M1 : Ingénierie du Logiciel UNIVERSITE PIERRE & MARIE CURIE

Examen Réparti 2eme partie 16 Décembre 2010

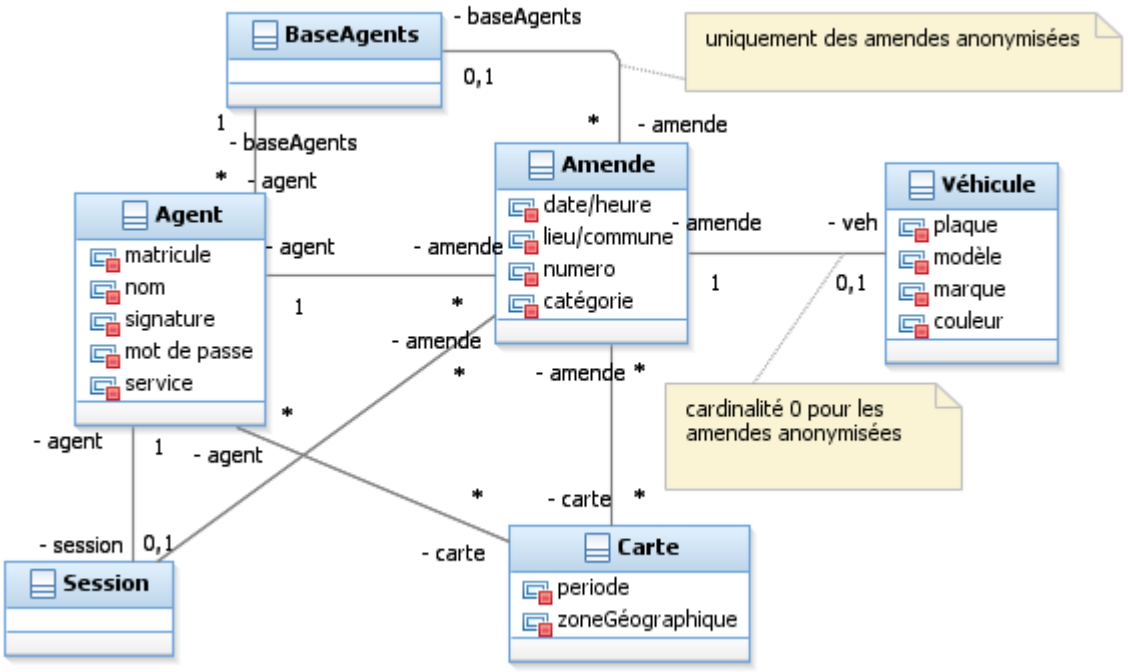
2 heures avec documents -- Barème indicatif sur 20 points.

Problème: Conception de @mende [20 Pts]

Extrait du cahier des charges (rappel): De retour au poste de police, l'agent va clore sa session et noter la fin de sa tournée. Il branche le smart-phone sur un terminal, qui récupère les données de la session et les envoie au fichier central des amendes. Ce transfert n'utilisera pas une connexion sans fil en raison des problèmes de sécurité que cela poserait, au vu de la sensibilité forte des données. La fin du téléchargement des données ferme la session de l'agent.

Les données des amendes sont transférées au système national de traitement des amendes **(SNTA)**. Si le serveur national est momentanément indisponible, les données seront stockées sur le terminal par l'application, qui retentera un transfert plus tard à intervalles réguliers. En plus, une copie anonymisée (les matricules d'agent sont conservés mais les plaques d'immatriculation des véhicules sont retirées) est stockée dans une base de données locale.

L'analyse de @mende a permis de construire ce diagramme de classes métier :



Partie I (4 points)

On propose dans un premier temps de construire un composant bout de chaîne pour représenter un ensemble d'amendes (anonymes ou non). On se limitera à la gestion des caractéristiques suivantes de l'amende, gérées à l'aide de types simples (String, Integer, Date…) : numéro de l'amende, catégorie, lieu, date, marque du véhicule, identifiant de l'agent, service de l'agent, et pour les amendes qui ne sont pas anonymes la plaque minéralogique du véhicule.

Question 1 (4 points): Proposez une conception de ce composant permettant la manipulation d'un ensemble d'amendes. On se limitera aux opérations de création et de lecture.

a) On commencera par représenter sur un diagramme de composant son/ses interfaces offertes et requises (avec opérations et signature dans les interfaces).

b) On réalisera ensuite un diagramme de classe expliquant une conception détaillée d'une réalisation possible du composant.

NB : Dans la suite on appellera IListAmendes l'interface offerte de ce composant permettant la manipulation d'un ensemble d'amendes. Sur les diagrammes de composant qui suivent, il n'est pas demandé de représenter la dépendance sur IListAmendes des divers composants.

Corrigé

a)

Deux options possibles :

Option 1 : par identifiants, une seule interface offerte, IListAmendes, porte au moins un accesseur getListeIdAmende (ou un size() et un accès indicé) + opération d'ajout d'amende, et des opérations getXXXAmende (IdAmende) pour chaque attribut XXX de l'amende.

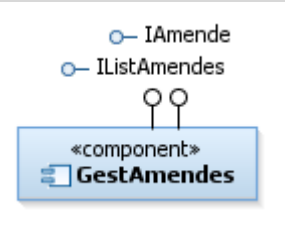
E.g. getAmendeLieu(idAmende) : String

Option 2 : Avec deux interfaces, une IListAmendes avec au moins un accesseur getAmendes() : IAmende* (ou autre variante d'itérateur ou d'accès indexé + size) et une opération add. Une interface IAmende qui porte des accesseurs pour tous les attributs des amendes.

Dans les deux cas, un composant, qui offre la ou les interfaces.

Pour la gestion de l'anonymat on doit trouver une façon de reconnaitre les amendes anonymes, soit une note indiquant que getPlaque rends null si l'amende est anonyme, soit un isAnonyme pour une amende donnée, soit une interface distincte AmendeAnonyme.

Ici option 2 suivie (signatures sur le diagramme plus bas)



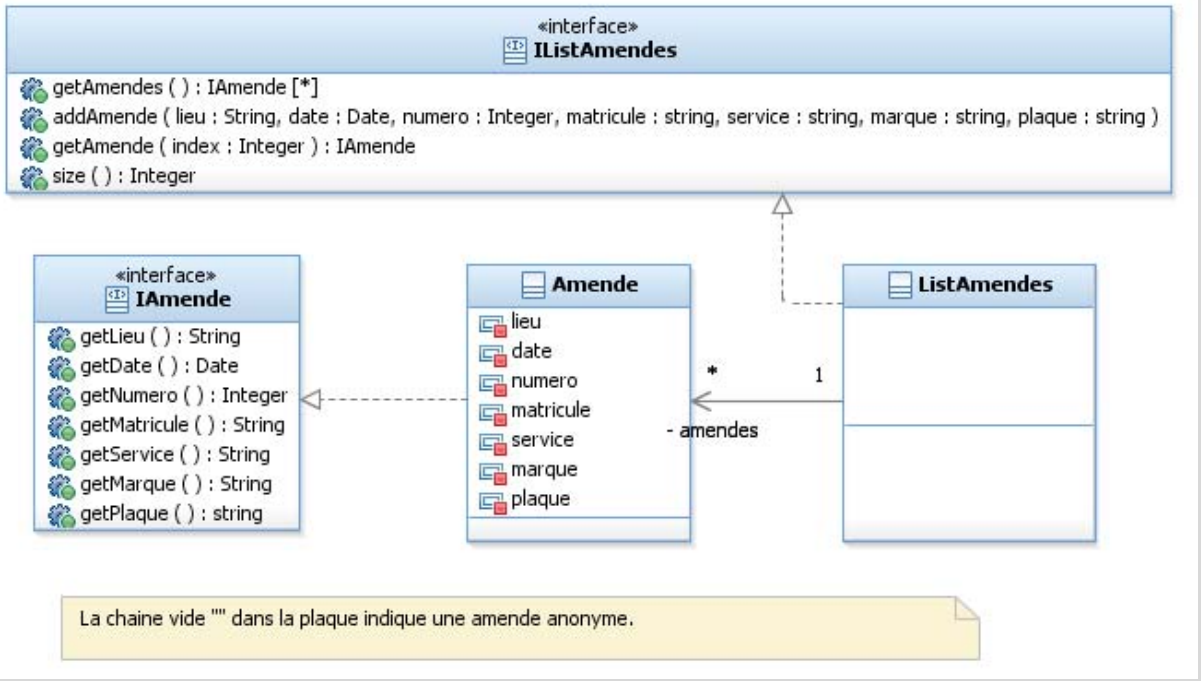
b)

A priori, il est inutile de garder toutes les classes métier. En particulier, on doit éviter de stocker toutes les données de l'agent (mot de passe…).

Le plus simple, une seule classe Amende qui porte les attributs demandés (et uniquement ceux là) + une classe GestAmende avec un lien sur * Amende.

Ensuite bien vérifier que les classes appropriées implémentent les interfaces proposées en a).

Une version qui suit l'option 2 :



Barème : TODO

1 point composant, 1,5 point interface, 1,5 points conception interne

Soit :

25% description du composant, avec une ou deux interfaces offertes selon le cas

35% Signatures et opérations. (10% chaque, 15% pour le premier correct : 15% 1/3, 25% 2/3, 35% 3/3)

1. On doit voir tous les attributs demandés (il y en a 7).
2. On doit pouvoir accéder aux éléments
3. et ajouter un élément.

40% conception interne :

On doit voir les réalisations des interfaces spécifiées en a) : 20%

On doit trouver deux classes (au moins) et ListeAmendes associé en * sur Amende : 20%

Partie II : **(7 points)**

On s'intéresse à présent à la gestion de la fin de session de l'agent. L'analyse détaillée du cas d'utilisation « Clore Session Agent » a identifié les scenarii de comportement suivant :

1. L'agent sélectionne l'action « Clore Session » **(offert par IHM)**
2. Le système **(IHM)** contrôle la connexion au Terminal.
3. Le système **(TERMINAL)** transmet les amendes de la session en cours au SNTA.
4. Le système **(TERMINAL)** stocke les amendes anonymisées dans une base locale.
5. Le système **(IHM)** affiche un message de confirmation.

Exception E1 :

En étape 2, si la connexion est indisponible, le système **(IHM)** affiche un message invitant l'agent à vérifier sa connexion physique (filaire) au terminal. Le cas d'utilisation est interrompu.

Alternative A1 :

En étape 5, si l'envoi de certaines données au SNTA a échoué à l'étape 3, le système **(IHM)** affiche un message indiquant la nature de l'erreur. Ces Amendes seront stockées **(TERMINAL)** sur le terminal qui retentera un transfert **(TERMINAL)** plus tard à intervalles réguliers. La session de l'agent est tout de même fermée **(IHM)**.

En débutant la conception, on a ajouté à cette description un raffinement pour distinguer les composants de l'application, en particulier le terminal et l'IHM de l'agent, notés en **MAJUSCULE**.

Question 2 (3 points) : (NB : les Questions 2 et 3 sont liées, il est recommandé de lire les deux avant de répondre)

On propose de s'appuyer sur l'interface suivante pour représenter la connexion de l'IHM au terminal. Interface IConnexion :

open() : boolean : démarre la connexion au terminal, on gère l'existence d'un problème avec un booléen pour éviter de modéliser des exceptions.

transmit (amendes : IListAmendes) : String[*] : transmet les amendes de la liste fournie, et rend un rapport d'erreurs potentielles de transmission, sous la forme d'une collection de String, une par amende n'ayant pas pu être transmise. Si cette liste est vide, c'est qu'il ne s'est pas produit d'erreurs.

close() : void : ferme la connexion.

a) Quel est l'intérêt, pour un composant donné, de réaliser une classe bouchon qui implémente les différentes interfaces qu'il requiert ?

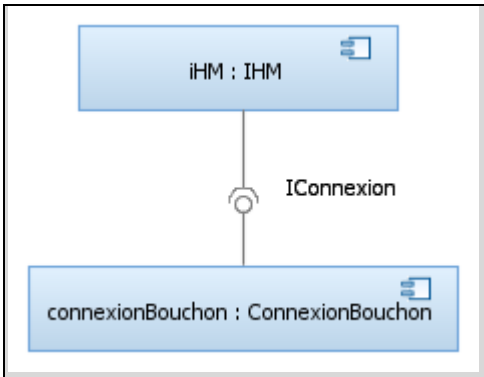
b) Comment faut-il spécifier cette classe dans le cas présent, pour permettre de substituer le bouchon au terminal réel ? Faites un diagramme de classe pour décrire la classe bouchon.

c) Modélisez l'assemblage de l'IHM et de ce composant bouchon dans une configuration de test à l'aide d'un diagramme de structure interne.

30% : Intérêt : permettre le dév de l'IHM en parallèle du dév du terminal. Préparer l'intégration.

35% : la classe bouchon réalise l'interface IConnexion

35% : Instanciation : diagramme avec deux instances, une de l'IHM et une du bouchon, dépendances orientées de IHM sur bouchon. 20% si on reconnait les composants mais que ce ne sont pas clairement des instances



Remarque :

On a ici une question de cours quasiment.

Question 3 (4 points)

Représentez par un ou plusieurs diagrammes de séquence les interactions entre l'IHM de l'agent et le Terminal. On couvrira tous les cas de figure identifiés par l'analyse.

On se place au niveau intégration, en considérant les composants Terminal et IHM uniquement. On se focalise dans cette question sur les responsabilités de l'IHM dans ces scenarii, on ne représentera donc pas les actions du terminal qui seront traitées dans les questions suivantes. Par contre on représentera les affichages réalisés par l'IHM. On considérera que l'IHM a déjà préalablement construit une IListAmendes représentant les données de la session.

On doit trouver 3 diag de séquence, ou une modélisation avec des **alt.**

a) Le smartphone n'est pas connecté au terminal : openSession echoue => message à afficher

b) La transmission d'une ou plusieurs amendes échoue (indisponibilité du serveur central) => liste d'erreurs non vide, affichage

c) La transmission se passe sans encombre (cas nominal). => liste d'erreurs vides

Dans les cas b et c, on doit avoir un close()

Ici une version avec des **alt**, bof, on ne peut pas dire que ça améliore la lisibilité. 3 diagrammes sont plus confortables.

Barème :

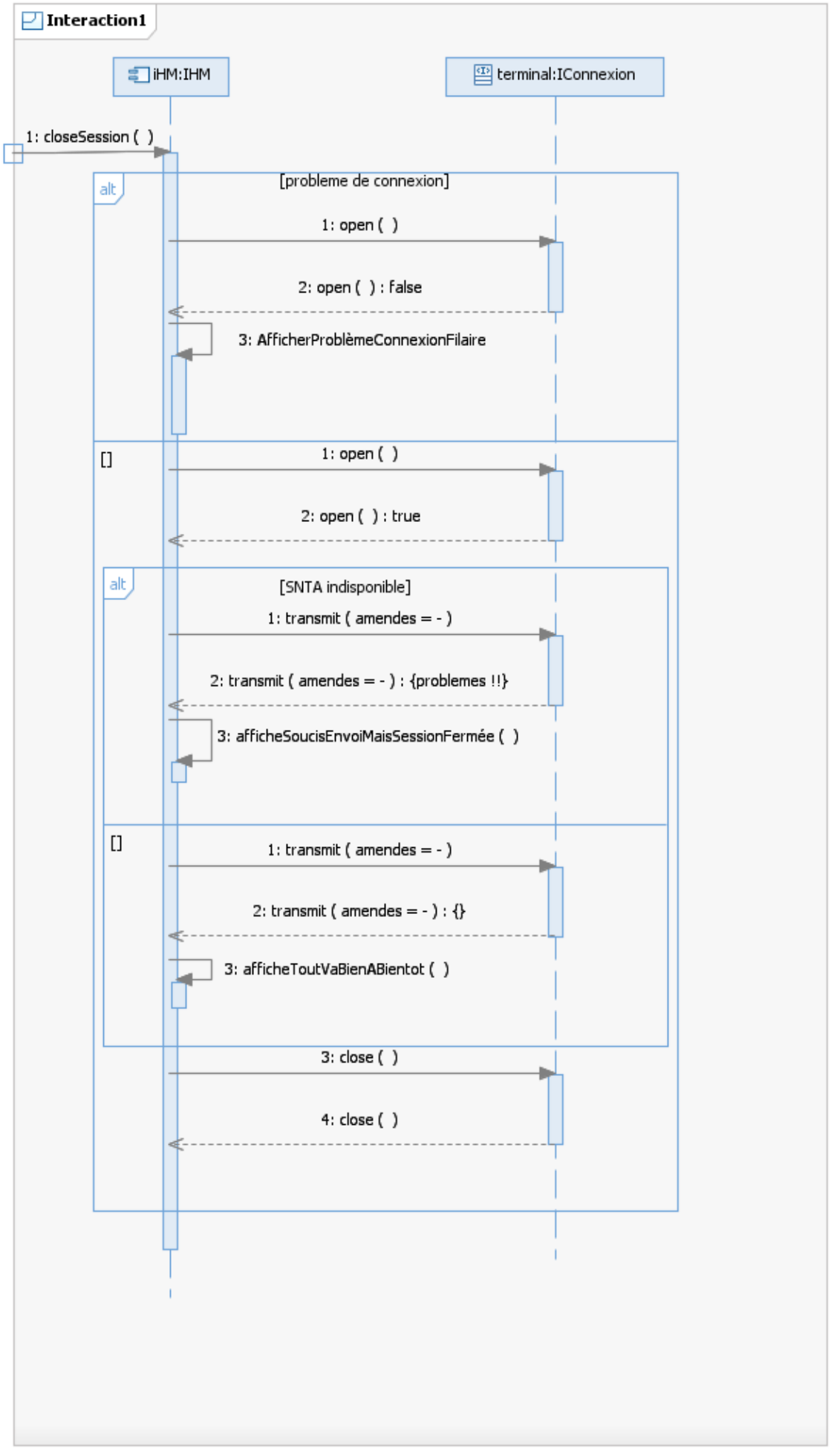
Scenario a) : 30% on doit voir l'invocation à open, et un affichage

Scenario b) : 35% on doit voir open et close (15%) et le transmit + affichage d'erreur (20%)

Scenario c) : 35% on doit voir open et close (15%) et le transmit + affichage OK(20%)

Pas de double peine s'il manque deux fois le close…

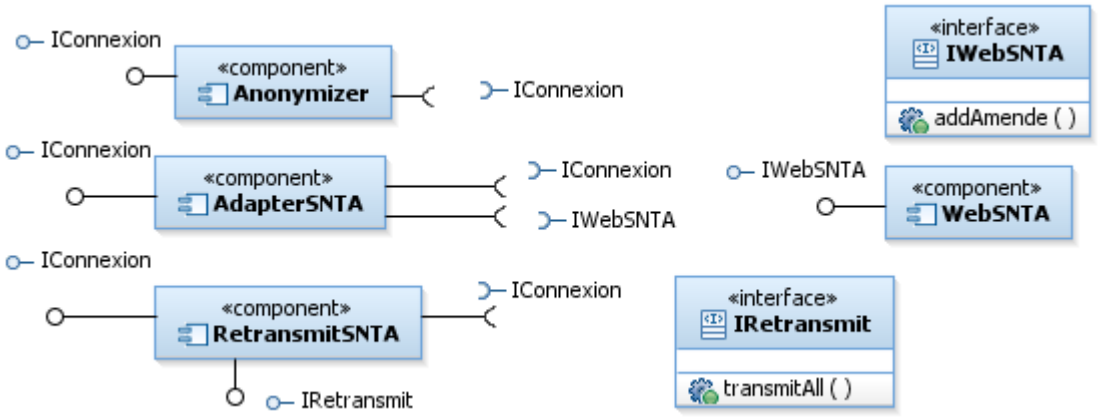
Au moins 30% si les 3 scenarios sont clairement identifié, indépendamment de si la modélisation est correcte.


Interaction1
iHM:IHM
terminal:IConnexion
1: closeSession ( )
alt
[probleme de connexion]
1: open ( )
2: open ( ) : false
3: AfficherProblèmeConnexionFilaire
[]
1: open ( )
2: open ( ) : true
alt
[SNTA indisponible]
1: transmit (amendes = -)
2: transmit (amendes = -) : {problemes !!}
3: afficheSoucisEnvoiMaisSessionFermée ( )
[]
1: transmit (amendes = -)
2: transmit (amendes = -) : {}
3: afficheToutVaBienABientot ( )
3: close ( )
4: close ( )

Partie III (9 points)

On s'intéresse à présent à la conception du terminal. Les responsabilités du terminal sont identifiées dans la description du cas d'utilisation de la question 2.

On propose le découpage suivant, détaillé dans la suite (**NB : il est fortement recommandé de lire les questions 4 à 6 avant de commencer à répondre)**:



Un composant **Anonymizer** réalise le stockage dans la base locale des amendes anonymes, un composant **AdapterSNTA** réalise la transmission au SNTA ou en cas de souci de connexion au composant **RetransmitSNTA** de retransmission périodique. Ces trois composants vont être chaînés ; ils offrent et requièrent chacun l'interface IConnexion. Le composant **Anonymizer** sera connecté à **l'IHM** dans le déploiement final ; il constitue le point d'entrée du Terminal.

Question 4 (2 points) : Anonymizer

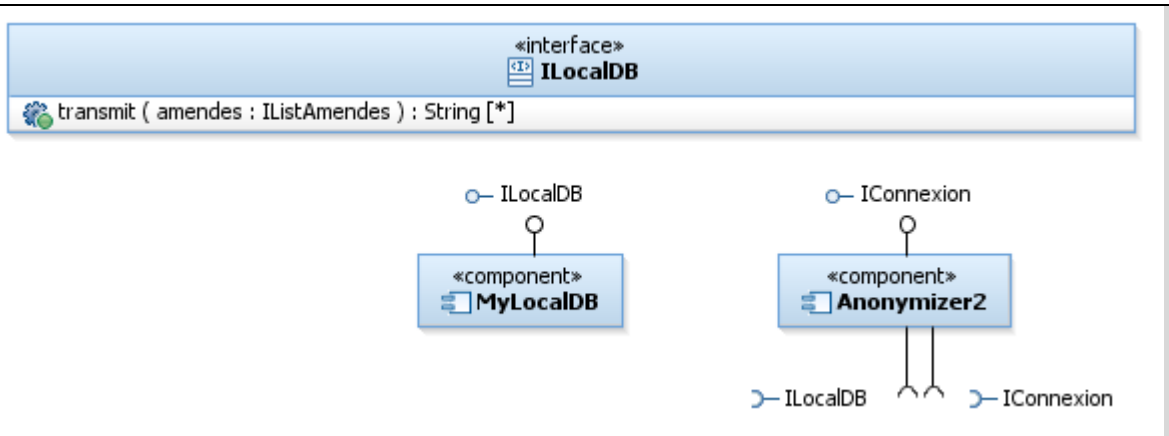
Ce composant n'est pas complètement décrit dans le schéma ci-dessus.

Il doit fonctionner de façon transparente, il retransmet directement les amendes reçues sur son interface offerte sur son interface de sortie.

Cependant au passage il stocke une version anonyme des amendes dans la base de données locale, gérée par un composant LocalBD muni d'une interface qui lui est propre.

a) Complétez la description du composant Anonymizer proposée ci-dessus en intégrant cette contrainte (sur un diagramme de composant).

b) Quelles opérations (+signatures) devrait offrir LocalBD ?



Barème :

a) (50%) On doit voir une dépendance interface requise ILocalDB ajoutée à Anonymizer.

b) (50%) On doit trouver une signature qui permet de transmettre les amendes, sans doute sous la forme d'une IListAmendes comme suggéré ici. On peut trouver une signature différente si l'on a souhaité mieux séparer les amendes anonymes ou pas en Q1.

Question 5 (4 points) : AdapterSNTA

Le système SNTA (acteur secondaire du système) offre une interface de manipulation via un webservice présenté sur une couche de connexion sécurisée (https). Pour nos besoins, l'équipe de développement web fournit le composant WebSNTA qui gère la connexion au webservice et l'envoi des données.

On donne la signature de l'unique opération de IWebSNTA :

addAmende (numero : int, lieu : string, date : string, matricule agent : string, service : string, plaque : string, marque : string, categoriePV : int) throws ConnexionException.

Le composant AdapterSNTA réalise IConnexion. Quand on (dans le système final, ce sera une occurrence du composant Anonymizer) invoque son opération ***transmit(liste),*** **AdapterSNTA** doit essayer d'envoyer les amendes contenues dans ***list*** au serveur web. S'il n'y parvient pas, il doit transmettre les amendes *qu'il n'a pas réussi à envoyer* à sa connexion sortante (IConnexion requise). Cette interface requise sera connectée au composant de retransmission dans l'application finale. De plus pour chaque amende qu'il n'a pas pu transmettre au SNTA, une String décrivant l'erreur (identifiant de l'amende et le message associé à la ConnexionException reçue) sera ajoutée dans la valeur de retour de ***transmit***.

Modélisez ce comportement à l'aide d'un diagramme de séquence représentant le comportement d'une instance de **AdapterSNTA** quand on invoque « transmit » de IConnexion et que certains envois échouent. On pourra user de syntaxe libre (notes de commentaire, pseudo-code) pour expliquer les points algorithmiques plus difficiles à modéliser (boucles, conditionnelles, exceptions…).

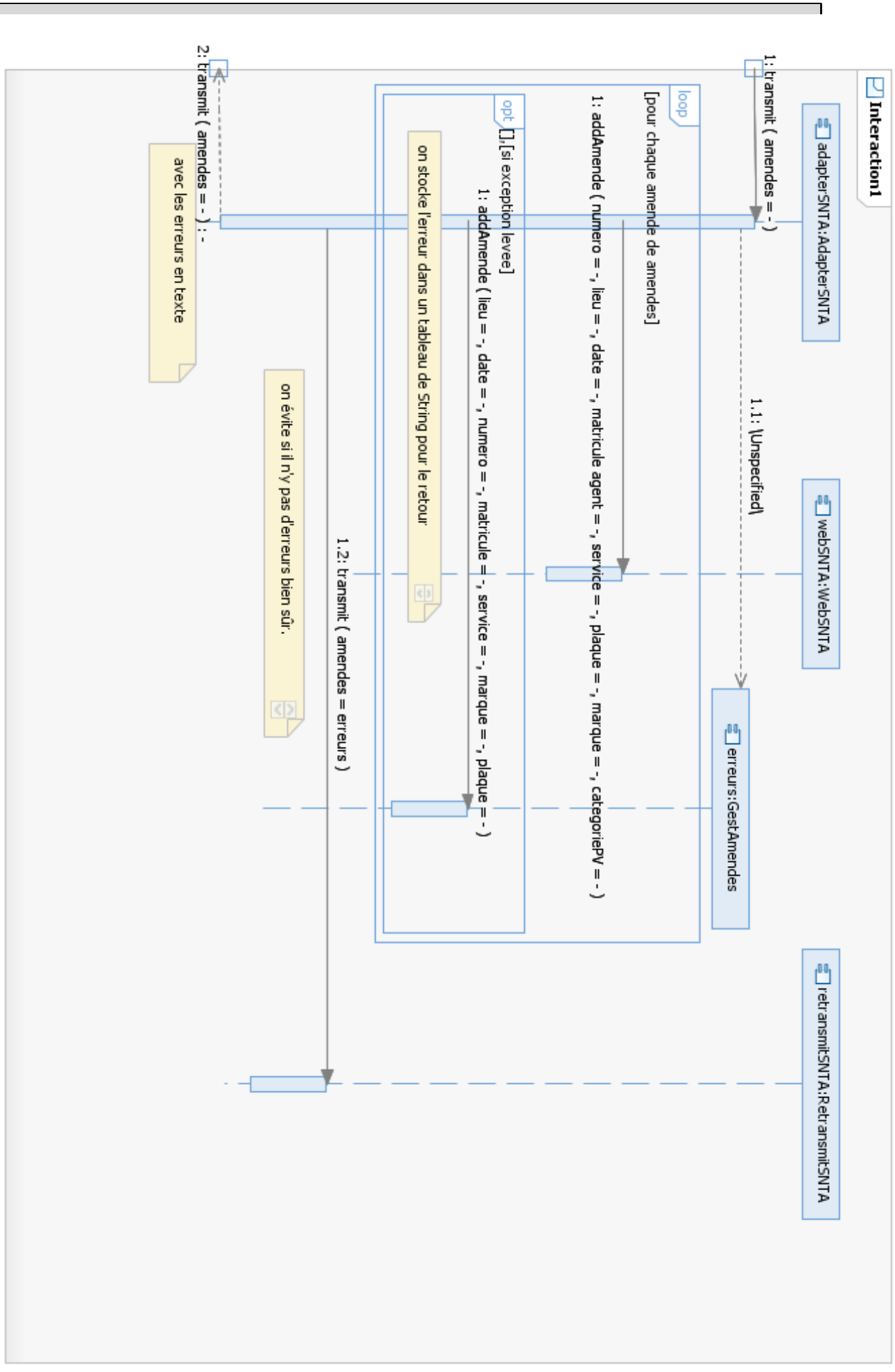
Barème :

30% : gestion Adapter : on doit trouver une spécification de boucle sur les amendes de la liste (20%), les occurrences doivent être correctes, AdapterSNTA et WebSNTA (10%)

60% : gestion des erreurs retransmises:

- on doit voir d'une manière ou d'une autre la construction de la liste d'amendes à retransmettre et le moment ou l'on ajoute des éléments dedans (40%). Des solutions moins explicites que le corrigé (texte, bulles) auront tout de même une portion des points (20 à 30%).
- On doit voir la transmission au retransmitSNTA de la liste construite (20%)

10% gestion des erreurs en sortie : on doit voir que la valeur de retour de transmit porte des String d'erreur.


Interaction1
adapterSNTA:AdapterSNTA
webSNTA:WebSNTA
erreurs:GestAmendes
retransmitSNTA:RetransmitSNTA
1: transmit (amendes = -)
1.1: \Unspecified\
loop
[pour chaque amende de amendes]
1: addAmende (numero = -, lieu = -, date = -, matricule agent = -, service = -, plaque = -, marque = -, categoriePV = -)
opt
[],[si exception levee]
1: addAmende (lieu = -, date = -, numero = -, matricule = -, service = -, marque = -, plaque = -)
on stocke l'erreur dans un tableau de String pour le retour
1.2: transmit (amendes = erreurs)
on évite si il n'y pas d'erreurs bien sûr.
2: transmit (amendes = -) : -
avec les erreurs en texte

Barème

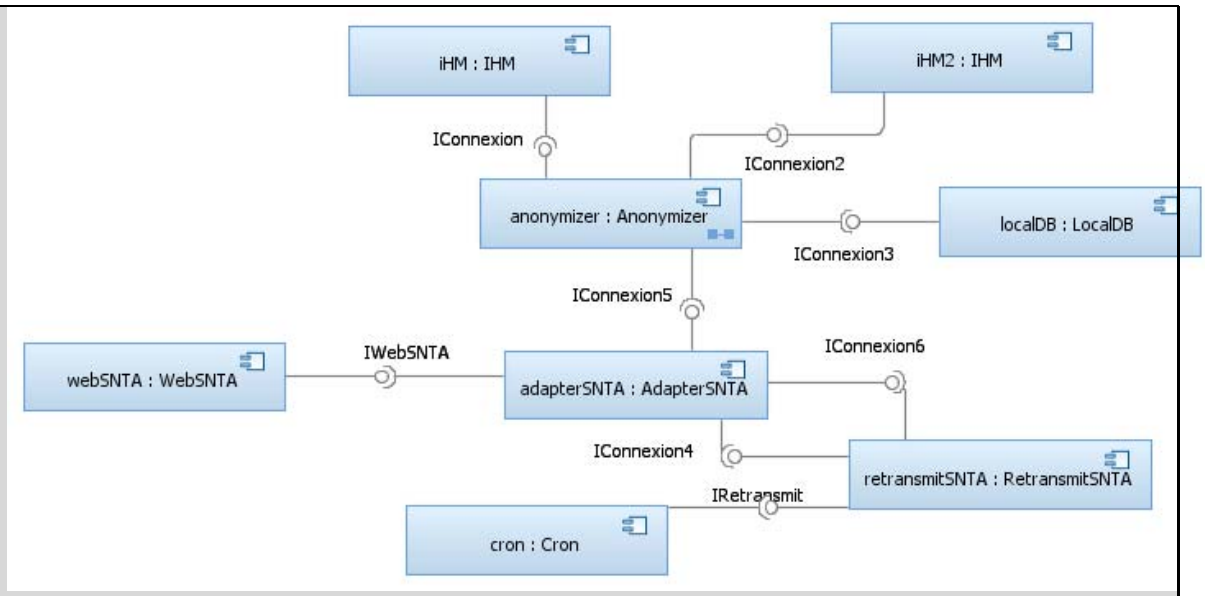
Question 6 (3 points) : RetransmitSNTA

Ce composant crypte les amendes qui lui sont passées et assure leur persistance. Quand on invoque « transmitAll » sur son interface IRetransmit, il recharge les amendes cryptées et les transmets sur sa connexion sortante. « transmitAll » est invoqué à intervalles réguliers (toutes les heures) par une tache de fond (de type cron).

Ce composant reçoit et transmet ses amendes à une même instance de AdapterSNTA dans l'architecture finale.

Représentez sur un diagramme de structure interne l'assemblage final du système. On relira attentivement les descriptions des divers composants dans les questions précédentes.

On devra trouver une instance de chacun des composants IHM, Anonymizer, LocalDB, AdapterSNTA, WebSNTA, RetransmitSNTA, et une instance d'un composant Cron modélisant la tache périodique. On fera attention à l'orientation des liens entre instances.



Barème :

30% : syntaxe correcte, instances présentes ; ne pas donner les points s'il n'est pas clair que ce sont des instances de composants.

70% : 10% par lien correct (instances et orientation).

Il n'est pas demandé les deux occurrences d'IHM ici, donc on attends 7 liens.