UPMC-LIP6

TD

MI-017-IL : Ingénierie du Logiciel

Yann Thierry-Mieg

Master Informatique – 2012/2013

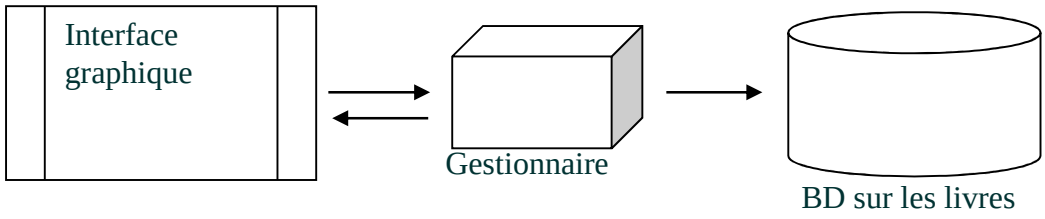
TD1

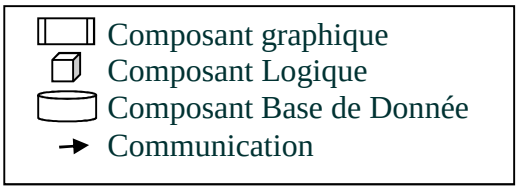
Considérons une application de commerce électronique permettant à des utilisateurs d'effectuer des achats de livres en ligne.
Cette application est composée de trois modules logiciels :

1. une base de données contenant toutes les informations sur les différents livres,
2. une interface graphique permettant aux utilisateurs d'effectuer leurs achats
3. et un gestionnaire permettant d'effectuer les virements bancaires.

Q1 - Décrivez cette architecture à l'aide d'un schéma. Vous expliquerez la notation graphique que vous utilisez (légende du schéma).
Q2 – A quoi peut servir ce schéma et quelles sont les qualités qu'il doit posséder ?

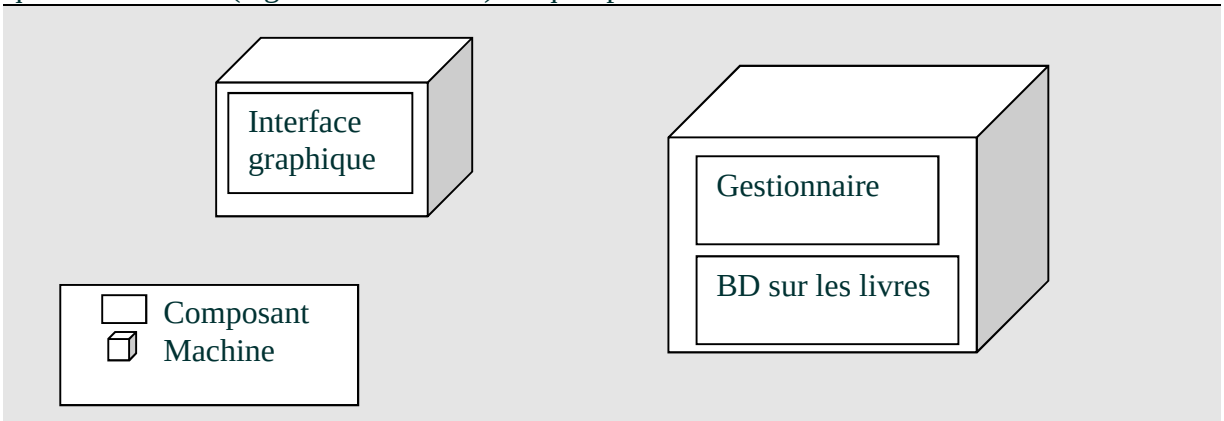
Cette application a été déployée par une société qui a décidé de mettre la base de données et le





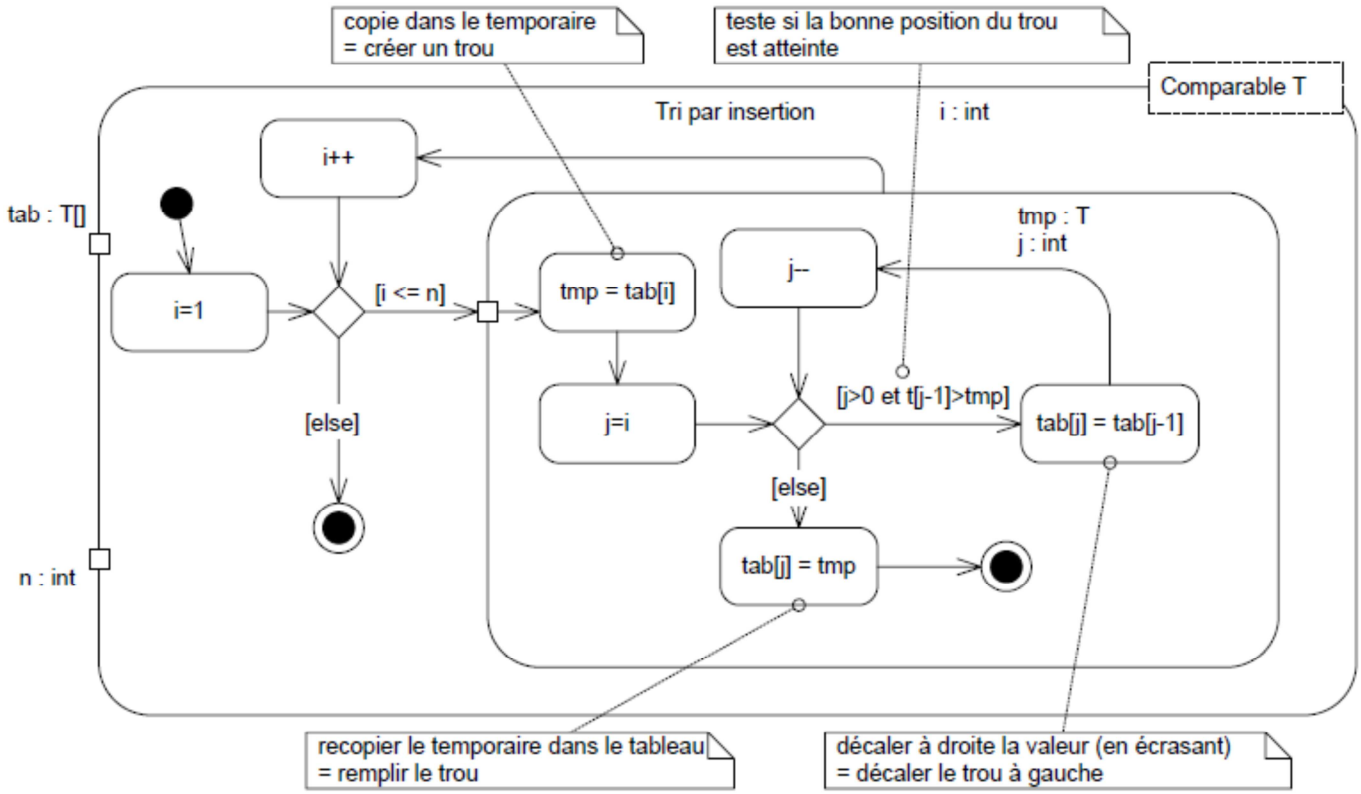
gestionnaire sur une même machine. Les interfaces graphiques s'exécutent dans les navigateurs web des clients. Les communications entre les machines s'effectuent via internet.

Q3 – Décrivez ce déploiement à l'aide d'un schéma. Vous expliquerez la notation graphique que vous utilisez (légende du schéma). A quoi peut servir ce schéma ?



Q4 – Le schéma d'architecture et le schéma de déploiement représentent deux points de vue d'une même application. Schématisez la relation qui lie ces deux schémas : on fera en sorte que les dépendances entre composants soient présentées ainsi que les connexions réseau entre les différentes entités supports au déploiement (machine). A quoi peut servir ce nouveau schéma ? A-t-on besoin des trois schémas (architecture, déploiement, lien) ou peut-on se contenter d'un seul ?

On souhaite modéliser un algorithme de tri.



Comparez le diagramme d'activité UML suivant et le code java correspondant.

```
public static <T extends Comparable<T>> void triSelect(T[] tab) {
      int n = tab.length;
      for (int i = 1; i <= n; i++) {
            T tmp = tab[i];
            int j;
            for (j = i; j > 0 && tab[j - 1].compareTo(tmp) > 0; j--) {
                  tab[j] = tab[j - 1];
            }
            tab[j] = tmp;
      }
}
```

Q5 – Quels sont les qualités et inconvénients de ces deux façons de présenter les choses ? Code et diagramme contiennent-ils la même information ? Quelle présentation préférer pour communiquer avec un collègue développeur Java ? Avec un informaticien qui ne parle pas Java ? Avec un mathématicien ? Avec un ordinateur ? Laquelle de ces deux représentations est la plus facile à construire ?

Q6 – Cet algorithme est lié au gestionnaire qui est déjà présent dans le schéma d'architecture. Schématisez cette relation entre l'algorithme et le gestionnaire. A quoi peut servir ce nouveau schéma ?

Q7 – Donnez une liste d'autres schéma qu'il serait intéressant de réaliser (expliquez leur intérêt).

Q8 – Expliquez les différences entre modèle et schéma. Un modèle a-t-il besoin d'une légende ? A quoi sert un modèle ? Un modèle doit-il être abstrait ou concret ? Qu'elles sont les qualités que doit posséder un modèle ?

Q9 – Expliquez la différence entre un modèle UML et un diagramme UML.

Q10 – Le code peut-il remplacer tous les schémas que vous venez de réaliser ? Expliquez les différences entre schéma et code. Expliquez les différences entre modèle et code.

Q11 – A quoi sert une méthode de conception logicielle ? Quelles doivent être les qualités d'une bonne méthode ?

TD2

Analyse

Notions abordées : Lire un cahier des charges, diagrammes de cas d'utilisation, diagrammes de classe métier.

Q1 : Rappelez l'objectif de la phase d'analyse.

Q2 : Identifiez les acteurs du SIGB.

Q3 : Un acteur est-il obligatoirement un utilisateur des fonctionnalités offertes par le système ?
Q4 : Identifiez les cas d'utilisation du SIGB, et élaborez le diagramme de cas d'utilisation du SIGB. En particulier, justifiez tout lien d'héritage entre acteur et toutes les relations (*include* et *extend*) entre cas d'utilisation.
Q7 : Identifiez les classes métier du SIGB.
Q8 : Elaborez le diagramme de classes métier du SIGB. Justifiez en particulier toutes les associations entre les classes.
Q9 : Echangez votre travail avec un autre groupe d'étudiants. Vérifiez que les réponses des autres étudiants ne contiennent pas d'erreurs. En particulier, prêtez attention à ce que, par rapport au cahier des charges, aucune information ne manque mais aussi à ce qu'aucune information ne soit en trop.

TD3

Spécifications détaillées

Notions abordées : Fiches détaillées, diagramme de séquence d'analyse, classe « système ».

Q1 : Réalisez la fiche détaillée des cas d'utilisation « enregistrer un retour » et « enregistrer un emprunt ».

Q2 : Pour chaque les cas d'utilisation « enregistrer retour » et « enregistrer emprunt » construisez les diagrammes de séquence d'analyse illustrant un fonctionnement nominal.

Q3 : Listez les évolutions nécessaires que vous avez faites aux classes de ces objets.
Q4 : Utilisez le temps restant pour appliquer la même démarche aux autres cas d'utilisation.

TD4

Notions abordées : Tests de validation, rupture Analyse/Conception du cycle de vie.

Q1. Rappelez l'objectif des tests de validation.

Q2 : A partir des diagrammes de séquence et des fiches détaillées élaborées, réalisez très précisément au moins deux tests de validation. Pour ce faire, vous remplirez le tableau suivant :

Titre :	**id**
Contexte :	
Entrée :	
Scénario :	
Résultat attendu :	
Moyens de vérification :	

Q3 : Définissez un ordre dans lequel les tests de validation doivent être réalisés. Justifiez l'ordre que vous avez choisi. Pensez-vous que votre ensemble de tests est complet ?

Q4 : Echangez votre travail avec un autre groupe d'étudiants. Vérifiez que les réponses des autres étudiants ne contiennent pas d'erreur. En particulier, prêtez attention à ce que toute fonctionnalité ait des tests de validation lui correspondant.

Q5 : Expliquez les différences entre le point de vue conception et le point de vue de l'analyse. Quand peut-on parler de conception architecturale ?

L'objectif principal de cette première étape de conception est de proposer une découpe cohérente avec une séparation en composants.
On va commencer par réfléchir sur une découpe structurelle des données.

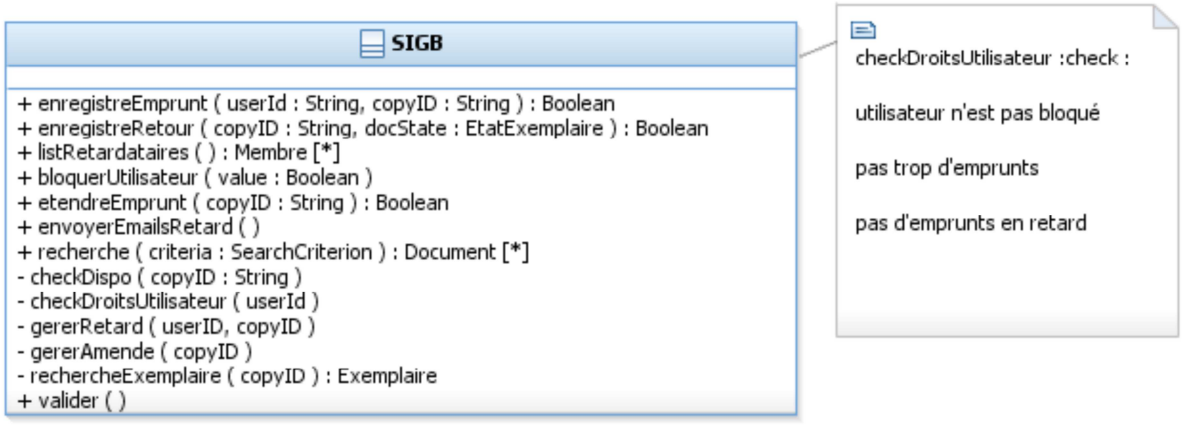
Q6. **Découpage orienté structure.**
En partant du diagramme de classes métier réalisé en analyse, identifiez une découpe possible sur des composants. Il faudra pour cela éliminer les dépendances structurelles, à l'aide d'identifiants. On peut être amené à séparer des classes et à ajouter des classes ou à modifier la répartition des données sur les classes. On proposera 3 ou 4 composants.

TD5

Notions abordées : Découpe fonctionnelle, interfaces, composants, configuration.

Découpage orienté fonctionnalités.

L'analyse a permis de dégager la liste (sans doute pas exhaustive) des responsabilités suivantes. Utilisez ce diagramme comme base, ou le travail réalisé aux séances précédentes. Les valeurs de retour Boolean sont utilisées pour noter que divers types d'erreurs peuvent se produire au cours de cette opération, empêchant de les mener à bien. En pratique, on utilisera des exceptions pour traiter ces cas en réalisation.



Q1. Proposez une découpe de ces responsabilités selon des critères logiques, et partitionnez-les sur des interfaces couvrant ces fonctionnalités. On pourra aussi essayer de mettre en valeur le découpage en fonctionnalités selon les acteurs.

Imaginons que chaque interface ainsi définie soit offerte par un composant dédié. On va travailler la cohérence entre les aspects structurel (TD 4) et fonctionnel, les interfaces définies devant correspondre avec les données manipulées. On affine donc les diagrammes élaborés pour

1. casser les dépendances structurelles entre composants en introduisant des identifiants.
2. S'assurer que chaque composant dispose bien des données nécessaires pour réaliser les traitements dont il est responsable
3. S'assurer que les types manipulés dans les opérations d'interface sont soit des types de base (String, int…) soit des interfaces. Introduire des interfaces au-dessus des classes concrètes au besoin.

Q2. A l'aide de diagrammes de séquence de niveau composant et en raisonnant sur les données nécessaires pour réaliser un traitement, identifier des dépendances entre composants. Essayez dans la mesure du possible d'orienter les dépendances identifiées pour éviter les cycles. Enrichissez vos interfaces existantes ou créez-en de nouvelles pour loger les opérations qui traduisent ces dépendances.
On cherche à se rapprocher de traitements réalisables par les composants.
Q3. Afin de faire apparaître l'IHM dans le modèle UML, nous choisissons une approche de conception qui consiste à construire un composant nommé « IHM ». Ce composant n'a pas d'interface offerte mais utilise les autres composants. Définissez ce composant.

Q4. Représentez à l'aide d'un diagramme de composants les composants proposés. On

représentera aussi les interfaces explicitement avec leurs signatures. Quelles sont les qualités et défauts principaux de l'architecture retenue ?

Q5 : Décrivez brièvement cette découpe en précisant le rôle des composants et leurs relations. Critiquez votre travail (forces et faiblesses).
Q6. Modélisez sur un diagramme de structure interne la configuration nominale des composants proposés.
Q7 : Expliquez en quoi le modèle de composants établi en conception générale est relativement indépendant des plateformes et langages de réalisation.

TD6

Notions abordées : Conception détaillée, réalisation de « composants » en Java (SE).

Q1. En partant de la conception architecturale réalisée, ajoutez à chaque composant une classe jouant le rôle de Façade et qui implémente l'interface offerte du composant.

Q2 : A l'aide des diagrammes de séquence de niveau composant, vérifiez que chaque composant a la capacité de traiter les opérations qu'il offre. Au besoin, enrichissez les interfaces offertes des composants.
Q3 : Les classes de conception du composant IHM représentent les écrans et les boutons de l'IHM. Proposez une approche pour les modéliser en UML

Q4 : Ajoutez une classe jouant le rôle de Factory pour les composants. On utilisera une forme simple portant des opérations *static*.

Q5 : Expliquez comment obtenir l'assemblage de composants décrit en Q6 du TD5 à l'aide de cette factory.

Q6 : Que manque-t-il aux modèles UML pour être pleinement exécutables ? Pourquoi n'est-ce pas souhaitable de chercher à décrire ce niveau de détail directement en UML ?

Q7 : Quels sont les avantages et inconvénients d'un modèle décrit au niveau d'abstraction « code » ?

Q8 : Pensez-vous avoir fini la phase de conception ?

TD7

Conception détaillée, Tests d'intégration, Tests Unitaires

Q1. On va se pencher à présent sur la conception détaillée du moteur de recherche de documents.
On supposera que l'on part d'une chaine de texte décrivant la requête utilisateur, par exemple : author= « E. Gamma » and year=1995
On commencera par proposer une représentation de l'arbre de syntaxe abstraite de la requête, à l'aide du DP Composite.
On décrira ensuite une façon de réaliser cette recherche, en supposant

a) Une implémentation OO
b) Une implémentation sur un moteur de base de données relationnel.

Tests d'intégration

Q2. Rappelez la nature et l'objectif des tests d'intégration.

Q3 : Rappelez en quoi les diagrammes de séquence inter-composant (cf. TD 5) peuvent être utilisés lors de la rédaction des tests d'intégration.

Q4 : Rappelez pourquoi ces diagrammes ne peuvent cependant pas être utilisés tels quels lors de la rédaction des tests d'intégration.

Q5. Définissez à l'aide de diagrammes de structure interne des configurations des composants permettant de tester chacun d'eux. On introduira pour cela au besoin des composants bouchon.

Q5 : Définissez le jeu de données qui sera utilisé lors de l'exécution des tests d'intégration. Rappelez en quoi votre jeu de données ne peut être complet et ne peut être que satisfaisant.

Q6 : Définissez les tests d'intégration de l'application SIGB.

Tests Unitaires

Q7 : Rappelez l'objectif des tests unitaires et expliquez la différence entre test unitaire et test d'intégration.

Q8 : Quelles métriques utiliser pour mesurer la qualité d'un jeu de tests unitaires.
Q9 : Proposez une extension à UML afin de pouvoir spécifier des tests.

TD8

Adapter, Façade

Réutilisation de l'existant

L'analyse d'un système de réservations de salles a produit le diagramme de cas d'utilisation suivant :

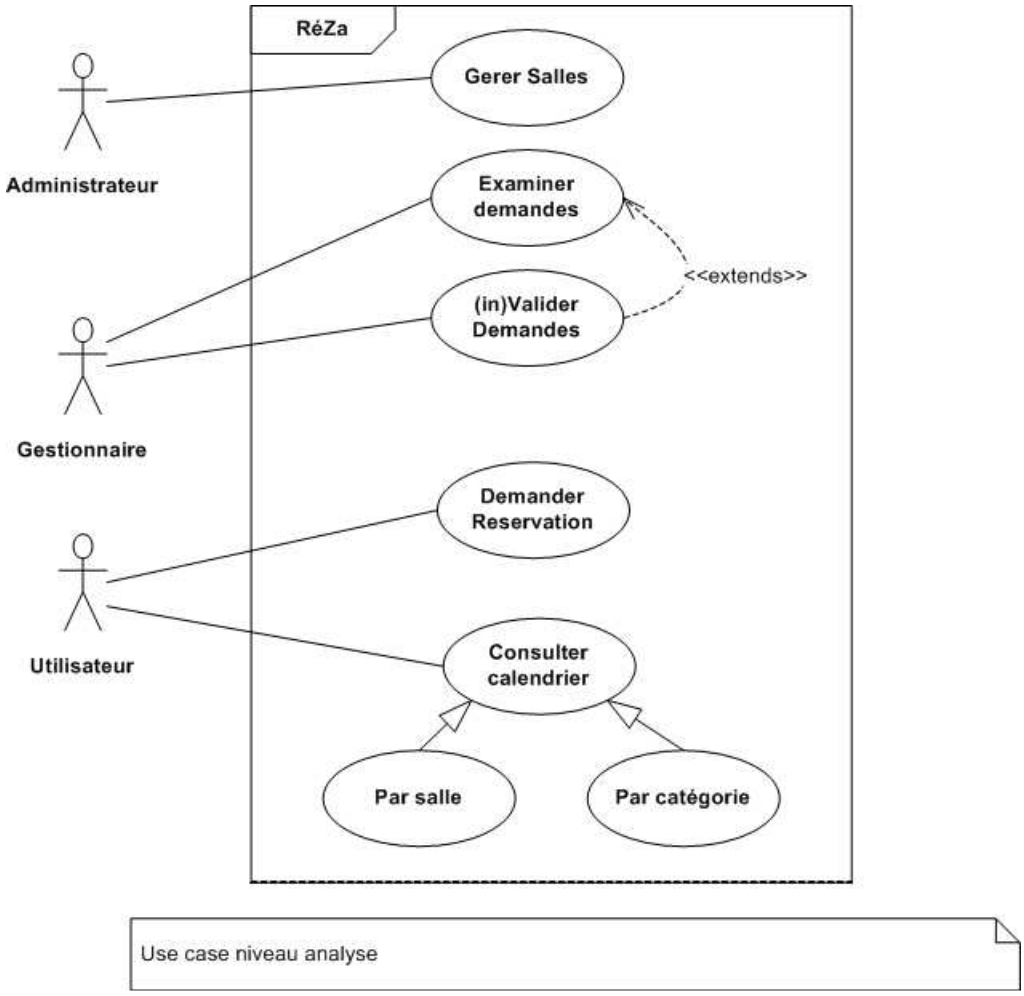


Fig. 1 : Use cases

Ces cas d'utilisation sont détaillés ici :

Acteur : Administrateur

- L'administrateur gère la liste des salles disponibles. On ne le détaillera pas dans la suite.

Acteur : Utilisateur

- Consulte les emplois du temps et fait des demandes de réservation

Demander Réservation :

Il existe deux types de réservations : les réservations ponctuelles et les réservations périodiques. Pout toute réservation le demandeur doit préciser son nom (par exemple « Y.Thierry-Mieg »), la catégorie à laquelle la réservation est liée (par exemple « ueMI017 »), un nom pour cette réservation (par exemple « Examen rattrapage »), la salle demandée (par exemple « Amphi B1 »), et la date et plage horaire désirée (par exemple « 22 Janvier, 15h30-17h30 »).

Si la réservation est périodique, l'utilisateur le précisera sur le même formulaire, et fixera la

périodicité (quotidienne, hebdomadaire ou mensuelle) et le nombre d'occurrences (par exemple pour « Cours MIO17 », 10 occurrences de périodicité hebdomadaire).
Spécifications complémentaires :

- Une demande périodique de n occurrences sera traitée comme si l'utilisateur avait fait n demandes ponctuelles du point de vue de la validation (chaque occurrence pouvant être individuellement validée ou non).

Consulter Calendrier

La consultation des emplois du temps permet aux utilisateurs d'obtenir un calendrier qui correspond à leurs critères de recherche. Il existe deux types de calendrier :

- Par catégorie : chaque réservation contient un champ « catégorie », qui permet de préciser à quel évènement cette réservation se rapporte. Par exemple « ue MI017 ». La recherche par catégorie permet d'obtenir un emploi du temps couvrant toutes les réservations qui se rapportent à la catégorie.
- Par salle: il est possible d'obtenir l'emploi du temps d'une salle particulière, sur une période donnée. L'utilisateur n'a qu'à préciser la salle et les dates de début et de fin du calendrier à produire.

Acteur : Gestionnaire

- Chargé de valider les demandes de salles

Examiner demandes et (In)Valider demande

Le gestionnaire peut consulter toutes les demandes de réservation non validées. Il les inspecte alors une par une et décide de les valider ou les invalider.
Spécifications complémentaires :

- Dans les deux cas, le demandeur de la réservation sera informé par mail de la mise à jour de l'état de sa réservation.

Q1. L'équipe en conception a défini le composant Reservations de la façon suivante (fig.2). Quel est l'intérêt d'un tel composant ? Comment le réaliser en pratique ? Quel design pattern est utilisé pour accéder à l'unique instance de Reservations ?

Q2. L'équipe a décidé de découper les responsabilités de la classe Système d'analyse sur des interfaces distinctes pour chaque use case principal (fig. 3). On suppose que ces interfaces sont offertes par un seul composant qui joue le rôle de contrôleur, représentez cette situation sur un diagramme de composant.

Q3. Interactions pour la gestion des réservations.
Modélisez à l'aide d'un ou plusieurs diagrammes de séquence de niveau intégration, c'est-à-dire où les lignes de vie représentent des composants, les interactions nécessaires pour demander une réservation ponctuelle de la salle B1 le 22 janvier de 15h à 17h, puis que le gestionnaire valide cette demande.

Q4. Le contrôleur s'appuie sur le composant Calendrier, trouvé sur étagère, permettant la génération d'images représentant des calendriers. L'API de ce composant est la suivante (fig.4). Quel pattern de création reconnaissez-vous ? Mettez à jour le diagramme de composants de la question 2 pour représenter que le contrôleur s'appuie sur ce composant.

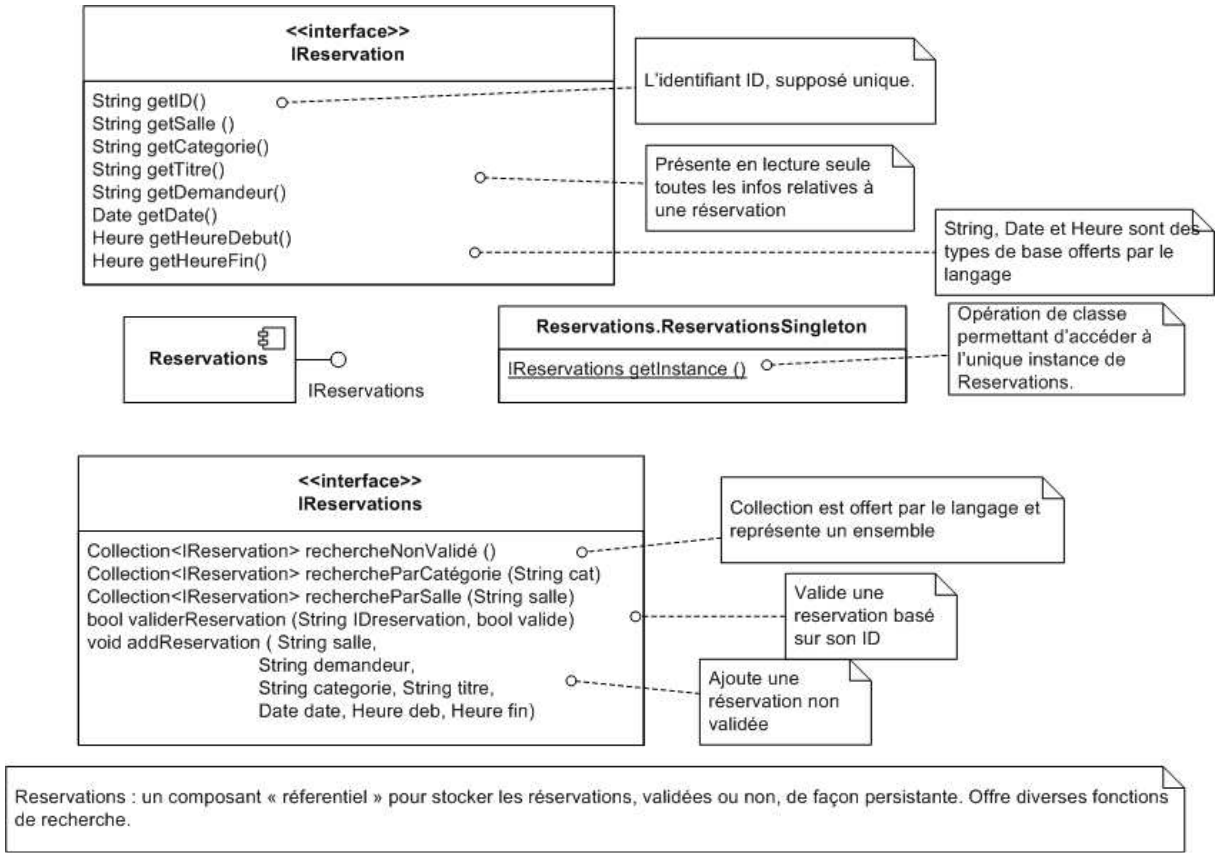


Fig.2 : Composant Reservation

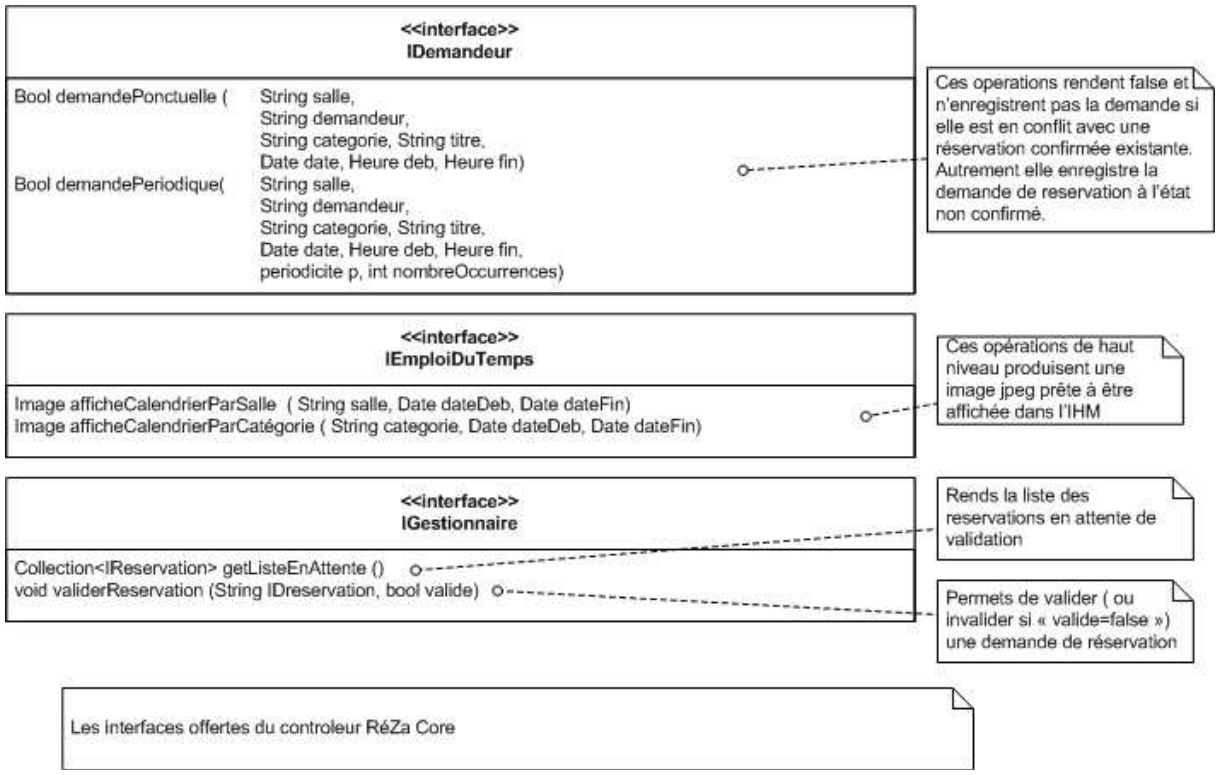
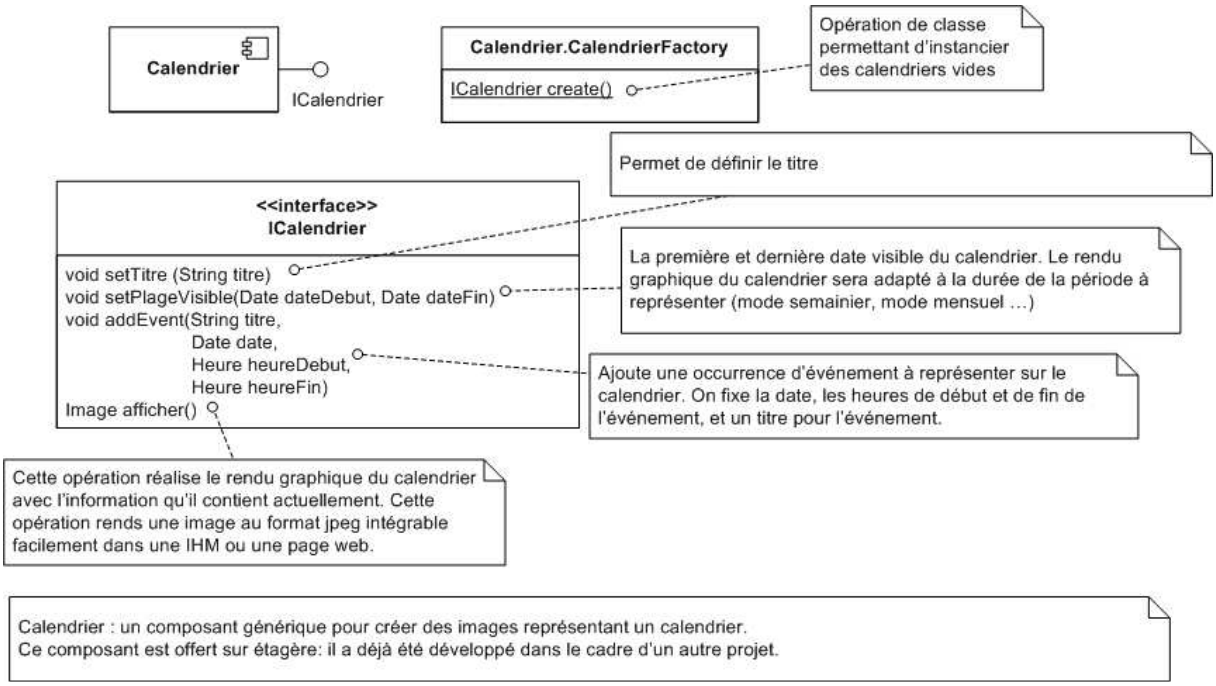


Fig.3 : Les interfaces



Q5. Modélisez à l'aide d'un ou plusieurs diagrammes de séquence de niveau intégration, c'est-à-dire où les lignes de vie représentent des composants, les interactions nécessaires pour qu'un utilisateur consulte un calendrier portant sur la salle B22 du 1er au 15 janvier.
Q6. Comment réaliser le composant contrôleur en pratique ? Quel(s) design pattern(s) appliquer ?

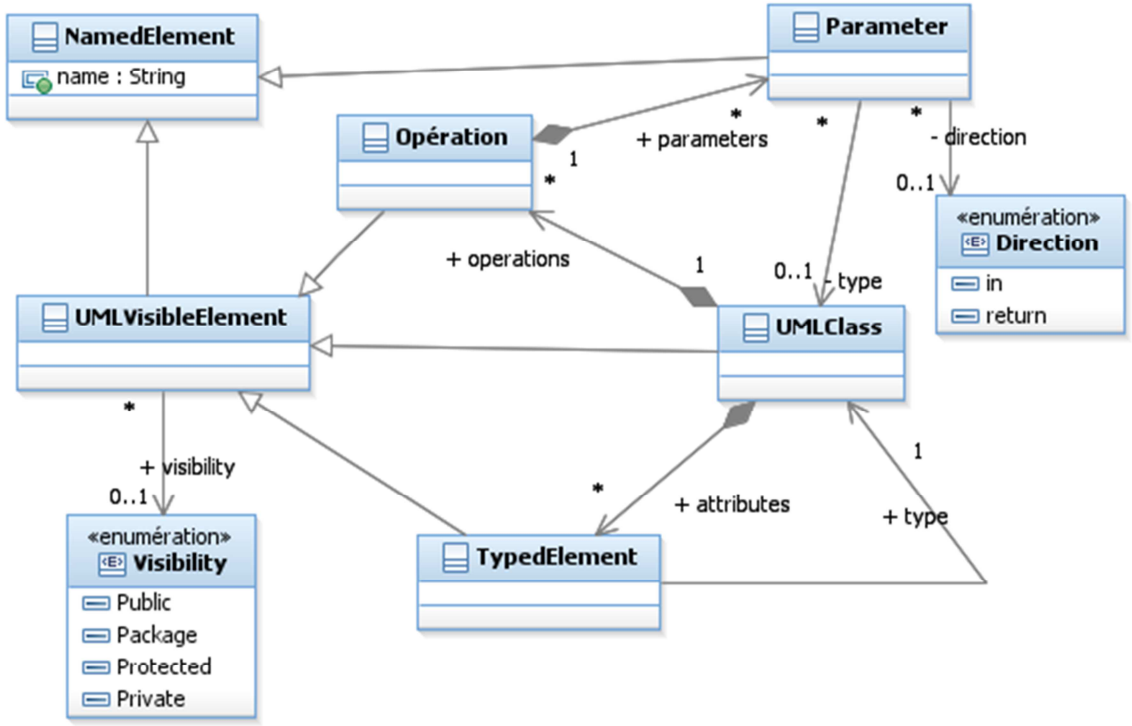
TD9

Cycles de développement
Opérations de production, Méta-modélisation

Q1 : Rappelez, pour chaque étape du cycle en V, les diagrammes UML utilisés. Expliquez le rôle de chacun de ces diagrammes et les différentes opérations de production qu'il est possible de réaliser (génération de code ou autre).
Q2 : Définissez les étapes du processus "Extreme Programming".
Q3 : Proposez une exploitation d'UML pour l'extreme programming XP

Q4 : Rappelez les différents types d'opérations de production sur les modèles et précisez l'intérêt de ces opérations. Donnez pour chaque type un exemple concret.

Q5 :
Soit l'extrait simplifié du méta-modèle des diagrammes de classe d'UML :



Décrivez les objets montrant comment ce méta-modèle s'instancie pour modéliser une classe Chainon, munie d'un attribut « next » de type Chainon, et d'une opération addNext(in c : Chainon) . On utilisera une représentation arborescente, proche de la structure du fichier XMI.
Q6. Nous allons travailler sur la génération de code qui est une opération de production "Modèle vers Texte". Reprécisez l'algorithme de génération de code pour les classes avec leurs attributs (sans prendre en compte l'héritage, la réalisation d'interface et les opérations). On s'intéressera particulièrement à la conversion des types des attributs.
Q7. Cette transformation est-elle facilement réversible ? Quelles opérations faut-il mettre en place pour le faire ?

M1 : Ingénierie du Logiciel UNIVERSITE PIERRE & MARIE CURIE

Examen Réparti 2eme partie 16 Décembre 2010

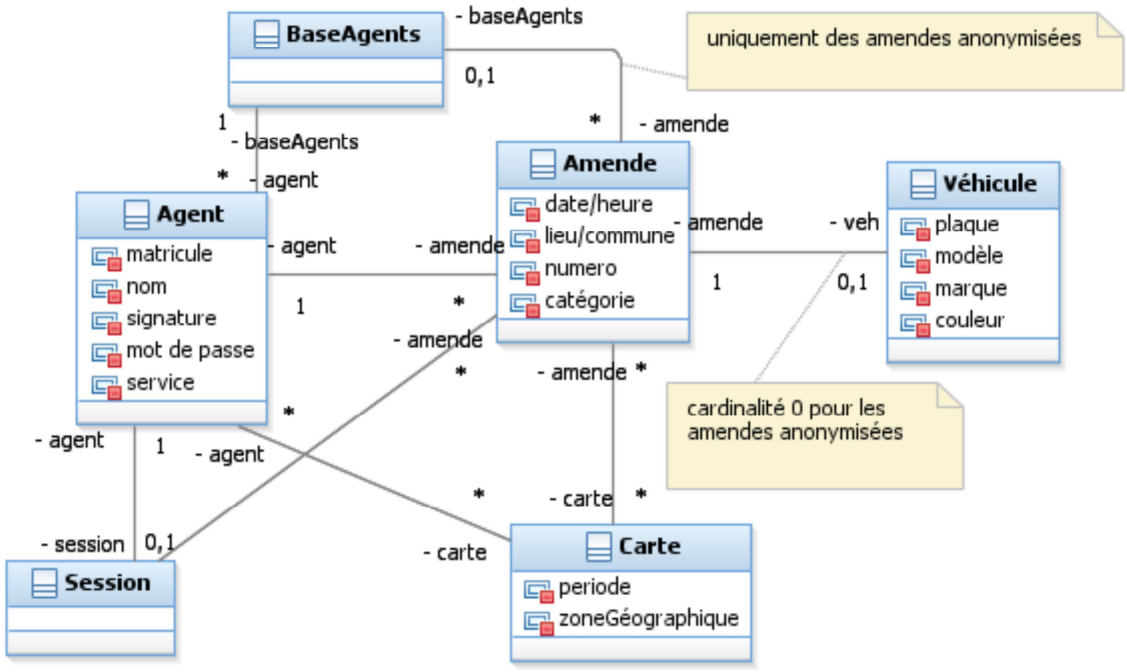
2 heures avec documents -- Barème indicatif sur 20 points.

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.

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.

Question 3 (4 points)

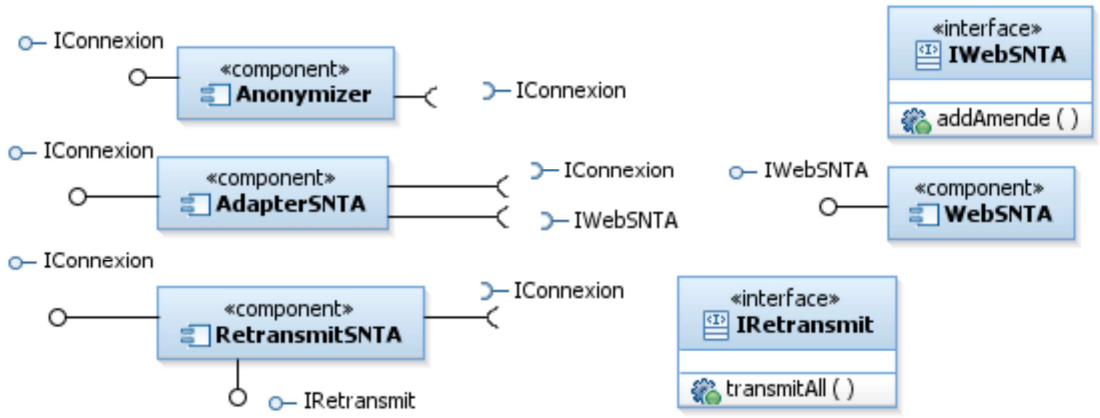
Représentez par un ou plusieurs diagrammes de séquence les interactions entre l'IHM de l'agent et le Terminal. On couvrira tous les cas de figure identifiés par l'analyse.

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.

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 ?

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…).

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.

UPMC-LIP6

TME

MI017- Ingénierie du Logiciel

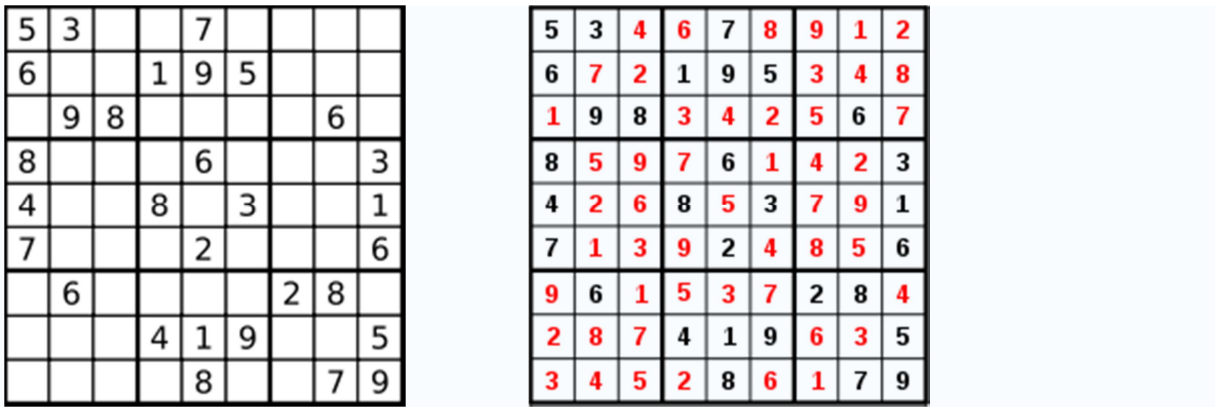
Yann Thierry-Mieg

2012/2013

iSudoku : l'assistant Sudoku à portée de main

Votre SSII à décidé de se lancer sur le marché du smartphone avec une « killer app » qui va faire du bruit.
iSudoku est une application pour plateforme mobile qui permet de s'entraîner et de progresser au Sudoku.

Le but du jeu de Sudoku est de remplir la grille composée de 9 sous-grilles avec une série de chiffres allant de 1 à 9 tous différents, qui ne se trouvent jamais plus d'une fois sur une même ligne, dans une même colonne ou dans une même sous-grille. Les sous-grilles sont des carrés de 3 × 3.
Quelques chiffres sont déjà disposés dans la grille, ce qui permet progressivement de déduire les autres valeurs.



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

L'application devra supporter les fonctionnalités suivantes :

- Permettre de façon intuitive et ergonomique d'afficher et de résoudre des grilles de Sudoku.
- Différentes difficultés seront proposées, de la plus difficile (on ne peut valider la grille que quand elle est finie) à la plus facile (dans chaque case, le système n'offre que les valeurs possibles au vu du reste de la grille, et/ou met en valeur les cellules qui n'ont plus qu'une option possible). En difficulté intermédiaire l'application signale une erreur au moment où elle est commise. Les options d'aide à l'utilisateur (signaler les erreurs, afficher des suggestions..) seront configurables par ce dernier.
- Générer des grilles de diverses difficultés, et utiliser des grilles existantes fournies avec l'application ou téléchargées.
- En prenant une photo d'une grille (typiquement celle de votre journal), le système permettra de la charger dans l'application et ainsi à l'utilisateur de la résoudre. Cette fonctionnalité s'appuie sur un module de reconnaissance de forme existant déjà développé dans la SSII.
- L'application permettra de mesurer le QI du joueur. Il s'appuie pour cela sur la difficulté des grilles traitées et le temps mis à les résoudre. On pourra voir les statistiques du joueur et sa progression au fil du temps. Le joueur sera encouragé à persévérer par des bonus quand il résout une grille.
- L'application permettra de publier ses résultats sur divers réseaux sociaux (face2bouc etc.) via un module de diffusion existant.

Organisation des TME

L'objectif des séances de TME est de construire une spécification et un implémentation du projet iSudoku. Le travail sera collaboratif par équipes de 6 à 8 étudiants.
Le travail de chaque étape sera réparti sur les membres de l'équipe. L'ensemble du travail du groupe sera présenté par l'équipe, à travers :

- Un document d'analyse et un cahier de tests de validation, rendu à mi semestre. Ce document présentera les acteurs et cas d'utilisation du système. Il pourra également comporter des ébauches d'écran d'interface, et les diagrammes de séquence de niveau analyse expliquant les responsabilités identifiées du système. La responsabilité de la rédaction des fiches sera partagée sur le groupe. Le cahier de tests offrira une couverture au moins partielle de chaque cas d'utilisation.
- Un document de conception, présentant l'architecture retenue pour la solution (composants, interfaces, instanciation nominale), des tests d'intégration pour au moins deux composants, et les principales séquences d'interaction niveau composant. La conception détaillée des composants sera présentée à l'aide d'un diagramme de classes par composant. Ce document sera rendu à la fin du semestre.
- Une présentation d'équipe, à la fin de l'ue (séance TME 10). En 20 à 30 minutes (selon le nombre de groupes d'étudiants dans le groupe de TD) le groupe devra présenter les grandes lignes du travail réalisé, et dans la mesure du possible faire la démo des fonctionnalités opérationnelles. Il est demandé qu'au cours de cette présentation tous les membres du groupe s'expriment.

L'ensemble de ces éléments participera à la note de TME.
Pour la coordination du groupe, il est recommandé de mettre en place rapidement un email de groupe, voire de créer un groupe sur un site communautaire (google group par exemple). Cela permet de disposer d'une zone où partager les documents facilement. Pour la gestion partagée des modèles et du code, une solution basée sur Mercurial pour la gestion de version sera mise en place à l'ARI.
La suite de ce document place quelques marqueurs pour vous guider au cours de votre travail ; l'objectif global est d'appliquer en groupe la méthodologie présentée en cours et TD à l'exemple du Sudoku.

TME1
Prise en main

L'objectif de ce TP est de présenter l'outil Rational Software Architect et d'illustrer les différentes possibilités qu'il offre.
Notez qu'il vous sera demandé une très grande **autonomie** lors de l'utilisation de cet outil tout au long des TME. En cela, n'oubliez jamais d'utiliser l'aide (**touche F1**) avant de poser vos questions. Il y a une page d'accueil (« Help->Welcome ») qui peut être utile aux débutants.
L'outil est disponible en version 8.5 sous linux (dans /usr/local/IBM/SDP/eclipse). Il faut un workspace, que vous placerez dans votre home. Attention, au premier démarrage, un certain nombre de fichiers de configuration sont créés ce qui ralentit le démarrage. Il peut être judicieux si le NFS peine d'utiliser un répertoire local pour travailler (e.g. dans /tmp/ sur linux) et de copier ce workspace dans votre espace personnel en fin de session.

Notons que cet outil est basé sur Eclipse, et qu'une familiarité avec cet IDE sera un sérieux avantage au cours des premières séances.

Partie 1 : Découverte d'une application aux travers des fichiers sources

Récupérez les sources d'une application java que vous avez écrite (d'au moins 500/1000 lignes), ou le petit projet exemple fourni « Chat » sur le site de l'UE.

A l'aide de l'outil, construisez un diagramme de classe par package.
Pour ce faire, utilisez la séquence :

- Sélection des classes à afficher
- Bouton droit->Visualize->Add to new diagram file->class diagram
- Déplacez/arrangez les classes
- Jouez un peu avec les filters, on peut le faire sur une multi sélection de classes:
    - Sélection d'une classe->Filters->Show/Hide Relationship puis décochez le « use » et cocher le « Collection »
    - Filters->show signature permet d'afficher la signature des opérations. Il faut par contre configurer le plugin Java pour avoir les types de retour via :Window->Preferences->Java->Appearance->show return type in signature
    - Filters->Stereotype and visibility style -> visibility style text permet de modifier l'affichage des visibilités

Voyagez entre le modèle et le code : double cliquez sur un attribut ou une méthode pour atteindre le source. Modifiez le source ou le modèle, l'autre est mis à jour de façon synchronisée.

Essayez de construire un diagramme de séquence, en sélectionnant une opération et en faisant Visualize->new diagram file->static sequence diagram. Choisissez une opération un peu complexe de préférence, par exemple le « run » de AbstractTCPServer.

Essayez l'outil Software Analyzer en cliquant sur un projet Java. Bouton droit sur le projet->Software Analyzer, créez une nouvelle configuration (un peu comme le menu Run) et choisissez les règles Java dans la deuxième page.

Partie 2 : Découverte d'une application au travers d'un modèle UML

Créez un nouveau projet de nature UML (New->Project->Modelling->Uml Project).
On utilisera le paramétrage par défaut avec un « General->Blank Package »

Créez un diagramme de use case, sélection du package dans le modèle -> Add Diagram->Use case diagram. Ajoutez un sous-système, quelques use case et un ou deux acteurs. Liez les acteurs aux use case par des « bidirectional association ».

Créez un diagramme de classes et au moins deux classes et une association entre les deux. Pour ajouter des opérations à la classe, utilisez le menu flottant, le rectangle ajoute des attributs, la roue crantée ajoute des opérations. On peut aussi quand on flotte sur une classe avec la souris utiliser les flèches qui apparaissent pour tirer des liens.
Ces menus s'adaptent à l'objet sélectionné, sur une opération cela permet d'ajouter des paramètres par exemple.

Créez un diagramme de séquence. Tirez un de vos acteurs et une de vos classes, du modèle à gauche directement sur le diagramme. Ajoutez une invocation de l'acteur sur la ligne de vie (synchronous message). De nouveau le plus facile est d'utiliser le menu flottant avec les flèches. On notera que le système propose d'invoquer une opération existante ou d'en créer une à la volée.

Essayez de créez une transformation UML->Java ou Java->UML. Sélection d'un projet->Transform->New configuration, on choisira le mode « Conceptual » puis on sélectionne une source (e.g. un modèle ou package UML) et une cible (un projet Java existant ou qui sera créé à la volée). Utilisez le bouton « Run » pour lancer la transformation.

S'il vous reste du temps, créez d'autres types de diagrammes et explorez les possibilités offertes par l'outil.

TME2

Analyse de iSudoku

L'objectif de ce TME est de réaliser la phase d'analyse de l'application « iSudoku ».

Lancez RSA.
Créez un projet UML vierge:

- nom : sudoku_Nom1_Nom2
  où x correspond à votre numéro de groupe et où Nom1 et Nom2 correspondent aux noms des binômes.

Ce projet sera l'unique projet pour tous les TME (TME 2 à TME7)
Ce projet contiendra les phases d'analyse, de conception et de réalisation de code !

Q1 : Construisez un modèle nommé « analyse ». Sélectionnez ce modèle et construisez un nouveau diagramme de cas d'utilisation. Dans ce diagramme, définissez les acteurs et les cas d'utilisation de l'application.

Q2 : Sélectionnez le package « analyse » et construisez un nouveau diagramme de classes. Dans ce diagramme de classes, définissez les classes d'analyse de l'application.
N'hésitez pas à utiliser beaucoup de notes pour commenter votre diagramme.
On les pose avec la palette du diagramme, à droite, dans l'onglet « UML common », ou via menu flottant : tirez un lien à l'aide des flèches flottantes depuis un objet à commenter du diagramme, terminez le geste dans le vide, « Create reference To->new element : Comment ».

TME3

Tests de validation

L'objectif de ce TME est de finir la phase d'analyse de « iSudoku » commencée lors du TME2. Les tests de validation seront écrits à l'issu de cette phase.

Lancez sur votre projet du TME2.

Q1 : Pour chaque cas d'utilisation, sélectionnez le cas d'utilisation et remplissez sa documentation, sous la forme d'une fiche détaillée. La documentation est accessible dans un des onglets de la vue Properties du cas d'utilisation (Window->view->Properties si elle n'est pas déjà visible). On dispose d'un éditeur RTF.
Essayez de générer la documentation de votre système, à l'aide du bouton Modeling->Publish, visible quand un modèle est sélectionné dans la perspective Modeling. Explorez les options et onglets disponible sur la page de publication ; cochez l'option de l'onglet « Properties » : Remove empty properties. Positionnez un répertoire de sortie raisonnable (attention ça crée toute une arborescence de fichier à partir de là).
Complétez un peu votre modèle pour améliorer sa documentation. On pourra aussi jouer avec les options du menu « publish ».
NB : Le suivi des besoins (requirements) se fait en principe dans la suite Rational à l'aide de l'outil « Requisite Pro » qui permet de lier les use case et autres artifacts du modèle à un modèle des requirements. Cet outil spécialisé est dédié au métier d'analyste du besoin, et s'intègre à RSA, mais n'est pas déployé sur vos machines. Il permet des fonctions avancées de suivi des besoins à travers le cycle de vie, génération de matrices de traçabilité entre autres. Cependant dans le cadre de ce module il parait un peu « overkill », on se contentera donc des notes de documentation.

Q2 : Ouvrez le diagramme de cas d'utilisation. Pour chaque cas d'utilisation, sélectionnez le cas d'utilisation et construisez un nouveau diagramme de séquence. Définissez dans ce diagramme la séquence nominale correspondant au cas d'utilisation. N'oubliez pas de lier les instances appartenant à vos diagrammes de séquence avec les classes que vous avez déjà définies. On introduira au passage la classe représentant l'application et on enrichira ses opérations.
Note : pour construire la signature des opérations, plusieurs options sont possibles. Via la « Properties view » on peut éditer dans l'onglet « Parameters » les arguments de l'opération. On ajoute les paramètres un à un et on édite les champs de la table. Une autre approche beaucoup plus commode consiste à utiliser la syntaxe textuelle, quand on édite le nom de l'opération : nomOperation ([in] param1 : Integer , [in] param2 : String) : Boolean.
La complétion/correction au cours de la frappe est disponible. Un click sur la méthode quand elle est sélectionnée (attention pas un double click) permet d'ouvrir l'édition textuelle.

Q3 : Pour chaque cas d'utilisation, construisez les autres diagrammes de séquence définissant l'ensemble des comportements voulus par le cahier des charges de l'application.

TME4

Q1 : A l'aide des diagrammes de séquence et à l'aide des fiches détaillées des cas d'utilisation, élaborez l'ensemble des tests de validation. On saisira les tests de validation dans les notes de documentation accompagnant les use case qu'ils valident.

Générez encore votre documentation. Il vous sera demandé de rendre ce travail pour contribuer à votre note CC.

Conception

Q2 : Construisez un package « conception » par copie de votre modèle d'analyse. On prendra soin de travailler dans cette copie, pour préserver l'état de votre spécification à la fin de l'analyse.

Q3 : Proposez une découpe en composants. Pour chaque composant, introduisez un package puis réalisez les diagrammes de classes du composant en découpant les classes métier.

TME5

Q1. Dans le modèle de « conception » du TME 4, réalisez un diagramme présentant tous les composants et leurs interfaces requises/offertes.
On prendra soin de distribuer les responsabilités de l'application sur des interfaces nouvellement introduites.

Q2. Elaborez les diagrammes de séquence correspondant aux cas d'utilisation de l'application (ceux définis en analyse). On reste sur une granularité « inter-composants ».
On enrichira au fur et à mesure les interfaces et les diagrammes de classe des composants de l'application élaborés à la séance 4.

Q3. Introduisez un composant « iSudoku » muni du stéréotype <<subsystem>>. Définissez sa structure interne en instanciant et en connectant les composants que vous avez défini, selon la configuration nominale envisagée.

TME6

Conception Détaillée

Q1 : A partir des diagrammes de classe partiellement établis pour les composants, complétez les à l'aide de Façade pour vous rapprocher d'une structure implémentable en Java. On évitera cependant de s'enfoncer trop profondément dans la conception détaillée.

NB : Si la conception n'est pas encore satisfaisante, itérez pendant une séance sur les questions des TME 4,5 et 6. Le sujet 7 est le dernier qui demande encore de modéliser. Les séances 8 et 9 seront consacrées à améliorer les spécifications et modèles déjà construits, et à attaquer la réalisation. Les projets de TME seront évalués au cours de la séance 10 (voir avec le chargé de TD pour les modalités).

TME7

Tests d'intégration

Q1 : A partir des séquences d'interaction inter composants, définissez les tests d'intégration. Construisez le document des tests d'intégration. Définissez en particulier les jeux de données nécessaires à ces tests.

Q2 : Construisez les classes/composant bouchons permettant un développement « en parallèle » de chacun des sous-modules.

Q3 : Représentez sur des diagrammes de structure interne des configurations de test d'intégration des composants.

TME8

Vers la réalisation

Q2 : Configurez et lancez une transformation UML vers Java, pour les package qui correspondent à vos composants.

Q3 : En travaillant avec les diagrammes « Vue sur le code » et en développement Java pur, implémentez un maximum des cas d'utilisation identifiés.

NB :

1. il peut y avoir des divergences entre votre code et le modèle de conception, essayez de les identifier, mais il n'est pas demandé de chercher à maintenir à tout prix la cohérence entre les deux.
2. La notation cherche à évaluer votre capacité à spécifier et à concevoir une architecture, pas vraiment votre niveau en Java. Le « produit fini » n'est donc pas le facteur prédominant dans la note de TP.