

# TP ASP.NET

Le texte qui suit fait référence aux documents suivants :

[ref1] : [http://tahe.ftp-developpez.com/fichiers-archiv/javaee.pdf]

[ref2] : http://tahe.ftp-developpez.com/fichiers-archiv/dotnet/exercices/simupaie-aspnet-csharp.pdf

Par facilité, ces références sont notées par la suite [ref1] et [ref2].

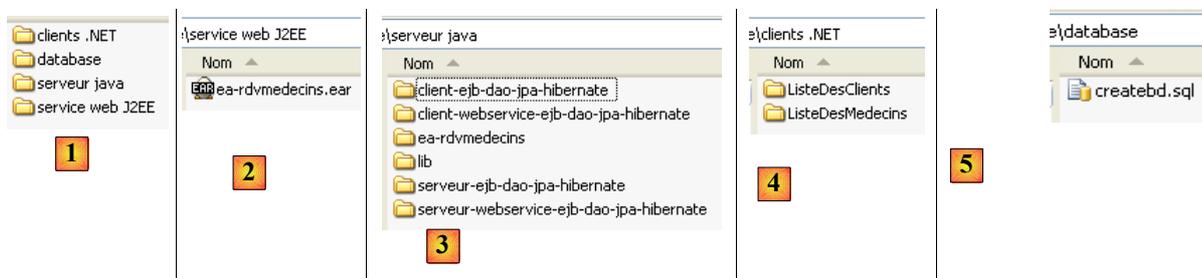
## 1 Objectif du TP

Le TP consiste en la réalisation d'une application client / serveur :

- le serveur est un service web J2EE s'exécutant sur le serveur Glassfish
- le client est une application ASP.NET / C# s'exécutant sur un serveur web .NET

Le service web J2EE est donné. Le TP consiste en la seule écriture du client ASP.NET décrit au paragraphe 6, page 41. Le texte qui précède la page 41 a une visée pédagogique. Il montre comment écrire et déployer le service web à l'aide de l'IDE Netbeans 6.5. Il montre également comment écrire des clients .NET pour ce service web.

Le texte du TP s'accompagne d'un fichier *zip* contenant les éléments suivants :



- [1] : le contenu du zip
- [2] : le service web J2EE
- [3] : un ensemble de projets Netbeans 6.5 visant à construire progressivement le service web J2EE
- [4] : des clients .NET du service J2EE - un client C# et un client ASP.NET / C#
- [5] : le script de création de la base de données utilisée par l'application

Le lecteur pressé et expérimenté pourra :

- lire le texte du TP
- créer la base de données MySQL (paragraphe 4.6, page 13).
- créer la ressource JDBC du serveur Glassfish pour la base de données précédente (paragraphe 4.9, page 18).
- déployer le service web J2EE (paragraphe 4.12, page 27).
- écrire le client ASP.NET du service web J2EE (paragraphe 6, page 41).

Pour les autres, il est conseillé de tout lire.

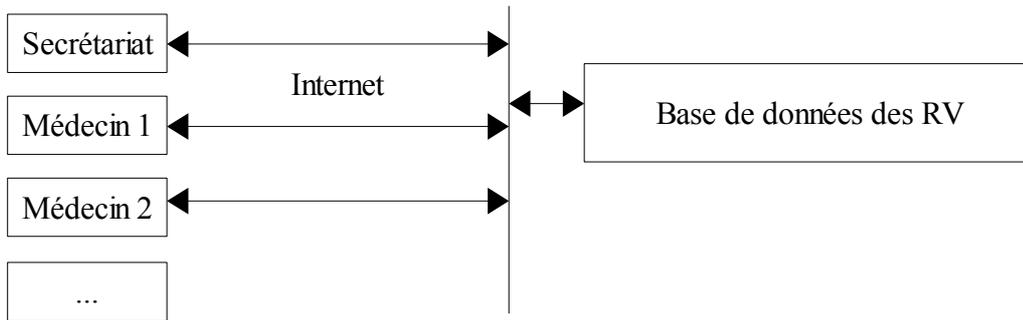
## 2 Le problème

Une société de services en informatique [ISTIA-IAIE] désire proposer un service de prise de rendez-vous. Le premier marché visé est celui des médecins travaillant seuls. Ceux-ci n'ont en général pas de secrétariat. Les clients désirant prendre rendez-vous téléphonent alors directement au médecin. Celui-ci est ainsi dérangé fréquemment au cours d'une journée ce qui diminue sa disponibilité à ses patients. La société [ISTIA-IAIE] souhaite leur proposer un service de prise de rendez-vous fonctionnant sur le principe suivant :

- un secrétariat assure les prises de RV pour un grand nombre de médecins. Ce secrétariat peut être réduit à une unique personne. Le salaire de celle-ci est mutualisé entre tous les médecins utilisant le service de RV.
- le secrétariat et tous les médecins sont reliés à Internet

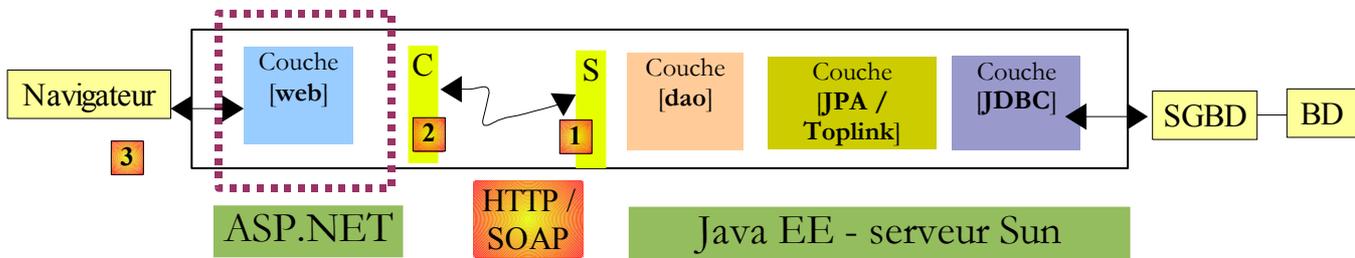
- les RV sont enregistrés dans une base de données centralisée, accessible par Internet, par le secrétariat et les médecins
- la prise de RV est normalement faite par le secrétariat. Elle peut être faite également par les médecins eux-mêmes. C'est le cas notamment lorsqu'à la fin d'une consultation, le médecin fixe lui-même un nouveau RV à son patient.

L'architecture du service de prise de RV est le suivant :



Les médecins gagnent en efficacité s'ils n'ont plus à gérer les RV. S'ils sont suffisamment nombreux, leur contribution aux frais de fonctionnement du secrétariat sera faible.

La société [ISTIA-IAIE] décide de confier à un stagiaire l'élaboration d'une première maquette du service, une application web basique. C'est cette maquette que nous nous proposons de réaliser ici. L'application aura l'architecture suivante :



- [1] : les couches [dao, jpa] permettant l'accès aux données sont réalisées à l'aide d'un service web J2EE
- [2] : une application web ASP.NET est cliente de ce service web. Elle propose à ses clients [3], un ensemble de pages permettant la prise de rendez-vous avec un médecin.

Dans cette architecture, il y a deux serveurs web :

- celui qui exécute le service web distant
- celui qui exécute le client ASP.NET du service web distant

### 3 Fonctionnement de l'application

Nous appellerons [RdvMedecins] l'application. Nous présentons ci-dessous des copies d'écran de son fonctionnement côté client. Comme il a été indiqué, l'interface client est une interface web réalisée avec ASP.NET. La page d'accueil de l'application est la suivante :

## Cabinet médical - LES MEDECINS ASSOCIES -

Prise de rendez-vous :

Médecin

Jour (JJ/MM/AAAA)

A partir de cette première page, l'utilisateur (Secrétariat, Médecin) va engager un certain nombre d'actions. Nous les présentons ci-dessous. La vue de gauche présente la vue à partir de laquelle l'utilisateur fait une **demande**, la vue de droite la **réponse** envoyée par le serveur.

### Cabinet médical - LES MEDECINS ASSOCIES -

Prise de rendez-vous :

Médecin

Jour (JJ/MM/AAAA)

L'utilisateur a sélectionné un médecin et a saisi un jour de RV

### Cabinet médical - LES MEDECINS ASSOCIES -

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action
8h0-8h20		<a href="#">Réserver</a>
8h20-8h40		<a href="#">Réserver</a>
8h40-9h0		<a href="#">Réserver</a>
9h0-9h20		<a href="#">Réserver</a>
9h20-9h40		<a href="#">Réserver</a>
9h40-10h0		<a href="#">Réserver</a>
10h0-10h20	Mr Jules JACQUARD	<a href="#">Supprimer</a>
10h20-10h40		<a href="#">Réserver</a>
10h40-11h0		<a href="#">Réserver</a>
11h0-11h20		<a href="#">Réserver</a>
11h20-11h40	Mme Christine GERMAN	<a href="#">Supprimer</a>

On obtient la liste (vue partielle ici) des RV du médecin sélectionné pour le jour indiqué.

## Cabinet médical - LES MEDECINS ASSOCIES -

Rendez-vous : 8h0-8h20, Jour : 2006:08:23, Médecin : Mme Marie PELISSIER

Client

On la renseigne

## Cabinet médical - LES MEDECINS ASSOCIES -

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action
8h0-8h20	Melle Brigitte BISTROU	<a href="#">Supprimer</a>
8h20-8h40		<a href="#">Réserver</a>
8h40-9h0		<a href="#">Réserver</a>

Le nouveau RV apparaît dans la liste

## Cabinet médical - LES MEDECINS AS

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action
8h0-8h20	Melle Brigitte BISTROU	<a href="#">Supprimer</a>
8h20-8h40		<a href="#">Réserver</a>

L'utilisateur peut supprimer un RV

## Cabinet médical - LES MEDECINS .

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action
8h0-8h20		<a href="#">Réserver</a>
8h20-8h40		<a href="#">Réserver</a>

Le RV supprimé a disparu de la liste des RV

## Cabinet médical - LES MEDECINS

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action
8h0-8h20		<a href="#">Réserver</a>
8h20-8h40		<a href="#">Réserver</a>

Une fois l'opération de prise de RV ou d'annulation de RV faite, l'utilisateur peut revenir à la page d'accueil

## Cabinet médical - LES MEDECINS ASSOCIES -

Prise de rendez-vous :

Médecin     
Jour (JJ/MM/AAAA)

Il la retrouve dans son dernier état

## Cabinet médical - LES MEDECINS .

Prise de rendez-vous :

Médecin

Jour (JJ/MM/AAAA)

On efface pour repartir sur une nouvelle opération

## Cabinet médical - LES MEDECINS ASSOCIES -

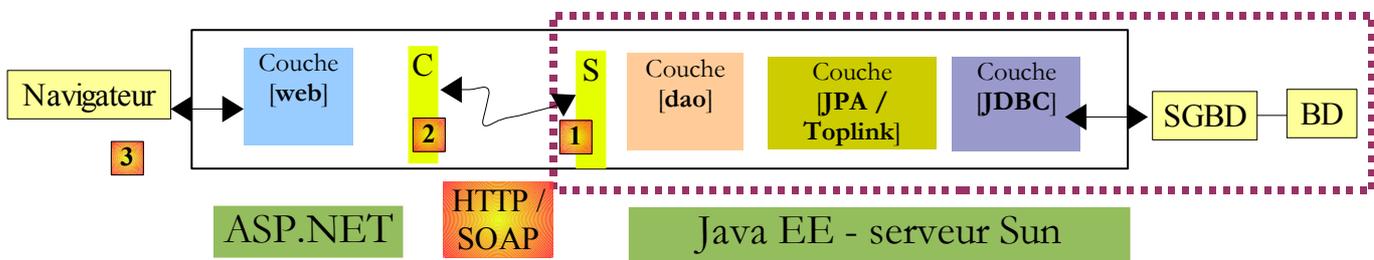
Prise de rendez-vous :

Médecin

Jour (JJ/MM/AAAA)

## 4 Le service web J2EE des rendez-vous

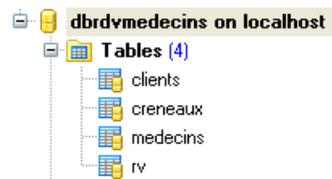
Revenons à l'architecture de l'application à construire :



Nous nous intéressons dans cette partie à la construction du service web J2EE [1] exécuté sur un serveur Sun / Glassfish.

### 4.1 La base de données

La base de données qu'on appellera [DBRDVMEDECINS] est une base de données MySQL5 avec quatre tables :



#### 4.1.1 La table [MEDECINS]

Elle contient des informations sur les médecins gérés par l'application [RdvMedecins].

Fields	Indices	Foreign Keys	Data	Description	DDL
Field Name	Field Type	Size	Precision	Not Null	
ID	BIGINT	20	0	✓	
VERSION	INTEGER	11	0	✓	
TITRE	VARCHAR	5	0	✓	
NOM	VARCHAR	30	0	✓	
PRENOM	VARCHAR	30	0	✓	

ID	VERSION	TITRE	NOM	PRENOM
1	1	Mme	PELISSIER	Marie
2	1	Mr	BROMARD	Jacques
3	1	Mr	JANDOT	Philippe
4	1	Melle	JACQUEMOT	Justine

- ID : n° identifiant le médecin - clé primaire de la table
- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- NOM : le nom du médecin
- PRENOM : son prénom
- TITRE : son titre (Melle, Mme, Mr)

### 4.1.2 La table [CLIENTS]

Les clients des différents médecins sont enregistrés dans la table [CLIENTS] :

Fields	Indices	Foreign Keys	Data	Description	DDL
Field Name	Field Type	Size	Precision	Not Null	
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	
VERSION	INTEGER	11	0	<input checked="" type="checkbox"/>	
TITRE	VARCHAR	5	0	<input checked="" type="checkbox"/>	
NOM	VARCHAR	30	0	<input checked="" type="checkbox"/>	
PRENOM	VARCHAR	30	0	<input checked="" type="checkbox"/>	

ID	VERSION	TITRE	NOM	PRENOM
1	1	Mr	MARTIN	Jules
2	1	Mme	GERMAN	Christine
3	1	Mr	JACQUARD	Jules
4	1	Melle	BISTROU	Brigitte

- ID : n° identifiant le client - clé primaire de la table
- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- NOM : le nom du client
- PRENOM : son prénom
- TITRE : son titre (Melle, Mme, Mr)

### 4.1.3 La table [CRENEAUX]

Elle liste les créneaux horaires où les RV sont possibles :

Fields	Indices	Foreign Keys	Data	Description	DDL	
Field Name	Field Type	Size	Precision	Not Null	Default	
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	Null	
VERSION	INTEGER	11	0	<input checked="" type="checkbox"/>		
HDEBUT	INTEGER	11	0	<input checked="" type="checkbox"/>		
MDEBUT	INTEGER	11	0	<input checked="" type="checkbox"/>		
HFIN	INTEGER	11	0	<input checked="" type="checkbox"/>		
MFIN	INTEGER	11	0	<input checked="" type="checkbox"/>		
ID_MEDECIN	BIGINT	20	0	<input checked="" type="checkbox"/>		

ID	VERSION	ID_MEDECIN	HDEBUT	MDEBUT	HFIN	MFIN
1	1	1	8	0	8	20
2	1	1	8	20	8	40
3	1	1	8	40	9	0
4	1	1	9	0	9	20
5	1	1	9	20	9	40
6	1	1	9	40	10	0
7	1	1	10	0	10	20
8	1	1	10	20	10	40
9	1	1	10	40	11	0
10	1	1	11	0	11	20
11	1	1	11	20	11	40
12	1	1	11	40	12	0
13	1	1	14	0	14	20
14	1	1	14	20	14	40
15	1	1	14	40	15	0

1

16	1	1	15	0	15	20
17	1	1	15	20	15	40
18	1	1	15	40	16	0
19	1	1	16	0	16	20
20	1	1	16	20	16	40
21	1	1	16	40	17	0
22	1	1	17	0	17	20
23	1	1	17	20	17	40
24	1	1	17	40	18	0
25	1	2	8	0	8	20
26	1	2	8	20	8	40
27	1	2	8	40	9	0
28	1	2	9	0	9	20
29	1	2	9	20	9	40
30	1	2	9	40	10	0
31	1	2	10	0	10	20

32	1	2	10	20	10	40
33	1	2	10	40	12	0
34	1	2	12	0	12	20
35	1	2	12	20	12	40
36	1	2	12	40	12	0
37	1	3	8	0	8	20
38	1	3	8	20	8	40
39	1	3	8	40	9	0
40	1	3	9	0	9	20
41	1	3	9	20	9	40
42	1	3	9	40	10	0
43	1	3	10	0	10	20
44	1	3	10	20	10	40
45	1	3	10	40	12	0
46	1	3	12	0	12	20

- ID : n° identifiant le créneau horaire - clé primaire de la table (ligne 8)
- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- ID\_MEDECIN : n° identifiant le médecin auquel appartient ce créneau – clé étrangère sur la colonne MEDECINS(ID).
- HDEBUT : heure début créneau

- MDEBUT : minutes début créneau
- HFIN : heure fin créneau
- MFIN : minutes fin créneau

La seconde ligne de la table [CRENEAUX] (cf [1] ci-dessus) indique, par exemple, que le créneau n° 2 commence à 8 h 20 et se termine à 8 h 40 et appartient au médecin n° 1 (Mme Marie PELISSIER).

#### 4.1.4 La table [RV]

Elle liste les RV pris pour chaque médecin :

Field Name	Field Type	Size	Precision	Not Null	Default
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	Null
JOUR	DATE	10	0	<input checked="" type="checkbox"/>	
ID_CLIENT	BIGINT	20	0	<input checked="" type="checkbox"/>	
ID_CRENEAU	BIGINT	20	0	<input checked="" type="checkbox"/>	

ID	JOUR	ID_CLIENT	ID_CRENEAU
1	22/08/2006	2	1
3	23/08/2006	4	20
4	10/09/2006	2	10
6	23/08/2006	3	7
9	23/08/2006	2	10

- ID : n° identifiant le RV de façon unique – clé primaire
- JOUR : jour du RV
- ID\_CRENEAU : créneau horaire du RV - clé étrangère sur le champ [ID] de la table [CRENEAUX] – fixe à la fois le créneau horaire et le médecin concerné.
- ID\_CLIENT : n° du client pour qui est faite la réservation – clé étrangère sur le champ [ID] de la table [CLIENTS]

Cette table a une contrainte d'unicité sur les valeurs des colonnes jointes (JOUR, ID\_CRENEAU) :

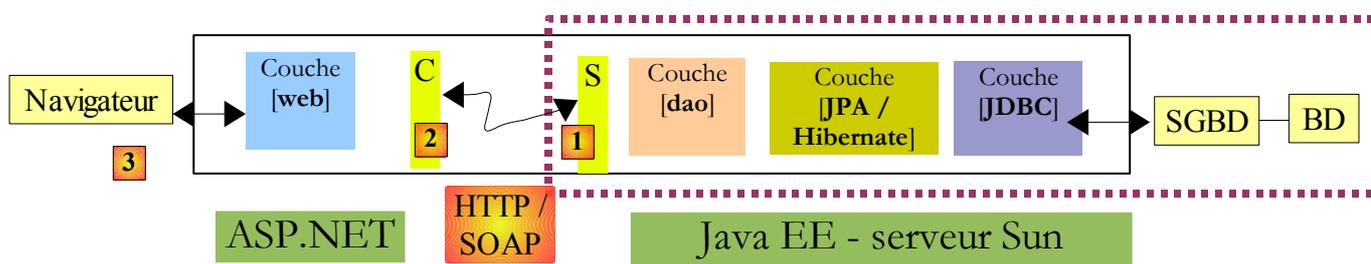
```
ALTER TABLE RV ADD CONSTRAINT UNQ1_RV UNIQUE (JOUR, ID_CRENEAU);
```

Si une ligne de la table [RV] a la valeur (JOUR1, ID\_CRENEAU1) pour les colonnes (JOUR, ID\_CRENEAU), cette valeur ne peut se retrouver nulle part ailleurs. Sinon, cela signifierait que deux RV ont été pris au même moment pour le même médecin. D'un point de vue programmation Java, le pilote JDBC de la base lance une *SQLException* lorsque ce cas se produit.

La ligne d'id égal à 3 (cf [1] ci-dessus) signifie qu'un RV a été pris pour le créneau n° 20 et le client n° 4 le 23/08/2006. La table [CRENEAUX] nous apprend que le créneau n° 20 correspond au créneau horaire 16 h 20 - 16 h 40 et appartient au médecin n° 1 (Mme Marie PELISSIER). La table [CLIENTS] nous apprend que le client n° 4 est Melle Brigitte BISTROU.

## 4.2 Les éléments de l'architecture côté serveur

Revenons à l'architecture de l'application à construire :



Côté serveur, l'application sera formée :

- d'une couche Jpa permettant de travailler avec la BD au moyen d'objets
- d'un Ejb chargé de gérer les opérations avec la couche Jpa
- d'un service web chargé d'exposer à des clients distants, l'interface de l'Ejb sous la forme d'un service web.

Les éléments (b) et (c) implémentent la couche [dao] représentée sur le schéma précédent. On sait qu'une application peut accéder à un Ejb distant via les protocoles RMI et JNDI. Dans la pratique, cela limite les clients à des clients Java. Un service web utilise un

protocole de communication standardisé que divers langages implémentent : .NET, Php, C++, ... C'est ce que nous voulons montrer ici en utilisant un client .NET.

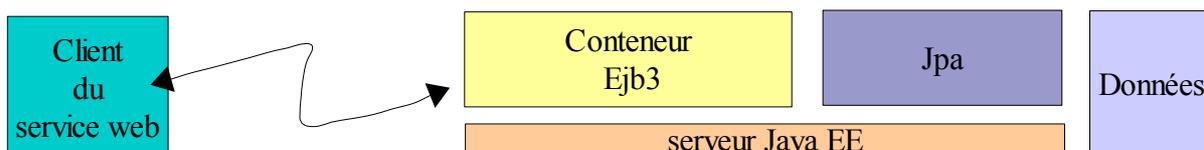
Pour une courte introduction aux services web, on pourra lire le cours [ref1], paragraphe 14, page 109.

Un service web peut être implémenté de deux façons :

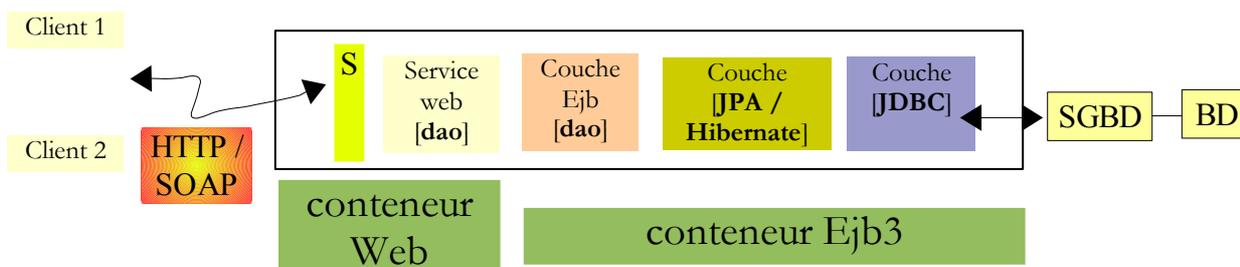
- par une classe annotée **@WebService** qui s'exécute dans un conteneur web



- par un Ejb annoté **@WebService** qui s'exécute dans un conteneur Ejb



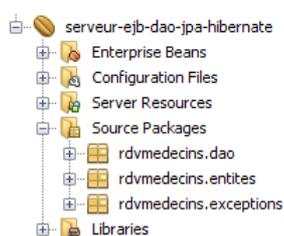
Nous allons utiliser ici la première solution :



Dans le cours [ref1], paragraphe 14, page 109, on trouvera un exemple utilisant la seconde solution.

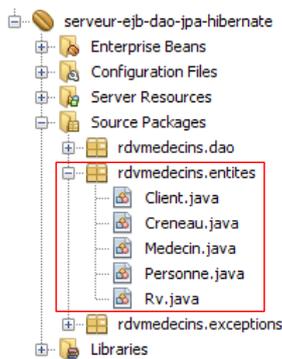
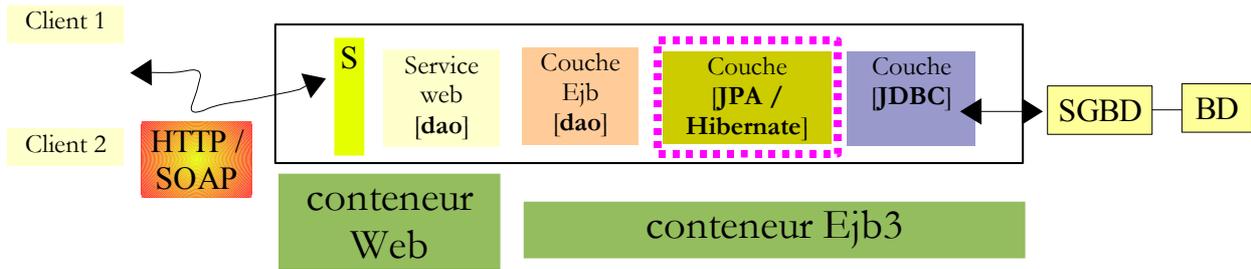
### 4.3 Le projet Netbeans du module Ejb

Le module Ejb de l'application est le suivant :



- le package [rdvmedecins.entites] regroupe les entités de la couche Jpa
- le package [rdvmedecins.dao] implémente l'Ejb de la couche [dao]
- le package [rdvmedecins.exceptions] implémente une classe d'exception spécifique à l'application

## 4.4 Les entités de la couche JPA



Le package [rdvmedecins.entites] implémente la couche [jpa] avec les classes et entités suivantes :

La classe **Personne** est utilisée pour représenter les médecins et les clients :

```

1. package rdvmedecins.entites;
2. ...
3. @MappedSuperclass
4. public class Personne implements Serializable {
5.     // caractéristiques d'une personne
6.
7.     @Id
8.     @GeneratedValue(strategy = GenerationType.AUTO)
9.     @Column(name = "ID")
10.    private Long id;
11.    @Version
12.    @Column(name = "VERSION", nullable = false)
13.    private Integer version;
14.
15.    @Column(name = "TITRE", length = 5, nullable = false)
16.    private String titre;
17.    @Column(name = "NOM", length = 30, nullable = false)
18.    private String nom;
19.    @Column(name = "PRENOM", length = 30, nullable = false)
20.    private String prenom;
21.
22.    // constructeur par défaut
23.    public Personne() {
24.    }
25.
26.    // constructeur avec paramètres
27.    public Personne(String titre, String nom, String prenom) {
28.        // on passe par les setters
29.    }
30.
31.
32.    // constructeur par copie
33.    public Personne(Personne personne) {
34.        // on passe par les setters

```

```

35. ...
36. }
37.
38. // toString
39. @Override
40. public String toString() {
41.     return "[" + titre + "," + prenom + "," + nom + "]";
42. }
43.
44. // getters et setters
45. ....
46. }

```

- ligne 3 : on notera que la classe [Personne] n'est pas elle-même une entité (@Entity). Elle va être la classe parent d'entités. L'annotation @MappedSuperClass désigne cette situation.

L'entité [Client] encapsule les lignes de la table [clients]. Elle dérive de la classe [Personne] précédente :

```

1. package rdvmedecins.entites;
2. ....
3. @Entity
4. @Table(name = "CLIENTS")
5. public class Client extends Personne implements Serializable {
6.
7.     // constructeur par défaut
8.     public Client() {
9.     }
10.
11.    // constructeur avec paramètres
12.    public Client(String titre, String nom, String prenom) {
13.        // parent
14.        super(titre, nom, prenom);
15.    }
16.
17.    // constructeur par copie
18.    public Client(Client client) {
19.        // parent
20.        super(client);
21.    }
22. }

```

- ligne 3 : la classe [Client] est une entité Jpa
- ligne 4 : elle est associée à la table [clients]
- ligne 5 : elle dérive de la classe [Personne]

L'entité [Medecin] qui encapsule les lignes de la table [medecins] suit le même modèle :

```

1. package rdvmedecins.entites;
2. ...
3. @Entity
4. @Table(name = "MEDECINS")
5. public class Medecin extends Personne implements Serializable {
6.
7.     // constructeur par défaut
8.     public Medecin() {
9.     }
10.
11.    // constructeur avec paramètres
12.    public Medecin(String titre, String nom, String prenom) {
13.        // parent
14.        super(titre, nom, prenom);
15.    }
16.
17.    // constructeur par copie
18.    public Medecin(Medecin medecin) {
19.        // parent
20.        super(medecin);
21.    }
22. }

```

L'entité [Creneau] encapsule les lignes de la table [creneaux] :

```

1. package rdvmedecins.entites;
2. ....

```

```

3. @Entity
4. @Table(name = "CRENEAUX")
5. public class Creneau implements Serializable {
6.
7.     // caractéristiques d'un créneau de RV
8.     @Id
9.     @GeneratedValue(strategy = GenerationType.AUTO)
10.    @Column(name = "ID")
11.    private Long id;
12.    @Version
13.    @Column(name = "VERSION", nullable = false)
14.    private Integer version;
15.    @ManyToOne
16.    @JoinColumn(name = "ID_MEDECIN", nullable = false)
17.    private Medecin medecin;
18.    @Column(name = "HDEBUT", nullable = false)
19.    private Integer hdebut;
20.    @Column(name = "MDEBUT", nullable = false)
21.    private Integer mdebut;
22.    @Column(name = "HFIN", nullable = false)
23.    private Integer hfin;
24.    @Column(name = "MFIN", nullable = false)
25.    private Integer mfin;
26.
27.    // constructeur par défaut
28.    public Creneau() {
29.
30.    }
31.
32.    // constructeur avec paramètres
33.    public Creneau(Medecin medecin, Integer hDebut,Integer mDebut, Integer hFin, Integer mFin) {
34.        // on passe par les setters
35.    ...
36.    }
37.
38.    // constructeur par copie
39.    public Creneau(Creneau creneau) {
40.        // on passe par les setters
41.    ...
42.    }
43.
44.    // toString
45.    @Override
46.    public String toString() {
47.        return "[" + getId() + "," + getVersion() + "," + getMedecin() + "," + getHdebut() + ":" +
48.            getMdebut() + "," + getHfin() + ":" + getMfin() + "j";
49.    }
50.    // setters - getters
51.    ...
52. }

```

L'entité [Rv] encapsule les lignes de la table [rv] :

```

1. package rdvmedecins.entites;
2. ...
3. @Entity
4. @Table(name = "RV")
5. public class Rv implements Serializable {
6.     // caractéristiques
7.
8.     @Id
9.     @GeneratedValue(strategy = GenerationType.AUTO)
10.    @Column(name = "ID")
11.    private Long id;
12.    @Column(name = "JOUR", nullable = false)
13.    @Temporal(TemporalType.DATE)
14.    private Date jour;
15.    @ManyToOne
16.    @JoinColumn(name = "ID_CLIENT", nullable = false)
17.    private Client client;
18.    @ManyToOne
19.    @JoinColumn(name = "ID_CRENEAU", nullable = false)
20.    private Creneau creneau;
21.
22.    // constructeur par défaut
23.    public Rv() {
24.    }

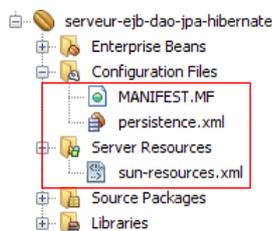
```

```

25.
26. // constructeur avec paramètres
27. public Rv(Date jour, Client client, Creneau creneau) {
28.     // on passe par les setters
29.     ...
30. }
31.
32. // constructeur par recopie
33. public Rv(Rv rv) {
34.     // on passe par les setters
35.     ...
36. }
37.
38. // toString
39. @Override
40. public String toString() {
41.     return "[" + getId() + "," + new SimpleDateFormat("dd/MM/yyyy").format(getJour()) + "," +
    getClient() + "," + getCreneau() + "];"
42. }
43.
44. // getters et setters
45. ...
46. }

```

## 4.5 Configuration de la couche JPA



La couche [JPA] est configurée par les fichiers [persistence.xml] et [sun-resources.xml] ci-dessus :

Le fichier [persistence.xml] est le suivant :

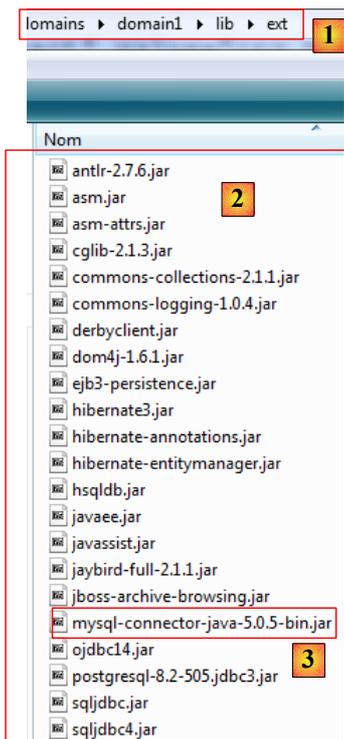
```

1. <?xml version="1.0" encoding="UTF-8"?>
2. <persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/
    ns/persistence http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
3.   <persistence-unit name="dbrdvmedecins" transaction-type="JTA">
4.     <provider>org.hibernate.ejb.HibernatePersistence</provider>
5.     <jta-data-source>jdbc/dbrdvmedecins</jta-data-source>
6.     <properties>
7.       <!-- logs SQL
8.         <property name="hibernate.show_sql" value="true"/>
9.         <property name="hibernate.format_sql" value="true"/>
10.        <property name="use_sql_comments" value="true"/>
11.       -->
12.       <!-- Dialecte -->
13.       <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect"/>
14.       <!-- création automatique du schéma
15.       <property name="hibernate.hbm2ddl.auto" value="update"/>
16.       -->
17.     </properties>
18.   </persistence-unit>
19. </persistence>

```

- ligne 3 : le type de transactions est JTA : les transactions seront gérées par le conteneur Ejb3 de Glassfish
- ligne 4 : une implémentation Jpa / Hibernate est utilisée. Pour cela, la bibliothèque Hibernate devra être ajoutée au serveur Glassfish.
- ligne 5 : la source de données JTA utilisée par la couche Jpa a le nom JNDI « jdbc/dbrdvmedecins ».

Selon sa version, le serveur Glassfish V2 peut ne pas avoir les bibliothèques Hibernate dont la couche Jpa / Hibernate a besoin. On ajoute ces bibliothèques dans le dossier [<glassfish>/domains/domain1/lib/ext] avant de (re)démarrer le serveur Glassfish :



- en [1], le dossier <glassfish>/.../lib/ext
- en [2], les bibliothèques Hibernate plus quelques pilotes Jdbc
- en [3], le pilote Jdbc de MySQL

La source de données "jdbc/dbrdvmedecins" est configurée dans le fichier [sun-resources.xml] suivant :

```

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE resources PUBLIC "-//Sun Microsystems, Inc.//DTD Application Server 9.0 Resource
   Definitions //EN" "http://www.sun.com/software/appserver/dtds/sun-resources_1_3.dtd">
3. <resources>
4.   <jdbc-resource enabled="true" jndi-name="jdbc/dbrdvmedecins" object-type="user" pool-
   name="dbrdvmedecinsPool">
5.     <description/>
6.   </jdbc-resource>
7.   <jdbc-connection-pool ...>
8.     <property name="URL" value="jdbc:mysql://localhost/dbrdvmedecins"/>
9.     <property name="User" value="root"/>
10.    <property name="Password" value="()"/>
11.  </jdbc-connection-pool>
12.</resources>

```

- lignes 8-10 : les caractéristiques Jdbc de la source de données (Url de la base, nom et mot de passe de l'utilisateur). La base de données MySQL *dbrdvmedecins* est celle décrite au paragraphe 4.1, page 5.
- ligne 7 : les caractéristiques du pool de connexions associé à cette source de données

## 4.6 Génération de la base de données

Créez la base de données MySql [dbrdvmedecins] avec l'outil de votre choix. Pour créer les tables et les remplir on pourra utiliser le script [createbd.sql] qui vous sera fourni. Son contenu est le suivant :

```

1. create table CLIENTS (
2.   ID bigint not null auto_increment,
3.   VERSION integer not null,
4.   TITRE varchar(5) not null,
5.   NOM varchar(30) not null,
6.   PRENOM varchar(30) not null,
7.   primary key (ID)

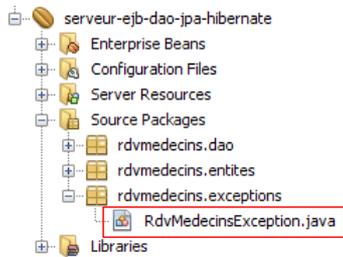
```

```

8.     ) ENGINE=InnoDB;
9.
10.    create table CRENEAUX (
11.        ID bigint not null auto_increment,
12.        VERSION integer not null,
13.        HDEBUT integer not null,
14.        MDEBUT integer not null,
15.        HFIN integer not null,
16.        MFIN integer not null,
17.        ID_MEDECIN bigint not null,
18.        primary key (ID)
19.    ) ENGINE=InnoDB;
20.
21.    create table MEDECINS (
22.        ID bigint not null auto_increment,
23.        VERSION integer not null,
24.        TITRE varchar(5) not null,
25.        NOM varchar(30) not null,
26.        PRENOM varchar(30) not null,
27.        primary key (ID)
28.    ) ENGINE=InnoDB;
29.
30.    create table RV (
31.        ID bigint not null auto_increment,
32.        JOUR date not null,
33.        ID_CLIENT bigint not null,
34.        ID_CRENEAU bigint not null,
35.        primary key (ID)
36.    ) ENGINE=InnoDB;
37.
38.    alter table CRENEAUX
39.        add index FK9BD7A197FE16862 (ID_MEDECIN),
40.        add constraint FK9BD7A197FE16862
41.        foreign key (ID_MEDECIN)
42.        references MEDECINS (ID);
43.
44.    alter table RV
45.        add index FKA4494D97AD2 (ID_CLIENT),
46.        add constraint FKA4494D97AD2
47.        foreign key (ID_CLIENT)
48.        references CLIENTS (ID);
49.
50.    alter table RV
51.        add index FKA441A673246 (ID_CRENEAU),
52.        add constraint FKA441A673246
53.        foreign key (ID_CRENEAU)
54.        references CRENEAUX (ID);
55.
56. INSERT INTO CLIENTS ( VERSION, NOM, PRENOM, TITRE) VALUES (1, 'MARTIN', 'Jules', 'Mr');
57. ...
58.
59. INSERT INTO MEDECINS ( VERSION, NOM, PRENOM, TITRE) VALUES (1, 'PELISSIER', 'Marie', 'Mme');
60. ...
61.
62. INSERT INTO CRENEAUX ( VERSION, ID_MEDECIN, HDEBUT, MDEBUT, HFIN, MFIN) VALUES (1, 1, 8, 0, 8,
63.    20);
64. ...
65. INSERT INTO RV ( JOUR, ID_CRENEAU, ID_CLIENT) VALUES ('2006-08-22', 1, 2);
66. ...
67.
68. ALTER TABLE RV ADD CONSTRAINT UNQ1_RV UNIQUE (JOUR, ID_CRENEAU);
69.
70. COMMIT WORK;

```

## 4.7 La classe d'exception



La classe d'exception [RdvMedecinsException] de l'application est la suivante :

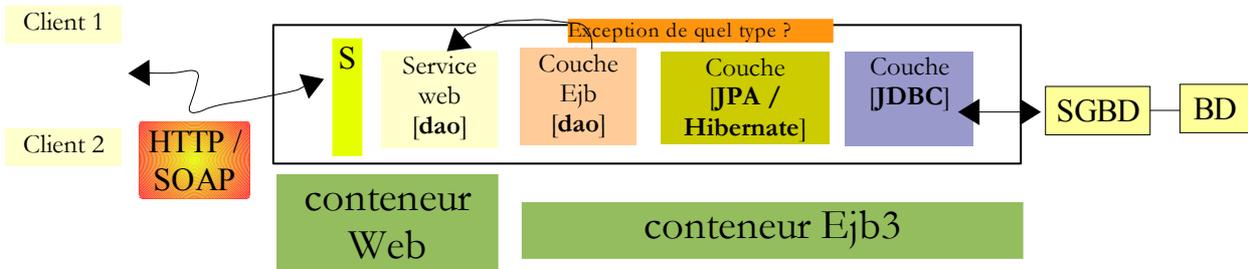
```

1. package rdvmedecins.exceptions;
2.
3. import javax.ejb.ApplicationException;
4.
5. @ApplicationException(rollback=true)
6. public class RdvMedecinsException extends RuntimeException {
7.
8.     private static final long serialVersionUID = 1L;
9.
10.    // champs privés
11.    private int code = 0;
12.
13.    // constructeurs
14.    public RdvMedecinsException() {
15.        super();
16.    }
17.
18.    public RdvMedecinsException(String message) {
19.        super(message);
20.    }
21.
22.    public RdvMedecinsException(String message, Throwable cause) {
23.        super(message, cause);
24.    }
25.
26.    public RdvMedecinsException(Throwable cause) {
27.        super(cause);
28.    }
29.
30.    public RdvMedecinsException(String message, int code) {
31.        super(message);
32.        setCode(code);
33.    }
34.
35.    public RdvMedecinsException(Throwable cause, int code) {
36.        super(cause);
37.        setCode(code);
38.    }
39.
40.    public RdvMedecinsException(String message, Throwable cause, int code) {
41.        super(message, cause);
42.        setCode(code);
43.    }
44.
45.    // getters - setters
46.    ...
47. }

```

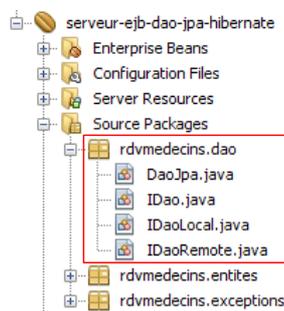
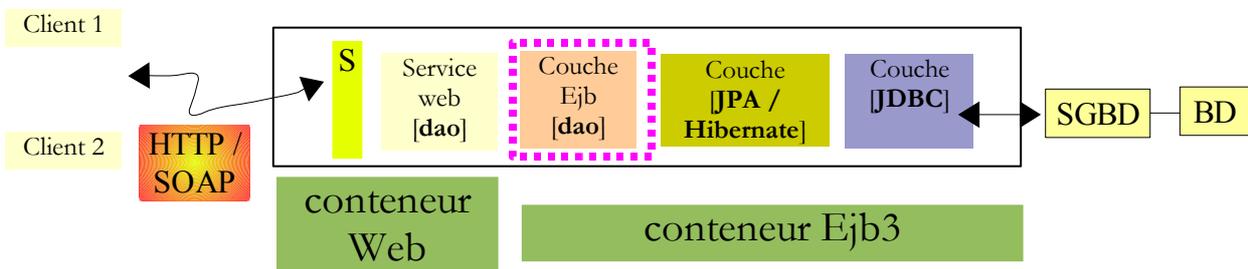
- ligne 6 : la classe dérive de la classe [RuntimeException]. Le compilateur ne force donc pas à la gérer avec des try / catch.
- ligne 5 : l'annotation **@ApplicationException** fait que l'exception ne sera pas "avalée" par une exception de type [EjbException].

Pour comprendre l'annotation **@ApplicationException** revenons à l'architecture utilisée côté serveur :



L'exception de type [RdvMedecinsException] sera lancée par les méthodes de l'Ejb de la couche [dao] à l'intérieur du conteneur Ejb3 et interceptée par celui-ci. Sans l'annotation **@ApplicationException** le conteneur Ejb3 encapsule l'exception survenue, dans une exception de type [EjbException] et relance celle-ci. On peut ne pas vouloir de cette encapsulation et laisser sortir du conteneur Ejb3 une exception de type [RdvMedecinsException]. C'est ce que permet l'annotation **@ApplicationException**. Par ailleurs, l'attribut (**rollback=true**) de cette annotation indique au conteneur Ejb3 que si l'exception de type [RdvMedecinsException] se produit à l'intérieur d'une méthode exécutée au sein d'une transaction avec un SGBD, celle-ci doit être annulée. En termes techniques, cela s'appelle faire un *rollback* de la transaction.

#### 4.8 L'Ejb de la couche [dao]



L'interface java [IDao] de la couche [dao] est la suivante :

```

1. package rdvmedecins.dao;
2. ...
3. public interface IDao {
4.
5.     // liste des clients
6.     public List<Client> getAllClients();
7.     // liste des Médecins
8.     public List<Medecin> getAllMedecins();
9.     // liste des créneaux horaires d'un médecin
10.    public List<Creneau> getAllCreneaux(Medecin medecin);
11.    // liste des Rv d'un médecin, un jour donné
12.    public List<Rv> getRvMedecinJour(Medecin medecin, String jour);
13.    // trouver un client identifié par son id
14.    public Client getClientById(Long id);
15.    // trouver un client idenbtifié par son id
16.    public Medecin getMedecinById(Long id);
17.    // trouver un Rv identifié par son id
18.    public Rv getRvById(Long id);
19.    // trouver un créneau horaire identifié par son id

```

```

20. public Creneau getCreneauById(Long id);
21. // ajouter un RV
22. public Rv ajouterRv(String jour, Creneau creneau, Client client);
23. // supprimer un RV
24. public void supprimerRv(Rv rv);
25. }

```

L'interface locale [IDaoLocal] de l'Ejb se contente de dériver l'interface [IDao] précédente :

```

1. package rdvmedecins.dao;
2.
3. import javax.ejb.Local;
4.
5. @Local
6. public interface IDaoLocal extends IDao{
7. }

```

Il en est de même pour l'interface distante [IDaoRemote] :

```

1. package rdvmedecins.dao;
2.
3. import javax.ejb.Remote;
4.
5. @Remote
6. public interface IDaoRemote extends IDao {
7. }

```

L'Ejb [DaoJpa] implémente les deux interfaces, locale et distante :

```

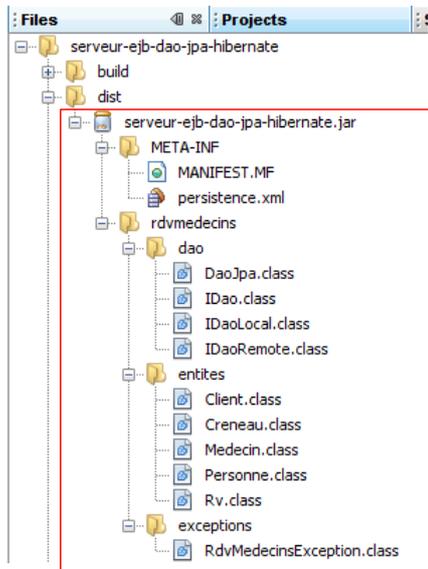
1. package rdvmedecins.dao;
2. ...
3. @Stateless(mappedName="rdvmedecins.dao")
4. @TransactionAttribute(TransactionAttributeType.REQUIRED)
5. public class DaoJpa implements IDaoLocal, IDaoRemote {
6. ...
7. }

```

Nous ne nous intéresserons pas au code de l'Ejb. Il n'a pas à être connu pour réaliser le TP. Cela nous permet de considérer la couche [Ejb dao] comme une boîte noire dont on connaît les interfaces locale et distante. C'est suffisant aux détails près suivants :

- la ligne 3 indique que l'Ejb distant porte le nom "rdvmedecins.dao"
- la ligne 4 indique que toutes les méthodes de l'Ejb se déroulent au sein d'une transaction gérée par le conteneur Ejb3.
- ce que ne dit pas le code ci-dessus est que l'exception lancée par l'Ejb est de type [RdvMedecinsException], le type présenté au paragraphe 4.7, page 15.

Le module Ejb présenté au paragraphe 4.3, page 8, une fois compilé donne naissance à un fichier .jar :

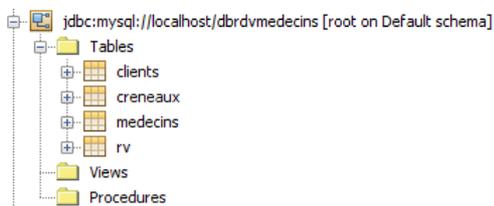


L'archive ci-dessus [serveur-ejb-dao-jpa-hibernate.jar] vous sera distribuée pour le TP.

## 4.9 Déploiement de l'archive de l'Ejb de la couche [dao]

Nous montrons ici comment déployer sur le serveur Glassfish un Ejb à partir de son archive .jar.

- lancez le serveur MySQL et assurez-vous que la base [dbrdvmedecins] est en ligne. Pour cela, vous pouvez suivre la procédure décrite au paragraphe 4.6, page 13.
- puis assurez-vous que vous êtes capable de créer une connexion Netbeans sur la base MySQL [dbrdvmedecins]. Cette procédure est décrite page 20 dans [ref1].



ID	VERSION	TITRE	NOM	PRENOM
1	1	Mr	MARTIN	Jules
2	1	Mme	GERMAN	Christine
3	1	Mr	JACQUARD	Jules
4	1	Melle	BISTROU	Brigitte

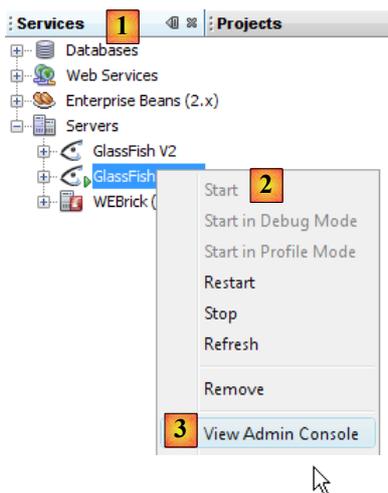
Rappelons la configuration Jpa du module Ejb qui va être déployé. Cette configuration est faite dans le fichier [persistence.xml] :

```

1. <?xml version="1.0" encoding="UTF-8"?>
2. <persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/
   ns/persistence http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
3.   <persistence-unit name="dbrdvmedecins" transaction-type="JTA">
4.     <provider>org.hibernate.ejb.HibernatePersistence</provider>
5.     <jta-data-source>jdbc/dbrdvmedecins</jta-data-source>
6.     <properties>
7.       <!-- Dialecte -->
8.       <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect"/>
9.     </properties>
10.  </persistence-unit>
11. </persistence>

```

La ligne 5 indique que la couche Jpa utilise une source de données JTA, c.a.d. gérée par le conteneur Ejb3, nommée "jdbc/dbrdvmedecins". Il nous faut créer cette source de données JTA. Nous suivons ici une procédure décrite au paragraphe 13.1.2, page 79 de [ref1].

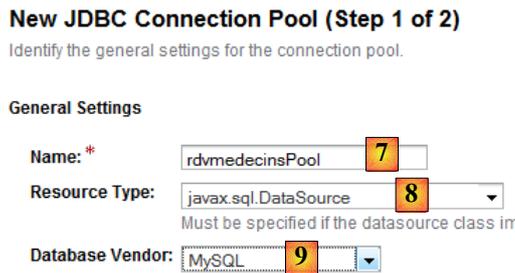


### Sun Java™ System Application Server Admin Console

User Name:

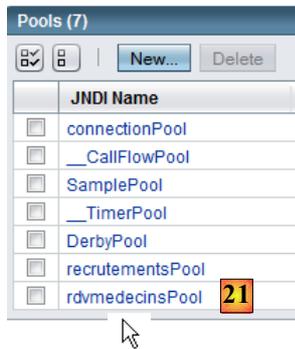
Password:

- dans l'onglet [services] [1] de Netbeans, lancez le serveur Glassfish [2] puis accédez [3] à sa console d'administration
- en [4], connectez-vous comme administrateur (mot de passe : *adminadmin* si vous n'avez pas changé celui-ci lors de l'installation ou après).



- en [5], sélectionnez la branche [Connection Pools] des ressources de Glassfish
- en [6], créez un nouveau pool de connexions. On rappelle qu'un pool de connexions est une technique pour limiter le nombre d'ouvertures / fermetures de connexions avec un SGBD. Au démarrage du serveur, N, un nombre défini par configuration, connexions sont ouvertes avec le SGBD. Ces connexions ouvertes sont ensuite mises à disposition des Ejb qui les sollicitent pour faire une opération avec le SGBD. Dès que celle-ci est terminée, l'Ejb rend la connexion au pool. La connexion n'est jamais fermée. Elle est partagée entre les différents threads qui accèdent au SGBD
- en [7], donnez un nom au pool
- en [8], la classe modélisant la source de données est la classe [javax.sql.DataSource]
- en [9], le SGBD qui détient la source de données est ici MySQL.
- en [10], passez à l'étape suivante

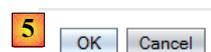
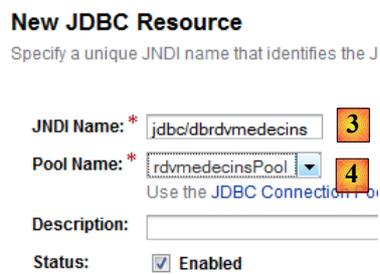
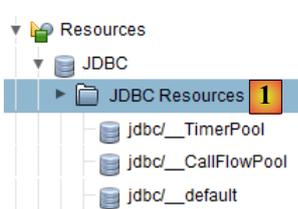




- en [21], le pool a été créé. On clique sur son lien.
- en [22], le bouton [Ping] permet de créer une connexion avec la bd [dbrdvmedecins]
- en [23], si tout va bien, un message indique que la connexion a réussi

23

Une fois le pool de connexions créé, on peut créer une ressource Jdbc :



- en [1], on sélectionne la branche [JDBC Resources] de l'arbre des objets du serveur
- en [2], on crée une nouvelle ressource JDBC
- en [3], on donne un nom à la ressource JDBC. Celui-ci doit correspondre au nom utilisé dans le fichier [persistence.xml] :

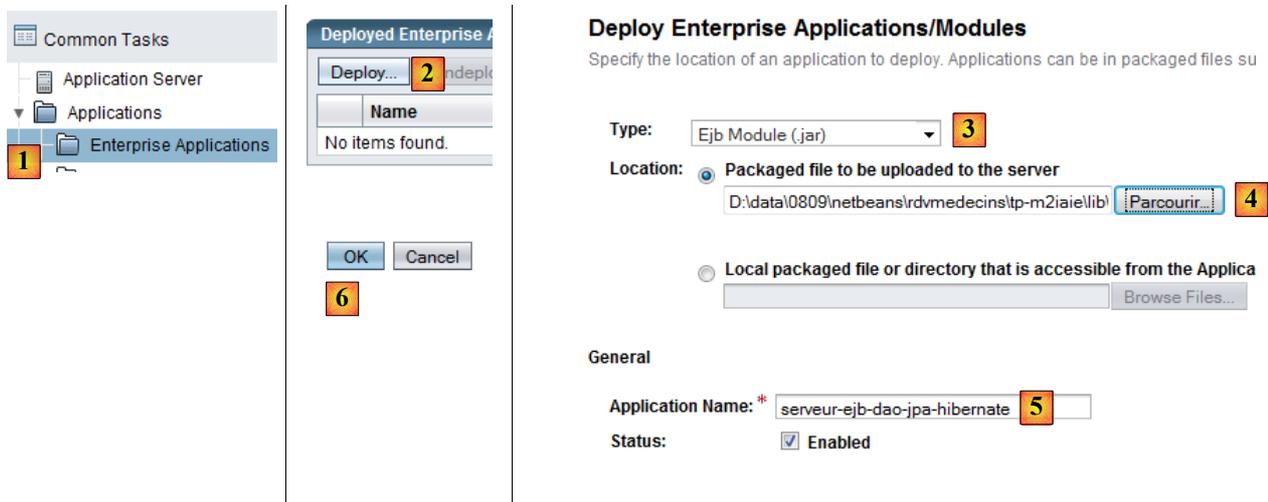
```
<jta-data-source>jdbc/dbrdvmedecins</jta-data-source>
```

- en [4], on précise le pool de connexions que doit utiliser la nouvelle ressource JDBC : celui qu'on vient de créer
- en [5], on termine l'assistant de création

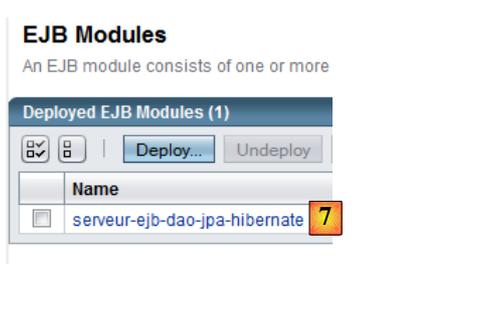
JNDI Name	Enabled	Connection Pool
jdbc/dbrdvmedecins <span style="border: 1px solid red; padding: 2px;">6</span>	true	rdvmedecinsPool
jdbc/sample	true	SamplePool
jdbc/__TimerPool	true	__TimerPool

- en [6] la nouvelle ressource JDBC

Maintenant que la ressource JDBC est créée, on peut déployer l'archive jar de l'Ejb :



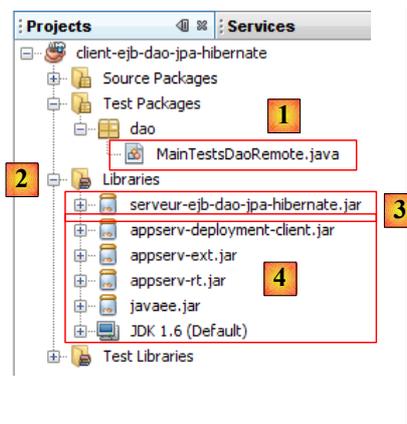
- en [1], sélectionnez la branche [Enterprise Applications]
- en [2], avec le bouton [Deploy], indiquez que vous voulez déployer une nouvelle application
- en [3], indiquez que l'application est un module Ejb
- en [4], sélectionnez le jar de l'Ejb [serveur-ejb-dao-jpa-hibernate.jar] qui vous aura été donné pour le TP.
- en [5], vous pouvez changer le nom du module Ejb si vous le souhaitez
- en [6], terminez l'assistant de déploiement du module ejb



- en [7], le module Ejb a été déployé. Il peut désormais être utilisé.

## 4.10 Tests de l'Ejb de la couche [dao]

Maintenant que l'Ejb de la couche [dao] de notre application a été déployée, nous pouvons le tester. Nous le ferons au moyen du client Java suivant :



La classe [MainTestsDaoRemote] [1] est une classe de test JUnit 4. Les bibliothèques en [2] sont constituées d'une part :

- du jar de l'Ejb de la couche [dao] [3]
- des bibliothèques Glassfish [4] nécessaires aux clients distants des Ejb. Elles vous seront fournies également.

La classe de test est la suivante :

```

1. package dao;
2. ...
3. public class MainTestsDaoRemote {
4.
5.     // couche [dao] testée
6.     private static IDaoRemote dao;
7.
8.     @BeforeClass
9.     public static void init() throws NamingException {
10.         // initialisation environnement JNDI
11.         InitialContext initialContext = new InitialContext();
12.         // instanciation couche dao
13.         dao = (IDaoRemote) initialContext.lookup("rdvmedecins.dao");
14.     }
15.
16.     @Test
17.     public void test1() {
18.         // données du test
19.         String jour = "2006:08:23";
20.         // affichage clients
21.         List<Client> clients = null;
22.         try {
23.             clients = dao.getAllClients();
24.             display("Liste des clients :", clients);
25.         } catch (Exception ex) {
26.             System.out.println(ex);
27.         }
28.         // affichage médecins
29.         List<Medecin> medecins = null;
30.         try {
31.             medecins = dao.getAllMedecins();
32.             display("Liste des médecins :", medecins);
33.         } catch (Exception ex) {
34.             System.out.println(ex);
35.         }
36.         // affichage créneaux d'un médecin
37.         Medecin medecin = medecins.get(0);
38.         List<Creneau> creneaux = null;
39.         try {
40.             creneaux = dao.getAllCreneaux(medecin);
41.             display(String.format("Liste des créneaux du médecin %s", medecin), creneaux);
42.         } catch (Exception ex) {
43.             System.out.println(ex);
44.         }
45.         // liste des Rv d'un médecin, un jour donné
46.         try {
47.             display(String.format("Liste des créneaux du médecin %s, le [%s]", medecin, jour),
48.                 dao.getRvMedecinJour(medecin, jour));
49.         } catch (Exception ex) {
50.             System.out.println(ex);

```

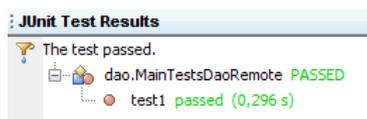
```

50.     }
51.     // ajouter un RV
52.     Rv rv = null;
53.     Creneau creneau = creneaux.get(2);
54.     Client client = clients.get(0);
55.     System.out.println(String.format("Ajout d'un Rv le [%s] dans le créneau %s pour le client %s",
    jour, creneau, client));
56.     try {
57.         rv = dao.ajouterRv(jour, creneau, client);
58.         System.out.println("Rv ajouté");
59.         display(String.format("Liste des Rv du médecin %s, le [%s]", medecin, jour),
    dao.getRvMedecinJour(medecin, "2006:08:23"));
60.     } catch (Exception ex) {
61.         System.out.println(ex);
62.     }
63.     // ajouter un RV dans le même créneau du même jour
64.     // doit provoquer une exception
65.     System.out.println(String.format("Ajout d'un Rv le [%s] dans le créneau %s pour le client %s",
    jour, creneau, client));
66.     try {
67.         rv = dao.ajouterRv(jour, creneau, client);
68.         System.out.println("Rv ajouté");
69.         display(String.format("Liste des Rv du médecin %s, le [%s]", medecin, jour),
    dao.getRvMedecinJour(medecin, "2006:08:23"));
70.     } catch (Exception ex) {
71.         System.out.println(ex);
72.     }
73.     // supprimer un RV
74.     System.out.println("Suppression du Rv ajouté");
75.     try {
76.         dao.supprimerRv(rv);
77.         System.out.println("Rv supprimé");
78.         display(String.format("Liste des Rv du médecin %s, le [%s]", medecin, jour),
    dao.getRvMedecinJour(medecin, "2006:08:23"));
79.     } catch (Exception ex) {
80.         System.out.println(ex);
81.     }
82. }
83.
84. // méthode utilitaire - affiche les éléments d'une collection
85. private static void display(String message, List elements) {
86.     System.out.println(message);
87.     for (Object element : elements) {
88.         System.out.println(element);
89.     }
90. }
91. }

```

- ligne 13 : on notera l'instanciation du proxy de l'Ejb distant. On utilise son nom JNDI "rdvmedecins.dao".

Si tout va bien, les tests doivent passer :

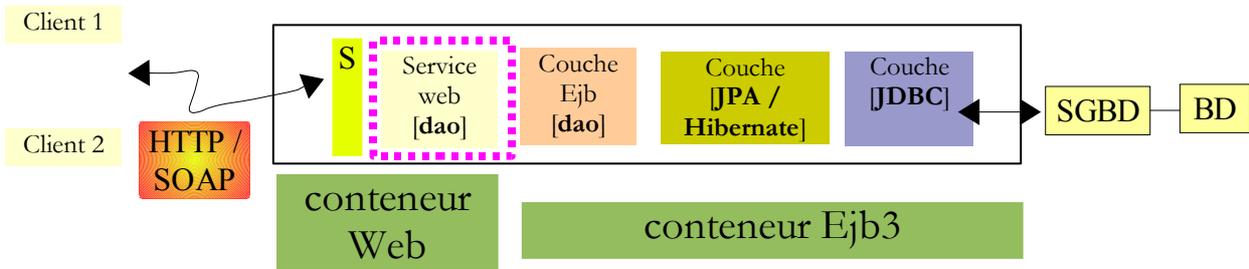


Maintenant que l'Ejb de la couche [dao] est opérationnel, on peut passer à son exposition publique via un service web.

## 4.11 Le service web de la couche [dao]

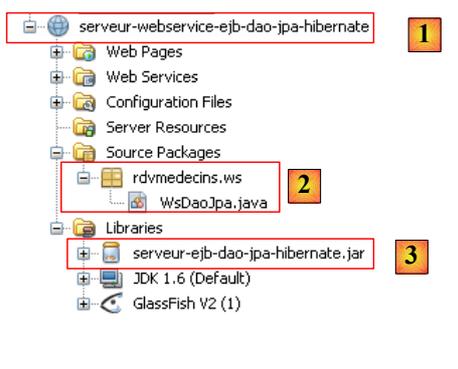
Pour une introduction à la notion de service web, on lira le paragraphe 14, page 111 de [ref1].

Revenons à l'architecture du serveur de notre application client / serveur :



Nous nous intéressons ci-dessus au service web de la couche [dao]. Ce service a pour seul rôle de rendre disponible l'interface de l'Ejb de la couche [dao] à des clients multi-plateformes capables de dialoguer avec un service web.

Le projet Netbeans du service web sera le suivant :



Les éléments notables du projet sont les suivants :

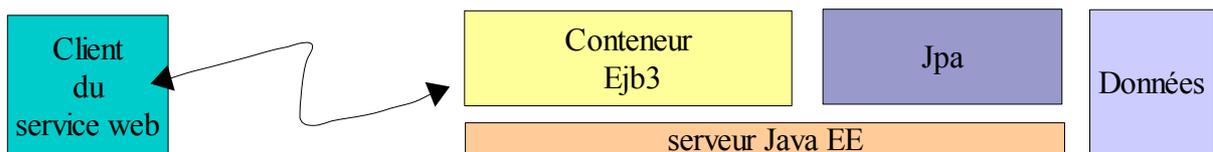
- [1] : le service web est implémenté par un projet Netbeans de type [Web Application].
- [2] : le service web est implémenté par la classe [WsDao]pa
- [3] : l'archive de l'Ejb de la couche [dao] qui permet à la classe [WsDao]pa d'avoir accès aux définitions des différentes classes, interfaces, entités des couches [dao] et [jpa].

Rappelons qu'il y a deux façons d'implémenter un service web :

- par une classe annotée **@WebService** qui s'exécute dans un conteneur web



- par un Ejb annoté **@WebService** qui s'exécute dans un conteneur Ejb



Nous utilisons ici la première solution. Dans l'IDE Netbeans, il faut construire un projet d'entreprise avec deux modules :

- le module Ejb qui s'exécutera dans le conteneur Ejb : l'Ejb de la couche [dao].
- le module web qui s'exécutera dans le conteneur web : le service web que nous sommes en train de construire.

Le projet actuellement étudié est le module web de la future application d'entreprise. Le service web est implémenté par la classe [WsDaoJpa] suivante :

```
1. package rdvmedecins.ws;
2. ...
3. @WebService()
4. public class WsDaoJpa implements IDao {
5.
6.     @EJB
7.     private IDaoLocal dao;
8.
9.     // liste des clients
10.    @WebMethod
11.    public List<Client> getAllClients() {
12.        return dao.getAllClients();
13.    }
14.
15.    // liste des médecins
16.    @WebMethod
17.    public List<Medecin> getAllMedecins() {
18.        return dao.getAllMedecins();
19.    }
20.
21.    // liste des créneaux horaires d'un médecin donné
22.    // medecin : le médecin
23.    @WebMethod
24.    public List<Creneau> getAllCreneaux(Medecin medecin) {
25.        return dao.getAllCreneaux(medecin);
26.    }
27.
28.    // liste des Rv d'un médecin donné, un jour donné
29.    // medecin : le médecin
30.    // jour : le jour
31.    @WebMethod
32.    public List<Rv> getRvMedecinJour(Medecin medecin, String jour) {
33.        return dao.getRvMedecinJour(medecin, jour);
34.    }
35.
36.    // ajout d'un Rv
37.    // jour : jour du Rv
38.    // creneau : créneau horaire du Rv
39.    // client : client pour lequel est pris le Rv
40.    @WebMethod
41.    public Rv ajouterRv(String jour, Creneau creneau, Client client) {
42.        return dao.ajouterRv(jour, creneau, client);
43.    }
44.
45.    // suppression d'un Rv
46.    // rv : le Rv supprimé
47.    @WebMethod
48.    public void supprimerRv(Rv rv) {
49.        dao.supprimerRv(rv);
50.    }
51.
52.    // récupérer un client donné
53.    @WebMethod
54.    public Client getClientById(Long id) {
55.        return dao.getClientById(id);
56.    }
57.
58.    // récupérer un médecin donné
59.    @WebMethod
60.    public Medecin getMedecinById(Long id) {
61.        return dao.getMedecinById(id);
62.    }
63.
64.    // récupérer un Rv donné
65.    @WebMethod
66.    public Rv getRvById(Long id) {
67.        return dao.getRvById(id);
68.    }
69.
70.    // récupérer un créneau donné
71.    @WebMethod
72.    public Creneau getCreneauById(Long id) {
73.        return dao.getCreneauById(id);
```

```
74. }
75. }
```

- ligne 4, la classe [Wsdao]pa implémente l'interface [IDao]. Rappelons que cette interface est définie dans l'archive de l'Ejb de la couche [dao] sous la forme suivante :

```
a) package rdvmedecins.dao;
b) ...
c) public interface IDao {
d)
e) // liste des clients
f) public List<Client> getAllClients();
g) // liste des Médecins
h) public List<Medecin> getAllMedecins();
i) // liste des créneaux horaires d'un médecin
j) public List<Creneau> getAllCreneaux(Medecin medecin);
k) // liste des Rv d'un médecin, un jour donné
l) public List<Rv> getRvMedecinJour(Medecin medecin, String jour);
m) // trouver un client identifié par son id
n) public Client getClientById(Long id);
o) // trouver un client identifié par son id
p) public Medecin getMedecinById(Long id);
q) // trouver un Rv identifié par son id
r) public Rv getRvById(Long id);
s) // trouver un créneau horaire identifié par son id
t) public Creneau getCreneauById(Long id);
u) // ajouter un RV
v) public Rv ajouterRv(String jour, Creneau creneau, Client client);
w) // supprimer un RV
x) public void supprimerRv(Rv rv);
y) }
```

- ligne 3 : l'annotation **@WebService** fait de la classe [WsDao]pa un service web.
- lignes 6-7 : la référence de l'Ejb de la couche [dao] sera injectée dans le champ de la ligne 7. Rappelons que c'est toujours l'implémentation locale (*IDaoLocal* ici) qui est ainsi injectée. Cette injection est possible parce que le service web s'exécute dans la même Jvm que l'Ejb.
- toutes les méthodes du service web sont taguées avec l'annotation **@WebMethod** pour en faire des méthodes visibles aux clients distants. Une méthode non taguée avec l'annotation **@WebMethod** serait interne au service web et non visible aux clients distants. Chaque méthode M du service web se contente d'appeler la méthode M correspondante de l'Ejb injecté en ligne 7.

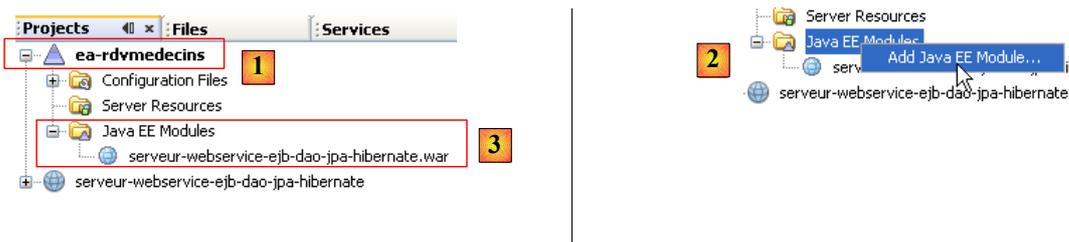
## 4.12 Déploiement du service web de l'application

Revenons à l'architecture du déploiement :



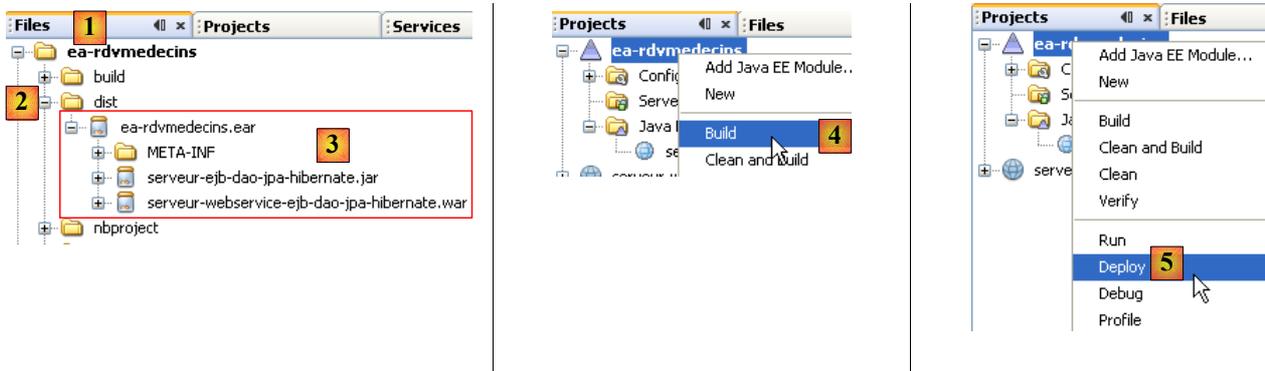
Avec l'IDE Netbeans, il nous faut construire un projet d'entreprise avec deux modules :

- le module Ejb qui s'exécutera dans le conteneur Ejb : l'Ejb de la couche [dao].
- le module web qui s'exécutera dans le conteneur web : le service web que nous sommes en train de construire.



- [1], on crée une application d'entreprise [ea-rdvmedecins], au départ sans aucun module.
- en [2], on ajoute le module web [serveur-webservice-ejb-dao-jpa-hibernate] étudié précédemment
- en [3], le résultat.

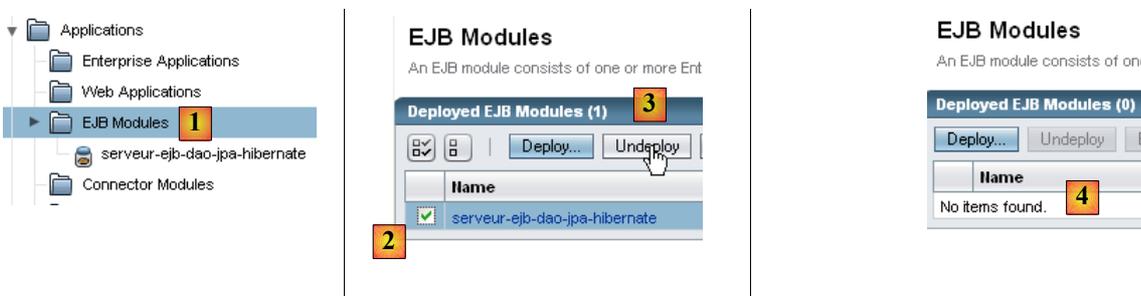
Telle quelle, l'application d'entreprise [ea-rdvmedecins] ne peut pas être déployée sur le serveur Glassfish à partir de Netbeans. On obtient une erreur (essayez). Il faut alors déployer à la main, l'archive **ear** de l'application [ea-rdvmedecins] :



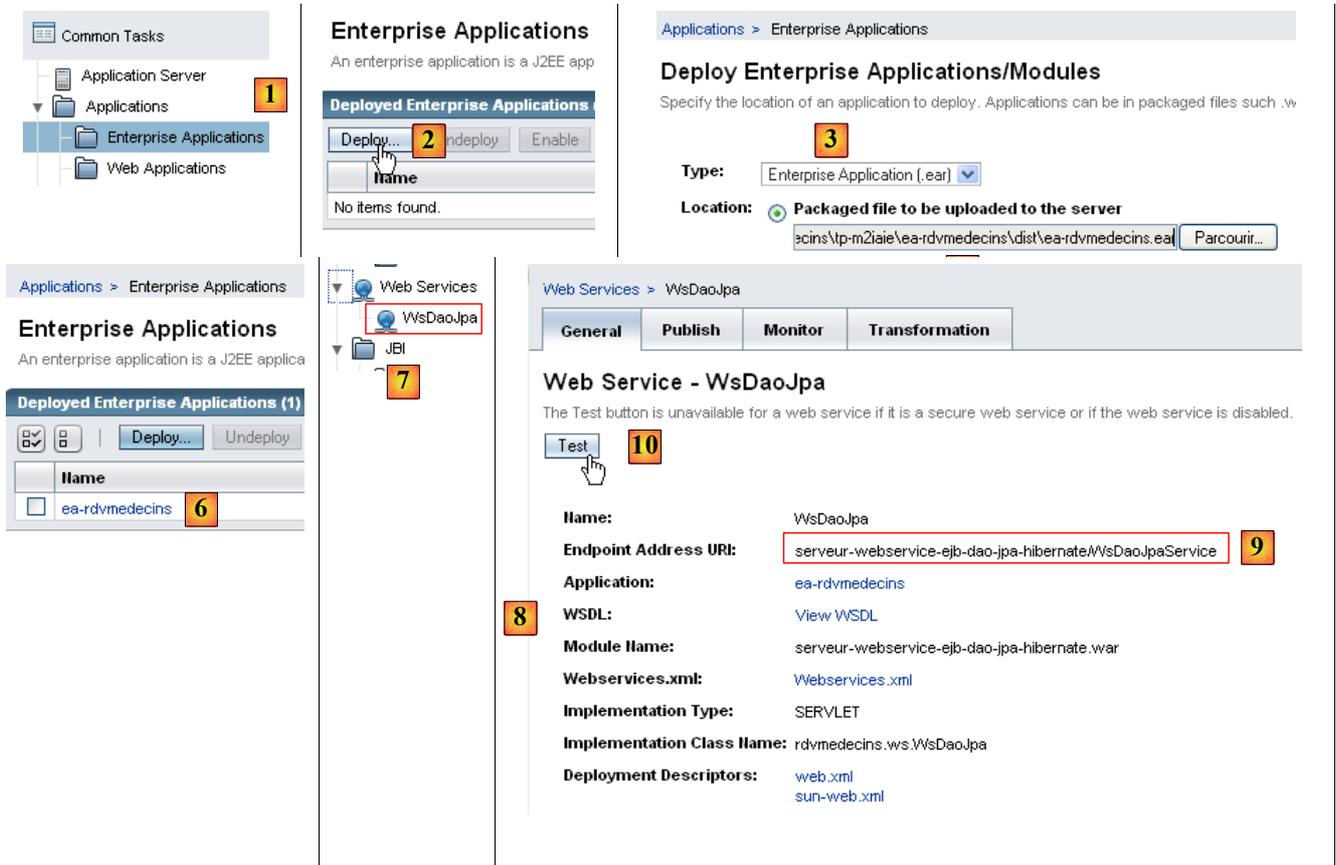
- l'archive [ea-rdvmedecins.ear] est trouvée dans le dossier [dist] [2] de l'onglet [Files] de Netbeans.
- dans cette archive [3], on trouve les deux éléments de l'application d'entreprise :
  - l'archive de l'Ejb [serveur-ejb-dao-jpa-hibernate]. Cette archive est présente parce qu'elle faisait partie des bibliothèques référencées par le service web.
  - l'archive du service web [serveur-webservice-ejb-dao-jpa-hibernate].
- l'archive [ea-rdvmedecins.ear] est construite par un simple *Build* [4] de l'application d'entreprise.
- en [5], l'opération de déploiement qui échoue.

Pour déployer l'archive [ea-rdvmedecins.ear] de l'application d'entreprise, nous procédons comme il a été montré lors du déploiement de l'archive de l'Ejb [serveur-ejb-dao-jpa-hibernate.jar] au paragraphe 4.6, page 13. Nous utilisons de nouveau le client web d'administration du serveur Glassfish. Nous ne répétons pas les étapes déjà décrites.

Tout d'abord, on commencera par "décharger" l'Ejb déployé au paragraphe 4.6, page 13 :



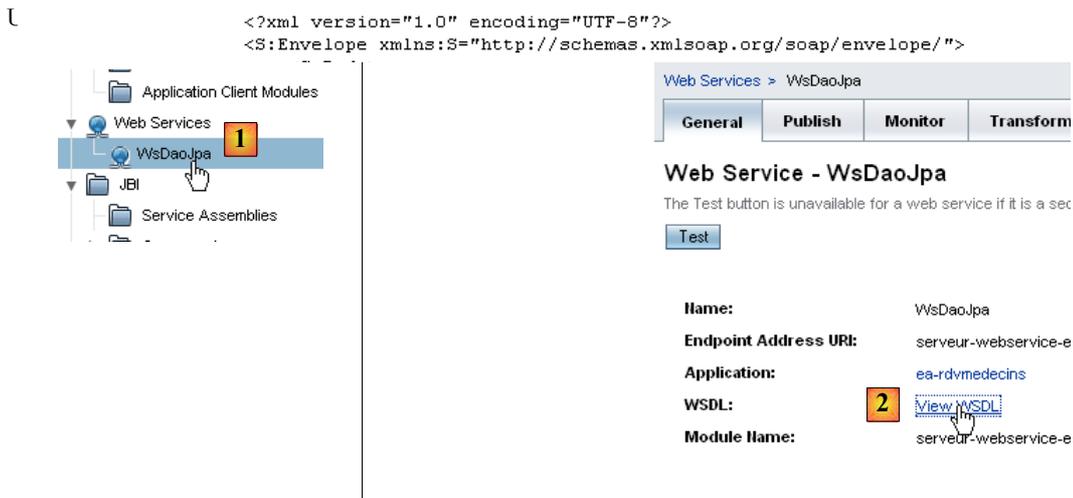
- [1] : sélectionnez la branche [EJB Modules] du serveur Glassfish
- en [2] sélectionnez l'Ejb à décharger puis en [3] déchargez-le
- en [4] il n'est plus là.



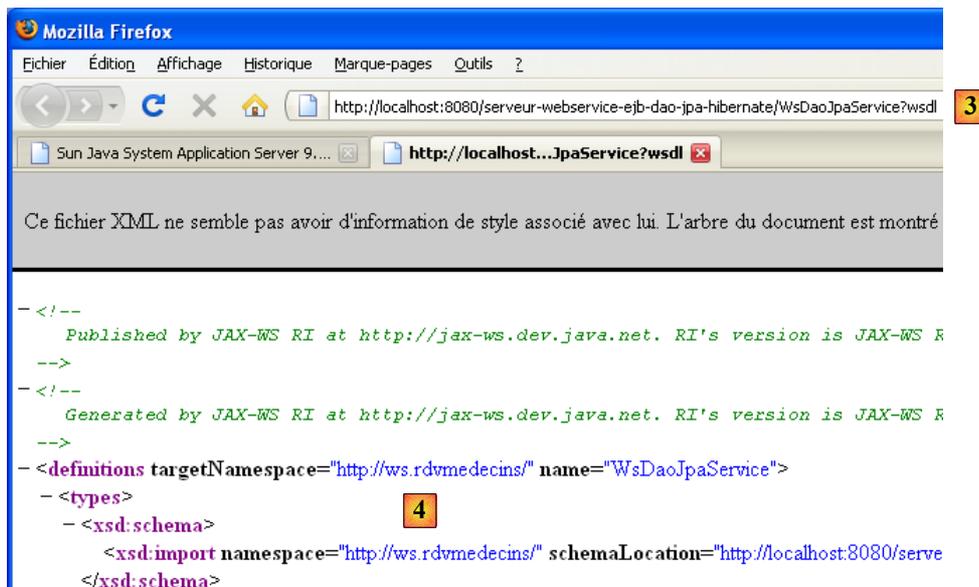
- en [6], l'application a été déployée
- en [7], le service web [WsDaoJpa] apparaît dans la branche [Web Services] du serveur Glassfish. On le sélectionne.
- en [8], on a diverses informations sur le service web. La plus intéressante pour un client est l'information [9] : l'uri du service web.
- en [10], on peut tester le service web



- en [11], l'uri du service web à laquelle on a ajouté le paramètre ?tester. Cette uri présente une page de test. Toutes les méthodes (@WebMethod) exposées par le service web sont affichées et peuvent être testées. Ici, on teste la méthode [13] qui demande la liste des clients.



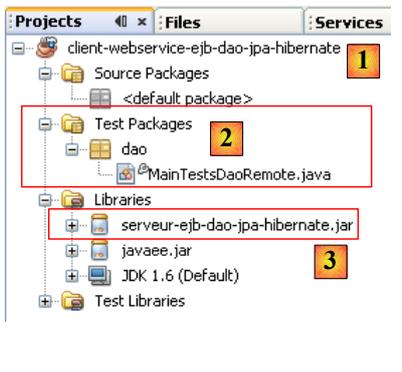
- en [1] dans l'outil web d'administration du serveur Glassfish, sélectionnez le service web [WsDaoJpa]
- en [2], suivez le lien [View WSDL]



- en [3] : l'uri du fichier WSDL. C'est une information importante à connaître. Elle vous sera nécessaire pour configurer les clients de ce service web.
- en [4], la description XML du service web. Nous ne commenterons pas ce contenu complexe.

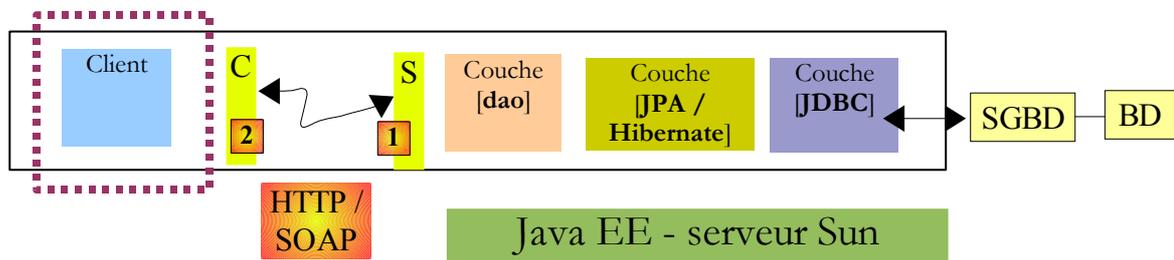
### 4.13 Tests JUnit du service web

Nous créons un projet Netbeans pour "jouer" les tests déjà joués avec un client Ejb avec cette fois-ci un client pour le service web nouvellement déployé. Nous suivons ici une démarche analogue à celle décrite au paragraphe 14.2.1, page 115 de [ref1].



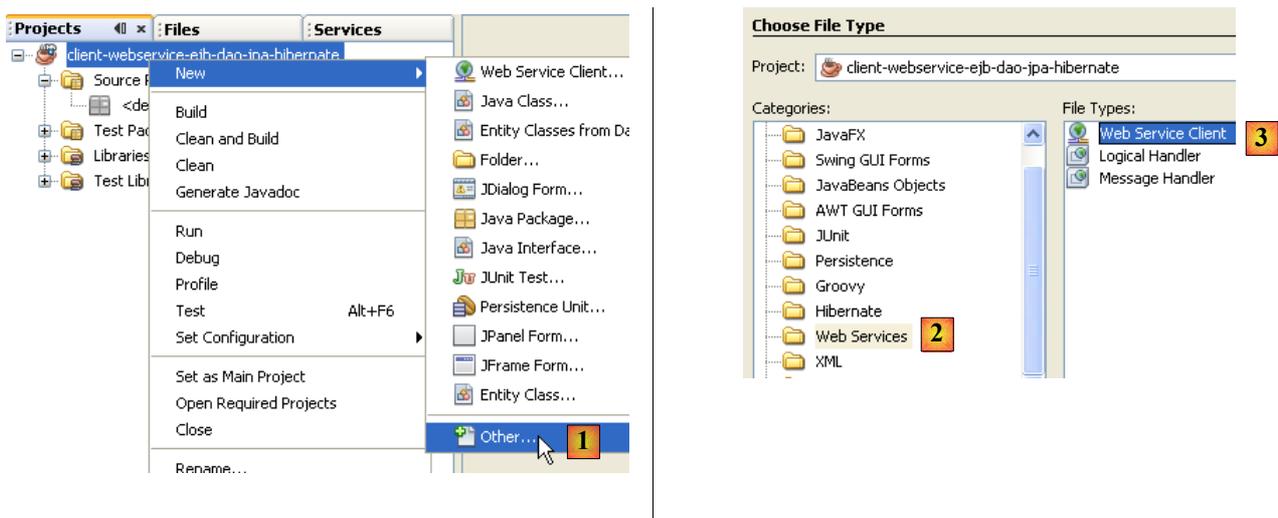
- en [1], un projet Java classique
- en [2], la classe de test
- en [3], le client utilise l'archive de l'Ejb pour avoir accès aux définitions de l'interface de la couche [dao] et des entités Jpa.

Pour accéder au service web distant, il est nécessaire de générer des classes proxy :



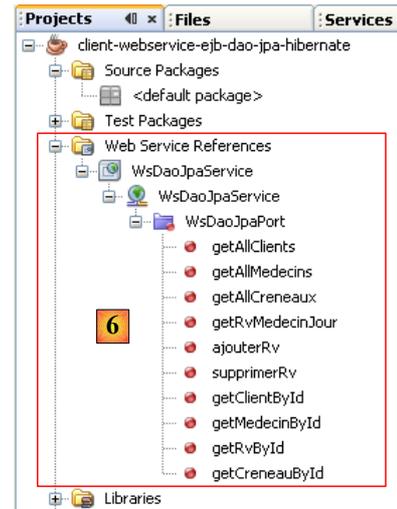
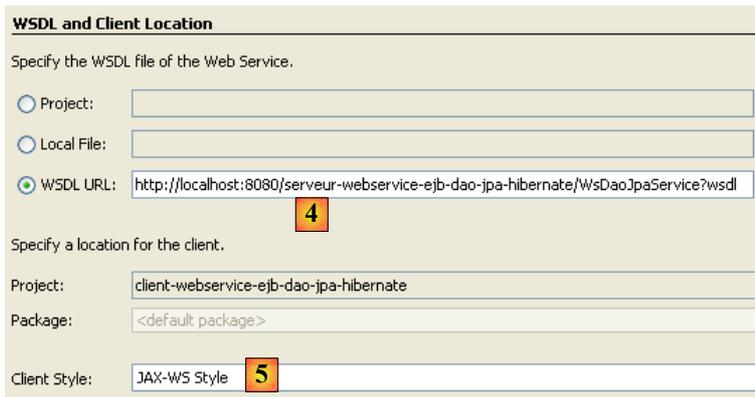
Dans le schéma ci-dessus, la couche [2] [C=Client] communique avec la couche [1] [S=Serveur]. Pour dialoguer avec la couche [S], le client [C] est amené à créer une connexion réseau avec la couche [S] et à dialoguer avec elle selon un protocole précis. Les connexions réseau sont des connexions TCP et le protocole de transport est HTTP. La couche [S] qui représente le service web est implémentée par une servlet Java exécutée par le serveur Glassfish. Nous n'avons pas écrit cette servlet. Sa génération est automatisée par Glassfish à partir des annotations `@WebService` et `@WebMethod` de la classe [WsDao]pa que nous avons écrite. De même, nous allons automatiser la génération de la couche [C] du client. On appelle parfois la couche [C], une couche **proxy** du service web distant, le terme *proxy* désignant un élément intermédiaire dans une chaîne logique. Ici, le proxy C est l'intermédiaire entre le client que nous allons écrire et le service web que nous avons déployé.

Avec Netbeans 6.5, le proxy C peut être généré de la façon suivante (pour la suite, il faut que le service web soit lancé) :

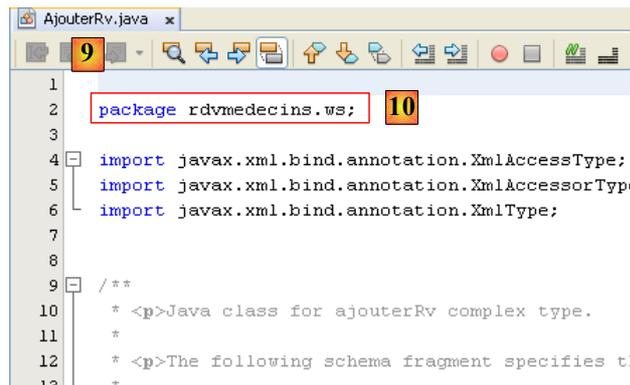
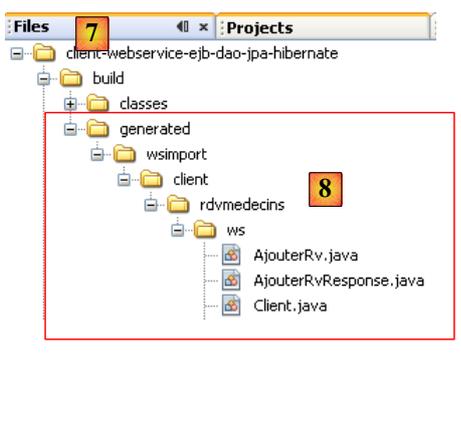


- en [1], ajoutez un nouvel élément au projet Java

- en [2], sélectionnez la branche [Web services]
- en [3], sélectionnez [Web Service Client]



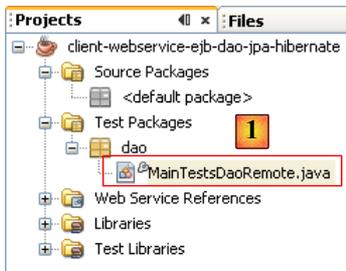
- en [4], fournissez l'uri du fichier WSDL du service web. Cette uri a été présentée au paragraphe 4.12, page 29.
- en [5], laissez la valeur par défaut [JAX-WS]. L'autre valeur possible est [JAX-RPC]
- après avoir validé l'assistant de création du proxy du service web, le projet Netbeans a été enrichi d'une branche [Web Service References] [6]. Cette branche montre les méthodes exposées par le service web distant.



- dans l'onglet [Files] [7], des codes sources Java ont été rajoutés [8]. Ils correspondent au proxy C généré.
- en [9] le code de l'une des classes. On y voit [10] qu'elles ont été placées dans un paquetage [rdvmedecins.ws]. Nous ne commenterons pas le code de ces classes qui est de nouveau assez complexe.

Pour le client Java que nous sommes en train de construire, le proxy C généré sert d'intermédiaire. Pour accéder à la méthode M du service web distant, le client Java appelle la méthode M du proxy C. Le client Java appelle ainsi des méthodes locales (exécutées dans la même Jvm) et de façon transparente pour lui, ces appels locaux sont traduits en appels distants.

Il nous reste à savoir appeler les méthodes M du proxy C. Revenons à notre classe de test JUnit :



En [1], la classe de test [MainTestsDaoRemote] est celle déjà utilisée lors du test de l'Ejb de la couche [dao] :

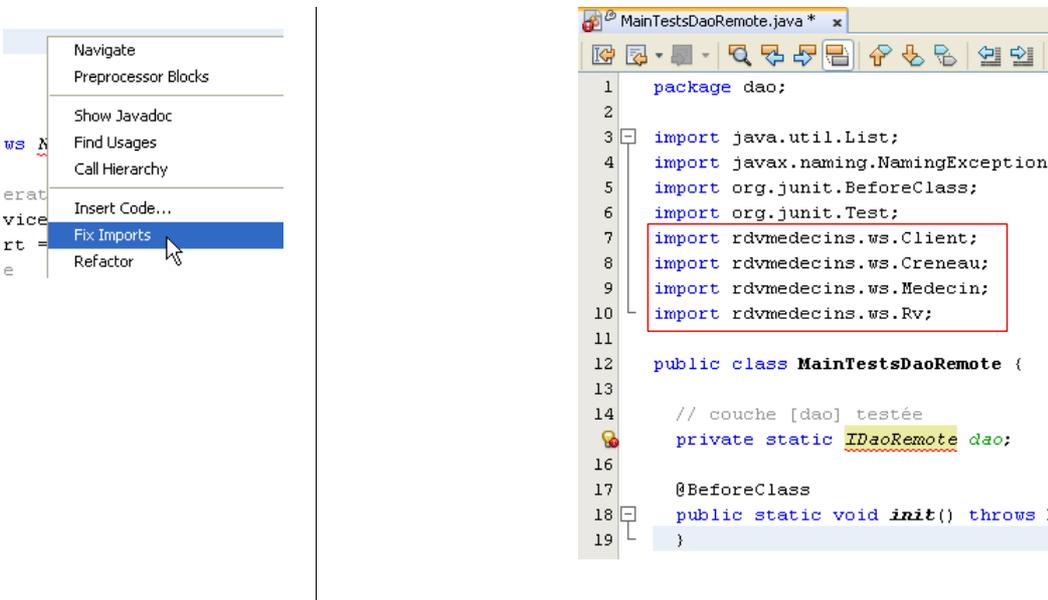
```

1. package dao;
2. ...
3. public class MainTestsDaoRemote {
4.
5.     // couche [dao] testée
6.     private static IDaoRemote dao;
7.
8.     @BeforeClass
9.     public static void init() throws NamingException {
10.    }
11.
12.     @Test
13.     public void test1() {
14.    ...
15.    }
16. }

```

- ligne [13], le test *test1* est conservé à l'identique.
- ligne [9], le contenu de la méthode [init] a été supprimé.

A ce stade, le projet présente des erreurs car la méthode de test [test1] utilise les entités [Client], [Medecin], [Creneau], [Rv] qui ne sont plus dans les mêmes packages qu'auparavant. Elles sont dans le package du proxy C généré. On supprime les instructions *import* concernées et on les régénère par l'opération *Fix Imports*.



Revenons au code de la classe de test [MainTestsDaoRemote] :

```

1. package dao;
2. ...
3.
4. public class MainTestsDaoRemote {
5.

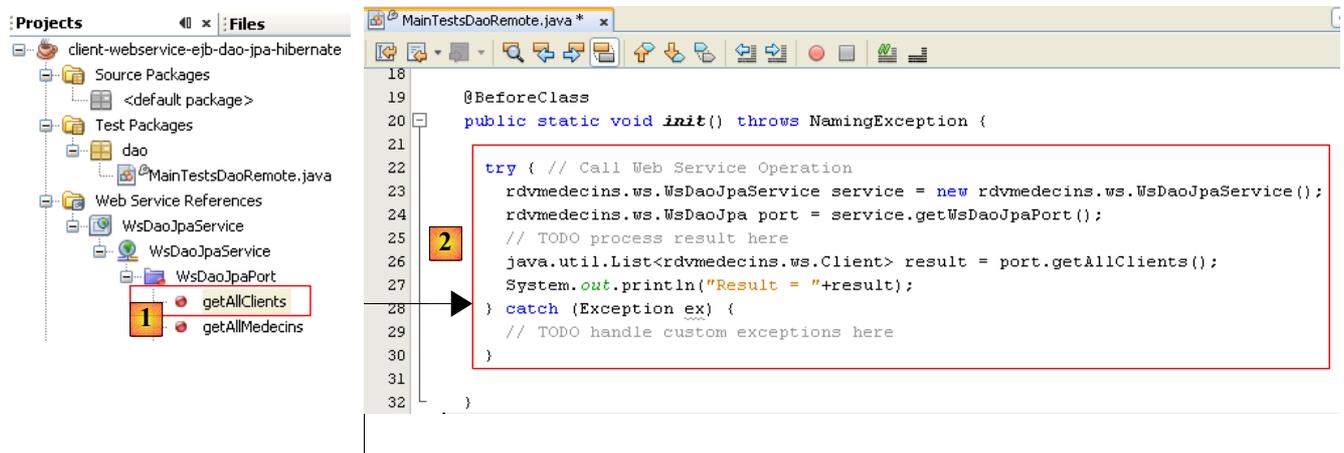
```

```

6. // couche [dao] testée
7. private static IDaoRemote dao;
8.
9. @BeforeClass
10. public static void init() throws NamingException {
11. }

```

La méthode [init] de la ligne 10 doit initialiser la référence de la couche [dao] de la ligne 7. Il nous faut savoir comment utiliser le proxy C généré, dans notre code. Netbeans nous aide dans cette démarche.



Sélectionnez en [1] la méthode [getAllClients] du service web et avec la souris, tirez cette méthode pour aller la déposer au sein de la méthode [init] de la classe de test. On obtient le résultat [2]. Ce squelette de code nous montre comment utiliser le proxy C généré :

```

1. try { // Call Web Service Operation
2.     rdvmedecins.ws.WsDaoJpaService service = new rdvmedecins.ws.WsDaoJpaService();
3.     rdvmedecins.ws.WsDaoJpa port = service.getWsDaoJpaPort();
4.     // TODO process result here
5.     java.util.List<rdvmedecins.ws.Client> result = port.getAllClients();
6.     System.out.println("Result = "+result);
7. } catch (Exception ex) {
8.     // TODO handle custom exceptions here
9. }

```

- la ligne [5] nous montre que la méthode [getAllClients] est une méthode de l'objet de type [WsDaoJpa] défini ligne 3. Le type [WsDaoJpa] est une interface présentant les mêmes méthodes que celles exposées par le service web distant.
- ligne [3], l'objet [WsDaoJpa port] est obtenu à partir d'un autre objet de type [WsDaoJpaService] défini ligne 2. Le type [WsDaoJpaService] représente le proxy C généré localement.
- l'accès au service web distant peut échouer, aussi l'ensemble du code est-il entouré d'un try / catch.
- les objets du proxy C sont dans le package [rdvmedecins.ws]

Une fois ce code compris, on voit que la référence locale du service web distant peut être obtenue par le code :

```
WsDaoJpa dao=new WsDaoJpaService().getWsDaoJpaPort();
```

Le code de la classe de test JUnit devient alors le suivant :

```

1. package dao;
2.
3. import rdvmedecins.ws.Client;
4. import rdvmedecins.ws.Creneau;
5. import rdvmedecins.ws.Medecin;
6. import rdvmedecins.ws.Rv;
7. import rdvmedecins.ws.WsDaoJpa;
8. import rdvmedecins.ws.WsDaoJpaService;
9. ...
10.
11. public class MainTestsDaoRemote {
12.
13.     // couche [dao] testée
14.     private static WsDaoJpa dao;
15.
16.     @BeforeClass

```

```

17. public static void init(){
18.     dao=new WsDaoJpaService().getWsDaoJpaPort ();
19. }
20.
21. @Test
22. public void test1() {
23. ...
24. }
25.
26. // méthode utilitaire - affiche les éléments d'une collection
27. private static void display(String message, List elements) {
28. ...
29. }
30. }

```

Nous sommes désormais prêts pour les tests :



En [1], le test JUnit est exécuté. En [2], il est réussi. Si on regarde les affichages sur la console Netbeans, on trouve des lignes comme les suivantes :

```

Liste des clients :
rdvmedecins.ws.Client@1982fc1
rdvmedecins.ws.Client@676437
rdvmedecins.ws.Client@1e4853f
rdvmedecins.ws.Client@1e808ca

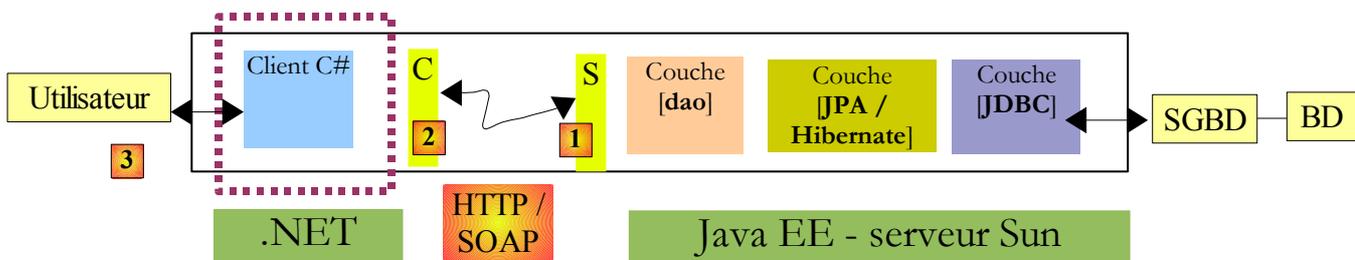
```

Côté serveur, l'entité [Client] a une méthode *toString* qui affiche les différents champs d'un objet de type [Client]. Lors de la génération automatique du proxy C, les entités sont créées dans le proxy C mais avec seulement les champs privés accompagnés de leurs méthodes get / set. Ainsi la méthode *toString* n'a pas été générée dans l'entité [Client] du proxy C. Ce qui explique l'affichage précédent. Ceci n'enlève rien au test JUnit : il a été réussi. On considèrera désormais qu'on a un service web opérationnel.

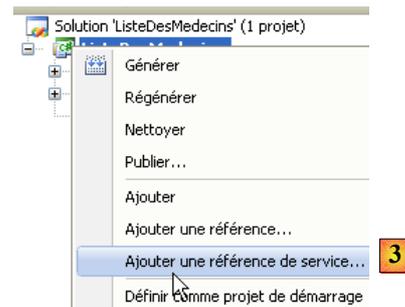
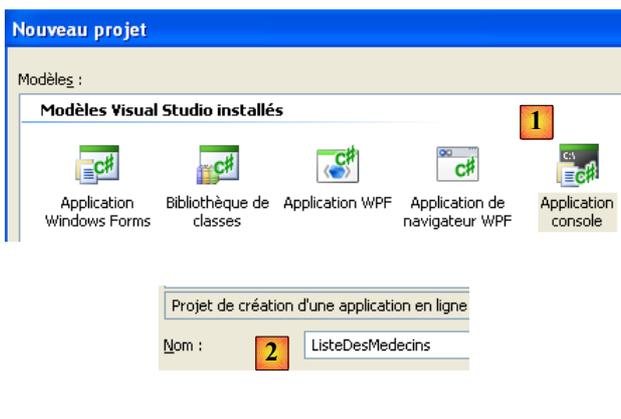
## 5 Clients .NET du service J2EE des rendez-vous

### 5.1 Un client C# 2008

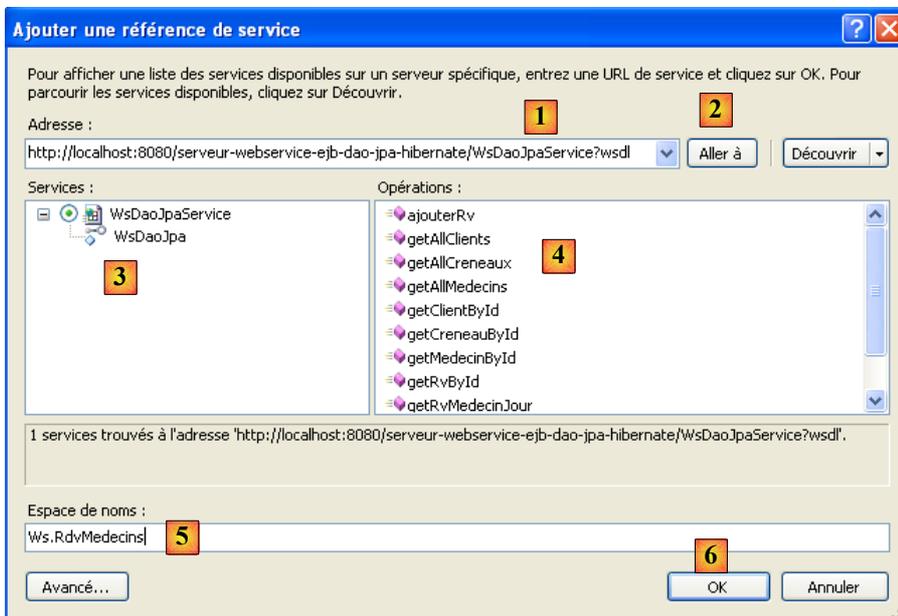
Nous supposons désormais que le service web précédent est disponible et actif. Un service web est utilisable par des clients qui peuvent être écrits en différents langages. Nous nous proposons ici d'écrire un client console C# pour afficher la liste des médecins.



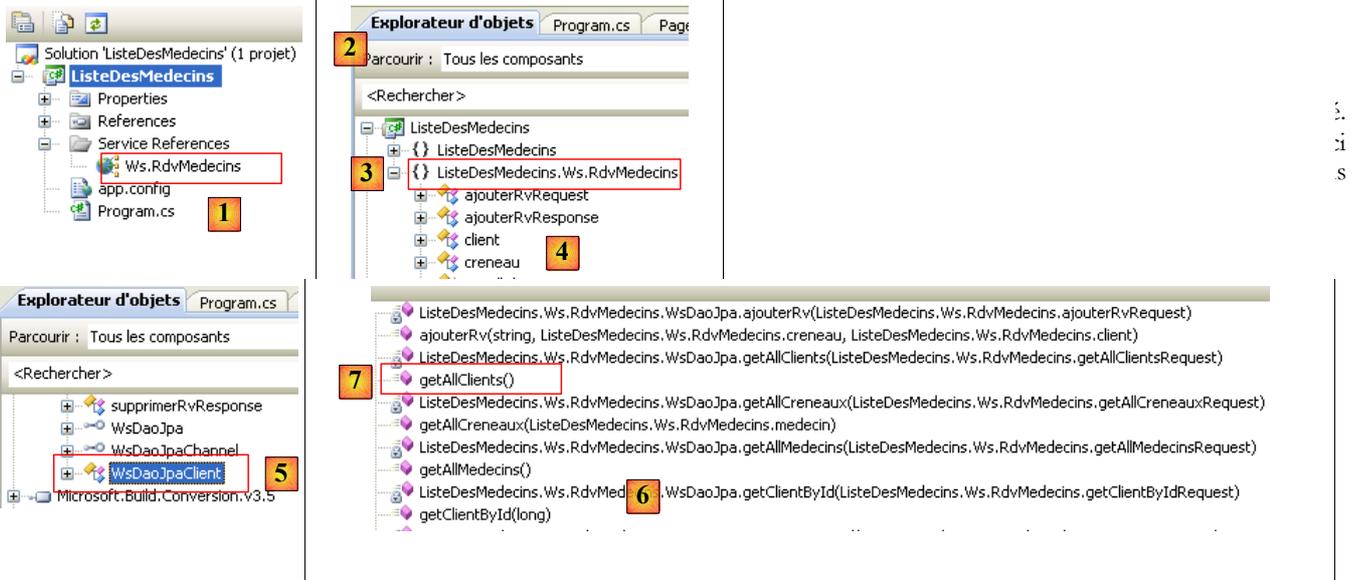
Commençons par créer le projet C# :



On crée une application console [1] appelée [ListeDesMedecins] [2]. En [3], on crée une référence sur un service web. Cet outil est similaire à celui utilisé dans Netbeans : il crée un proxy local du service web distant.

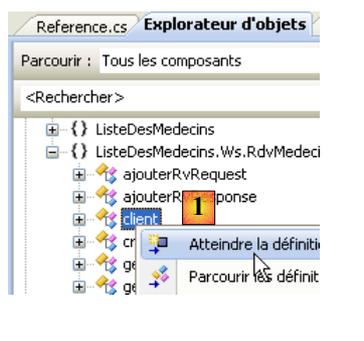


- en [1], on indique l'uri du fichier WSDL du service web (cf paragraphe 4.12, page 29).
- en [2], on demande à l'assistant de découvrir les services web exposés à cette uri
- en [3], le service web trouvé et en [4] les méthodes qu'il expose.
- en [5], on précise dans quel espace de noms, les classes du proxy C doivent être générées
- en [6], on génère le proxy C.



- en [5], [WsDao]paClient] est la classe implémentant localement les méthodes du service web distant.
- en [6], les méthodes de la classe [WsDao]paClient]. On y retrouve les méthodes du service web distant comme par exemple la méthode *getAllClients* en [7].

Examinons le code des entités générées :



En [1] ci-dessus, on demande à voir le code de la classe [client] :

```

1. ...
2.     public partial class client : personne {
3.     }
4.
5. ....
6.     public partial class personne : object, System.ComponentModel.INotifyPropertyChanged {
7.     ...
8.         public long id {
9.             ...
10.        }
11.
12. ...
13. }

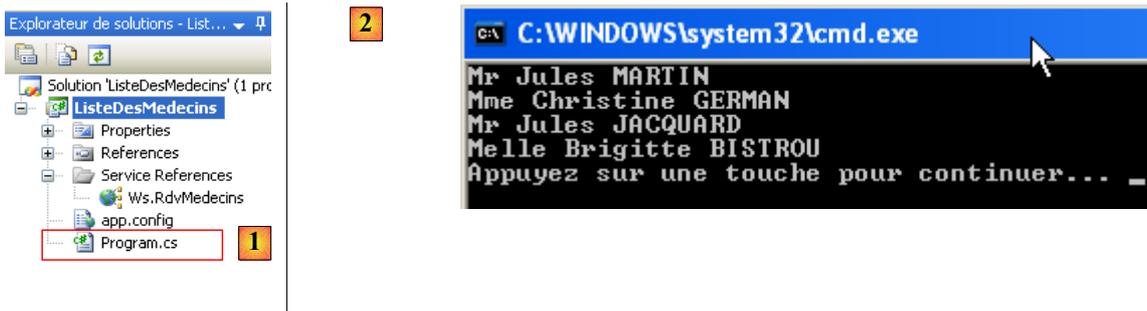
```

Ci-dessus, nous n'avons gardé que le code nécessaire à notre étude.

- ligne 2 : la classe [client] dérive de la classe [personne] comme dans le service web.
- ligne 8 : une propriété publique de la classe [personne]

Ce qu'on remarque, c'est que le code généré ne respecte pas les normes de codage habituelles de C#. Les noms des classes et des propriétés publiques devraient commencer par une majuscule. Cela s'avère assez gênant dans la pratique.

Revenons à notre application C# :



[Program.cs] est la classe de test. Son code est le suivant :

```

1. using System;
2. using ListeDesMedecins.Ws.RdvMedecins;
3. using Client = ListeDesMedecins.Ws.RdvMedecins.client;
4.
5. namespace ListeDesMedecins {
6.     class Program {
7.         static void Main(string[] args) {
8.             try {
9.                 // instanciation couche [dao] du client
10.                WsDaoJpaClient dao = new WsDaoJpaClient();
11.                // liste des médecins
12.                foreach (Client client in dao.getAllClients()) {
13.                    Console.WriteLine(String.Format("{0} {1} {2}", client.titre, client.prenom, client.nom));
14.                }
15.            } catch (Exception e) {
16.                Console.WriteLine(e);
17.            }
18.        }
19.    }
20. }

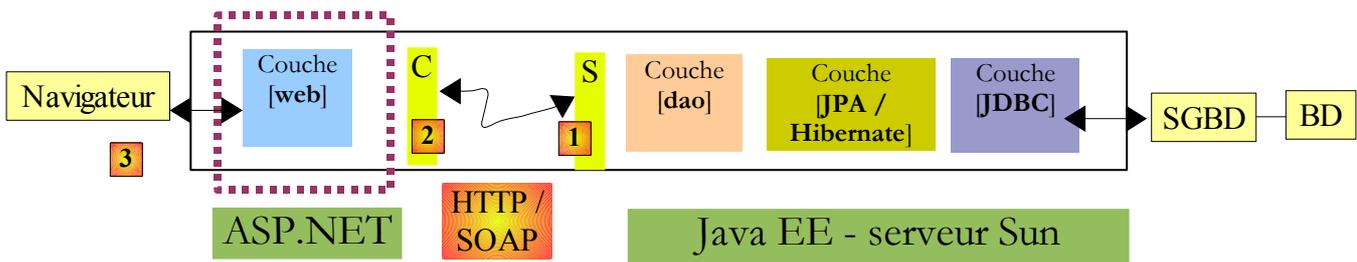
```

Ligne 12, la classe [Client] est utilisée alors que nous venons de voir qu'elle s'appelait en réalité [client]. C'est la déclaration de la ligne 3 qui nous permet d'utiliser [Client] au lieu de [client]. Cela nous permet de respecter la norme de codage des noms de classe. Toujours ligne 12, la méthode [getAllClients] est utilisée parce c'est ainsi qu'elle s'appelle dans le proxy C. La norme de codage C# voudrait qu'elle s'appelle [GetAllClients].

L'exécution du programme [1] précédent donne les résultats montrés en [2].

## 5.2 Un client ASP.NET 2008

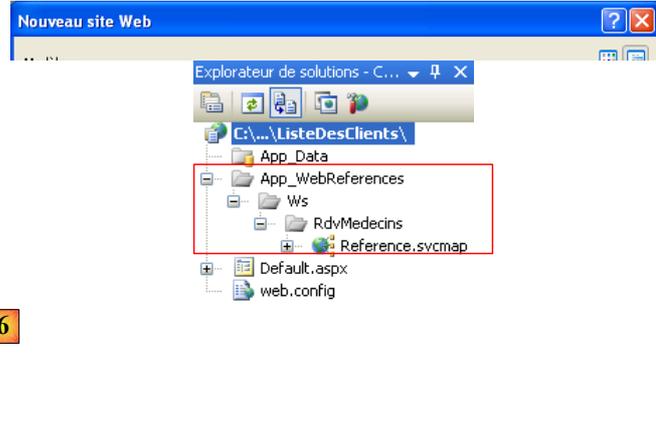
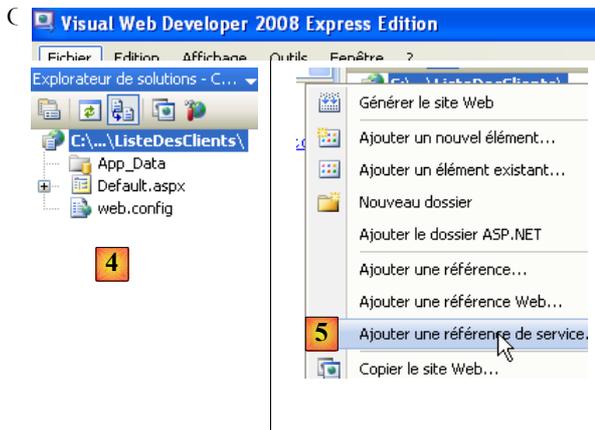
Nous écrivons maintenant un client ASP.NET / C# pour afficher la liste des clients.



Dans cette architecture, il y a deux serveurs web :

- celui qui exécute le service web distant
- celui qui exécute le client ASP.NET du service web distant

Créons le projet web du client .NET :



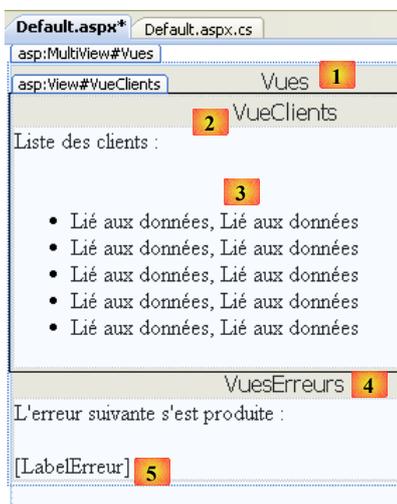
- en [4], le projet web
- en [5], par un clic droit sur le nom du projet, on ajoute une référence de service comme il a été fait avec le projet C#. On suivra la procédure décrite alors.
- en [6], le projet une fois la référence au service web distant ajoutée

Contrairement au projet C# étudié précédemment, on n'a pas accès aux objets du client du service web via l'explorateur d'objets. C'est très dommage... En pratique, il faut savoir que cela dépend de la version Visual Web 2008 dont on dispose. Dans la version de base, il n'est pas possible de créer un projet web (Fichier / Nouveau / Projet). On peut seulement créer un site web (Fichier / Nouveau / Site web). Une mise à jour de Visual web 2008 amène de nouvelles fonctionnalités dont celle de créer un projet. Si on avait pu créer un projet, on aurait eu accès aux objets du proxy C généré.

Ici, il faut savoir diverses choses :

- l'espace de noms du proxy C généré est celui défini dans son assistant de création : **Ws.RdvMedecins**
- la classe proxy qui implémente localement les méthodes du service web distant s'appelle **WsDaoJpaClient**.
- les noms des classes et des propriétés commencent par une minuscule

Nous pouvons écrire une page [Default.aspx] affichant la liste des médecins. La page est la suivante :



N°	Type	Nom	Rôle
[1]	MultiView	Vues	Conteneur de vues
[2]	View	VueClients	la vue qui contient la liste des clients
[3]	Repeater	RepeaterClients	affiche la liste des clients sous la forme d'une liste à puces

N°	Type	Nom	Rôle
4	View	VueErreurs	la vue qui affiche une éventuelle erreur
5	Label	LabelErreur	le texte de l'erreur

Le code de présentation de cette page est comme suit :

```

1. <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
2. <%@ Import Namespace="Ws.RdvMedecins" %>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4. <html xmlns="http://www.w3.org/1999/xhtml">
5. <head id="Head1" runat="server">
6. <title>Liste des clients</title>
7. </head>
8. <body>
9. <form id="form1" runat="server">
10. <div>
11. <asp:MultiView ID="Vues" runat="server">
12. <asp:View ID="VueClients" runat="server">
13. Liste des clients :<br />
14. <br />
15. <ul>
16. <asp:Repeater ID="RepeaterClients" runat="server">
17. <ItemTemplate>
18. <li>
19. <%#(Container.DataItem as client).nom%>, <%#(Container.DataItem as
client).prenom%>
20. </li>
21. </ItemTemplate>
22. </asp:Repeater>
23. </ul>
24. </asp:View>
25. <asp:View ID="VuesErreurs" runat="server">
26. L'erreur suivante s'est produite :<br />
27. <br />
28. <asp:Label ID="LabelErreur" runat="server"></asp:Label></asp:View>
29. </asp:MultiView><br />
30. </div>
31. </form>
32. </body>
33. </html>

```

On notera, ligne 2, la nécessité d'importer l'espace de noms [Ws.RdvMedecins] du proxy C du service web afin que la classe [client] de la ligne 19 soit accessible.

Le code de contrôle [Default.aspx.cs] associé à la page de présentation [Default.aspx] est le suivant :

```

1. using System;
2. using Ws.RdvMedecins;
3.
4. public partial class _Default : System.Web.UI.Page
5. {
6.     protected void Page_Load(object sender, EventArgs e)
7.     {
8.         try
9.         {
10.            // instantiation proxy local de la couche [dao] distante
11.            WsDaoJpaClient dao = new WsDaoJpaClient();
12.            // vue client
13.            Vues.ActiveViewIndex = 0;
14.            // initialisation vue client
15.            RepeaterClients.DataSource = dao.getAllClients();
16.            RepeaterClients.DataBind();
17.        }
18.        catch (Exception ex)
19.        {
20.            // vue erreurs
21.            Vues.ActiveViewIndex = 1;
22.            //initialisation vue erreurs
23.            LabelErreur.Text = ex.ToString();
24.        }
25.    }
26. }

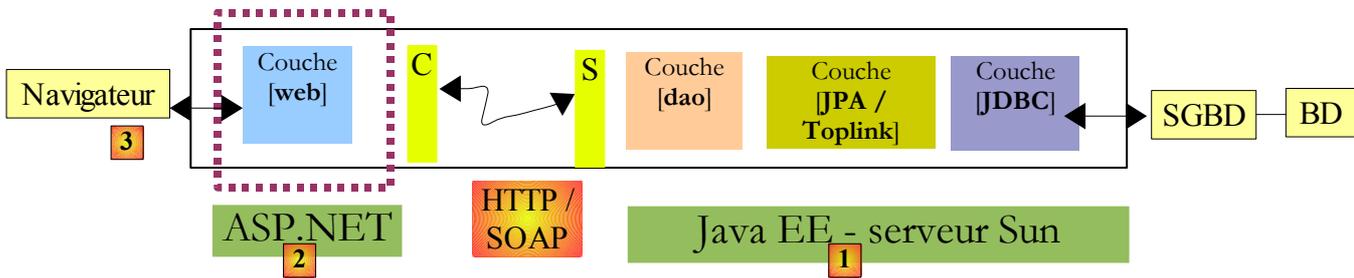
```

Nous laissons le lecteur prendre possession de ce code. Un code similaire devra être écrit par la suite.

L'exécution du projet web donne le résultat suivant :



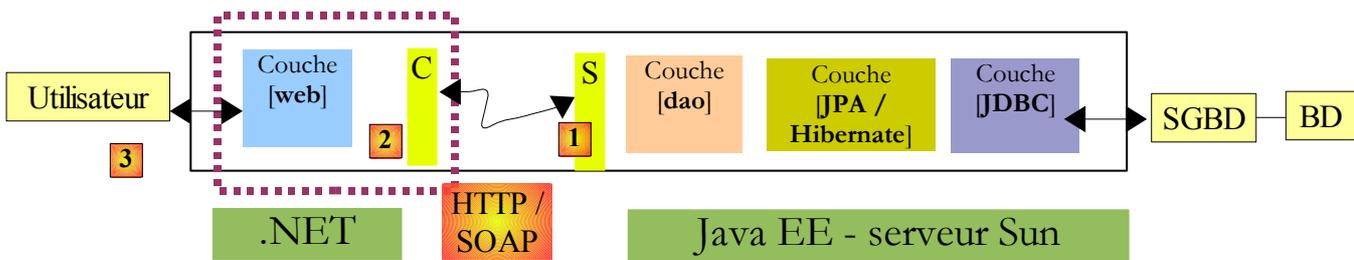
Nous en savons désormais assez pour développer la couche [web] [1] de notre application de rendez-vous :



Dans l'architecture ci-dessus, il y a deux serveurs web :

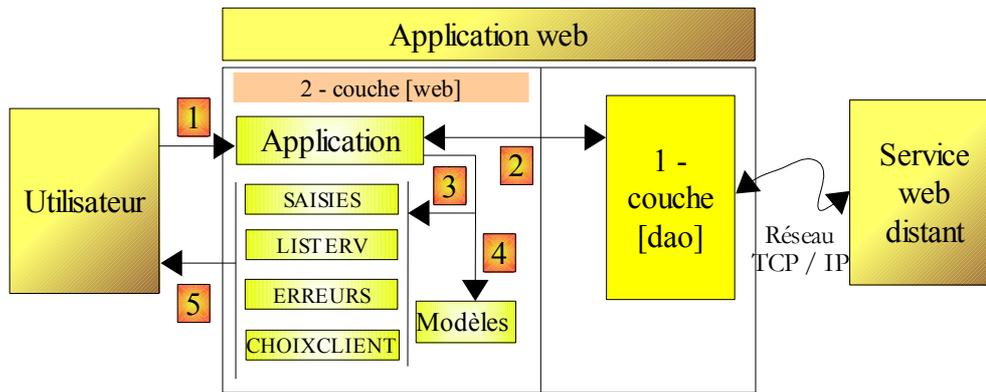
- [1] : un serveur d'application Sun ou Glassfish pour exposer le service web Java
- [2] : un serveur web ASP.NET qui gère l'application web cliente du service web Java. C'est cette application web ASP.NET qui présente les pages de gestion des rendez-vous des médecins aux navigateurs clients [3].

## 6 Client ASP.NET de gestion des rendez-vous



La couche [1-dao] est en réalité une couche proxy (c.a.d. servant d'intermédiaire) vers un service [dao] distant. Néanmoins, la couche [2-web] l'ignore et pour elle, cette couche *proxy* représente la couche d'accès aux données. Aussi appellerons-nous cette couche, la couche [dao] de l'application web ASP.NET.

Le bloc [2] du schéma ci-dessus est détaillé ci-dessous :



Le traitement d'une demande d'un client se déroule selon les étapes suivantes :

1. le client fait une demande à l'application.
2. l'application traite cette demande. Pour ce faire, elle peut avoir besoin de l'aide de la couche [dao] si des données doivent être échangées avec la base de données. L'application reçoit une réponse de la couche [dao].
3. selon celle-ci, elle choisit (3) la vue (= la réponse) à envoyer au client en lui fournissant (4) les informations (le modèle) dont elle a besoin.
4. la réponse est envoyée au client (5)

On a ici une architecture web dite MVC (Modèle – Vue – Contrôleur) :

- [Application] est le **contrôleur**. Il voit passer toutes les requêtes du client.
- [Saisies, ListeRv, Erreurs, ChoixClient] sont les **vues**. Une vue en .NET est du code ASP / HTML standard qui contient des composants qu'il convient d'initialiser. Les valeurs qu'il faut fournir à ces composants forment le **modèle** de la vue.

Nous allons ici implémenter le modèle (design pattern) MVC de la façon suivante :

- les vues seront des composants [View] au sein d'une page unique [Default.aspx]
- le contrôleur est alors le code [Default.aspx.cs] de cette page unique.

Le texte qui suit comporte des questions intermédiaires. L'étudiant n'est pas obligé de suivre la méthodologie proposée. Il peut délivrer l'application qui lui convient le mieux, tant qu'elle répond aux fonctionnalités demandées.

## 6.1 Les vues de l'application

Les différentes vues présentées à l'utilisateur ont déjà été vues au paragraphe 3, page 2. On les rappelle de nouveau :

- la vue [Saisies] qui présente le formulaire de saisie (choix du médecin, choix du jour de Rv) :

**Cabinet médical - LES MEDECINS AS**

---

Prise de rendez-vous :

Médecin

Jour (JJ/MM/AAAA)

- la vue [ListeRv] utilisée pour afficher les Rv d'un médecin donné, un jour donné :

## Cabinet médical - LES MEDECINS ASSOCIES -

## Cabinet médical - LES MEDECINS ASSOCIES -

Rendez-vous : 8h0-8h20, Jour : 2006-08-23, Médecin : Mme Marie PELISSIER

Client

8h0-8h20		<a href="#">Réserver</a>
9h20-9h40		<a href="#">Réserver</a>
9h40-10h0		<a href="#">Réserver</a>
10h0-10h20	Mr Jules JACQUARD	<a href="#">Supprimer</a>
10h20-10h40		<a href="#">Réserver</a>

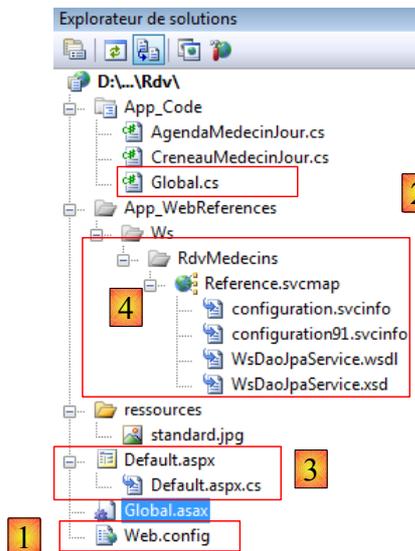
## Cabinet médical - LES MEDECINS ASSOCIES -

### Les erreurs suivantes se sont produites :

- L'erreur suivante s'est produite lors de l'initialisation de l'application :  
System.InvalidOperationException: La réponse n'est pas du code XML correct. --->  
System.Xml.XmlException: Fin de fichier inattendue. Les éléments suivants ne sont pas fermés : message, ns2:cause, ns2:cause, ns2:exception, detail, ns2:Fault, S:Body, S:Envelope.  
Ligne 560, position 86. à System.Xml.XmlTextReaderImpl.Throw(Exception e) à System.Xml.XmlTextReaderImpl.Throw(String res, String arg) à

## 6.2 Le projet Visual Web Developer de la couche [web]

Le projet Visual Web Developer de la couche [web] est le suivant :



- en [1] : le fichier de configuration [Web.config] de l'application.
- en [2] : le fichier [Global.cs] qui gère les événements de l'application web, principalement son démarrage.
- en [3] : le formulaire [Default.aspx] de l'application – contient les différentes vues de l'application.
- en [4] : la référence du service web. On notera que son espace de noms est [Ws.RdvMedecins].

## 6.3 Le contrôleur d'application [Global.cs]

Le fichier [Global.cs] est lié au fichier [Global.asax] et gère les événements de l'application web.

### Global.asax

```
<%@ Application Language="C#" inherits="Web.Global" %>
```

### Global.cs

```
1. using System;
2. using Medecin = Ws.RdvMedecins.medecin;
3. using Client = Ws.RdvMedecins.client;
4. using System.Collections.Generic;
5. using Ws.RdvMedecins;
6.
7. namespace Web
8. {
9.     public class Global : System.Web.HttpApplication
10.    {
11.        // propriétés statiques et automatiques de l'application
12.        public static string Msg { get; set; }
13.        public static WsDaoJpaClient Dao { get; set; }
14.        public static Boolean Erreur { get; set; }
15.        public static Medecin[] Medecins { get; set; }
16.        public static Client[] Clients { get; set; }
17.        public static Dictionary<long, Client> DicoClients { get; set; }
18.        public static Dictionary<long, Medecin> DicoMedecins { get; set; }
19.
20.        // initialisation de l'application
21.        protected void Application_Start(object sender, EventArgs e)
22.        {
23.            // on gère les erreurs de configuration
24.            try
25.            {
26.                // proxy [dao] du service web
27.                Dao = new WsDaoJpaClient();
28.                // dictionnaires
29.                DicoClients = new Dictionary<long, Client>();
30.                DicoMedecins = new Dictionary<long, Medecin>();
31.                // liste des médecins
32.                ...
33.                // liste non vide ?
34.                if (Medecins.Length == 0)
35.                {
36.                    ...
37.                }
38.                // liste des clients
39.                ...
40.                // liste non vide ?
41.                if (Clients.Length == 0)
42.                {
43.                    ...
44.                }
45.                // on crée le dictionnaire des médecins
46.                ...
47.                // on crée le dictionnaire des clients
48.                ...
49.                // on note le succès
50.                Msg = "Initialisation réussie ...";
51.            }
52.            catch (Exception ex)
53.            {
54.                // on note l'erreur
55.                Erreur = true;
56.                Msg = String.Format("L'erreur suivante s'est produite lors de l'initialisation de
l'application : {0}", ex.ToString());
57.            }
58.        }
59.    }
60.
61.    protected void Session_Start(object sender, EventArgs e)
62.    {
```

```

63.     ...
64.     }
65.
66. }
67. }

```

- la classe [Global] (ligne 9) a des propriétés statiques publiques (lignes 12-18) qui seront partagées par tous les utilisateurs
- lignes 13, 27 : instantiation de la couche [dao] (en réalité de la classe proxy de la couche [dao] distante)
- ligne 15 : le tableau de tous les médecins de la table [MEDECINS]
- ligne 16 : le tableau de tous les clients de la table [CLIENTS]
- ligne 17 : un dictionnaire des médecins indexés par leur n°. Il permettra à la couche [web] de retrouver un médecin via son n°.
- ligne 18 : un dictionnaire des clients indexés par leur n°. Il permettra à la couche [web] de retrouver un client via son n°.
- ligne 14 : un booléen indiquant si l'initialisation de l'application web s'est bien passée ou non
- ligne 12 : un message initialisé aux lignes 50 et 56

**Question** : compléter le code de la méthode [Application\_Start] ci-dessus.

## 6.4 La page [Default.aspx]

**Note** : on trouvera dans [ref2], paragraphe 10.5 - page 76, un formulaire analogue à celui étudié ici. On pourra s'en inspirer.

### 6.4.1 Vue d'ensemble

La page [Default.aspx] contient plusieurs composants [View], un pour chaque vue. Son squelette est le suivant :

```

1. <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
2.
3. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4. <html xmlns="http://www.w3.org/1999/xhtml">
5. <head runat="server">
6. <title>Médecins associés</title>
7.
8. <script language="javascript">
9.     function effacerInfos() {
10.         with(document.frmRvMedecins) {
11.             TextBoxJour.value="";
12.             DropDownListMedecins.selectedIndex=0;
13.         } //with
14.     } //effacerInfos
15. </script>
16.
17. </head>
18. <body background="ressources/standard.jpg">
19. <form id="frmRvMedecins" runat="server">
20. <asp:ScriptManager ID="ScriptManager1" runat="server" />
21. <h2>
22.     Cabinet médical - LES MEDECINS ASSOCIES -</h2>
23. <hr/>
24. <asp:MultiView ID="Vues" runat="server">
25. <asp:View ID="ViewSaisies" runat="server">
26. ...
27.     <input type="button" value="Effacer" onclick="effacerInfos()" /></p>
28.
29. </asp:View>
30. <asp:View ID="ViewErreurs" runat="server">
31. ...
32. </asp:View>
33. <asp:View ID="ViewListeRv" runat="server">
34. ...
35. </asp:View>
36. <asp:View ID="ViewChoixClient" runat="server">
37. ...
38. </asp:View>
39. </asp:MultiView>
40. </form>
41. </body>
42. </html>

```

- lignes 21-23 : code ASP en-dehors de tout composant [View]. Il sera présent dans chaque vue dont il formera l'entête.
- lignes 24-38 : un composant [MultiView] avec quatre vues : la vue [ViewSaisies] lignes 25-29, la vue [ViewErreurs] lignes 30-32, la vue [ViewListeRv] lignes 33-35, la vue [ViewChoixClient] lignes 36-38.

## 6.4.2 La vue [ViewSaisies]

Le composant [View] nommé [ViewSaisies] est le suivant :

N°	Type	Nom	Rôle
1	DropDownList	DropDownListMedecins	Contient la liste des médecins
2	TextBox	TextBoxJour	le jour du RV sous la forme JJ/MM/AAAA
3	RequiredFieldValidator	RequiredFieldValidatorJour	vérifie que le champ [2] [TextBoxJour] n'est pas vide
4	RegularExpressionValidator	RegularExpressionValidatorJour	vérifie que le champ [2] [TextBoxJour] est au format JJ/MM/AAAA
5	CustomValidator	CustomValidatorJour	vérifie que le champ [2] [TextBoxJour] contient une date valide
6	Button	ButtonValider	poste le formulaire
7	balise HTML	<input type="button" ...>	lié à du code Javascript (ligne 27 du code source de la page), il efface les saisies

## 6.4.3 La vue [ViewListeRv]

Le composant [View] nommé [ViewListeRv] est le suivant :

N°	Type	Nom	Rôle
1	Button	ButtonAccueil	réaffiche la vue [ViewSaisies]
2	Label	LabelRvMedecinJour	affiche le nom du médecin ainsi que le jour pour lequel on affiche les Rv de ce médecin sous la forme " Rendez-vous de Mme Marie PELISSIER le 23/08/2006 "
3	GridView	GridViewRVMedecinJour	une table qui affiche les Rv d'un médecin donné pour un jour donné. Cette table permet d'ajouter et de supprimer des Rv.

La table des Rv est affichée sous la forme suivante :

## Cabinet médical - LES MEDECINS ASSOCIES -

Accueil

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action	7
8h0-8h20		<a href="#">Réserver</a>	
8h20-4 D	5	<a href="#">Réserver</a>	6
8h40-9h0		<a href="#">Réserver</a>	
9h0-9h20		<a href="#">Réserver</a>	
9h20-9h40		<a href="#">Réserver</a>	
9h40-10h0		<a href="#">Réserver</a>	
10h0-10h20	Mr Jules JACQUARD	<a href="#">Supprimer</a>	
10h20-10h40		<a href="#">Réserver</a>	
10h40-11h0		<a href="#">Réserver</a>	
11h0-11h20		<a href="#">Réserver</a>	
11h20-11h40	Mme Christine GERMAN	<a href="#">Supprimer</a>	

N°	Propriétés
----	------------

4 Type : **BoundField**, HeaderText : **Créneau horaire**, DataField : **CreneauHoraire**

5 Type : **BoundField**, HeaderText : **Client**, DataField : **Client**

6 Type : **ButtonField**, HeaderText : **Action**, DataTextField : **Commande**, CausesValidation : **false**,

Les champs [CreneauHoraire, Client, Commande, IdCreneau] sont ceux de la classe [CreneauMedecinJour] suivante :

```
1. using Rv = Ws.RdvMedecins.rv;
2.
3. namespace Web
4. {
5.     public class CreneauMedecinJour
6.     {
7.         // propriétés publiques automatiques
8.         public long IdCreneau {get;set;}
9.         public string CreneauHoraire {get;set;}
10.        public string Client {get;set;}
11.        public string Commande {get;set;}
12.        public Rv Rv {get;set;}
13.    }
14. }
```

- ligne 8 : le n° du créneau tel que défini dans la table [CRENEAUX] de la base de données
- ligne 9 : heure de début – heure de fin du créneau sous la forme "8h20-8h40"
- ligne 10 : identité du client sous la forme "Mr Alain Dupont" ou "Mme Sylvie Gabono"
- ligne 11 : action à faire sur le Rv : "Réserver" ou "Supprimer" selon les cas
- ligne 12 : si le créneau est réservé, l'instance [Rv] correspondante. Elle permet de retrouver le n° du Rv à supprimer dans la table [RV] de la base de données.

Le GridView [GridViewRVMedecinJour] sera associé à une collection d'objets de type [CreneauMedecinJour].

### 6.4.4 La vue [ViewChoixClient]

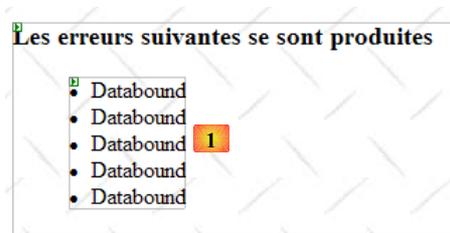
Le composant [View] nommé [ViewChoixClient] est le suivant :



N°	Type	Nom	Rôle
1	Label	LabelIdRv	un texte précisant le nom du médecin, le créneau horaire et le jour du Rv
1	DropDownList	DropDownListClients	Contient la liste des clients enregistrés dans la table [CLIENTS] de la base
3	Button	ButtonValider	poste le formulaire
4	Button	ButtonAnnuler	revient à la vue [ViewListeRv]

### 6.4.5 La vue [ViewErreurs]

Le composant [View] nommé [ViewErreurs] est le suivant :



N°	Type	Nom	Rôle
1	Repeater	RptErreurs	affiche une liste de messages d'erreur

Le composant [Repeater] permet de répéter un code ASP / HTML pour chaque objet d'une source de données, généralement une collection. Ce code est défini directement dans le code source ASP de la page :

```

1.     <asp:Repeater ID="RptErreurs" runat="server">
2.         <ItemTemplate>
3.             <li>
4.                 <%=# Container.DataItem %>
5.             </li>
6.         </ItemTemplate>
7.     </asp:Repeater>

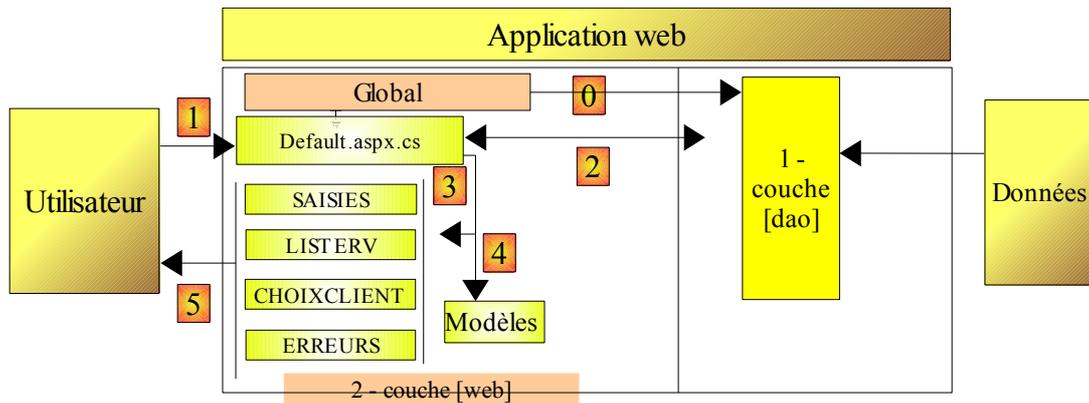
```

- ligne 2 : <ItemTemplate> définit le code qui sera répété pour chaque élément de la source de données.
- ligne 4 : affiche la valeur de l'expression *Container.DataItem* qui désigne l'élément courant de la source de données. Cet élément étant un objet, c'est la méthode *ToString* de cet objet qui est utilisée pour inclure celui-ci dans le flux HTML de la page. Notre collection d'objets sera une collection *List<String>* contenant des messages d'erreur. Les lignes 3-5 inclueront des séquences *<li>Message</li>* dans le flux HTML de la page.

## 6.5 Le contrôleur [Default.aspx.cs]

### 6.5.1 Vue d'ensemble

Revenons à l'architecture MVC de l'application :



- [Default.aspx.cs] qui est le code de contrôle de la page unique [Default.aspx] est le contrôleur de l'application.
- [Global] est l'objet de type [HttpApplication] qui initialise l'application et qui dispose d'une référence sur la couche [dao].

Le squelette du code du contrôleur [Default.aspx.cs] est le suivant :

```

1. using System;
2. using System.Web.UI.WebControls;
3. using Client = Ws.RdvMedecins.client;
4. using Medecin = Ws.RdvMedecins.medecin;
5. using Creneau = Ws.RdvMedecins.creneau;
6. using Rv = Ws.RdvMedecins.rv;
7. using System.Collections.Generic;
8. using Web;
9.
10. public partial class _Default : System.Web.UI.Page
11. {
12.     // données globales
13.     private string dateJour;
14.
15.     // chargement de la page
16.     protected void Page_Load(object sender, EventArgs e)
17.     {
18.         // traitement requête initiale
19.         if (!IsPostBack)
20.         {
21.             // des erreurs d'initialisation ?
22.             if ...
23.             {
24.                 // affichage vue [erreurs]
25.                 ...
26.                 return;
27.             }
28.             // chargement des noms des médecins dans le combo
29.             ...
30.             // chargement des noms des clients dans le deuxième combo
31.             ...
32.             // affichage vue [saisie]
33.             ...
34.         }
35.     }
36.
37.     protected void CustomValidatorJour_ServerValidate(object source, ServerValidateEventArgs args)
38.     {
39.         ...
40.     }
41.
42.     protected void ButtonValider_Click(object sender, EventArgs e)
43.     {
44.         // données valides ?
45.         Page.Validate();
46.         if (!Page.IsValid)
47.         {
48.             // on renvoie la page
49.             return;
50.         }
51.         // on affiche la liste des Rv du médecin
52.         ...
53.         try

```

```

54.     {
55.         ...
56.         // vue Agenda
57.         Vues.ActiveViewIndex = 2;
58.     }
59.     catch (Exception ex)
60.     {
61.         ...
62.         // affichage vue [erreurs]
63.         ...
64.         // on quitte
65.         return;
66.     }
67. }
68.
69. protected void ButtonAccueil_Click(object sender, EventArgs e)
70. {
71.     // retour à la page d'accueil
72.     Vues.ActiveViewIndex = 0;
73. }
74.
75.
76. protected void ButtonvaliderRv_Click(object sender, EventArgs e)
77. {
78.     // ajout d'un Rv
79.     ...
80. }
81.
82. protected void ButtonAnnulerRv_Click(object sender, EventArgs e)
83. {
84.     // annulation - on revient à l'agenda du médecin
85.     ...
86. }
87.
88. protected void GridViewRVMedecinJour_RowCommand(object sender, GridViewCommandEventArgs e)
89. {
90.     // quelle est la ligne de l'agenda concernée ?
91.     long iCreneau = long.Parse(e.CommandArgument.ToString());
92.     // on récupère le type de la commande
93.     ...
94.     // exécution commande
95.     if (commande == "Réserver")
96.     {
97.         ...
98.         // affichage ViewChoixRvPourClient
99.         Vues.ActiveViewIndex = 3;
100.        return;
101.    }
102.    if (commande == "Supprimer")
103.    {
104.        try
105.        {
106.            // suppression Rv
107.            ...
108.        }
109.        catch (Exception ex)
110.        {
111.            ...
112.            // affichage vue [erreurs]
113.            ...
114.            // on quitte
115.            return;
116.        }
117.
118.        // agenda du médecin le jour DateJour
119.        ...
120.        // on le remet dans la session de l'utilisateur
121.        ...
122.        // régénération du Grid
123.        ...
124.    }
125. }
126. }

```

A la requête initiale (GET) de l'utilisateur, seul l'événement *Load* des lignes 16-35 est traité. Aux requêtes suivantes (POST) faites via les boutons du formulaire, deux événements sont traités :

1. l'événement *Load* (16-35)

- l'événement lié au bouton qui a été cliqué

## 6.5.2 L'événement [Page\_Load]

La première vue présentée à l'utilisateur est la suivante :

**Cabinet médical - LES MEDECINS ASSOCIES -**

---

Prise de rendez-vous :

Médecin

Jour (JJ/MM/AAAA)

L'initialisation de l'application nécessite l'accès à un service web qui peut échouer. Dans ce cas, la première page est une page d'erreurs :

## Cabinet médical - LES MEDECINS ASSOCIES -

---

### Les erreurs suivantes se sont produites :

- L'erreur suivante s'est produite lors de l'initialisation de l'application :  
System.InvalidOperationException: La réponse n'est pas du code XML correct. --->  
System.Xml.XmlException: Fin de fichier inattendue. Les éléments suivants ne sont pas fermés : message, ns2:cause, ns2:cause, ns2:exception, detail, ns2:Fault, S:Body, S:Envelope.  
Ligne 560, position 86. à System.Xml.XmlTextReaderImpl.Throw(Exception e) à System.Xml.XmlTextReaderImpl.Throw(String res, String arg) à

---

**Question** : compléter le code de la procédure Page\_Load du contrôleur [Default.aspx.cs] de la page 49.

---

## 6.5.3 L'événement [ButtonValider\_Click]

## Cabinet médical - LES MEDECINS AS

---

Prise de rendez-vous :

Médecin

Jour (JJ/MM/AAAA)

L'utilisateur a sélectionné un médecin et a saisi un jour de RV

## Cabinet médical - LES MEDECINS ASSOCIES -

---

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action
8h0-8h20		<a href="#">Réserver</a>
8h20-8h40		<a href="#">Réserver</a>
8h40-9h0		<a href="#">Réserver</a>
9h0-9h20		<a href="#">Réserver</a>
9h20-9h40		<a href="#">Réserver</a>
9h40-10h0		<a href="#">Réserver</a>
10h0-10h20	Mr Jules JACQUARD	<a href="#">Supprimer</a>
10h20-10h40		<a href="#">Réserver</a>
10h40-11h0		<a href="#">Réserver</a>
11h0-11h20		<a href="#">Réserver</a>
11h20-11h40	Mme Christine GERMAN	<a href="#">Supprimer</a>

On obtient la liste (vue partielle ici) des RV du médecin sélectionné pour le jour indiqué.

**Question :** compléter le code de la procédure [ButtonValider\_Click] de la page 49.

**Conseils :**

1. une exception peut survenir puisqu'ici le service web distant est sollicité.
2. certaines informations sont à mettre dans la session pour les requêtes suivantes (médecin, jour du Rv)
3. la procédure doit construire la collection d'objets de type [CreneauMedecinJour] qui devra être associée au GridView [GridViewRVMedecinJour] de la vue [ViewListeRv].
4. pour se conformer aux deux points précédents, on pourra utiliser la classe [AgendaMedecinJour] suivante :

```

5. using System;
6.
7. namespace Web
8. {
9.     public class AgendaMedecinJour
10.    {
11.        // propriétés publiques automatiques
12.        public long IdMedecin {get;set;}
13.        public String DateJour {get;set;}
14.        public CreneauMedecinJour[] Creneaux;
15.    }
16. }

```

- ligne 12 : le n° du médecin sélectionné dans la vue [ViewSaisies]
- ligne 13 : la date du jour saisie dans la vue [ViewSaisies].
- ligne 14 : avec les champs des lignes 12 et 13, on peut obtenir le tableau d'objets de type [CreneauMedecinJour] qui devra être associée au GridView [GridViewRVMedecinJour] de la vue [ViewListeRv].
  - la méthode [getCreneauxMedecin] de la couche [dao] permet de connaître les créneaux horaires du médecin identifié par le champ de la ligne 12
  - la méthode [getRvMedecinJour] de la couche [dao] permet de connaître les créneaux réservés pour le médecin de la ligne 12 et la date de la ligne 13
  - on peut alors savoir si un créneau est libre (absence de Rv) ou occupé (présence d'un Rv).
- une fois l'objet [AgendaMedecinJour] construit, **on le mettra dans la session** afin d'y avoir accès dans les requêtes suivantes.

## 6.5.4 L'événement [GridViewRVMedecinJour\_RowCommand]

Cet événement se produit lorsqu'on clique sur les liens [Réserver] ou [Supprimer] de la liste des Rv d'un médecin :

Accueil

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action
8h0-8h20		<a href="#">Réserver</a>
8h20-8h40		<a href="#">Réserver</a>
8h40-9h0		<a href="#">Réserver</a>

On réserve

**Cabinet médical - LES MEDECINS ASSOCIES -**

Rendez-vous : 8h0-8h20, Jour : 2006-08-23, Médecin : Mme Marie PELISSIER

Client :

On obtient une page à renseigner

**Cabinet médical - LES MEDECINS AS**

Accueil

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action
8h0-8h20	Melle Brigitte BISTROU	<a href="#">Supprimer</a>
8h20-8h40		<a href="#">Réserver</a>

L'utilisateur peut supprimer un RV

**Cabinet médical - LES MEDECINS .**

Accueil

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action
8h0-8h20		<a href="#">Réserver</a>
8h20-8h40		<a href="#">Réserver</a>

Le RV supprimé a disparu de la liste des RV

**Question :** compléter le code de la procédure [GridViewRVMedecinJour\_RowCommand] de la page 49.

**Note :** une exception peut survenir dans le cas de la suppression d'un Rv puisqu'alors le service web distant est sollicité.

## 6.5.5 L'événement [ButtonvaliderRv\_Click]

Cet événement se produit lors du choix d'un client pour un Rv donné :

**Cabinet médical - LES MEDECINS ASSOCIES -**

Rendez-vous : 8h0-8h20, Jour : 2006-08-23, Médecin : Mme Marie PELISSIER

Client :

On la renseigne

**Cabinet médical - LES MEDECINS ASSOCIES -**

Accueil

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

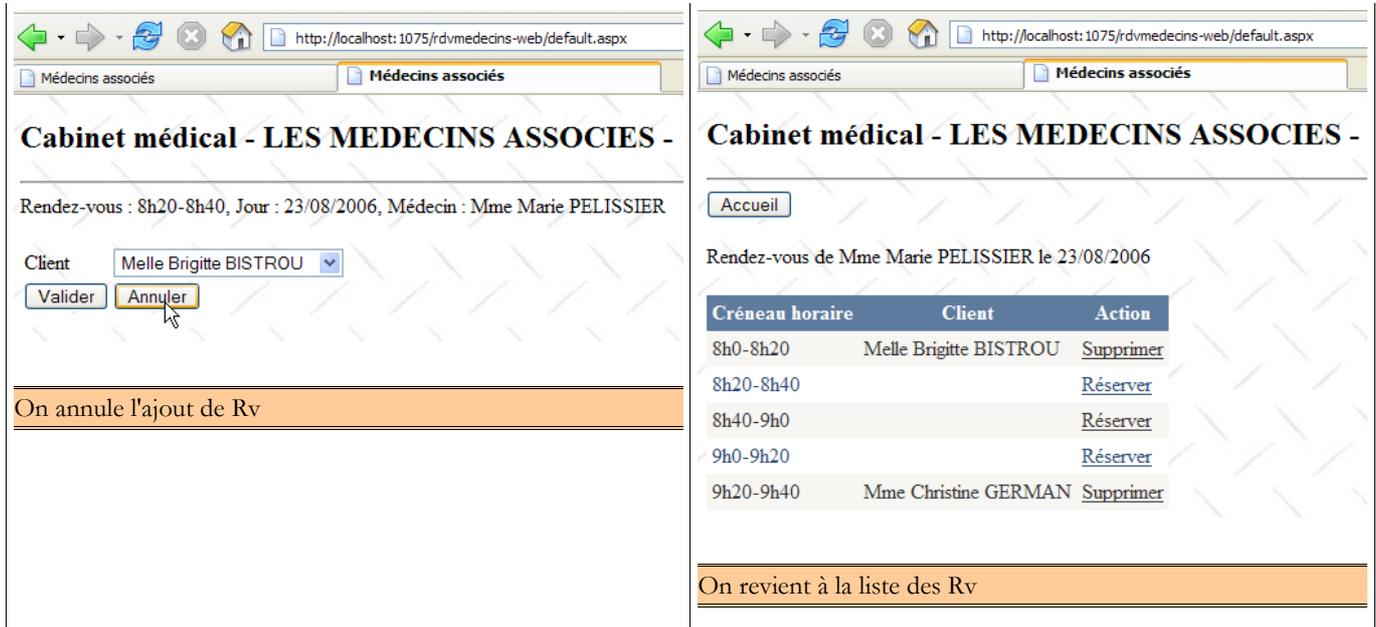
Créneau horaire	Client	Action
8h0-8h20	Melle Brigitte BISTROU	<a href="#">Supprimer</a>
8h20-8h40		<a href="#">Réserver</a>
8h40-9h0		<a href="#">Réserver</a>

Le nouveau RV apparaît dans la liste

**Question** : compléter le code de la procédure [ButtonvaliderRv\_Click] de la page 49.

**Note** : une exception peut survenir puisqu'ici le service web distant est sollicité.

### 6.5.6 L'événement [ButtonAnnulerRv\_Click]



**Question** : compléter le code de la procédure [ButtonAnnulerRv\_Click] de la page 49.

### 6.5.7 L'événement [ButtonAccueil\_Click]

Cet événement se produit lorsque l'utilisateur veut revenir à la page d'accueil à partir de la vue [ViewListeRv] :



**Question** : compléter le code de la procédure [ButtonAccueil\_Click] de la page 49.

## 6.6 Ajaxification de l'application web

---

**Question** : Ajaxifier l'application finale à l'aide d'un composant **UpdatePanel**.

---

**Note** : on pourra suivre la méthode exposée dans [ref2] au paragraphe 7, page 32.

# Table des matières

<b>1</b>	<b>OBJECTIF DU TP.....</b>	<b>1</b>
<b>2</b>	<b>LE PROBLÈME.....</b>	<b>1</b>
<b>3</b>	<b>FUNCTIONNEMENT DE L'APPLICATION.....</b>	<b>2</b>
<b>4</b>	<b>LE SERVICE WEB J2EE DES RENDEZ-VOUS.....</b>	<b>5</b>
<b>4.1</b>	<b>LA BASE DE DONNÉES.....</b>	<b>5</b>
<b>4.1.1</b>	<b>LA TABLE [MEDECINS].....</b>	<b>5</b>
<b>4.1.2</b>	<b>LA TABLE [CLIENTS].....</b>	<b>6</b>
<b>4.1.3</b>	<b>LA TABLE [CRENEAUX].....</b>	<b>6</b>
<b>4.1.4</b>	<b>LA TABLE [RV].....</b>	<b>7</b>
<b>4.2</b>	<b>LES ÉLÉMENTS DE L'ARCHITECTURE CÔTÉ SERVEUR.....</b>	<b>7</b>
<b>4.3</b>	<b>LE PROJET NETBEANS DU MODULE EJB.....</b>	<b>8</b>
<b>4.4</b>	<b>LES ENTITÉS DE LA COUCHE JPA.....</b>	<b>9</b>
<b>4.5</b>	<b>CONFIGURATION DE LA COUCHE JPA.....</b>	<b>12</b>
<b>4.6</b>	<b>GÉNÉRATION DE LA BASE DE DONNÉES.....</b>	<b>13</b>
<b>4.7</b>	<b>LA CLASSE D'EXCEPTION.....</b>	<b>14</b>
<b>4.8</b>	<b>L'EJB DE LA COUCHE [DAO].....</b>	<b>16</b>
<b>4.9</b>	<b>DÉPLOIEMENT DE L'ARCHIVE DE L'EJB DE LA COUCHE [DAO].....</b>	<b>18</b>
<b>4.10</b>	<b>TESTS DE L'EJB DE LA COUCHE [DAO].....</b>	<b>22</b>
<b>4.11</b>	<b>LE SERVICE WEB DE LA COUCHE [DAO].....</b>	<b>24</b>
<b>4.12</b>	<b>DÉPLOIEMENT DU SERVICE WEB DE L'APPLICATION.....</b>	<b>27</b>
<b>4.13</b>	<b>TESTS JUNIT DU SERVICE WEB.....</b>	<b>30</b>
<b>5</b>	<b>CLIENTS .NET DU SERVICE J2EE DES RENDEZ-VOUS.....</b>	<b>35</b>
<b>5.1</b>	<b>UN CLIENT C# 2008.....</b>	<b>35</b>
<b>5.2</b>	<b>UN CLIENT ASP.NET 2008.....</b>	<b>38</b>
<b>6</b>	<b>CLIENT ASP.NET DE GESTION DES RENDEZ-VOUS.....</b>	<b>41</b>
<b>6.1</b>	<b>LES VUES DE L'APPLICATION.....</b>	<b>42</b>
<b>6.2</b>	<b>LE PROJET VISUAL WEB DEVELOPER DE LA COUCHE [WEB].....</b>	<b>43</b>
<b>6.3</b>	<b>LE CONTRÔLEUR D'APPLICATION [GLOBAL.CS].....</b>	<b>44</b>
<b>6.4</b>	<b>LA PAGE [DEFAULT.ASPX].....</b>	<b>45</b>
<b>6.4.1</b>	<b>VUE D'ENSEMBLE.....</b>	<b>45</b>
<b>6.4.2</b>	<b>LA VUE [VIEWSAISIES].....</b>	<b>46</b>
<b>6.4.3</b>	<b>LA VUE [VIEWLISTERV].....</b>	<b>46</b>
<b>6.4.4</b>	<b>LA VUE [VIEWCHOIXCLIENT].....</b>	<b>47</b>
<b>6.4.5</b>	<b>LA VUE [VIEWERREURS].....</b>	<b>48</b>
<b>6.5</b>	<b>LE CONTRÔLEUR [DEFAULT.ASPX.CS].....</b>	<b>48</b>
<b>6.5.1</b>	<b>VUE D'ENSEMBLE.....</b>	<b>48</b>
<b>6.5.2</b>	<b>L'ÉVÈNEMENT [PAGE_LOAD].....</b>	<b>51</b>
<b>6.5.3</b>	<b>L'ÉVÈNEMENT [BUTTONVALIDER_CLICK].....</b>	<b>51</b>
<b>6.5.4</b>	<b>L'ÉVÈNEMENT [GRIDVIEWRVMEDECINJOUR_ROWCOMMAND].....</b>	<b>53</b>
<b>6.5.5</b>	<b>L'ÉVÈNEMENT [BUTTONVALIDERV_CLICK].....</b>	<b>53</b>
<b>6.5.6</b>	<b>L'ÉVÈNEMENT [BUTTONANNULERV_CLICK].....</b>	<b>54</b>
<b>6.5.7</b>	<b>L'ÉVÈNEMENT [BUTTONACCUEIL_CLICK].....</b>	<b>54</b>
<b>6.6</b>	<b>AJAXIFICATION DE L'APPLICATION WEB.....</b>	<b>54</b>