Serveurs d'applications Java

Projet Etudiants
Exigences & Conseils

Dan Dimcea (dimcead@fr.ibm.com)

Master 2 MIAGE IF
Université Paris - Dauphine

Objectif pédagogique

- Le projet consiste à vous familiariser au développement Jakarta EE à travers la réalisation d'une application de type client/serveur Web.

- L'objectif est de démontrer :
  - la compréhension et le bon usage des technologies et des outils abordés dans le cadre des sessions du cours « Serveurs d'Applications Java »
  - la capacité de travail en équipe, de planification des travaux, de répartition des tâches et de leur réalisation en respectant les contraintes imposées.

Soutenance

- **La soutenance permettra aux étudiants de** :
  - Présenter leur travail au sens large,
  - Faire une démonstration de leur application,
  - Montrer leurs connaissances en répondant à un questionnaire sur la totalité des matières abordées dans le cours. Les détails sur le questionnaire seront fournis ultérieurement, au plus tard lors de la dernière session du cours.
- **La soutenance dure 1h15 par groupe**

Planning projet

- **Le planning et les échéances du projet étudiant** :
  - Points d'avancement du projet (1-2 pages) à envoyer par mail à l'adresse dimcead@fr.ibm.com toutes les deux semaines à partir du démarrage officiel du projet, à savoir, après la dernière session de cours.
  - Remise du projet complet par email à l'adresse dimcead@fr.ibm.com. La date finale de remise est le 9 février 2021 (~ 8 semaines après le début de la réalisation du projet étudiant).
  - Remise d'un exemplaire papier du rapport projet au secrétariat du Master.
  - La date de la soutenances et l'ordre de passage des groupes seront communiqués ultérieurement.
  - Possibilité de poser des questions tout au long du projet.

Livrables projet (1/3)

- **Synthèse Projet** : 5 pages au maximum de synthèse sous forme de fichier PDF, avec les rubriques suivantes :
  - noms / prénoms des membres du groupe,
  - rôles de chacun au sein du groupe,
  - macro-planning + séquencement des activités + charge de travail,
  - liste détaillée des fonctionnalités et exigences techniques réalisées et pas réalisées dans l'application
  - ce que vous retenez du projet et ce que vous en avez appris,
  - ce que vous changeriez si c'était à refaire.

Livrables projet (2/3)

- **Rapport Projet** : minimum 30 pages sous forme de fichier PDF et contenant au minimum les chapitres suivants
  - Court rappel du cahier des charges
  - Hypothèses, contraintes et exigences considérées,
  - Analyse :
    - modèle UML des cas d'utilisation (voir annexe 1) avec description de chaque cas d'utilisation
    - modèle UML des classes métier (voir l'annexe 2) et description de chaque classe
  - Conception technique
    - architecture applicative et décisions d'architecture,
      - diagramme d'architecture applicative avec les principaux types d'objets et leurs Interactions (voir l'annexe 3); description des rôles de chaque type d'objet
      - description du mécanisme de gestion transactionnelle
      - description du mécanisme de gestion de la sécurité
    - diagrammes UML d'interaction des principaux use case (voir l'annexe 4)
    - description de l'IHM (interface homme/machine) de l'application
    - description du mécanisme de gestion des erreurs et des logs; typologie des exceptions
    - choix des technologies (JSP/JSF, EJB, Servlet, JDBC, etc.)
  - Implémentation
    - choix des frameworks (MVC, persistance, Java Script, log etc.)
    - choix d'implémentation

Livrables projet (3/3)

- **L'application** :
  - Le code binaire (exécutable) sous forme de EAR ou WAR,
  - Le code source. Celui-ci doit être inclus dans le WAR/EAR,
    - **Remarque** : si vous faites héberger votre projet sur GitHub ou un autre hébergeur de ce type, vous pouvez fournie l'adresse URL du projet;
  - Le script SQL d'installation du schéma de la base de données,
  - Le script SQL d'insertion des données dans la base,
  - Le document PDF contenant la procédure d'installation de l'application dans le serveur d'application.

Enoncé du projet

- L’énoncé du projet détaillé sous forme de PDF peut être trouvé dans le répertoire « *Projet Etudiant* » de l’espace de partage du cours.

Exigences techniques (1/3)

- **Architecture générale** :
  - Architecture trois tiers basée sur un serveur d'application (WildFly 8.x de préférence mais pas obligatoire),
  - Utilisation du pattern MVC,
  - Gestion de la sécurité : implémentation d'un mécanisme de gestion de l'authentification/autorisation pour limiter l'accès à l'application et restreindre les fonctionnalités par rapport aux profiles des utilisateurs,
  - Gestion transactionnelle : utilisation du mode CMT – tous les accès en base doivent être fait dans le cadre des transactions JTA,

Exigences techniques (2/3)

- **Tiers client (interface utilisateur)** :
  - L'interface utilisateur s'exécute dans un navigateur internet (IE, Firefox etc.) à travers lequel l'utilisateur accède à la partie serveur de l'application,
  - Les pages peuvent être implémentées dans la technologie JSF ou JSP, générant du HTML5,
  - Possibilité d'utiliser le java script dans les pages (dans ce cas, l'utilisation d'un framework java script est conseillée).
- **Tiers serveur de traitement** :
  - Technologies exigées :
    - Servlets,
    - Managed Beans (Obligatoire dans le cas de l'utilisation des JSF),
    - Stateless Session Beans EJB,
    - Entity Beans (JPA),

Exigences techniques (3/3)

- **Tiers serveur de données** :
  - Les données doivent être stockées dans une base de données relationnelle de type MySQL
  - L'accès au données à l'aide de JDBC, JPA et d'un framework de persistance de type ORM (de préférence Hibernate mais un autre framework ORM pourrait également être utilisé).

- **Développement** :
  - Il est préférable que le développement soit réalisé avec l'IDE Eclipse mais ce n'est pas une obligation.
  - D'autres frameworks de développement (Spring MVC, Struts etc.) peuvent être utilisé mais au moins 50% des use cases doivent être implémenté à l'aide des composants Jakarta EE natifs (Servlet, JSP ou JSF, EJB session, EJB Entity)

Options (bonus)

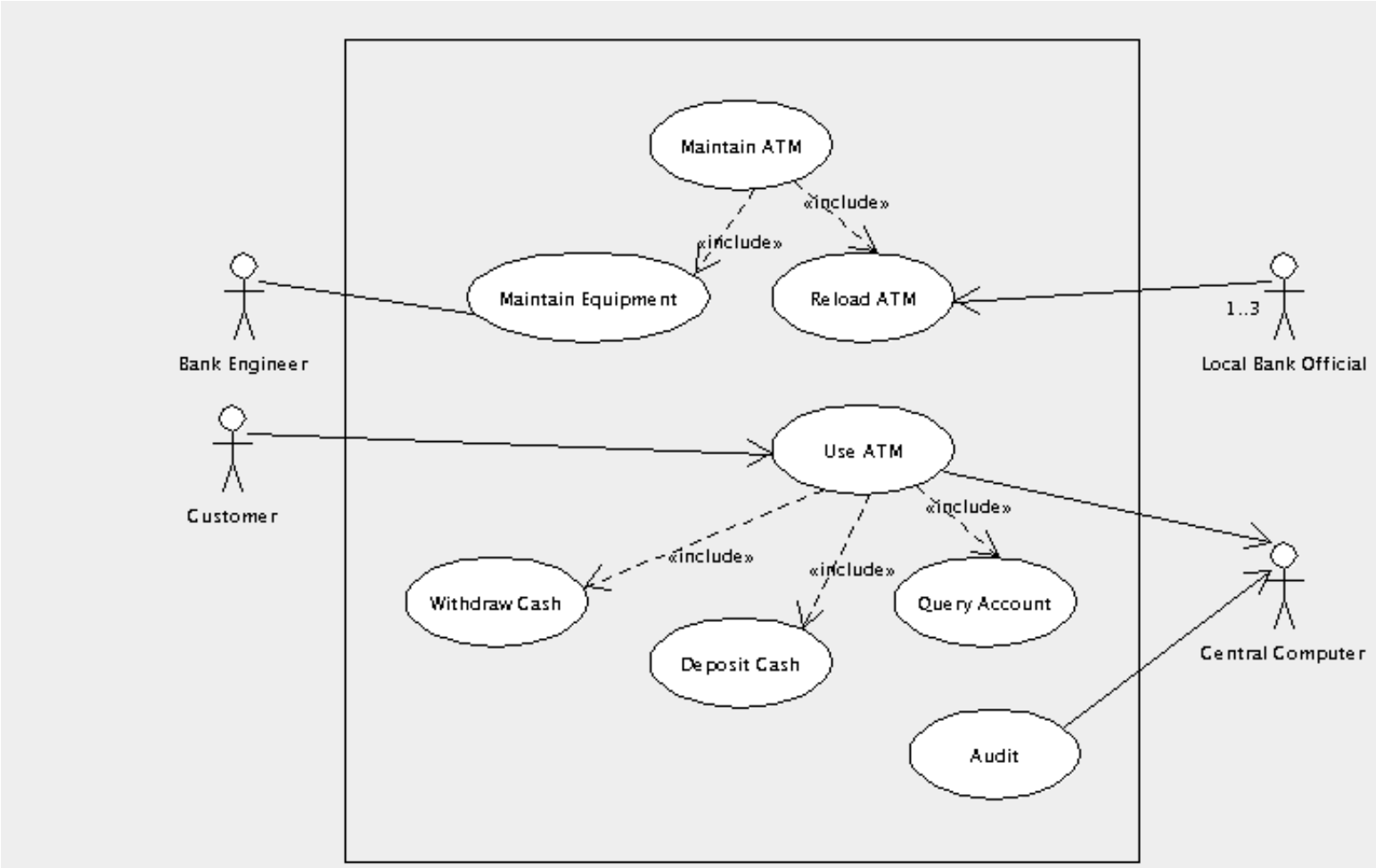
- Utilisation des EJB asynchrone.
- Utilisation des applets dans au moins deux page HTML.
- Utilisation des appels serveur de type AJAX dans au moins 50% des pages.
- Gestion du verrouillage des données.
- Implémentation des tests unitaires de l'application avec le framework Junit – au moins un test par use case. L'exécution avec succès des tests dans l'outil de développement (Eclipse ou celui de votre choix) doit faire partie de la démonstration de l'application lors de la soutenance.
- Utilisation de MAVEN pour la structuration du projet et l'assemblage.

- **Remarque** : chaque bonus apporte un ½ point dans la note finale.

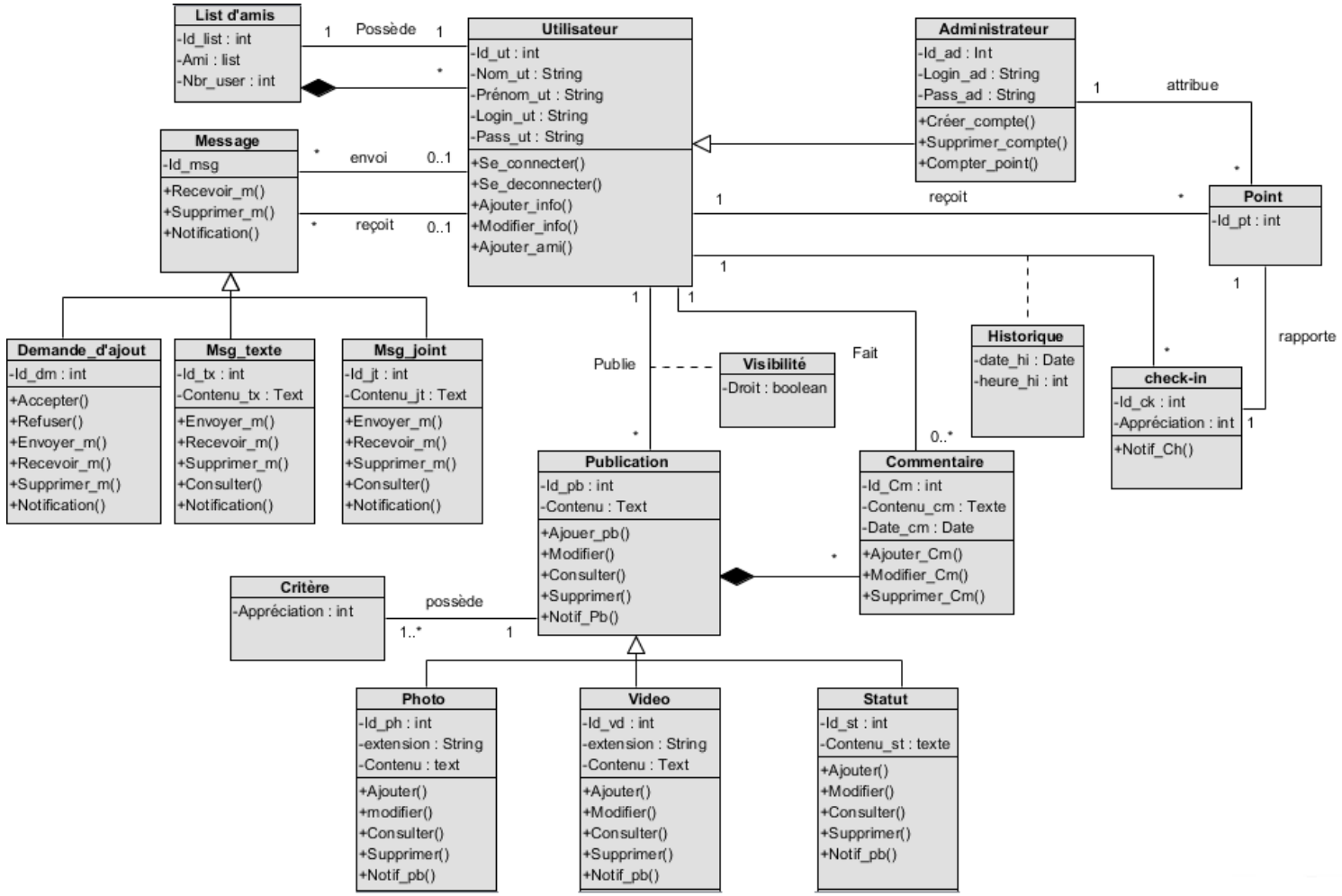
Critères de notation

- Notation déterminée sur la base des critères suivants :
  - Complétude du matériel livré,
  - Qualité du rapport projet (description de l'architecture, modélisation UML de l'application, description des mécanismes de gestion de la sécurité, de gestion transactionnelle, de gestion des erreurs, des principes d'ergonomie)
  - Clarté et fluidité de la présentation et de la démonstration de l'application lors de la soutenance,
  - Périmètre fonctionnel réalisé,
  - Exigences techniques réalisées,
  - Absence d'anomalies lors de l'exécution de l'application,
  - Organisation de l'équipe & plan projet,
  - Conformité à l'état de l'art de l'architecture applicative,
  - Conformité à l'état de l'art de la conception technique,
  - Qualité du modèle objet et de données,
  - Qualité graphique et ergonomique de l'IHM,
  - Qualité du code par rapport aux « best practices » de programmation java,
  - Volume du code,
  - Régularité et ponctualité des rendus (livrables projet et points projet),
  - Contribution individuelle au groupe et à la soutenance,
  - Réponses au questionnaire lors de la soutenance,
  - Présence aux cours,
  - Bonus.

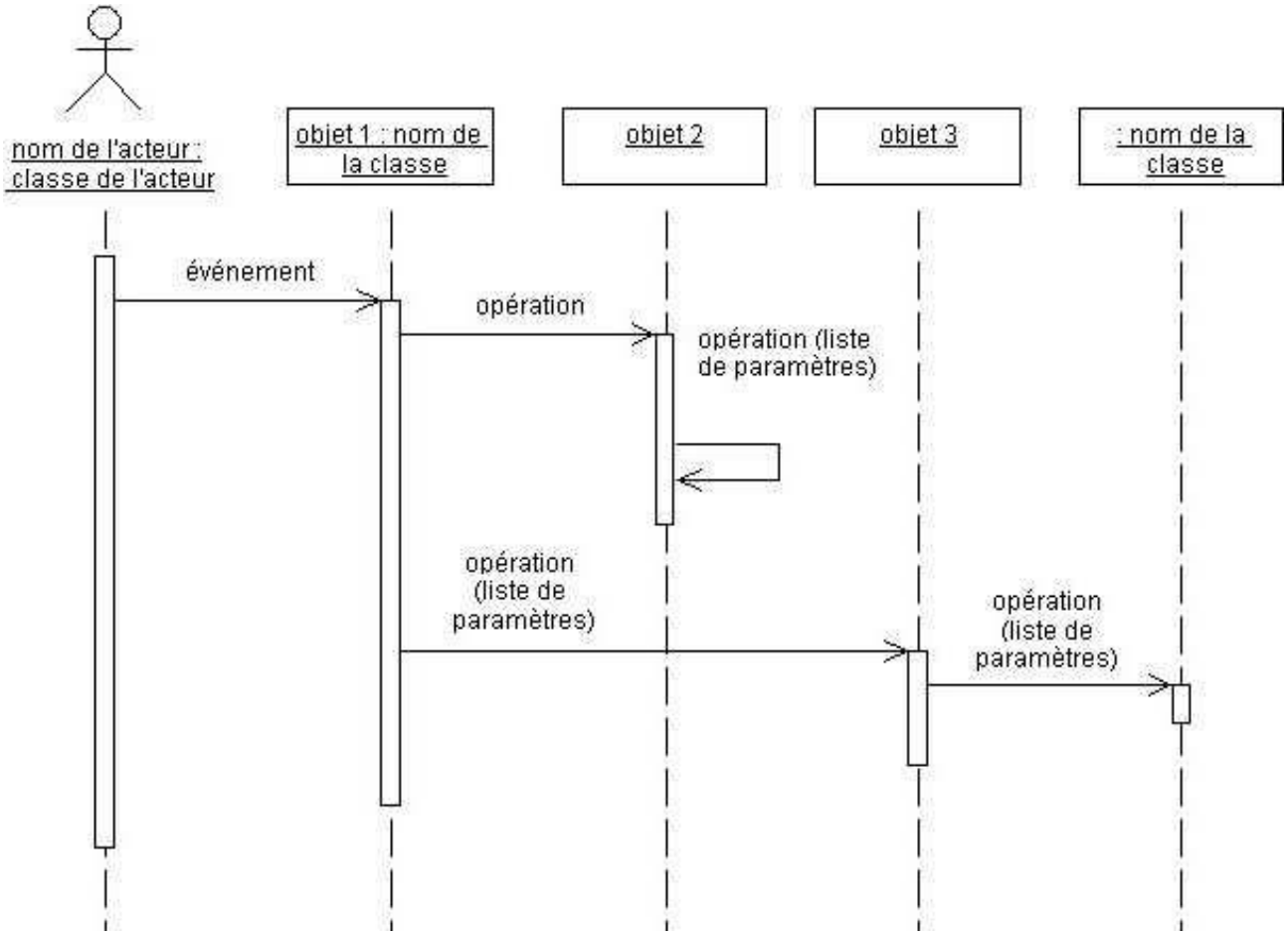
Annexe 1 : Exemple de diagramme des cas d'utilisation



Annexe 2 : Exemple de diagramme de classes

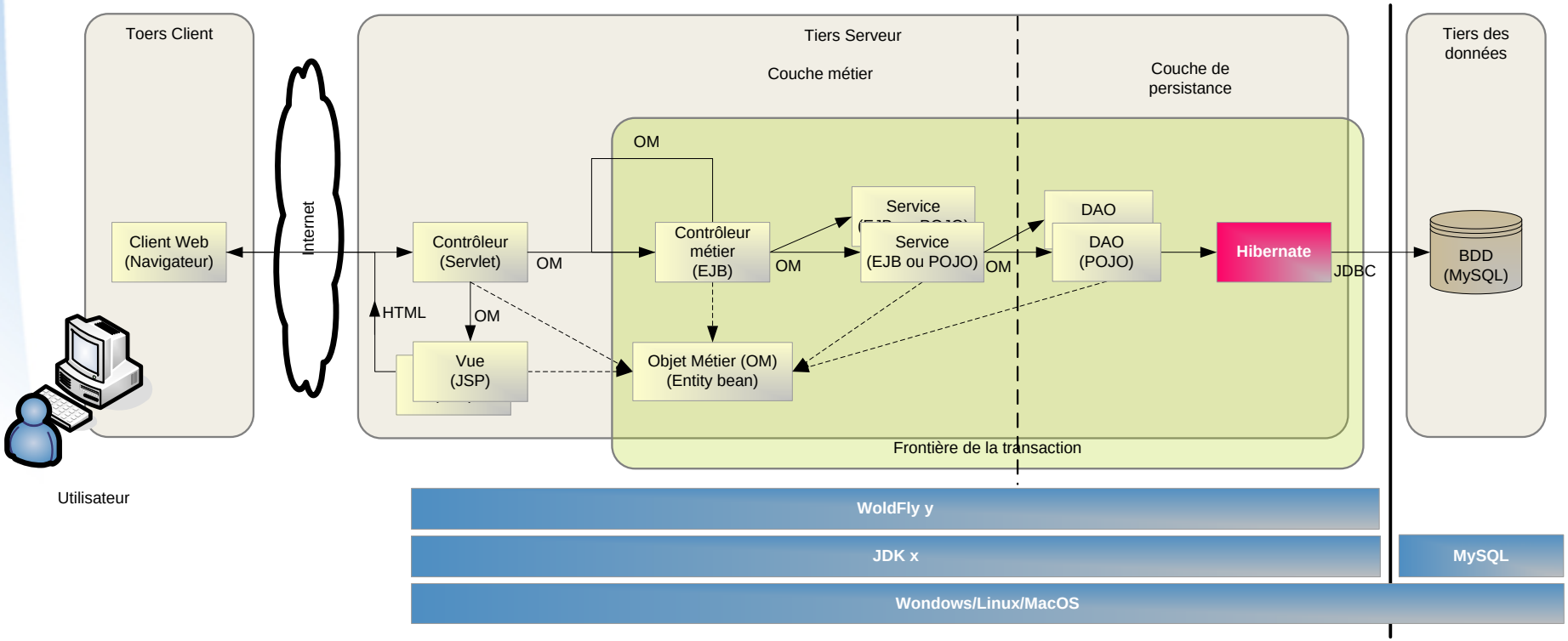


Annexe 3 : Exemple de diagramme d'intéraction



Annexe 4 : architecture applicative

Schéma d'architecture



Annexe 4 : architecture applicative

Implémentation d'un use case

- **L'implémentation d'un use case dans le cas de l'utilisation des JSP (à quelques détails près, s'applique aussi au cas des JFS) est supportée par:**
  - un *Contrôleur* dédié car, généralement, il n'est pas recommandé de réutiliser le contrôleur par plusieurs use case;
    - les contrôleurs sont implémentés par le biais des servlets,
    - les contrôleurs sont invoqués directement par les requêtes utilisateur.
  - une vue (rarement plusieurs vues), généralement dédiée, car il n'est pas recommandé de réutiliser les vues par plusieurs use case;
    - les vues sont implémentées par le biais des JSP ou JSF,
    - la vue est invoqués par un Contrôleur, donc, par une servlet, généralement, par la méthode *forward* de celle-ci
    - on pourrait utiliser plusieurs vues lorsque, par exemple, chaque vue comporte une validation intermédiaire; autrement dit, chaque vue remonte les informations saisies par l'utilisateurs dans la vue pour être stockées provisoirement par le serveur. La dernière vue comporte la validation finale qui rendra toutes les données remontées persistantes. Il est à noter que toutes les vues invoquent le même contrôleur de use case.
  - un ou plusieurs *Contrôleurs métier*. Les *Contrôleurs métier* peuvent être réutilisés par plusieurs use cases;
    - les *Contrôleurs métier* sont implémentés par le biais des *EJB session stateless*.
    - Les *Contrôleurs métiers* sont invoqués par les *Contrôleurs* (les servlets)

Annexe 4 : architecture applicative
Implémentation d'un use case

- un ou plusieurs objets *Service*. Les *Services* peuvent être réutilisés par plusieurs use case;
  - les *Services* sont implémentés par le biais des objets POJO (Plein Old Java Object) mais aussi par des EJB session en cas de besoin de gestion transactionnel plus fin,
  - les *Services* sont invoqués par les *Contrôleurs métier*. Un *Contrôleur métier* invoque un ou plusieurs *Services*. Si les *Services* sont implémentés par le biais des POJO, tous les *Services* s'exécuteront dans la même transaction créés par le *Contrôleur métier*. Pour plus de détails, voir l'Annexe 5.
- un ou plusieurs *Objets Métier* (OM). Les *Objets Métier* représentent les objets les plus réutilisés dans une application;
  - les *Objets Métier* sont implémentées par le biais des objets *EJB Entity*,
  - généralement, on crée un *Objet Métier* par table.
- un ou plusieurs objets *DAO* (Data Access Object), à savoir, des objets qui son spécialisés dans l'accès à la base de données, que ce soit par le biais du JPA (le moyens d'accès à privilégier), des requêtes HQP (dans le cas des recherches avec des critères de recherche plus complexes), voire, par le biais des requêtes SQL natives (JPA/Hibernet permettent d'utiliser ce genre de requêtes également) ;
  - les *DAOs* sont implémentés par le biais des objets POJO,
  - généralement, on crée un *DAO* par *Objet Métier* et donc, ce qui revient à créer un *DAO* par table,
  - les *DAOs* sont invoqués par les Services.

Annexe 4 : architecture applicative
Rôles des objets

- **Rôles des objets :**
  - *Contrôleur (servlet)* :
    - Être le point d'entrée du use case – autrement dit, intercepter la requête utilisateur,
    - Récupérer les données (paramètres) envoyés dans la requête,
    - Faire une première vérification des paramètres de la requête. Généralement, il s'agit des vérifications dites « de surface », à savoir, des contrôles plutôt techniques tels que la validité des formats des dates, contrôle de présence des paramètres obligatoires etc. Donc, le *Contrôleur* ne fait pas de vérifications de règles de gestions fonctionnelles. Ceci est le rôle des objets *Service*.
    - Si c'est le cas, instancier des *Objets Métier* et les initialiser avec les données de la requête
    - Invoquer le ou les *Contrôleurs Métiers* appropriés pour traiter la requête utilisateur et lui passe en paramètre (si c'est la cas) les *Objets Métier* créés
    - Récupérer les données éventuellement retournées par les *Contrôleurs métier*, d'habitude sous forme *d'Objets Métier*
    - Transmettre à la vue les données retournées par les *Contrôleur métier*
    - Invoquer la vue (généralement par la méthode *forward*)

Annexe 4 : architecture applicative

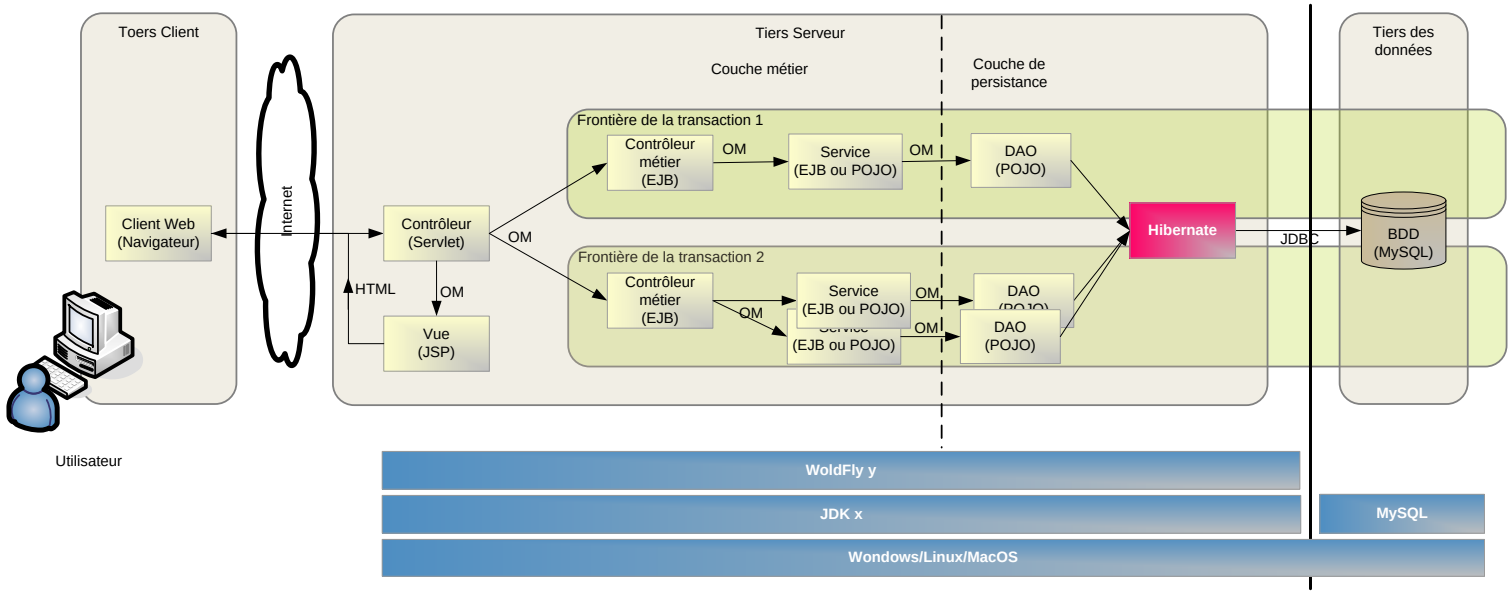
Rôles des objets

- *Contrôleur Métier* (EJB session) :
  - Créer une transaction, généralement une nouvelle car le *Contrôleur* (la servlet) qui l'l'invoque ne crée pas de transaction et donc, il n'a pas de transaction à lui transmettre. Dans cette transaction s'exécute le code de la méthode du *Contrôleur* métier (celle invoquée par le *Contrôleur*) ainsi que les méthodes des objets en aval qui seront invoqués depuis cette méthode
  - Invoque un ou plusieurs objets *Service* et leurs transmet les *Objets Métier* reçus de la part du *Contrôleur*. S'il invoque un seul objet Service, le Contrôleur métier ne sert, en quelque sorte, que de « passe plat » pour l'objet *Service*. En revanche, dans le cas où l'objet *Contrôleur métier* invoque plusieurs objets *Service*, son rôle est « d'orchestrer » ces objets, à savoir, de les invoquer dans le bon ordre et, si nécessaire, de passer des données retournées par un service vers les *Services* en aval.
  - Commiter ou rollbacker la transaction.
    - ***Remarque*** : Etant donné qu'il est demandé dans les exigences technique de gérer les transactions d'une manière CMT (Container Managed Transactions), le *Contrôleur Métier* ne commite/rolback pas la transaction explicitement car cela est fait par le conteneur WEB. Néanmoins, vous devez bien gérer les exceptions éventuellement levées dans la transaction car c'est en fonction de cela que le conteneur fera le commit ou le rollback de la transaction.
  - Retourner au *Contrôleur Métier* le résultat de l'exécution du (ou des) Service(s) – généralement un ou plusieurs Objets Métier.

Annexe 4 : architecture applicative

Rôles des objets

- *Contrôleur Métier* (EJB session) - suite :
  - Dans certains cas nous avons besoin d'exécuter plusieurs traitements dans le même use case et qui sont « transactionnellement » indépendants les uns des autres. Autrement dit, l'annulation d'un traitement en aval ne doit annuler les traitement effectués avec succès en amont du use case. Dans ce cas, l'objet *Contrôleur* va invoquer plusieurs *Contrôleurs Métier* et les orchestrer. Chaque *Contrôleur Métier* effectuera un traitement dans sa propre transaction. Il pourra ainsi commiter ou rollbacker sa transaction sans impacter les autres traitements.



Annexe 4 : architecture applicative
Rôles des objets

- *Service* (EJB session ou POJO) - suite :
  - C'est l'objet censé réaliser les traitements fonctionnels de un ou plusieurs du use cases (cet objet n'est pas dédié à un use case mais peut être réutilisé par plusieurs use cases)
  - Il doit d'abord vérifier toutes les données transmises par l'objet *Contrôleur Métier* qui la invoqué (généralement sous forme d'*Objets Métier*). Pour cela, il utilise des méthodes spécialement conçues pour la vérification des règles de gestion. En cas d'erreurs, il lèvera une exception et arrêtera le traitement.
  - Si le traitement concerne la base de données (récupérer des données depuis la bas ou modifier/insérer des données en base – voire les deux – l'objet Service invoquera des objets *DAO* qui réaliseront cette tâche. La raison en est que l'on veut garder les objets *Service* purement fonctionnels, à savoir, sans code technique d'accès en base etc.
  - Un objet *Service* est d'habitude créé en binôme avec *l'Objet Métier*. Autrement dit, pour chaque *Objet Métier* on crée un objet *Service* correspondant. Par exemple, pour *l'Objet Métier* ClientBO (BO comme Business Object), on créerait un objet *Service* ClientService. Cet objet *Service* s'occuperait de tous les traitements liés à l'*Objet Métier* Client.
  - Un objet *Service* pourrait être amené à invoquer plusieurs objets *DAO*

Annexe 4 : architecture applicative

Rôles des objets

- *DAO :*
  - Le rôle de cet objet est d'effectuer les traitement liés directement à la base de données. Etant donné l'exigence technique d'utiliser le JPA pour l'accès à la base de données, c'est dans cet objet que l'on doit coder l'accès à la base par JPA.
  - D'habitude, on crée un *DAO* par *Objet Métier*; donc pour l'objet ClientBO (pour reprendre l'exemple du plus haut) on créerait un *DAO* Client DAI qui serait en charge d'implémenter tous les accès en base liés à *l'Objet Métier* ClientBO
- *Objet Métier* (EJB Entity) :
  - C'est l'objet qui supporte les concepts du métier de l'application et également les données de la base. Généralement, on crée un *Objet Métier* par table. Par exemple, pour la table CLIENT, on créerait un *Objet Métier* ClientBO (ou vice versa, selon que l'on fait du « Bottom-up » ou du « Top-down » respectivement)