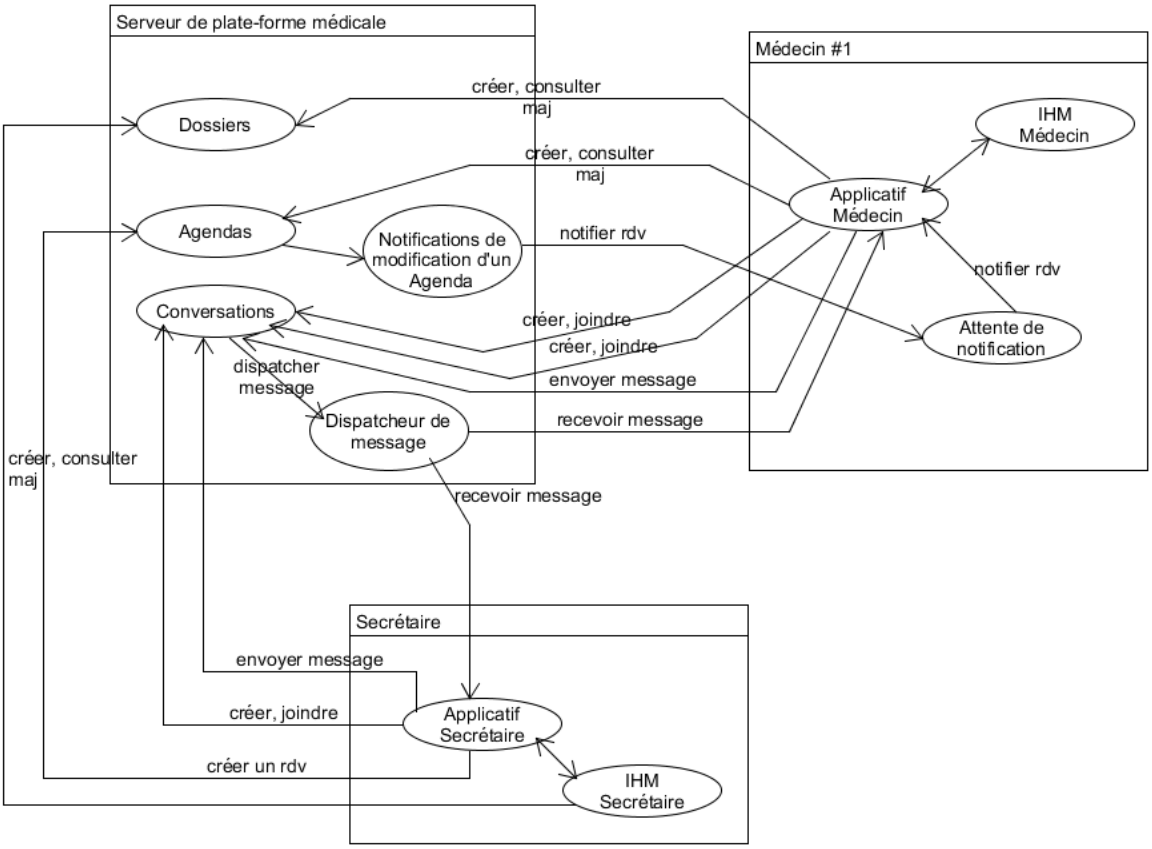
2ème PARTIE – AVEC DOCUMENT (durée: 1h30)

3. PROBLEME (50 points)

.

1/ Le schéma d'architecture logicielle de votre solution (composants logiciels et sous-composants, liens fonctionnels entre les composants logiciels et les sous-composants).
Commentez votre schéma (fonctionnement, rôle, fonctions).

Un composant logiciel correspond à une JVM.



Le Système d'Information, comme précisé dans le sujet, et comme le montre ce schéma est composé de 3 types de composants logiciels distants :

- un serveur,
- un poste pour la secrétaire et
- des postes pour chacun des médecins.

Le serveur centralise la gestion de toutes les données (CRUD) :

- **Dossiers** des patients,
- **Agendas** des médecins

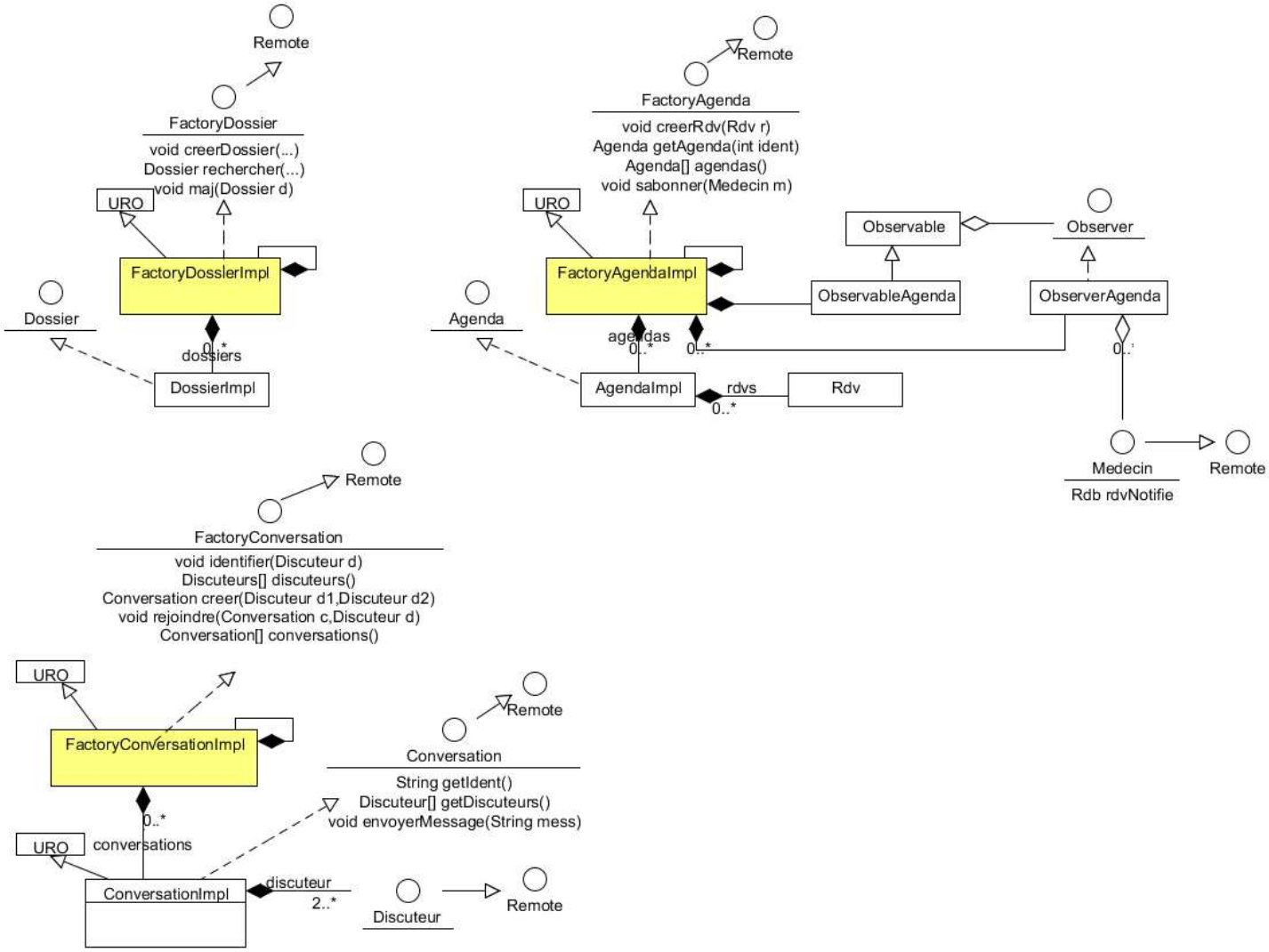
Ces données sont créées, mises à jour et consulter à distance par la secrétaire et les médecins.
Le serveur permet de gérer des **Conversations** permettant d'échanger des messages entre la secrétaire et les médecins et entre les médecins. Une conversation est un groupe de discussion qui dispatche un message à toutes les personnes du groupe.

Quand un Agenda est mis à jour avec un nouveau rendez-vous, la **Notification de modification d'un agenda** préviens instantanément le médecin concerné via le **Attente de notification** qui prévient **l'Applicatif médecin** puis **l'IHM Médecin**.

2/

Diagrammes de classe du COMPOSANT 1 : le serveur

Le serveur est composé de 3 Factory qui sont des singletons car unique sur le serveur.
Chaque factory est accessible à distance via le DP d'Objet Distant par héritage. Chacun de ces factory centralise les méthodes de création, de consultation, de modification et de suppression des données : Dossier des patients, Agenda de chaque médecin, et les Conversation en cours.

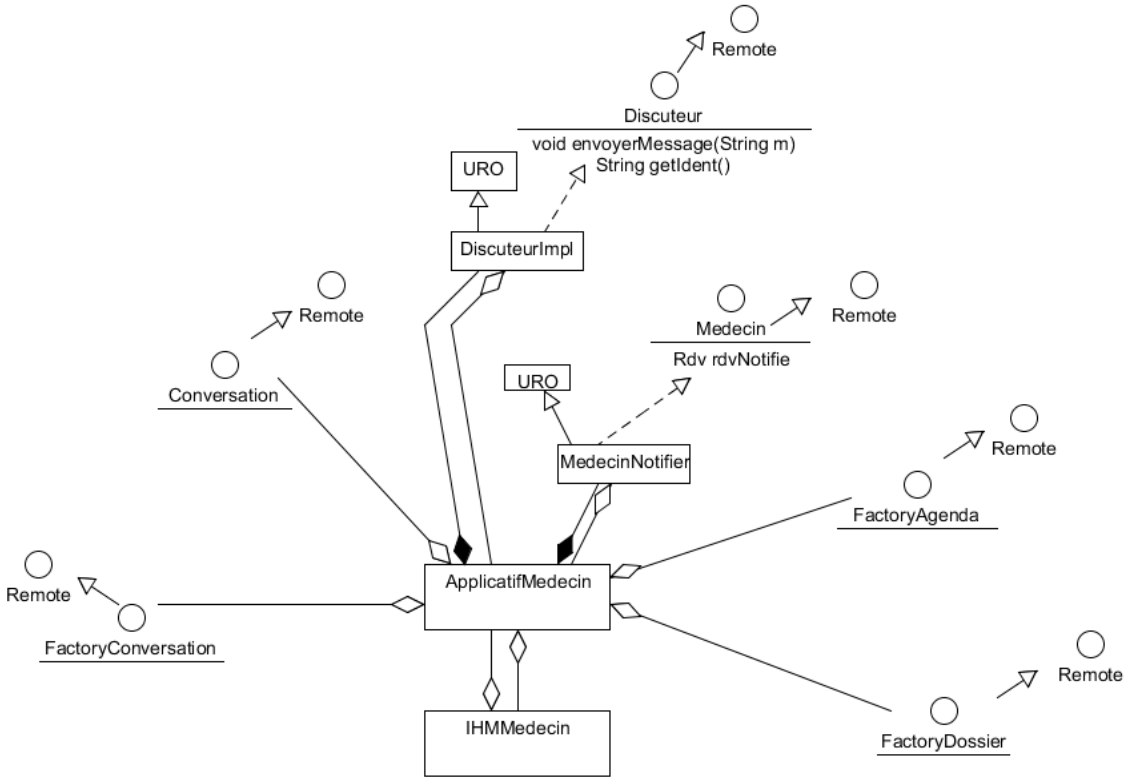


Le Factory des Agendas crée un DP Observer/Observable distant qui permet à chaque Medecin de s'abonner à la notification d'ajout, de modification et de suppression d'un rendez-vous de son agenda.

Les Factory d'agenda et de dossier gère leurs données à distance à travers leur interface remote respective.

Le Factory des Conversation crée des conversations utilisées de manière distante à travers l'interface remote Conversation. Chaque ConversationImpl utilise les interfaces remote Discuteur pour notifier toutes les personnes du groupe de discussion (dont soi-même mais le message est filtré pour lui).
Tous les médecins et la secrétaire utilisent la méthode identifier de FactoryConversation pour s'identifier au près du factory de conversation. Ainsi, tout le monde connait tout le monde et peut donc commencer une conversation avec quelqu'un.

Diagrammes de classe du COMPOSANT 2 : le médecin

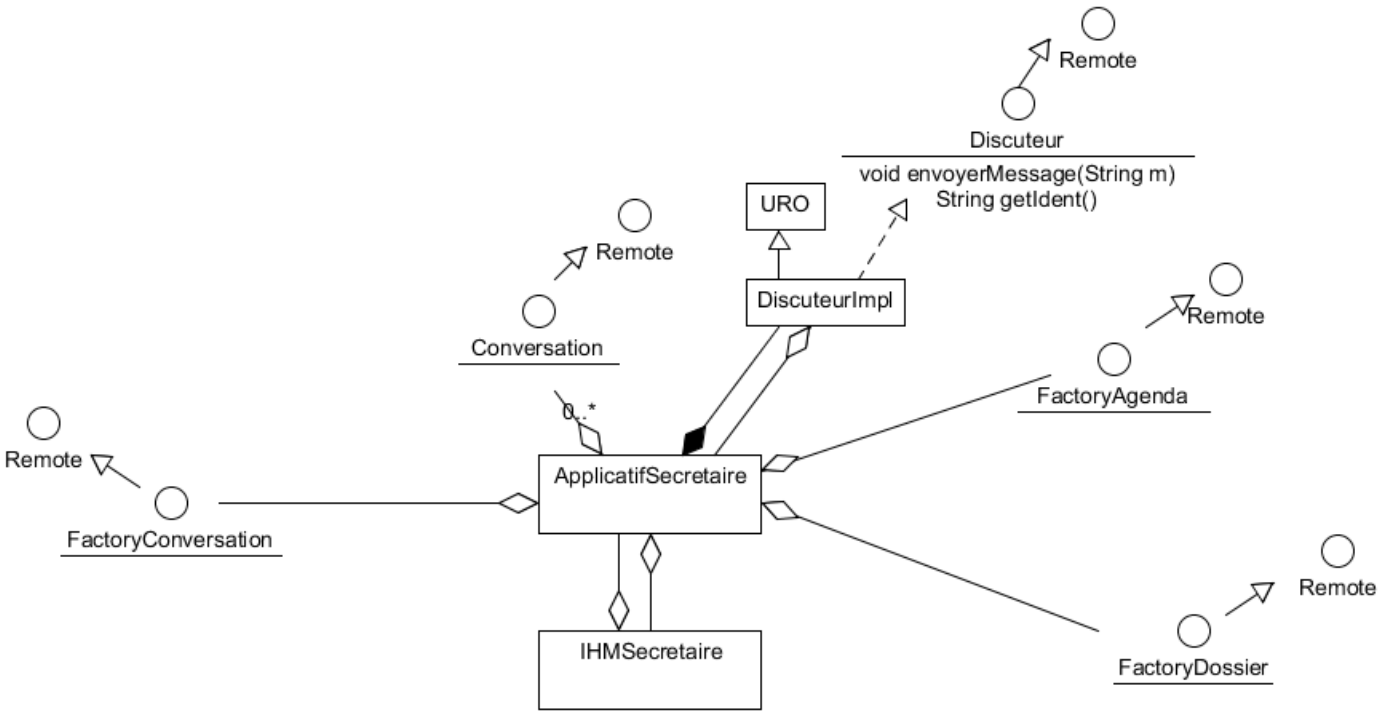


La classe applicative d'un mèdecin utilisent tous les factory du serveur à travers leurs interfaces remote respectives.

Le médecin comme la secrétaire (voir plus bas) crée un DiscuteurImpl permettant de recevoir les messages de chaque conversation. Il est le stub utilisé par chaque conversation gérée par le serveur.

De plus, le médecin crée une MedecinNotifier pout pouvoir se faire notifier d'une maj de son agenda situé sur le serveur qui est un adaptateur à l'interface distante Observer du modèle Observer/Observable distant géré par le factory d'agenda.

Diagrammes de classe du COMPOSANT 3 : la secrétaire



Ce composant est à l'identique de celui du médecin (sans la notification de modification d'un agenda).