

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

**Document présentation**

<b>Description</b>	Examen de CARA
--------------------	----------------

**Document certification**

	<i>Name</i>	<i>Fonction</i>
<i>Author</i>	DJEBIEN Tarik	Etudiant Miage IPI NT
<i>Decidor</i>	ROOS Jean-François	Enseignant CARA

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

## Synthese

### Clients légers / riches

#### Clients riches

Avantages :

Inconvénients :

#### Clients légers

Avantages :

Inconvénients :

### Services

### Technique 1

### Technique 2

#### Question 1

### Problème

#### Définition de l'architecture


#### Question 1

#### Question 2

#### Question 3.

Avantages :

Inconvénients :

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

## Synthese

### Clients légers / riches

#### Clients riches


##### *Avantages :*

- **Ergonomie** : étant exécutée sur le poste client le développeur a accès à l'ensemble des fonctionnalités de l'OS (exemple: le système de fichiers) ainsi que toutes les ressources et évènements matériels.
- **Performance** : pour le C++ par exemple, l'application étant compilée pour génération d'un code assembleur adapté à l'architecture processeur (x86 par exemple) les instructions sont optimisées. Ceci n'est plus vrai sur les technologies client lourd actuelles comme le Java Swing ou le C# en client lourd où le principe de machine virtuelle a fait son apparition.
- **Répartition** : en mode client serveur, le code d'affichage côté IHM et de gestion de l'interaction utilisateur est exécuté au niveau du poste client ce qui offre l'avantage de solliciter les ressources serveurs uniquement pour la restitution des données et/ou traitements centralisés.
- **Interconnecté** : ayant accès aux ressources matérielles les solutions de type client lourd permettent le pilotage et/ou la réception d'informations de périphériques matériels connectés au poste client. Ces périphériques peuvent être des produits grands publics comme une douchette à code barre ou bien un scanner mais également des périphériques conçu sur mesure comme une chaîne de production, un réseau ferroviaire ou bien un ensemble pyrotechnique. Ces besoins sont d'ailleurs toujours couverts par des solutions de type client lourd avec si besoin des systèmes d'exploitation temps réel (on parle alors selon les cas d'informatique industrielle et non d'informatique de gestion).
- **Connectivité** : même lorsqu'elle est implémentée en architecture client / serveur, une solution de type client lourd peut facilement disposer d'un mécanisme de base temporaire permettant de fonctionner en mode déconnecter. A l'inverse, ce mode de fonctionnement est très difficile à mettre en œuvre sur une architecture client léger.

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

### *Inconvénients :*

- **Déploiement et montée de version** : étant installée sur chacun des postes clients, le déploiement d'une solution de type client lourd est un projet en soi qui peut devenir très complexe lorsque l'on fait face à un parc informatique hétérogène. Par la suite, à chaque montée de version il est nécessaire de redéployer la nouvelle version sur chacun des postes.
- **Compatibilité** : étant compilé pour une architecture processeur et faisant appel à des fonctionnalités au niveau OS, une solution de type client lourd offre une compatibilité restreinte à une configuration matérielle et un système d'exploitation donné. Cette problématique a été résolue notamment avec Java et C# par la mise en place de la notion de machine virtuelle. Le code Java par exemple n'est pas compilé en assembleur mais dans un code intermédiaire (bytecode) compréhensible par la machine virtuelle Java installée sur le poste et qui va alors faire la traduction en code assembleur : ce principe permet la compatibilité du code à tout système d'exploitation et processeur permettant l'installation d'une machine virtuelle Java. Cependant ce nouveau mode de fonctionnement limite les performances des applications compilées en bytecode de part la mise en place d'une couche intermédiaire, de plus il devient très difficile de dialoguer avec des périphériques extérieurs car la machine virtuelle Java n'a qu'un nombre très limité d'interactions possible avec l'OS et ses périphériques.
- **Limitation du choix technologique sur une architecture répartie client/serveur** : le client riche nécessite un choix des technologies sous-jacentes du projet restreint. En effet, si on prend l'exemple des EJB en JEE, on peut avoir des difficultés concernant le RPC des Services Remote si le client riche n'est pas déployé sur la même JVM que le Serveur (Conflits possibles sur les recherches des objets distribués dans l'annuaire), on doit alors passer par un administrateur réseaux par exemple pour la configuration des ports pour accéder aux ressources JNDI pour chaque site de déploiement. Une alternative reste la solution Java WebStart qui propose au client sur un navigateur web de télécharger le client riche Swing par exemple.

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

## Clients légers

### *Avantages :*

Ainsi pour faciliter le déploiement et la maintenance des applications de gestion, la majorité des entreprises se sont tournée vers les architectures client léger. Un client léger est une application dont les écrans sont calculés côté serveur puis transmis au poste client à travers un navigateur Web. Ce mode de fonctionnement permet de résoudre les inconvénients vus précédemment sur les clients lourds. En effet, le passage par un navigateur Web **évite la phase de déploiement de l'application sur l'ensemble des postes clients**. Ainsi lors des montées de version seul l'applicatif côté serveur a besoin d'être mise à jour. Quant à la compatibilité, une application en client léger est **compatible avec tous les OS** et processeurs permettant l'exécution d'un navigateur Web supportant l'application.

### *Inconvénients :*

Ces nouvelles architectures applicatives (PHP, Java JEE et Microsoft ASP et ASP .NET) ont également des inconvénients majeurs :

- **Adhérence au navigateur Web** : bien qu'étant basé sur des standards comme l'HTML, le CSS ou bien le JavaScript le code envoyé au navigateur Web peut entraîner des différences de fonctionnement selon le navigateur (IE, Chrome, Mozilla...). Il est alors nécessaire pour les développeurs de s'assurer de la compatibilité de leur application avec les versions majeures des navigateurs disponibles sur le marché.
- **Limites ergonomiques** : pour des raisons de sécurité, les technologies Web n'ont qu'un accès très limité aux ressources système du poste de travail.
- **Performances** : le serveur se chargeant de calculer l'ensemble des pages pour tous les utilisateurs, les performances de l'application peuvent être très dégradé lorsqu'un nombre important d'utilisateurs

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

sont connectés. Nous recommandons pour ces architectures des tests de performances systématiques.

## Services

### Atouts des environnements orientés services

Dans un modèle «économique» d'entreprise, toute branche fonctionnelle de l'entreprise est assimilable à un service. Les environnements orientés services dispose de nombreux atouts issue de l'architecture SOA qu'elle implémente :

- **Couplage** lâche de façon à minimiser les interdépendances, dépenses
- **Contrat** explicite entre les services qui interagissent, communication ne se font que via un accord entre services
- **Autonomie** : Le service contrôle complètement la logique qu'il encapsule
- **Abstraction** : au-delà de ce qui est visible dans le contrat, le reste est caché, on masque la logique du service au monde extérieur
- **Réutilisation** : la logique encapsulée est elle-même organisée en services de façon a promouvoir la réutilisation
- **Composabilité** : la capacité à regrouper plusieurs services et les coordonner de façon a former un nouveau service 'composite '
- **Absence de contexte** : interdiction de conserver des informations spécifiques à une activité.
- **Auto-description** : organisation du service de façon a ce que ses capacités puissent être découvertes avec les outils de recherche disponibles
- **Optimisation** : services peuvent être optimisés individuellement
- **Encapsulation existant** : encapsulé pour pouvoir être réutilisé avec SOA

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

## **Avantages et inconvénients d'un environnement à composants type EJB aux Web Services**

### **Avantages**


- Les EJB, grâce à leur containers, fournissent des services de gestion des transactions (JTA), persistance (JPA), de gestion des pools de connexion, des datasources, de l'annuaire de nommage des ressources (JNDI), des messages orientés middleware (JMS) et de sécurité (JASS) alors que les web services ne gèrent pas la sécurité (sécurité basée sur la sécurité du protocole sous-jacent (ex. HTTPS)) , le contexte Transactionnelle.

### **Inconvénients**

- Les EJBs sont construit au-dessus de RMI, RMI et EJB sont dépendants du langage JAVA : Si vos clients doivent être écrits dans quelque chose d'autre (par exemple.NET, PHP, etc) mieux vaut utiliser les services Web que les composants de type EJB. En effet, ils sont plus intéropérable car ils utilisent un protocole standard (HTTP) couplé avec un langage de description de donnée standardisé par le W3C (XML), tout cela basé sur SOAP.

Les EJBs n'ont pas d'intéropérabilité, comparé au web services.

De plus, les EJBS nécessitent la configuration du réseau (numéro de port , proxy...)

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

## Technique 1

Je choisi d'utiliser la technologie des EJB JEE.

EJB Entite Notification.java

@Entity

public class Notification implements Serializable {

    @Id

    @GeneratedValue(strategy=GenerationType.AUTO)

    @Column(name="NOTIFICATION\_ID")

    private Long id;

    @Temporal(TemporalType.DATE)

    private Date date;

    private String resultat ;

    private String detail ;

    @ManyToMany(mappedBy="notifications")

    protected Set<Utilisateur >utilisateurs ;


    ...

Constructeurs, getters, setters, equals, hashCode

    ...

}



 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

### EJB Entité Utilisateur .java

@Entity

public class Utilisateur implements Serializable {

    @Id

    @GeneratedValue(strategy=GenerationType.AUTO)

    @Column(name="UTILISATEUR\_ID")

    private Long id;

    private String nom ;

    private String password ;

    @ManyToMany

    @JoinTable(name="UTILISATEURS\_NOTIFICATIONS",

        joinColumns=

            @JoinColumn(

                name="UN\_UTILISATEUR\_ID",

                referencedColumnName ="UTILISATEUR\_ID"),

        inverseJoinColumns=

            @JoinColumn(

                name="UN\_NOTIFICATION\_ID",

                referencedColumnName ="NOTIFICATION\_ID"))


    protected Set<Notification > notifications;

    ...

    Constructeurs, getters, setters, equals, hashCode

    ...

}

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

Contrat de l'EJB Service Facade PushNotificationServiceRemote .java

@Remote

public interface PushNotificationServiceRemote {

/\*\*

\* Indique au bureau de vote que les notifications de résultats ont bien été lue

\* par l'utilisateur lors du clic bouton par l'électeur sur le client léger

\*/

void push(Utilisateur electeur, List<Notification> notificationsLues) ;


/\*\*

\* Connaître le detail explicite de la notification de vote choisie pour un usager

\*/

String getNotificationDetail(Notification notificationChoisie) ;

}

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

Implementation de l'EJB Service Facade PushNotificationServiceRemote .java

```


@Stateful(mappedName="PushNotificationService")
public class PushNotificationServiceImpl implements PushNotificationServiceRemote {

    @PersistenceContext(unitName = "Cara-examPU")
    protected EntityManager entityManager;

    @Override
    public void push(Utilisateur electeur, List<Notification> notificationsLues) {
        electeur.getNotifications().removeAll(notificationsLues);
        entityManager.merge(electeur);
    }

    @Override
    public String getNotificationDetail(Notification notificationChoisie) {
        return notificationChoisie.getDetail();
    }
}

```

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

### **Producteur de notification cote serveur**

- On a ici un producteur de notification via JMS, couplé à un EJB timer service de déclenchements de notification périodiques, qui se lance au démarrage de l'application sur le serveur JEE (Glassfish par exemple).

**@Singleton**

**@Startup**

**@LocalBean**

public class Producer {

// l'id de la fabrique de connection

@Resource(mappedName="jms/TopicConnectionFactory")

private TopicConnectionFactory connectionFactory;

// l'id du topic d'ecoute des electeurs

@Resource(mappedName = "jms/Topic")

private Topic destination;

@Resource

private TimerService ts;

@PostConstruct

public void configureTimer( Notification n ) {

ts.createTimer(1000,25000,n);

}

@Timeout

public void monitorerNotification( Timer timer ) {

sendNotificationToUsers((Notification)timer.getInfo());


}

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

```
// méthode qui envoie les notifications de cloture de vote aux electeurs
public void sendNotificationToUsers(Notification voteTerminee){
try {
    Connection connection = connectionFactory.createConnection();
    Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
    MessageProducer producer = session.createProducer(destination);
    TextMessage message = session.createTextMessage();
    message.setText(
        "The vote "+voteTerminee.getId()+" is ending with the score :
        "+voteTerminee.getResultat()
    );
    producer.send(message);
}

/* Send a non-text control message indicating end of messages. */
producer.send(session.createMessage());
} catch (JMSEException e) {
    System.err.println("Exception occurred: " + e.toString());
} finally {
    if (connection != null) {
        try {
            if (session != null) {
                session.close();
            }
            connection.close();
        } catch (JMSEException e) {}
    }
}
}
```

 <b>Université Lille1</b> <small>Sciences et Technologies</small>	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

 <b>Université Lille 1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

**On suppose que le client léger destiné à l'électeur s'exécute dans le même serveur d'application,(sinon faire un lookup dans l'annuaire JNDI)**

### Client consommateur


```
public class NotificationConsumer implements MessageListener {

    private List<Notification> messages ;

    @Resource(mappedName="jms/TopicConnectionFactory") //reference a la factory
    private TopicConnectionFactory connectionFactory;

    @Resource(mappedName = "jms/Topic")
    private Topic dest ;

    /**
     * Casts the message to a TextMessage and displays its text.
     * @param message the incoming message
     */
    public void onMessage(Message message) {
        TextMessage msg = null;
        try {
            if (message instanceof TextMessage) {
                msg = (TextMessage) message;
                Notification n = new Notification();
                n.setResultat (msg.getResultat());
                n.setDescription (msg.getText());
                messages.add(n);
            } else {
                System.err.println("Message is not a TextMessage");
            }
        } catch (JMSException e) {
            System.err.println("JMSException in onMessage(): " + e.toString());
        }
    }
}
```

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

```

    } catch (Throwable t) {
        System.err.println("Exception in onMessage(): " + t.getMessage());
    }
}

public static void main(String[] args) {
    messages = new ArrayList<Notification>();
    String destType = null;
    Connection connection = null;
    Session session = null;
    MessageConsumer consumer = null;
    TextMessage message = null;

    try {
        connection = connectionFactory.createConnection();
        session = connection.createSession(false,
            Session.AUTO_ACKNOWLEDGE);
        consumer = session.createConsumer(dest);
        consumer.setMessageListener(this);
        connection.start();
    } catch (JMSEException e) {
        System.err.println("Exception occurred: " + e.toString());
    } finally {
        if (connection != null) {
            try { connection.close(); } catch (JMSEException e) {}
        }
    }
}

```

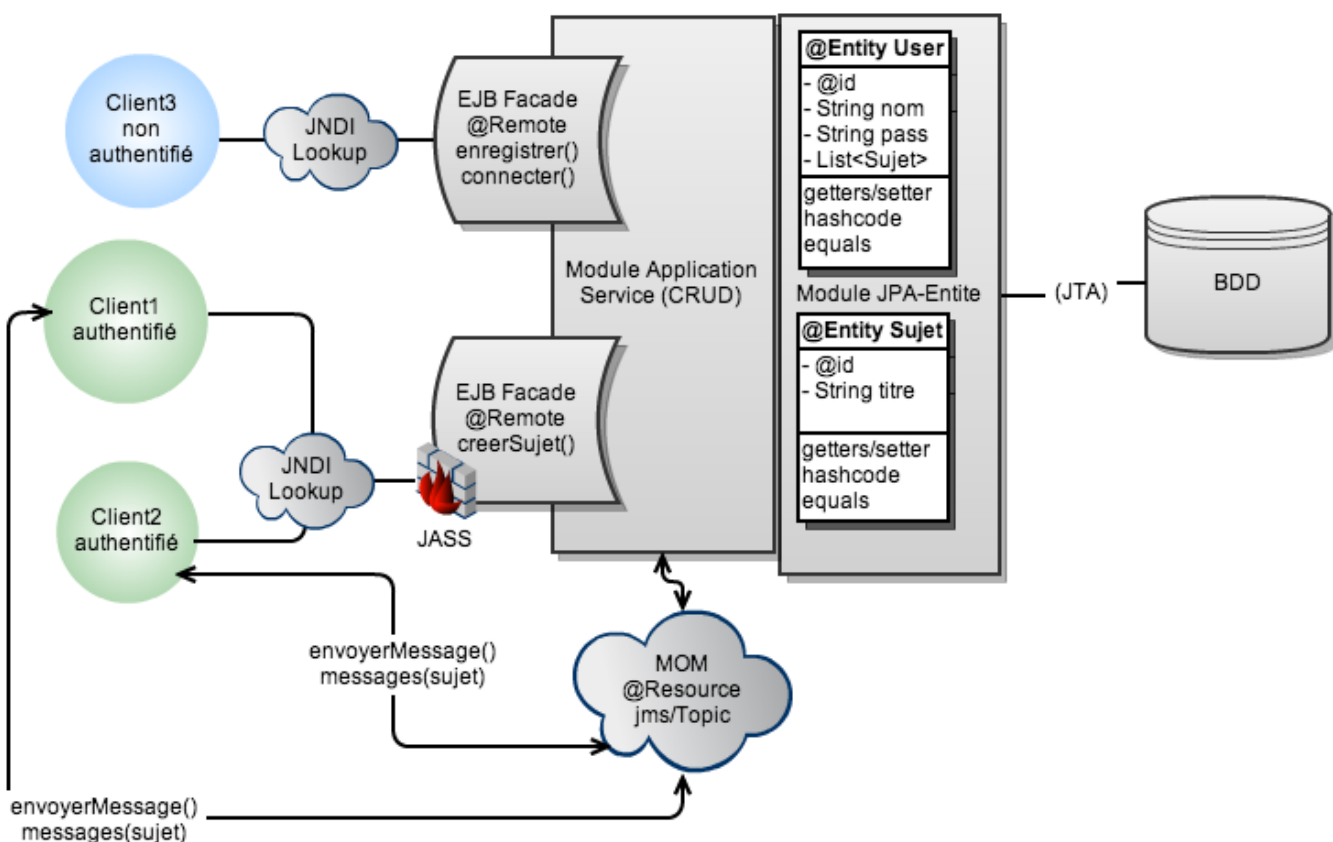


 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

## Technique 2

### Question 1

Je choisis la technologie JEE (**EJB** et **JMS**) sur une architecture **Client / Serveur 3-tiers**.



 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

Voici un descriptif pour l'architecture de l'application.


- Un **Client** est caractérisé par un membre du forum qui a effectué une inscription via la remote méthode **enregistrer(nom\_client, mot\_de\_passe)** .

Il est à la fois un **consommateur** des messages JMS (**messages(sujet)**) diffusé sur le **topic JMS** (= **sujet** en paramètre) après s'être identifié au préalable sur le système (via la remote méthode **se\_connecter(nom\_client, mot\_de\_passe)**).

Il est également un **producteur** de messages JMS sur le topic sur lequel il est connecté (via la méthode **envoyer\_message(sujet, message)** ).

Pour finir, il peut également créer un nouveau sujet sur le forum, donc un nouveau Topic JMS dynamiquement sur le serveur via la remote méthode **creer\_sujet(sujet)**.
- Le **Forum** est caractérisé par notre Serveur d'application JEE, il est composé de sujets multiples disponibles pour chaque clients, un sujet est caractérisé par un identifiant et un titre (**EJB entité Sujet.java**).

Il stocke (via **Java Persistence API**) également l'ensemble des utilisateurs inscrits sur le forum pour contrôler leur identification sur une base de donnée.. Un utilisateur est caractérisé par un identifiant, un nom, un mot de passe, et une liste de Sujet (**EJB entité Utilisateur.java**). Chaque EJB entité dispose d'un EJB Application Service doté de l'Entity Manager, qui fournit les opérations CRUD nécessaire à la persistance des utilisateurs et des sujets du forum.
- Toute les Remotes méthodes sont exposées via un Service EJB façade (design pattern application service) pour ne proposer aux clients uniquement les opérations autorisées (on utilisera le service de gestion de sécurité JAAS proposé dans le container d'EJB pour gérer les droits d'authentification des utilisateurs sur les appels de méthode distante qui nécessite d'être authentifiée , comme la création de sujet par exemple)

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

## Question 2

@Entity

public class Utilisateur implements Serializable {

    @Id

    @GeneratedValue(strategy=GenerationType.AUTO)

    @Column(name="UTILISATEUR\_ID")

    private Long id;

    private String nom ;

    private String password ;

    @ManyToMany

    @JoinTable(name="UTILISATEURS\_SUJETS",

        joinColumns=

            @JoinColumn(

                name="US\_UTILISATEUR\_ID",

                referencedColumnName ="UTILISATEUR\_ID"),

        inverseJoinColumns=

            @JoinColumn(

                name="US\_SUJET\_ID",

                referencedColumnName ="SUJET\_ID"))


    protected Set<Sujet > sujetsAbonnes;

    ...

Constructeurs, getters, setters, equals, hashCode

    ...

}

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

@Entity

public class Sujet implements Serializable {

    @Id

    @GeneratedValue(strategy=GenerationType.AUTO)

    @Column(name="SUJET\_ID")

    private Long id;

    private String titre;

    @ManyToMany(mappedBy="sujetsAbonnes")


    protected Set<Utilisateur > utilisateursInscrits ;

    ...

    Constructeurs, getters, setters, equals, hashCode

    ...

}

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

```

/**
 * @author tarik
 */
@Remote
public interface IUtilisateurDAO {


    /**
     * Find an entity by its unique id as primary key
     * @param id the primary key
     * @return the entity
     */
    Utilisateur findById(final String idNom);

    /**
     * Load all entities
     * @return the list of entities
     */
    List<Utilisateur> findAll();

    /**
     * Save an entity. This can be either a insert or update
     * @param entity the entity to save
     * @return the saved entity
     */
    Utilisateur save(final Utilisateur entity);

    /**
     * Delete an entity from the database
     * @param entity the entity to delete
     */
    void delete(final Utilisateur entity);

```

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

}

/\*\*

\* Session Bean implementation class UtilisateurDAO

\*/

@Stateful

public class UtilisateurDAO implements IUtilisateurDAO {

    @PersistenceContext

    protected EntityManager entityManager;

    @Override

    public Utilisateur findById(String idNom) {

        final Utilisateur result = entityManager.find(Utilisateur.class, idNom);

        return result;

    }

    @Override

    public void delete(Utilisateur entity) {

        Utilisateur attached = entityManager.merge(entity);

        entityManager.remove(attached);

    }

    @Override

    public Utilisateur save(Utilisateur entity) {


        final Utilisateur savedUtilisateurEntity = entityManager.merge(entity);

        return savedUtilisateurEntity;

    }

    @Override

    @SuppressWarnings("unchecked")

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0


```

        public List<Utilisateur> findAll() {
            return entityManager.createQuery("select u from Utilisateur u").getResultList();
        }
    }
    /**
     * @author tarik
     */
    @Remote
    public interface IUtilisateurServiceRemote {

        /**
         * Un utilisateur peut s'inscrire auprès du forum.
         * Pour cela, il doit fournir son nom, ainsi qu'un mot de passe.
         * Le forum doit vérifier qu'il n'existe pas deux utilisateurs ayant le même nom.
         *
         * @param nom
         * @param password
         * @return
         * @throws UserAlreadyExistException
         */
        Utilisateur enregistrer(String nom, String password)
        throws UserAlreadyExistException;

        /**
         * Un utilisateur doit s'identifier auprès de la salle des ventes
         * pour pouvoir envoyer ou lire des messages.
         * Pour cela, l'application de l'utilisateur envoie son nom et son mot de passe
         * qui sont alors vérifiés par le forum. Si le nom et/ou le mot de passe sont incorrects,
         * une erreur est retournée à l'application cliente sinon une référence sur l'objet
         * représentant l'utilisateur au sein du serveur de forum de messagerie est retournée.
         * Cet objet permet alors à l'utilisateur de proposer des sujets au forum ou de lire des
         * messages sur les sujet désiré.
         *
         * @param nom

```

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

```

    * @param password
    * @return
    * @throws UserBadLoginException
    */
    Utilisateur se_connecter(String nom, String password) throws UserBadLoginException;}

/**
 * @author tarik
 * Session Bean implementation class IUtilisateurServiceRemote
 */
@Stateless(mappedName="UtilisateurService")
public class IUtilisateurService implements IUtilisateurServiceRemote {

    @EJB
    private IUtilisateurDAO utilisateurDAO;

    @Override
    public Utilisateur enregistrer(String nom, String password)
        throws UserAlreadyExistException {

        if(utilisateurDAO.findById(nom) != null){
            throw new UserAlreadyExistException();
        }


        Utilisateur user = new Utilisateur();
        user.setNom(nom);
        user.setMotDePasse(password);
        return utilisateurDAO.save(user);
    }

    @Override
    public Utilisateur se_connecter(String nom, String password)
        throws UserBadLoginException {

        Utilisateur utilisateur = utilisateurDAO.findById(nom);

```



 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0


```

        if( utilisateur == null || !utilisateur.getMotDePasse().equals(password)){
            throw new UserBadLoginException(nom, password);
        }
        return utilisateur;
    }}
@Remote
public interface ISujetServiceRemote {

    /**
     * Un utilisateur inscrit auprès du forum peut créer un sujet.
     * Pour cela, il doit fournir son nom, son mot de passe ainsi que le titre du sujet
     * qu'il souhaite créer.
     * Le forum doit vérifier qu'il n'existe pas deux sujets ayant le même titre.
     *
     * @param nom
     * @param titre
     * @param password
     * @return
     * @throws SujetAlreadyExistException
     */
    Sujet creer_sujet(String nom, String password, String titre)
    throws SujetAlreadyExistException, UtilisateurNotGrantedException;

}

```

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

```

/*
 * En supposant que l'entity Sujet dispose aussi de son Dao CRUD
 */
@Stateless(mappedName="SujetService")
public class ISujetService implements ISujetServiceRemote {

    @EJB
    private ISujetDAO sujetDAO;

    @EJB
    private IUtilisateurDAO utilisateurDAO;

    @EJB (name = "UtilisateurService")
    private IUtilisateurServiceRemote utilisateurService;


    @Override
    public Sujet creer_sujet(String nom, String password, String titre)
        throws SujetAlreadyExistException, UtilisateurNotGrantedException {

        Utilisateur user = utilisateurService.se_connecter(nom,password);
        if( user == null){
            throw new UtilisateurNotGrantedException();
        }

        if(sujetDAO.findById(titre) != null){
            throw new SujetAlreadyExistException();
        }

        Sujet sujet = new Sujet();
        sujet.setTitre(nom);

```

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

```

        user.getSujetAbonne().add(sujet);
        utilisateurDAO.save(user);
        return sujetDAO.save(sujet);
    }

```

```

public class MessageForumClient implements MessageListener {

```

```

    private List<Sujet> sujets ;

```

```

    @Resource(mappedName="jms/TopicConnectionFactory") //reference a la factory
    private TopicConnectionFactory connectionFactory;

```

```

    @Resource(mappedName = "jms/Topic")
    private Topic dest ;

```

```

    /**

```

```

    * Casts the message to a TextMessage and displays its text.

```

```

    * @param message the incoming message

```

```

    */

```

```

    public void onMessage(Message message) {
        .... on recupere les message diffusé sur le Topic en les filtrant
        via Message Selectors :

```

L'exemple suivant sélectionne tout message qui a la propriété sujet à 'Sport' ou 'Opinion' :

```

        sujet = 'Sports' OR sujet = 'Opinion'

```

```

        ....

```

```

    }

```

```

    public void envoyer_message(String sujet, Message message){
        // on ajoute la date d'envoi dans l'entete avec currentMillisTime() de
        java.util.Date()
        // on teste avec un if si message.getText().length <= 1000

```

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

```
// Si c'est OK, on envoi sur la dest le message avec le sujet = "nomDuSujet" en propriété
du message
}
```

## Problème

### Définition de l'architecture

#### Question 1

- WebApp
- JEE Server / N clients (donc Topic)
- protocole SOAP (HTTP simple, sans état, /XML)
- On utilise les Webservices écrire/lire dans le document partagé commun car chaque client est le seule à modifier sa partie, pas de contexte transactionnelle nécessaire.

A la fin de chaque appel WS d'écriture, on diffuse un message JMS de notification de mise à jour à l'ensemble des clients qui sont à l'écoute sur le topic.

- Chaque document partagé est relié sur un topic au sein du groupe d'utilisateurs.

Chaque clients souscrit au Topic et recoit une notification en mode push lors d'une mise a jour du document. Il peut alors appeler explicitement demandé en mode pull avec le WS la methode de mise a jour du document pour la partie modifié.

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

## Question 2

- WebApp
- JEE Server / N clients à l'écoute de N messages
- Chaque message peut avoir plusieurs consommateurs. (donc Topic JMS)
- EJB déployé dans Glassfish disponible via JNDI dans JAR, client WAR web, EAR(jar+war)
- On utilise les EJB en Remote Statefull pour écrire avec un état correspondant à la partie du document dédié partagé commun. Chaque écriture dans le fichier est gérée par le container pour les accès concurrents (+ true en paramètre pour gérer les transactions JMS).
- A la fin de chaque écriture, on diffuse un message de notification de mise à jour à l'ensemble des clients qui sont à l'écoute du topic. On passe l'EJB en état pause dans son cycle de vie.
- Au début d'une écriture, on le passe en READY, ceci preserve les perfs de la JVM.
- On libère la session EJB avec @Remove lors de la déconnexion du client.
- Chaque document partagé est relié sur un topic au sein du groupe d'utilisateurs.
- On filtre les messages via les titres des documents avec une expression SQL92.
- Chaque client souscrit au Topic en participant à l'édition du document et reçoit une notification en mode push lors d'une mise à jour du document.

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

### Question 3.

#### Avantages :

- Q1 indépendance du langage sur client et serveur VS Q2 100% java

#### Inconvénients :

- on doit gérer les accès concurrents dans Q2 contrairement à Q1 car ils travaillent sur le même fichier donc risque de conflit
- Q1 moins performants que Q2

#### Avantages EJB VS WebServices

- monter en abstraction / objet
- architecture logicielle

 <b>Université Lille1</b> Sciences et Technologies	<b>Conception d'applications réparties avancées</b>	<b>Examen CARA</b>
Latest update : 25/03/2013	Copie étudiant	Current doc version : 1.0

séparation des préoccupations (SRP (Single Responsibility Principle))

- développer le métier indépendamment des préoccupations non fonctionnelles
- composants plus facilement réutilisable

inversion du contrôle

- container prenant en charge l'exécution du code métier (composant)
- container assure lien avec la partie technique
- configurer plutôt que programmer
- approche framework vs librairie

injection de dépendances

- vers d'autres composants, vers des services techniques
- retirer du code métier la gestion des liens vers les autres composants

métiers

- faire gérer l'architecture applicative par le container