Projet base de données

Traitement des emplois du temps du département informatique
Enseignant responsable : Claude Sabatier

Lionel Dromer Éloi Perdereau

11 mai 2013

Table des matières

1 Cahier des charges

1.1 Étude de l'existant

Nous ne pouvons avoir qu'un point de vue de l'utilisateur final, mais en étudiant différents modèles d'emploi du temps, nous avons pu relever des fonctionnalités récurrentes, avec des subtilités :

- Deux sens possibles d'affichage des jours : vertical ou horizontal.
- Afficher l'emploi du temps de promotions, d'enseignants ou de salles. Possibilité de les combiner.
- Choisir la période.
- Différents formats d'exportation (PDF, impression, ...).
- Modification de l'emploi du temps par un administrateur.
- Couleurs par UE [1].
- Libelle d'un créneau adapté selon l'affichage.
- Version mobile disponible.

En voici deux exemples :

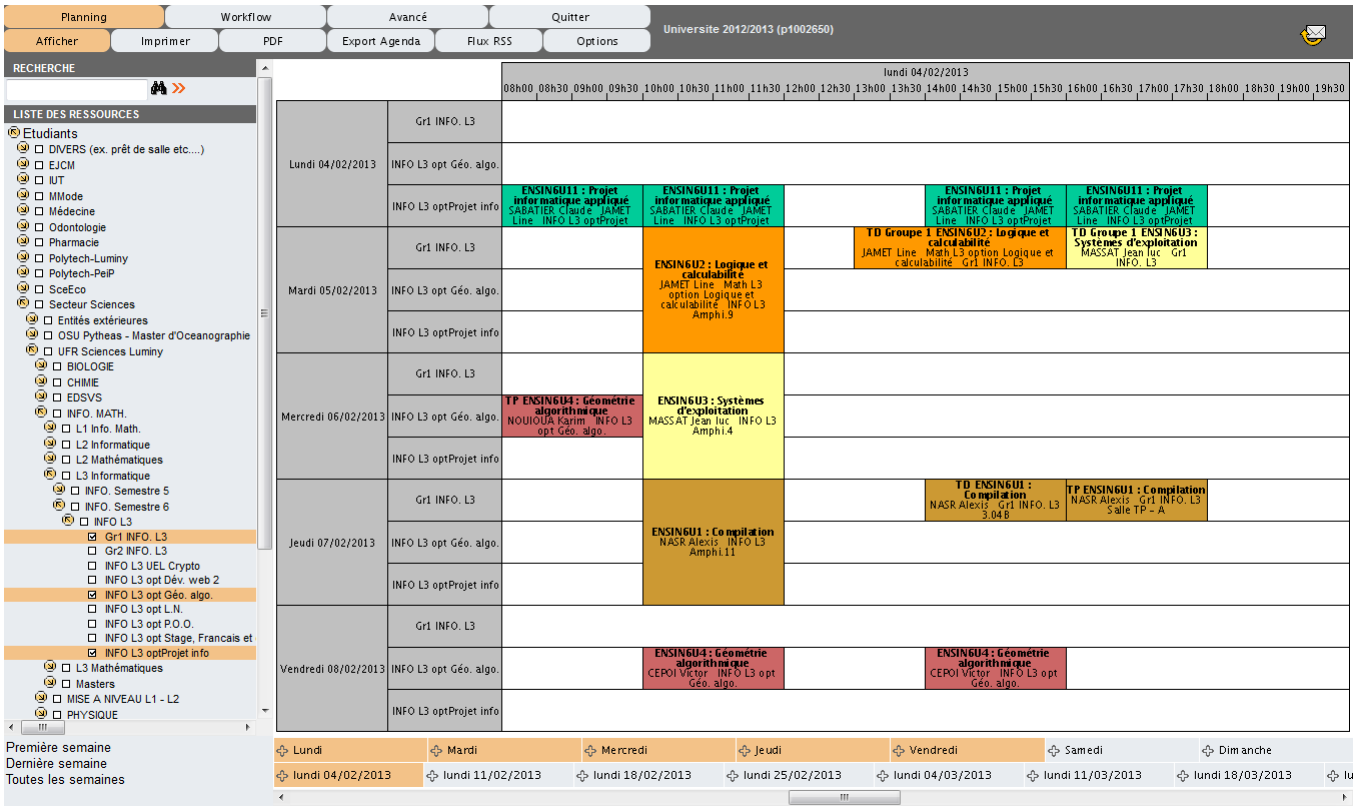


FIGURE 1 – ADE : Emploi du temps de l'AMU

1. UE = Union d'Enseignements

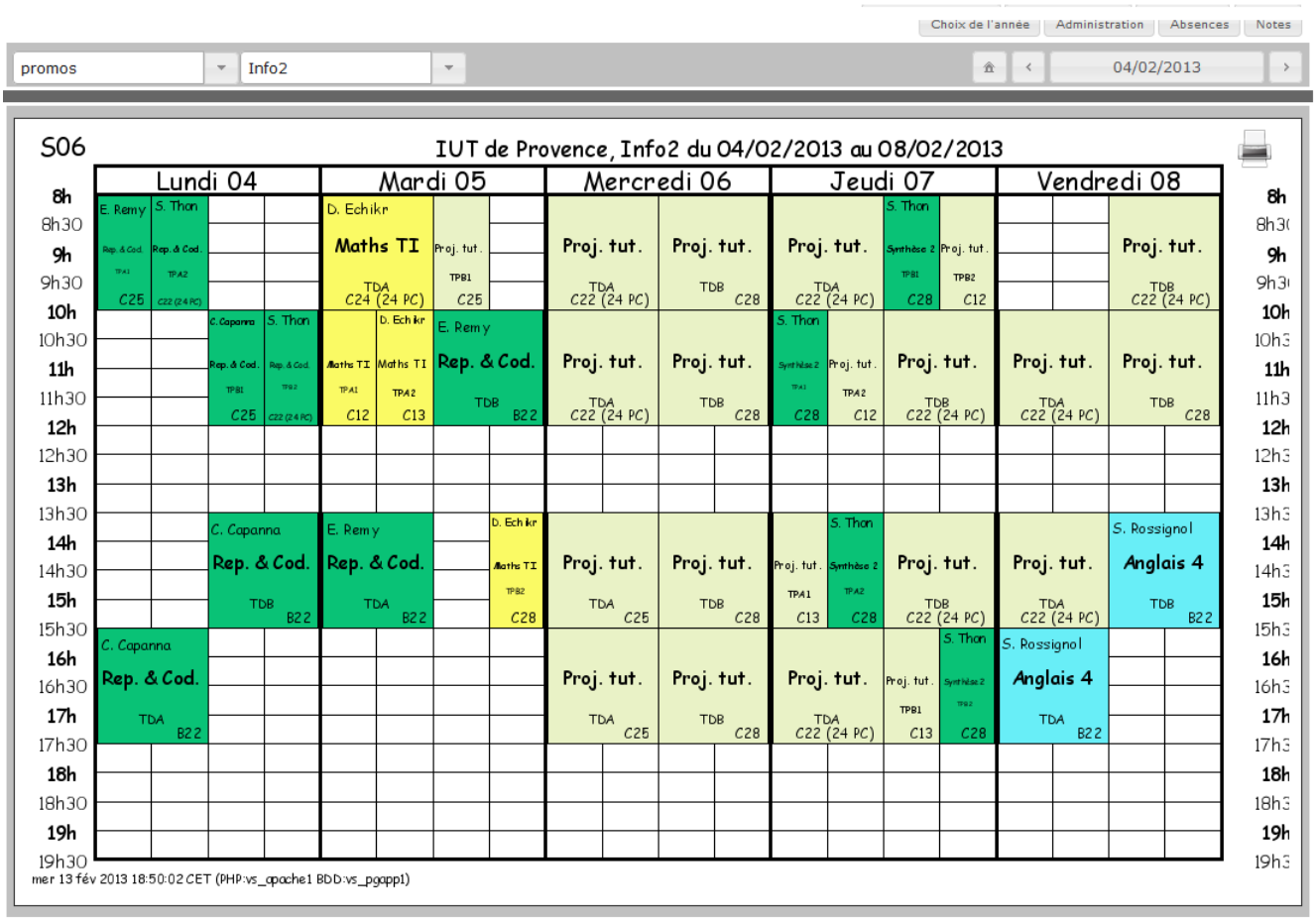


FIGURE 2 – Emploi du temps de l'IUT d'Arles

1.2 Étude des besoins

Afin de répondre à la problématique posée, on a pu relever les fonctionnalités suivantes :

- Vérifier la cohérence de toutes les horaires, au niveau des enseignants, des étudiants et des salles.
- Modification de l'emploi du temps : déplacer une séance. Vérifier la disponibilité des participants (étudiants, enseignant, salle) avant de l'enregistrer.
- Placement des séances selon un programme pédagogique donné.
- Éditer des récapitulatifs lisibles (couleurs et libellés adaptés) :
  - Emploi du temps d'une promotion donnée (tout groupes de TD confondus ou d'un groupe donné), pour une période donnée.
  - Emploi du temps d'un enseignant donné, pour une période donnée.
  - Occupation d'une salle donnée pour une période donnée.
  - Nombre total d'heures d'enseignement reçues par un étudiant d'une promotion donnée.
  - Nombre d'heures de Cours, TD et TP assurés par un enseignant donné (et les équivalents TD correspondant) pour toutes les UE et toutes les promotions.

1.3 Étude de faisabilité (outils)

Technologies à notre disposition :
- Langage Java (Java SE 6) : Très polyvalent, de plus nous l'avons beaucoup pratiqué depuis le début de notre formation.
- SGBDR Oracle : Égelement beaucoup utilisé, il est parfaitement adapté pour ce projet en offrant une très grande configuration, et une simplicité d'écriture de déclencheurs.
- JPA (*Java Persistence API*) : Cette API nous permet de faire le lien entre Java et le SGBD via un mapping des classes. Elle permet une grande abstraction en ne manipulant que des objets, de plus nous utilisons des DAO pour simplifier le code lorsqu'on interagit avec le SGBD.

Gestion de projet :
- Maven : Outil pour la gestion et l'automatisation de production des projets logiciels Java. L'utilité principale dont nous en faisons est la gestion de dépendances et l'empaquetage du projet en fichier jar exécutable.
- Git : Gestionnaire de version décentralisé nous permettant de partager et de versionner le code.
- Eclipse : IDE (*Integrated Development Environment*) très puissant et particulièrement adapté pour des projets Java. Grâce à son extensibilité, nous pouvons utiliser Maven via un plugin sur les machines du département de l'université.
- Équipe de deux développeurs.
- Gestion de projet Gantt.

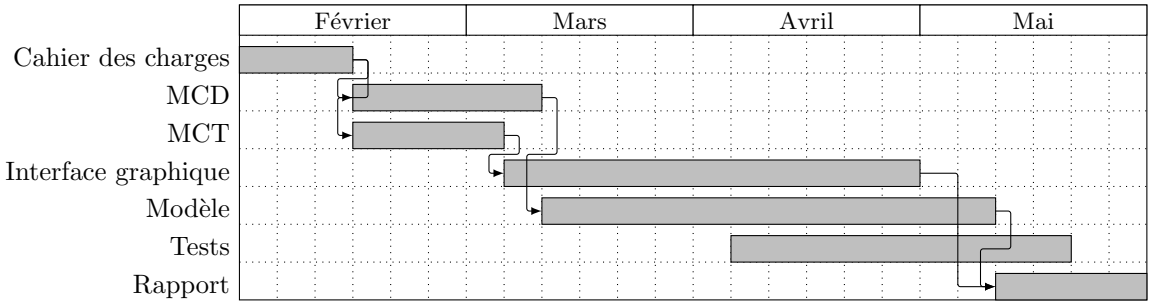


Figure 3 – Diagramme de Gantt

1.4 Analyse préalable

1.4.1 Scénarios

La structure pédagogique est fixe. Cela veux dire que les UE et leurs spécificités sont déjà présentes dans la base. Aucune interface n'est prévu pour modifier ces données. C'est à dire que seul un accès direct à la base de donnée permet la modification de la structure pédagogique.

Du point de vue de l'utilisateur final :
Il y a deux types d'utilisateurs finaux : les enseignants et le public. Le public ne peut que consulter l'emploi tu temps. Les enseignants, ont également le droit de consulter l'état de leur service. Certains enseignants peuvent être gestionnaire (cf. paragraphe suivant). Lorsqu'un utilisateur veut consulter l'emploi du temps, il doit d'abord dérouler un arbre dans lequel il sélectionne les différentes UE à afficher. Par salle, par année et groupe ou par enseignant.

Du point de vue des gestionnaires de l'emploi du temps :
Il y a plusieurs gestionnaires de l'emploi du temps ; ils travailleront de manière concurrente. On distingue en distingue deux types avec des droits spécifiques :

- Le gestionnaire d'année : il peut placer les séances auquel participerons les étudiants de l'année (des années) dont le gestionnaire à les droits.
- Le gestionnaire des professeurs : il peut modifier le nombre d'heures de services effectués par tous les enseignants, ainsi que leur ajouter une période d'indisponibilité.

Un gestionnaire peut faire les mêmes actions qu'un utilisateur classique. Lorsqu'il veut effectuer des tâches administratives, il doit se connecter via un bouton. Dès lors, une zone contenant différents boutons se présente à lui. Ces boutons affichent des panneaux dans lesquels le gestionnaire peut effectuer des tâches administratives.

- Pour un gestionnaire d'année(s) : La zone contient un bouton par année gérée. Chacun affiche un panneau contenant les cours/TD/TP/heures de projet restant à placer (pour l'année en question), avec le nombre d'heures associés. Le panneau contient également des champs à remplir pour placer une séance (date de début, durée). Lorsqu'il veux placer un cours, il sélectionne le cours en question dans le panneau. Une fois le cours sélectionné et les champs remplis, un bouton valider calcule si tous les participants sont disponibles (étudiants, enseignants, salle). Si oui, une salle est attribuée automatiquement et une boîte de dialogue s'affiche permettant au gestionnaire de choisir le professeur à affecter. Ensuite le créneau est placé dans l'emploi du temps. Si les étudiants ne sont pas disponibles où qu'aucune salle n'est libre, un message d'erreur averti le gestionnaire avec le motif de refus. Les champs précédemment remplis pourront être modifiés en conséquence. Un créneau est compris entre 8 heures et 20 heures du lundi au vendredi.

- Pour un gestionnaire d'enseignant : Le panneau se divise en deux avec une liste déroulante contenant la liste des enseignants en en-tête. Une partie lui montre un formulaire pour ajouter une période d'indisponibilité. Cet ajout peut être refusé si l'enseignant en question a déjà un cours de prévu dans le créneau indiqué. L'autre partie lui permet de modifier le nombre d'heures d'administrations effectués.

Un gestionnaire peut être à la fois gestionnaire de plusieurs années et gestionnaire d'enseignant en même temps.

1.4.2 Cas particuliers

- Avertissement lorsqu'une séance est placée dans un créneau peu commode (pause déjeuner, après 18 heures, etc...).
- Avertissement lorsqu'une séance est placée moins de 48 heures avant sa date de début.
- Avertissement si le nombre d'heures de TD, TP ou Projet est différent pour différents groupes d'une même promotion.

2 Analyse détaillée

2.1 Étude des données

2.1.1 Dictionnaire des données

Nom attribut	Description	Contrainte(s)
IdNiveau	Id du niveau d'une promotion (son année)	unique
LibelleNiveau	Libellé du niveau d'une promotion (e.g : 'L1', 'M2')	$\emptyset$
IdUE	Identifiant d'une UE	unique
LibelleUE	Libelle d'une UE	unique
NbHeuresCours	Nombre d'heures de cours concernant une UE	> 0
NbHeuresTD	Nombre d'heures de TD concernant une UE	> 0
NbHeuresTP	Nombre d'heures de TP concernant une UE	> 0
NbHeuresProjet	Nombre d'heures de Projet concernant une UE	≥ 0
IdPromo	Identifiant d'une promotion	unique
EffectifPromo	Effectif d'une promotion	> 0
IdGroupe	Identifiant d'un groupe de TD	unique
EffectifGroupe	Effectif d'un groupe	> 0
NomEns	Nom d'un enseignant	$\emptyset$
RattachementEns	Site de rattachement d'une enseignant	$\in$ {'Departement informatique', 'Exterieur'}
NoTelEns	Numéro de téléphone correspondant du bureau qu'un enseignant occupe	$\emptyset$
MailEns	Adresse e-mail d'un enseignant	$\emptyset$
NbHeuresAdminEns	Nombre d'heures qu'un enseignant a effectué	> 0
IdGrade	Identifiant d'un grade d'un enseignant	unique
LibelleGrade	Libelle d'un grade d'un enseignant	$\emptyset$
DateDebutIndisponibilite	Date de début d'une période d'indisponibilité pour un enseignant	$\emptyset$
DureeIndisponibilite	Durée d'une période d'indisponibilité pour un enseignant	> 0
IdSalle	Identifiant d'une salle	unique
NoSalle	Numéro d'une salle	$\emptyset$
EffectifMax	Effectif maximal d'une salle	> 0
IdTypeSalle	Identifiant d'un type de salle	unique
LibelleTypeSalle	Libelle d'un type de salle (e.g : 'Amphi', 'TD', 'TP').	$\emptyset$
IdTypeSeance	Identifiant du type de séance [2]	unique
LibelleTypeSeance	Libelle du type de séance (e.g : 'Cours', 'TD', 'TP', 'Projet')	$\emptyset$
CoefEquivalentTD	Coefficient d'équivalent TD d'un type de séance	> 0
DateDebutSeance	Date de début d'une séance	$\emptyset$
DureeSeance	Durée d'une séance	> 0 ; ≤ 4
IdGest	Identifiant numérique d'un gestionnaire	unique
LoginGest	Identifiant servant à un gestionnaire d'emploi du temps pour se connecter	$\emptyset$
PwGest	Mot de passe servant à un gestionnaire d'emploi du temps pour se connecter	$\emptyset$
IsGestEns	Indique si l'utilisateur est un gestionnaire d'enseignant ou non	$\emptyset$

2.1.2 Modèle Conceptuel des Données

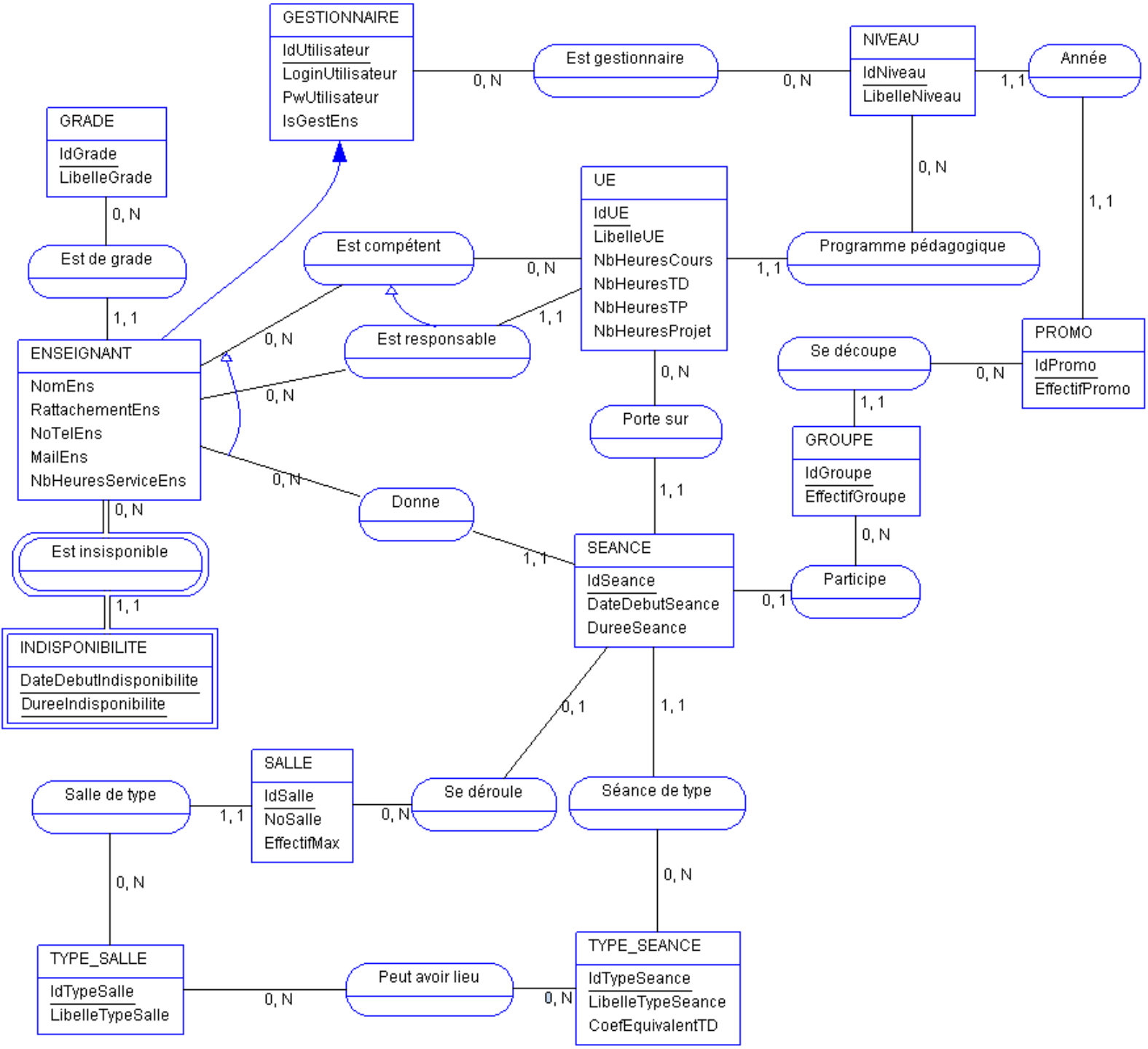


FIGURE 4 – MCD

On conviendra que si aucun groupe ne participe à une séance, alors c'est toute la promotion du niveau de l'UE qui y participe.

2.1.3 Contraintes

- Un enseignant ne peut participer à une séance uniquement s'il ne participe pas déjà à une séance dont la période chevauche celle de la séance en question. De plus, il ne doit pas être indisponible durant la période de la séance.
- Une promotion ne peut participer à une séance uniquement si elle ne participe pas déjà à séance dont la période chevauche celle de la séance en question. Idem pour un groupe.
- Une salle ne peut être affecté à une séance uniquement si elle n'est pas déjà affecté à une séance dont la période chevauche celle de la séance en question.
- Une séance ne peut avoir lieu uniquement dans une salle qui a un effectif plus faible que l'effectif du groupe ou de la promotion qui participe à la séance.
- Une séance ne peut avoir lieu uniquement dans une salle dont le type est compatible à celui de la séance.
- Une séance ne peut pas durer plus de 4 heures.
- Aucune séance ne doit avoir lieu avant 8 heures, après 20 heures ou le week-end.
- La somme des durées des séances pour une UE donnée et un type de séance donné ne doit pas dépasser le nombre d'heures prévu pour cette UE pour ce type de séance.
- La date de début d'une séance ajoutée ou modifiée doit être postérieur au présent. Idem pour la date de début d'une indisponibilité.
- Une séance ne peut être retirée que par un utilisateur qui possède le droit d'administration pour le niveau de l'UE de la séance concernée.
- L'effectif de l'ensemble des groupes d'une promotion doit être égal à l'effectif de la promotion concernée.
- Une promotion (ou un groupe) ne peut participer uniquement à des séances de son niveau.
- Un groupe ne peut pas participer à une séance de type 'Cours'.
- Une période d'indisponibilité ne peut être ajoutée (ou modifiée) si l'enseignant concerné participe à une séance chevauchant la période sus-dite.
- Si un individu est connecté et qu'il n'est pas gestionnaire, c'est un enseignant. En effet, seulement trois types d'individus pourront se connecter : les enseignants, les gestionnaires d'années et les gestionnaires d'enseignants. Malheureusement, cette contrainte ne pourra pas être vérifiée en base. Prenons un exemple : nous voulons ajouter un nouveau gestionnaire d'année. Il faut d'abord insérer le gestionnaire en question avec ses identifiants, puis insérer le (premier) lien qui le lie à une année. Entre ces deux insertions, la contrainte ne sera pas satisfaite. Néanmoins, nous pouvons empêcher un individu de se connecter s'il n'est ni gestionnaire ni enseignant.

2.2 Étude des fonctionnalités de l'application

Les fonctionnalités de l'application se déduisent de l'analyse ci-dessus. Les figures 9, 10 et 11 en annexe présentent le modèle conceptuel des traitements (MCT) qui donne une sorte de manuel utilisateur formel.

3 Programmation et interfaces

Nous faisons ici une application locale en Java (Java SE 6) qui interagira avec une base de données Oracle distante. Nous utilisons la bibliothèque `Swing` pour l'interface graphique, et l'API de persistance `JPA` pour l'interface avec la base de données.

Bien que toute l'analyse ait été faite en Français, l'implémentation sur machine (tables, classes, ...) sera en Anglais.

3.1 Modèle Logique des Données

T_ADMIN	(ID_ADMIN, LOGIN, PW, IS_TEACHER_ADMIN, ADMIN_STATUS)
T_COMPETENT_TEACHER	(ID_TEACHER#, ID_TU#)
T_CROOM	(ID_CROOM, NUM_CROOM, MAX_SIZE, ID_CROOM_TYPE)
T_CROOM_TYPE	(ID_CROOM_TYPE, LIBEL_CROOM_TYPE)
T_GROUP	(ID_GROUP, SIZE_GROUP, ID_PROMO#)
T_LEVEL	(ID_LEVEL, CODE_LEVEL, LIBEL_LEVEL)
T_LEVEL_ADMIN	(ID_ADMIN#, ID_LEVEL#)
T_PROMO	(ID_PROMO, SIZE_PROMO, ID_LEVEL#)
T_RANK	(ID_RANK, LIBEL_RANK)
T_SESSION	(ID_SESSION, SESSION_START_DATE, SESSION_DURATION, ID_CROOM#, ID_GROUP#, ID_SESSION_TYPE#, ID_TU#, ID_TEACHER#)
T_SESSION_CROOM_COMPATIBILITY	(ID_SESSION_TYPE#, ID_CROOM_TYPE#)
T_SESSION_TYPE	(ID_SESSION_TYPE, LIBEL_SESSION_TYPE, TUTOR_WORTH_COEF)
T_TEACHER	(ID_ADMIN, NAME_TEACHER, MAIL_TEACHER, TEL_TEACHER, ADMIN_HOURS, ID_RANK_TEACHER#, ATTACHMENT_SITE)
T_TU	(ID_TU, CODE_TU, LIBEL_TU, COLOR_TU, CLASS_HOURS, TUTOR_HOURS, PRACTICAL_HOURS, PROJECT_HOURS, ID_LEVEL#, ID_RESP_TEACHER#)
T_UNAVAILABILITY	(ID_UNAVAILABILITY, START_UNAV_DATE, UNAV_DURATION, ID_TEACHER#)

Les clés primaires sont soulignés, les clés étrangères sont suivis d'un `#`.

3.2 Fonctions PL/SQL

Elles sont utilisés pour faciliter l'écriture de requêtes complexes et de déclencheurs. Voici la liste des fonctions que nous avons écrites. Leur code est présent en annexe.

- `DATE_OVERLAP` : renvoie `1` si les deux périodes passées en arguments se chevauchent, `0` sinon.
- `CROOM_OCCUPIED` : renvoie `1` si la salle passée en premier argument est occupée durant la période passée en deuxième et troisième argument (date de début et durée), `0` sinon.
- `CROOM_COMPATIBLE` : renvoie `1` si la salle passée en premier arguments est compatible avec le type de séance passé en deuxième argument, `0` sinon.
- `GROUP_AVAILABLE` : renvoie `1` si le groupe passé en premier argument n'a pas de séance prévu durant la période passée en deuxième et troisième arguments (date de début et durée), `0` sinon.
- `TEACHER_IN_CLASS` : renvoie `1` si l'enseignant en premier argument a une séance prévu durant la période passée en deuxième et troisième arguments (date de début et durée), `0` sinon.
- `TEACHER_UNAVAILABLE` : renvoie `1` si l'enseignant passé en premier argument a une indisponibilité prévue durant la période passée en deuxième et troisième arguments (date de début et durée), `0` sinon.
- `TEACHER_BUSY` : renvoie `1` si l'enseignant passé en premier argument est occupé (indisponible ou en classe) pour la période passée en deuxième et troisième arguments (date de début et durée), `0` sinon.
- `TEACHER_SERVICE_HOURS` : renvoie le nombre total d'heures de service effectuées par l'enseignant passé en argument.
- `UNPLANNED_SESSIONS` : renvoie les séances qui n'ont pas encore été planifiées dans l'emploi du temps, mais qui sont prévues par le programme pédagogique. Cette fonction renvoie donc une liste de tuples, dont chacun contient :
  - le libellé de l'UE
  - le libellé 'Cours'
  - le nombre d'heures de cours prévues
  - le libellé 'TD'
  - le nombre d'heures de TD prévues
  - le libellé 'TP'
  - le nombre d'heures de TP prévues
  - le libellé 'Projet'
  - le nombre d'heures de projets prévues.

  Pour y parvenir, nous avons du créé deux types :

```
CREATE OR REPLACE TYPE rec_unplanned_sessions AS OBJECT
      (libel_tu         VARCHAR2(255 BYTE),
       class_libel      VARCHAR2(16),
       class_h_left     NUMBER,
       tutor_libel      VARCHAR2(16),
       tutor_h_left     NUMBER,
       practical_libel  VARCHAR2(16),
       practical_h_left NUMBER,
       project_libel    VARCHAR2(16),
       project_h_left   NUMBER);
```

  et

```
CREATE OR REPLACE TYPE unplanned_sessions_table AS TABLE OF rec_unplanned_sessions;
```

  et faire que la fonction `UNPLANNED_SESSIONS` renvoie un `unplanned_session_table`. Ainsi, on peux récupérer ce résultat en Java sous la forme d'un objet de type `Iterable<Object[]>`.

3.3 Déclencheurs (*triggers*)

Seulement trois déclencheurs ont étés utiles. Leur code est également en annexe.

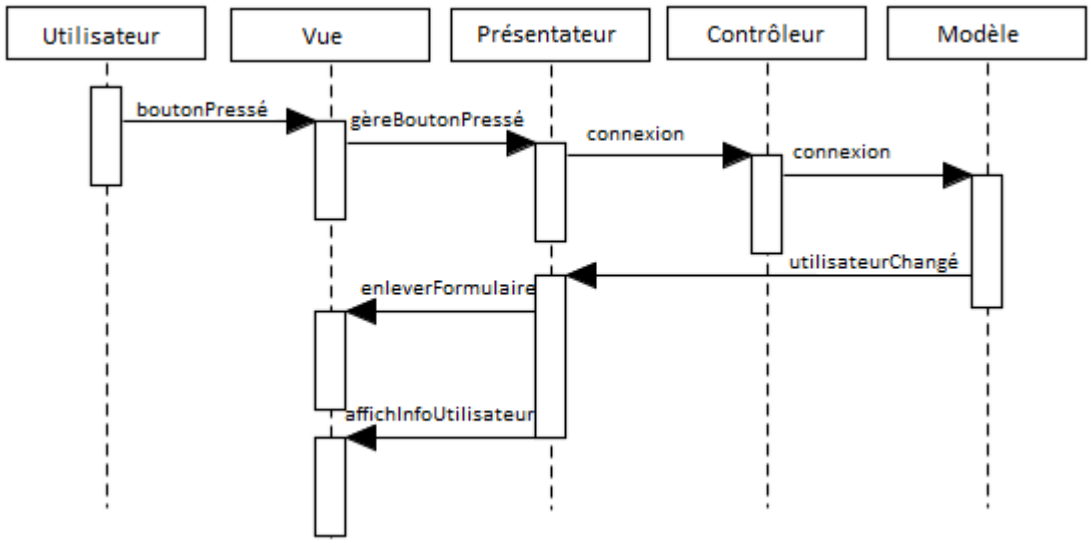
- `SESSION_CONSTRAINTS` pour vérifier les contraints lors de l'ajout ou la modification d'une séance.
- `UNAVAILABILITY_CONSTRAINTS` pour vérifier les contraintes lors de l'ajout ou la modification d'une indisponibilité pour un enseignant.
- `ATTACHMENT_SITE_CONSTRAINT` pour vérifier que le site de rattachement d'un enseignant est bien `Departement informatique` ou `Exterieur` lors de l'ajout ou la modification d'un enseignant.

3.4 Architecture MVPC

Au début de l'implémentation, nous étions parti sur un modèle MVC classique. C'est un pattern architectural qui sépare les données (le modèle), l'interface homme-machine (la vue) et la logique de contrôle (le contrôleur). Or, ce modèle nous oblige à coupler la logique de but utilisateur et celle de la présentation. En effet, les actions et les présentations qui en résultent sont indissociables. Pour pallier à ce défaut, nous utilisons en plus un présentateur qui inclut les deux entités de manière indépendante. C'est le MVCP (Modèle-Vue-Présentateur-Contrôleur). En voici les entités :

- le **Modèle** est un ensemble d'objets qui représentent les données de l'application. Ces objets implémentent la logique métier. Ils ignorent tout de l'interface utilisateur, et donc ne connaissent rien de la vue ou du contrôleur. Ils définissent aussi l'interaction avec la base de données et le traitement de ces données.
- la **Vue** représente l'interface utilisateur, ce avec quoi il interagit. Elle répond à ses actions et en délègue le traitement au présentateur. Elle n'effectue donc aucun traitement, et se contente simplement d'afficher les données que lui fournit le modèle.
- le **Présentateur** est chargé de la logique de présentation. Il modifie la vue en fonction des règles de l'application et délègue au contrôleur la logique de but utilisateur (actions sur le modèle).
- le **Contrôleur** est chargé par le présentateur de déléguer au modèle les opérations métier.

Pour mieux illustrer ce modèle, voici ce qui se passe lorsque l'utilisateur clique sur le bouton de connexion après avoir entré son identifiant et son mot de passe.



Lorsque l'utilisateur clique sur le bouton de connexion, la vue délègue le traitement au présentateur, lequel le délègue à son tour au contrôleur. Ce dernier demande au modèle une connexion (vérification du couple identifiant-mot de passe, et récupération de l'utilisateur si valide). Le modèle déclenche alors un événement, auquel le présentateur répond en enlevant le formulaire de connexion et en affichant les informations propres à l'utilisateur. Si erreur se produit, elle est remonté jusqu'au présentateur qui peut par exemple demander à la vue d'afficher un message d'erreur.
Le mécanisme des événements est géré par le pattern *Observer*. C'est à la création des entités que le présentateur est enregistré en tant qu'observateur du modèle.
La figure 5 présente le diagramme de classes simplifié de ce schéma.
On peut d'ores et déjà écrire le code qui va lancer l'application :

```
public class App {
    public static void main( String[] args ) {
        Model model = new Model();
        Controller controller = new Controller(model);
        Presenter presenter = new Presenter(controller); // creates the view
        model.addObserver(presenter);
        presenter.showView();
    }
}
```

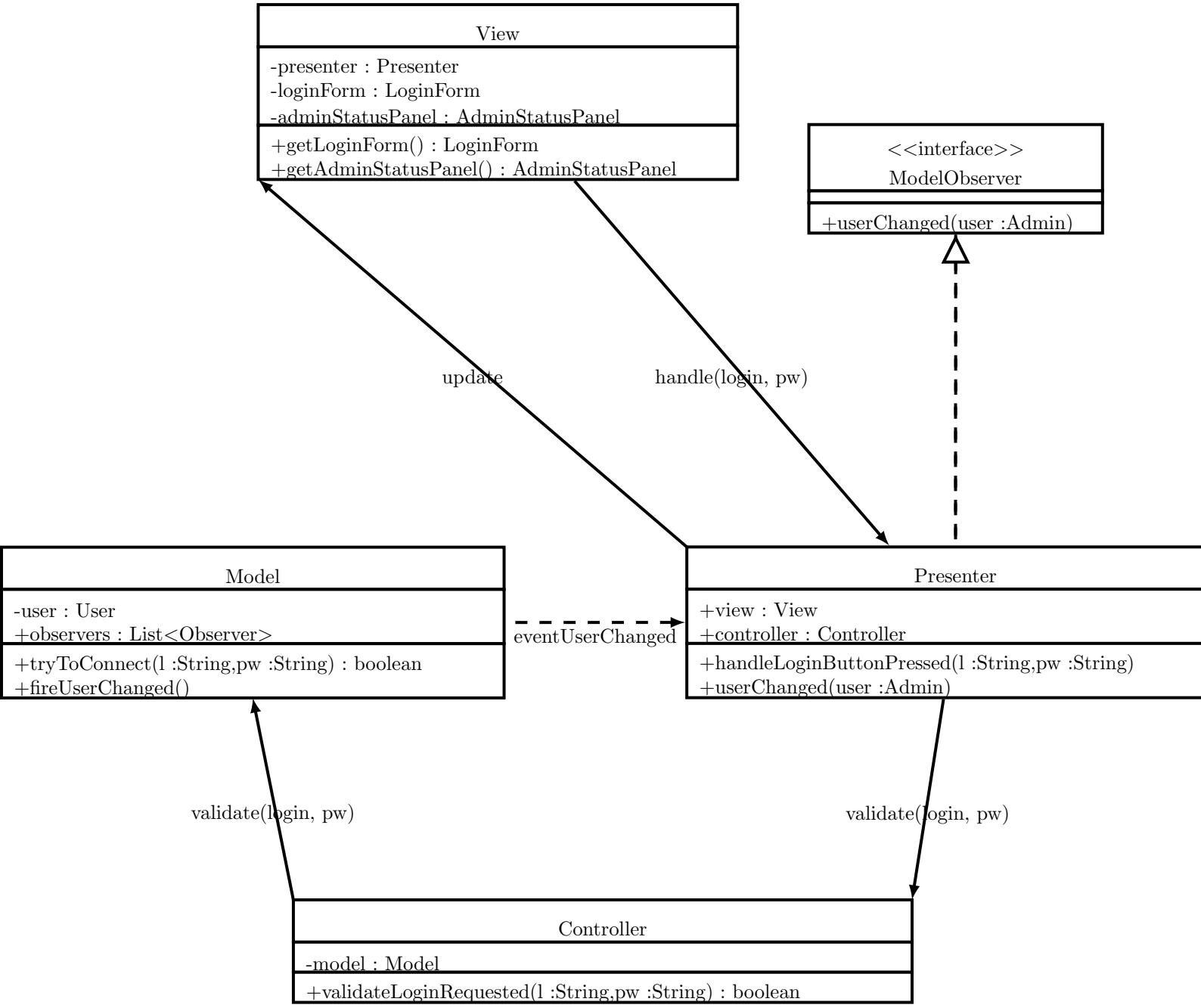


FIGURE 5 – Diagramme de classes MVPC

3.5 Les DAOs (Data Access Objects)

Le pattern DAO nous permet de nous abstraire de la façon dont les données sont stockés. Ainsi, il est possible de changer le mode de stockage sans remettre en cause le reste de l'application. Par exemple, pour passer de Oracle à MySQL, il suffit d'ajouter le connecteur et de modifier le fichier `persistence.xml` qui décrit la configuration de JPA.

Les DAO sont des objets servant à manipuler des objets (ils font donc évidemment parti du modèle). Nous en avons donc un par type d'objet métier. Par exemple, `DAOTeacher` pour les enseignants, `DAOGroup` pour les groupes, etc... Chacun de ces objets doit impérativement pouvoir exécuter les opérations élémentaires sur données : CRUD (*Create, Retrieve, Update, Delete*). Ces opérations sont déclarés ainsi, par exemple pour la DAO des enseignants :

- `Teacher create(Teacher teacher);`
- `List<Teacher> findAll();`
- `Teacher getById(int id);`
- `void update(Teacher teacher);`
- `void delete(Teacher teacher);`

Or, nous avons une douzaine de types d'objets, et les déclarations des DAOs varient que très peu. Nous avons donc créé une interface générique `DAO<T>` qui contient ces cinq méthodes, ainsi qu'une classe `DAOGeneriqueJPA<T>` qui implémente cette interface en utilisant JPA. Il nous suffirait maintenant de faire étendre `DAOTeacher` par `DAOGenerique<Teacher>` et d'ajouter, si nécessaire des méthodes de manipulation spécifiques aux enseignants. Mais nous serions donc obliger d'utiliser JPA. Pour s'en abstraire, il suffit de créé des interfaces pour les DAOs de chaque types d'objets, et ainsi créer les classes DAO spécifiques à JPA qui implémentent ces interfaces et étendent `DAOGeneriqueJPA<T>`, par exemple : `class DAOTeacherJPA extends DAOGeneriqueJPA<Teacher> implements DAOTeacher`
Le diagramme UML correspondant à ce modèle est présenté en figure 6.

Nous pouvons maintenant manipuler nos objets avec JPA. Si l'envi nous prend de remplacer JPA par JDBC par exemple, il nous faudrait juste réimplémenter les classes de type `DAOXXXJPA` en `DAOXXXJDBC`. Comme nous avons utilisé des interfaces, l'utilisation de ces DAO ne change pas. Mais leur initialisation oui, par exemple `DAOTeacher daoTeacher = new DAOTeacherJPA();`. Il faudrait donc changer toutes les initialisations des DAOs. Un autre défaut de cette méthode est que l'on peut avoir une multitude d'instances DAO pour un seul type d'objet. Bien que cela ne pose pas de problème majeur, il est préférable de n'avoir qu'une seule DAO par type d'objet.
Pour palier à ces problèmes, nous ne modifions pas le modèle déjà établi, mais nous introduisons le pattern `Factory` avec l'interface `DAOFactory`. Cette interface contiendra les méthodes de créations de toutes les DAOs, par exemple `DAOTeacher getDAOTeacher();`. Nous devons ensuite implémenter cette interface en utilisant JPA, et c'est maintenant le module qui gère la création de ses DAO (à condition de rendre les DAOs visibles uniquement par le package). Donc la factory portera les instances uniques de chaque DAOs qui seront créé lors du premier appel leur accesseur. Et tant qu'on y est, faisons la factory singleton également. De plus, c'est elle qui initialisera une bonne fois pour toute l'`EntityManager` nécessaire à JPA.
Allons encore plus loin et abstrayions nous JPA lors de la création de la factory. Cela se fait très simplement en créant une classe `DAOFactoryManager` qui ne fait que retourner l'instance de `DAOFactoryJPA` qui ne sera visible plus que dans le package du `DAOFactoryManager`. En conclusion, nous pouvons rendre publique uniquement les interfaces DAOs et la classe `DAOFactoryManager`. Ainsi l'utilisateur ne sait rien de leur implémentation, et le passage de JPA à JDBC ne modifie en rien le code utilisant le module DAO.

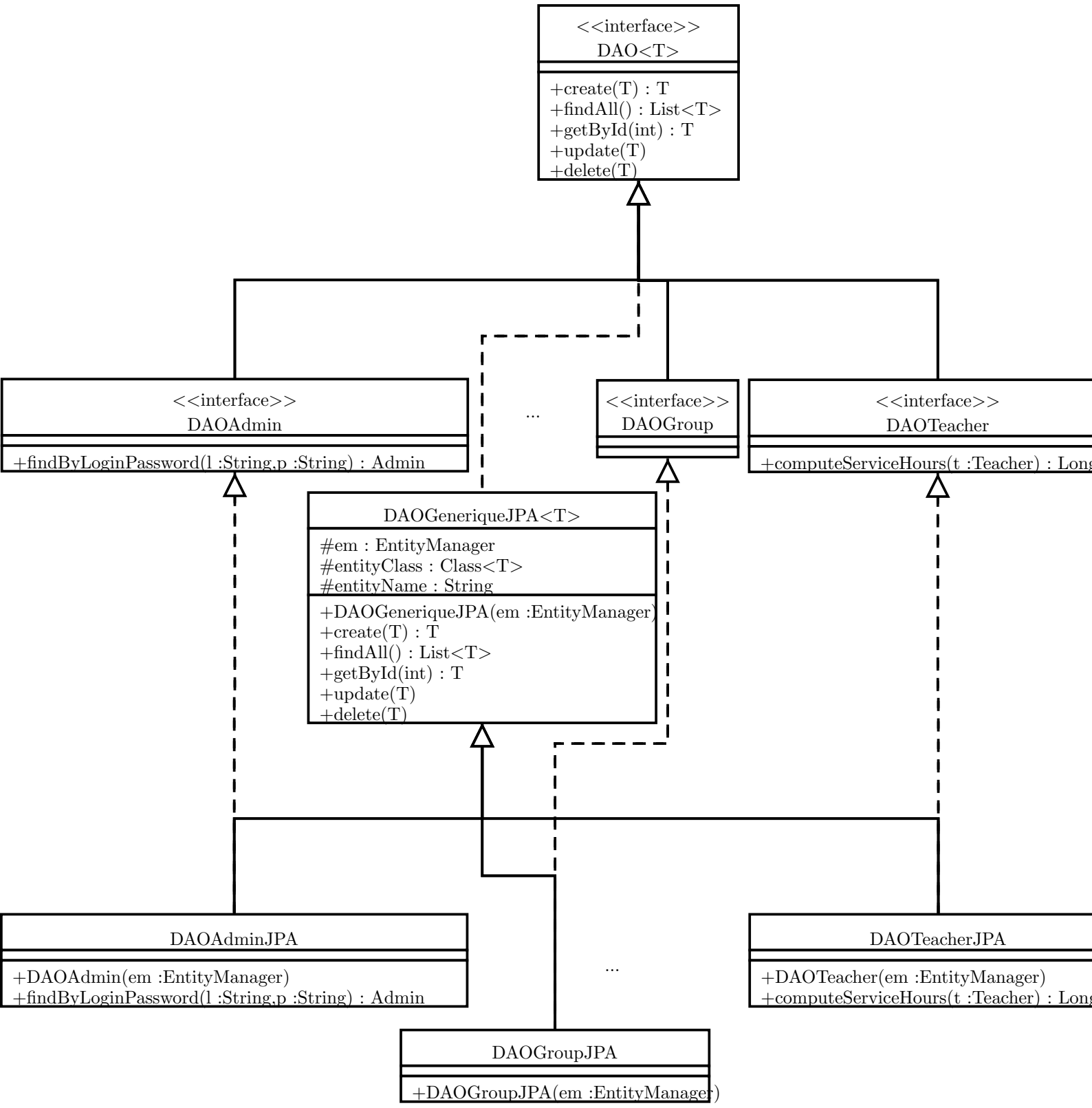


FIGURE 6 – Diagramme de classes des DAOs

3.6 Les rendus

Les conteneurs graphiques (`JPanel`, `JTable`, `JTree`, ...) contiennent par définition des objets qui seront disposés à la vue de l'utilisateur. Lorsqu'un conteneur est dessiné, les objets contenus sont eux même dessinés. Cela se fait principalement avec la méthode `paint()` pour des `JComponent`, ou pour des objets dits utilisateurs (*user object*), en mettant dans un "porteur" (souvent un `JLabel`) le résultat de l'appel à la méthode `toString()` (hérité de `Object`) sur l'objet à dessiner. Il suffirait donc de redéfinir cette méthode correctement pour chaque classes d'objet à afficher. Hors nos classes métiers ont déjà leur méthode `toString()` implémentée pour permettre la sérialisation. Cette méthode n'est donc plus adaptée pour l'affichage car non *user friendly*. Pour afficher les objets métiers correctement sans perdre leur aspect sérialisable, nous avons donc du utiliser un autre mécanisme. Bien qu'assez complexe à mettre en place, il permet au final une plus grande maniabilité.

La difficultée ici, c'est que pour un objet, par exemple un `JTree`, nous ne pouvons spécifier qu'un seul moteur de rendu (*renderer*) qui sera utilisé pour dessiner tous les objets (ce n'est pas le cas pour les `JTable`s car l'API nous oblige à spécifier la classe des objets lorsqu'on ajoute des moteurs de rendus [3]). C'est pour cela que la méthode `toString()` est très utilisée (car partagée par tous les objets).
Notre solution se base sur un moteur de rendu d'affichage délégué, basé sur la classe de l'objet (*Class-Based Display Delegation Renderer*). On a un objet qui fait office de moteur de rendu pour le conteneur. Lorsq'il est appelé, au lieu de créer directement un rendu, il délègue ce travail à un autre moteur de rendu selon un tableau associatif ayant pour clé la classe de l'objet à dessiner et pour valeur le moteur de rendu à appeler. Ainsi, peu importe la classe des objets à dessiner, nous pouvons utiliser le bon moteur de rendu (si aucun n'est trouvé, on utilise celui par défaut). Il faut bien entendu enregistrer ceux-ci avant tout affichage (c'est le rôle du présentateur).

Mais en pratique cette solution ne peux que très rarement être utilisée tel quel. En effet, les conteneurs graphiques requièrent des moteurs de rendus spécialisés. Par exemple, les `JComboBox`s ne peuvent qu'utiliser des rendus implémentant l'interface `ListCellRenderer`. La solution la plus simple est d'encapsuler un DDR [4] dans une classe dérivant de `DefaultListCellRenderer` qui implémente `ListCellRenderer` et dérive de `JLabel`. Lorsqu'un rendu est demandé, on appel d'abord la méthode `DefaultListCellRenderer` qui créera un rendu par défaut, puis on le modifie avec notre DDR.
Un autre problème se pose avec les `JTree`s : ce ne sont pas les objets métiers qui sont dessinés mais les noeuds (`TreeNode`s). La solution est similaire à celle du problème des moteurs de rendus spécialisés. Pour résumer, voici les états d'encapsulations des moteurs de rendus pour notre `JTree` : Le renderer de l'arbre créé un rendu par défaut et le modifie en appelant le renderer de noeud qui extrait l'objet métier et appel le DDR qui appel le bon renderer et renvoie le rendu tant désiré au premier renderer. Le diagramme de classes correspondant est présenté en figure 7.

3. Nous pouvons toujours appliquer notre méthode en utilisant `Object.class`
4. DDR : Display Delegation Renderer

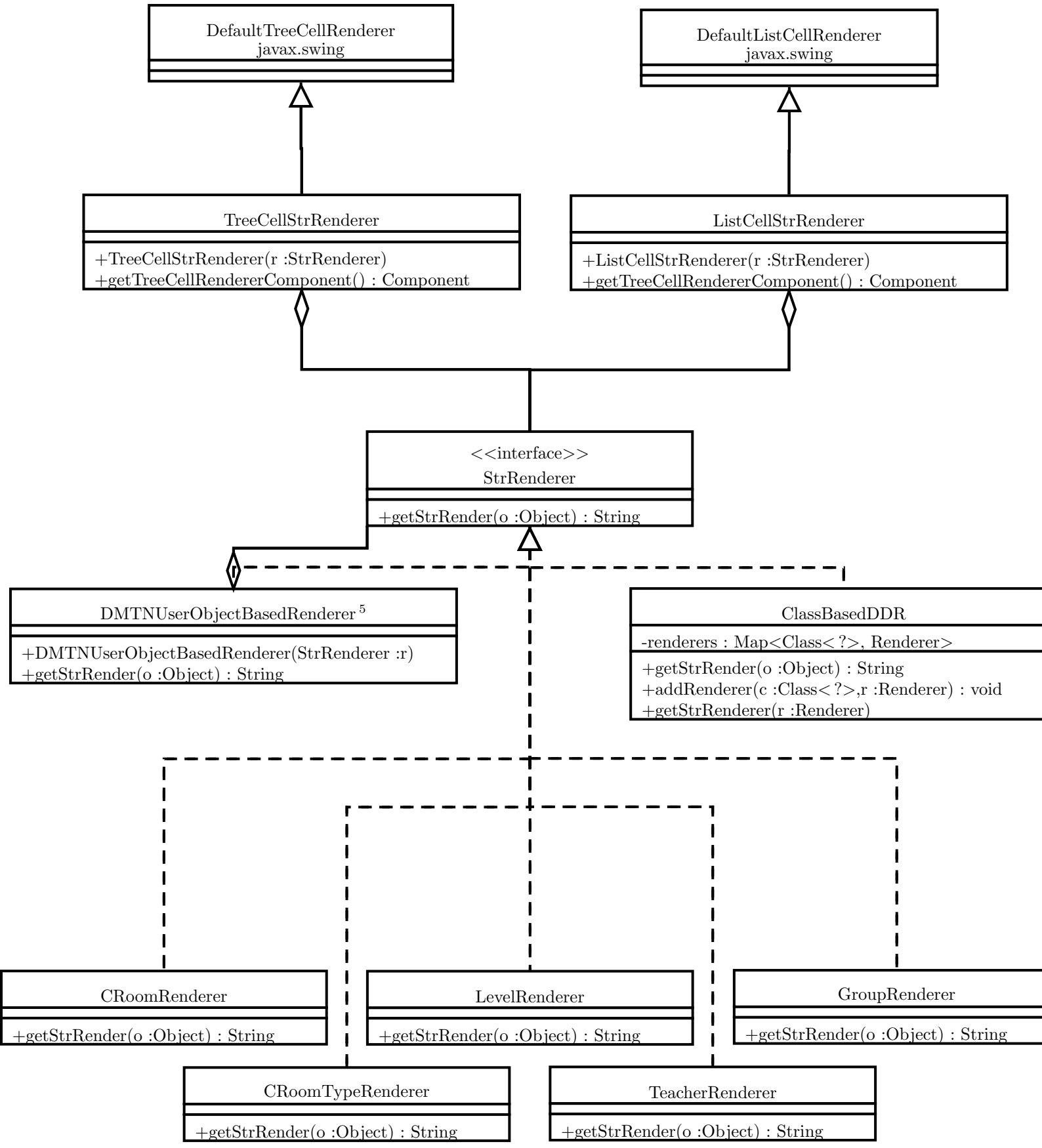


FIGURE 7 – Diagramme de classes des rendus

3.7 La table

Pour la représentation de l'emploi du temps lui-même, nous utilisons la classe fournie par `swing` : `JTable`. Ce faisant, plusieurs difficultés se sont posées :

- Il a fallu trouver les coordonnées d'une séance dans la table. C'est à dire convertir le jour de la semaine en index de ligne, et l'heure dans la journée en index de colonne. Avec des colonnes représentant deux heures, on peux utiliser la formule suivante :

  $$numero\ ligne = num\_jour - num\_lundi$$

  (en java (`classe Calendar`), le numéro représentant le jour de la semaine ne commence pas forcément à `0`).

  $$numero\ colonne = heure - (7 + \frac{heure - 8}{2})$$

  Finalement, comme les séances peuvent également durer une ou trois heures, nous utilisons une colonne par heure. La conversion pour pour l'indice de colonne deviens plus simple :

  $$numero\ colonne = heure - 8$$

- Pour les séances durant plusieurs heures, nous voulions fusionner horizontalement les cellules correspondantes dans la table (*column spanning*). Cela n'est pas initialement prévu par la classe `JTable`. Mais celle-ci semble avoir été concue pour autoriser la distribution non-uniforme des cellules.
  Les méthodes intéressantes à redéfinir sont :
  `public Rectangle getCellRect(int row, int column, boolean includeSpacing);`
  Retourne un rectangle englobant la cellule aux positions donnés.
  `public int columnAtPoint(Point p);`
  `public int rowAtPoint(Point p);`
  Retournent respectivement la ligne et la colonne de la cellule contenant un point donné dans la table. Dans notre cas, nous n'auront que `columnAtPoint()` à redéfinir, car les cellules ne seront fusionner que horizontalement.
  Une fois ceci accompli, le `TableCellRenderer` s'occupe du rendu des cellules.
  Mais cela ne suffit pas. Contrairement à ce que l'on pourrais penser, beaucoups de composants `swing` ne se dessinent pas directement avec la méthodes `paint` mais délèguent le travail à un objet `ComponentUI` pour permettre une plus grande flexibilité. C'est le cas avec les `JTable`s. Il nous faut donc définir une classe qui va dessiner la table à notre convenance. Pour cela, nous dérivons la classe `BasicTableUI` et redéfinissons la méthode paint.
  Enfin, il nous faut une structure pour stocker les cellules qui seront fusionnées. Elle fera parti du modèle de notre table que nous redéfinirons car le simple `DefaultTableModel` ne suffit plus. Le diagramme de classes est présenté figure 7.

4 Les tests

Comme notre application est assez simple, nous n'avons pas fait de tests unitaires. De plus, des tests unitaires complets nous auraient pris plus de temps que imparti. Seul un petit avec de la classe `DAORank` a été écrit. Le but était ici de se familiariser avec *Derby* et *DBUnit*.
Cependant, à chaque ajout d'une nouvelle fonctionnalité, nous la testions sur plusieurs exemples. Nous n'avons juste pas pris le temps d'écrire du code automatisant tout cela.

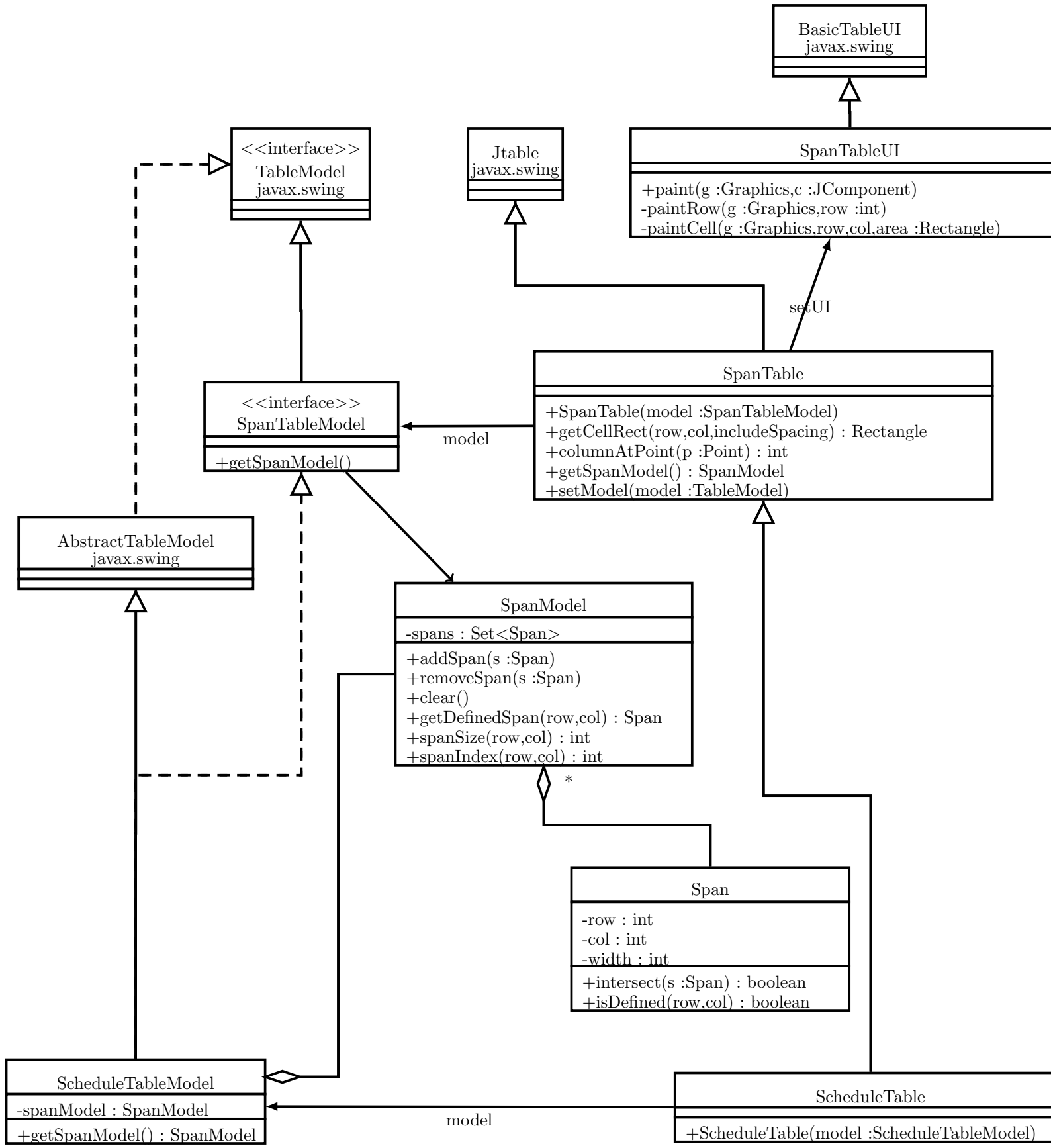


FIGURE 8 – Diagramme de classes de la table

A MCT

Voir à la fin du document.

B Fonctions PL/SQL

```
CREATE OR REPLACE FUNCTION date_overlap(d1_start DATE, d1_duration NUMBER,
                                        d2_start DATE, D2_duration NUMBER)
RETURN NUMBER
IS
  res BOOLEAN;
BEGIN
  res := (d1_start < d2_start + d2_duration/24) AND
         (D2_start < d1_start + d1_duration/24);
  IF res = TRUE THEN RETURN 1;
  ELSE RETURN 0;
  END IF;
END date_overlap;

-------------------------------------------------

CREATE OR REPLACE FUNCTION croom_occupied(p_id_croom NUMBER, p_start_date DATE, p_duration NUMBER)
RETURN NUMBER
IS
  res NUMBER;
BEGIN
  SELECT dIStinct NULL INTO res
  FROM t_croom cr, t_session s
  WHERE cr.id_croom = p_id_croom AND
        s.id_croom (+) = cr.id_croom AND
        (s.id_session IS NULL OR
         0 = (SELECT COUNT(*) FROM t_session
              WHERE id_croom = s.id_croom AND
                    date_overlap(p_start_date, p_duration, start_session_date, session_duration) = 1
  RETURN 0; -- not occupied
EXCEPTION
  WHEN no_data_found THEN RETURN 1; -- occupied
END croom_occupied;

-------------------------------------------------

CREATE OR REPLACE FUNCTION croom_compatible(p_id_croom NUMBER, p_id_session_type NUMBER)
RETURN NUMBER
IS
  res NUMBER;
BEGIN
  SELECT DISTINCT NULL INTO res
  FROM t_croom cr, t_session_croom_compatibility scrc
  WHERE cr.id_croom = p_id_croom AND
        cr.id_croom_type = scrc.id_croom_type AND
        scrc.id_session_type = p_id_session_type;
  RETURN 1; -- compatible
EXCEPTION
  WHEN no_data_found THEN RETURN 0; -- not compatible
END;

-------------------------------------------------
```

```
CREATE OR REPLACE FUNCTION group_available(p_group_id NUMBER, p_start_date DATE, p_duration NUMBER)
RETURN NUMBER
IS
  res NUMBER;
BEGIN

  IF p_group_id IS NULL THEN
    SELECT NULL INTO res
    FROM t_session
    WHERE id_group IS NULL AND
          date_overlap(p_start_date, p_duration, start_session_date, session_duration) = 1;
  ELSE
    SELECT NULL INTO res
    FROM t_session
    WHERE id_group = p_group_id AND
          date_overlap(p_start_date, p_duration, start_session_date, session_duration) = 1;
  END IF;
  RETURN 0; -- unavailable
EXCEPTION
  WHEN no_data_found THEN RETURN 1; -- available
END group_available;

-------------------------------------------------

CREATE OR REPLACE FUNCTION teacher_in_class(p_id_teacher NUMBER, p_start_date DATE, p_duration NUMBE
RETURN NUMBER
IS
  res NUMBER;
BEGIN
  SELECT DISTINCT NULL INTO res
  FROM t_teacher t, t_session s
  WHERE t.id_admin = p_id_teacher AND
        s.id_teacher (+) = t.id_admin AND
        (s.id_session IS NULL OR
         0 = (SELECT COUNT(*) FROM t_session
              WHERE id_teacher = s.id_teacher AND
                    date_overlap(p_start_date, p_duration, start_session_date, session_duration) = 1
  RETURN 0; -- not in class
EXCEPTION
  WHEN no_data_found THEN RETURN 1; -- in class
END teacher_in_class;

-------------------------------------------------

CREATE OR REPLACE FUNCTION teacher_unavailable(p_id_teacher NUMBER, p_start_date DATE, p_duration NU
RETURN NUMBER
IS
  res NUMBER;
BEGIN
  SELECT DISTINCT NULL INTO res
  FROM t_teacher t, t_unavailability u
  WHERE t.id_admin = p_id_teacher AND
        u.id_teacher (+) = t.id_admin AND
        (u.id_unavailability IS NULL OR.
         0 = (SELECT COUNT(*) FROM t_unavailability
              WHERE id_teacher = u.id_teacher AND
                    date_overlap(p_start_date, p_duration, start_unav_date, unav_duration) = 1));
  RETURN 0; -- unavailable
```

```
EXCEPTION
  WHEN no_data_found THEN RETURN 1; -- available
END teacher_unavailable;

-------------------------------------------------

CREATE OR REPLACE FUNCTION teacher_busy(p_id_teacher NUMBER, p_start_date DATE, p_duration NUMBER)
RETURN NUMBER
IS
BEGIN
  IF teacher_in_class(p_id_teacher, p_start_date, p_duration) = 1 OR
     teacher_unavailable(p_id_teacher, p_start_date, p_duration) = 1 THEN
    RETURN 1;
  ELSE
    RETURN 0;
  END IF;
END teacher_busy;

-------------------------------------------------

CREATE OR REPLACE FUNCTION teacher_service_hours(p_id_teacher NUMBER)
RETURN NUMBER
IS
  h NUMBER;
  s NUMBER;
BEGIN
  -- get admin hours
  SELECT admin_hours INTO h
  FROM t_teacher
  WHERE id_admin = p_id_teacher;
  -- get sessions hours worth
  SELECT SUM(s.session_duration * st.tutor_worth_coef) INTO s
  FROM t_teacher t, t_session s, t_session_type st
  WHERE t.id_admin = p_id_teacher AND
        t.id_admin = s.id_teacher AND
        s.id_session_type = st.id_session_type;
  RETURN s + h;
END teacher_service_hours;

-------------------------------------------------

CREATE OR REPLACE FUNCTION unplanned_sessions(p_id_level NUMBER)
RETURN unplanned_sessions_table
IS
   res unplanned_sessions_table;
   id_type_cours NUMBER;
   id_type_td NUMBER;
   id_type_tp NUMBER;
   id_type_projet NUMBER;
BEGIN
  SELECT id_session_type INTO id_type_cours
  FROM t_session_type
  WHERE libel_session_type = 'Cours';
  SELECT id_session_type INTO id_type_td
  FROM t_session_type
  WHERE libel_session_type = 'TD';
  SELECT id_session_type INTO id_type_tp
  FROM t_session_type
```

```
  WHERE libel_session_type = 'TP';
  SELECT id_session_type INTO id_type_projet
  FROM t_session_type
  WHERE libel_session_type = 'Projet';

  SELECT rec_unplanned_sessions(libel_tu,
                                'Cours',
                                class_hours - (SELECT NVL(SUM(session_duration), 0)
                                               FROM t_session
                                               WHERE id_tu = t.id_tu AND
                                                     id_session_type = id_type_cours),
                                'TD',
                                tutor_hours - (SELECT NVL(SUM(session_duration), 0)
                                               FROM t_session
                                               WHERE id_tu = t.id_tu AND
                                                     id_session_type = id_type_td),
                                'TP',
                                practical_hours - (SELECT NVL(SUM(session_duration), 0)
                                                   FROM t_session
                                                   WHERE id_tu = t.id_tu AND
                                                         id_session_type = id_type_tp),
                                'Projet',
                                project_hours - (SELECT NVL(SUM(session_duration), 0)
                                                 FROM t_session
                                                 WHERE id_tu = t.id_tu AND
                                                       id_session_type = id_type_projet))
  BULK COLLECT INTO res FROM t_tu t
  WHERE id_level = p_id_level;
  RETURN res;
END unplanned_sessions;
```

C Déclencheurs Oracle

```
CREATE OR REPLACE TRIGGER session_constraints
BEFORE INSERT OR UPDATE on t_session
FOR EACH ROW
DECLARE
  n_students t_group.size_group%TYPE;
  croom_size t_croom.max_size%TYPE;
  remaining_planned_hours NUMBER;
  level_group t_level.id_level%TYPE;
  level_tu t_level.id_level%TYPE;
BEGIN
  -- check teacher
  IF teacher_busy(:new.id_teacher, :new.start_session_date, :new.session_duration) = 1 THEN
    RAISE_APPLICATION_ERROR(-20001, 'enseignant occupé');
  END IF;

  -- check group (students)
  IF group_available(:new.id_group, :new.start_session_date, :new.session_duration) = 0 THEN
    RAISE_APPLICATION_ERROR(-20002, 'groupe ou promotion occupé');
  END IF;

  -- check croom occupation
  IF croom_occupied(:new.id_croom, :new.start_session_date, :new.session_duration) = 1 THEN
    RAISE_APPLICATION_ERROR(-20003, 'salle non dISponible');
  END IF;
```

```
-- check croom size
IF :new.id_group IS NULL THEN
  -- may raISe too many rows IF one-one relationship ISn't satISfied
  SELECT p.size_promo INTO n_students
  FROM t_promo p, t_tu t
  WHERE t.id_tu = :new.id_tu AND
        t.id_level = p.id_level;
ELSE
  SELECT size_group INTO n_students
  FROM t_group
  WHERE id_group = :new.id_group;
END IF;
SELECT max_size INTO croom_size
FROM t_croom
WHERE id_croom = :new.id_croom;
IF croom_size < n_students THEN
  RAISE_APPLICATION_ERROR(-20004, 'salle trop petite');
END IF;

-- check session TYPE
IF croom_compatible(:new.id_croom, :new.id_session_TYPE) = 0 THEN
  RAISE_APPLICATION_ERROR(-20005, 'salle non compatible avec le type de la seance');
END IF;

IF :new.id_session_TYPE = 3995 AND :new.id_group IS not NULL THEN
  RAISE_APPLICATION_ERROR(-50006, 'un group ne peut pas participer à un cours');
END IF;

-- check duration
IF :new.id_session_TYPE = 3995 THEN -- Cours
  SELECT class_hours INTO remaining_planned_hours
  FROM t_tu
  WHERE id_tu = :new.id_tu;
END IF;
IF :new.id_session_TYPE = 3993 THEN -- TD
  SELECT tutor_hours INTO remaining_planned_hours
  FROM t_tu
  WHERE id_tu = :new.id_tu;
END IF;
IF :new.id_session_TYPE = 3994 THEN -- TP
  SELECT practical_hours INTO remaining_planned_hours
  FROM t_tu
  WHERE id_tu = :new.id_tu;
END IF;
IF :new.id_session_TYPE = 3996 THEN -- Projet
  SELECT project_hours INTO remaining_planned_hours
  FROM t_tu
  WHERE id_tu = :new.id_tu;
END IF;
-- ELSE reference constraint will fail

IF :new.session_duration > 4 OR :new.session_duration > remaining_planned_hours THEN
  RAISE_APPLICATION_ERROR(-20007, 'durée trop importante');
END IF;

-- check time
IF SYSDATE > :new.start_session_date THEN
  RAISE_APPLICATION_ERROR(-20008, 'date de début de session déjà passé');
```

```
  END IF;

  IF date_overlap(to_date(to_char(:new.start_session_date,'dd/mm/yyyy')||' - 08:00',
                                  'dd/mm/yyyy - hh24:mi'), 12,
                 :new.start_session_date, :new.session_duration) = 0 THEN
    RAISE_APPLICATION_ERROR(-20009, 'date de début de session non conforme');
  END IF;

  IF to_char(:new.start_session_date, 'd') >= 6 THEN
    RAISE_APPLICATION_ERROR(-20010, 'date de début de session un week-end');
  END IF;

  -- check level of group / tu
  IF :new.id_group IS not NULL THEN
    SELECT p.id_level INTO level_group
    FROM t_promo p, t_group g
    WHERE g.id_group = :new.id_group AND
          p.id_promo = g.id_promo;
    SELECT id_level INTO level_tu
    FROM t_tu
    WHERE id_tu = :new.id_tu;
    IF level_group <> level_tu THEN
      RAISE_APPLICATION_ERROR(-20011, 'niveau du groupe non adapté à l''UE');
    END IF;
  END IF;

EXCEPTION
  WHEN too_many_rows THEN NULL;
END session_constraints;

-------------------------------------------------

CREATE OR REPLACE TRIGGER unavailability_constraints.
BEFORE INSERT OR UPDATE OF start_unav_date, unav_duration ON T_UNAVAILABILITY
FOR EACH ROW
BEGIN
  IF :new.start_unav_date < SYSDATE THEN
    RAISE_APPLICATION_ERROR(-20051, 'date de début de l''indisponibilité déjà passé');
  END IF;
  IF teacher_in_class(:new.id_teacher, :new.start_unav_date, :new.unav_duration) = 1 THEN
    RAISE_APPLICATION_ERROR(-20052, 'enseignant donne une séance pendant cette période');
  END IF;
  IF teacher_unavailable(:new.id_teacher, :new.start_unav_date, :new.unav_duration) = 1 THEN
    RAISE_APPLICATION_ERROR(-20053, 'enseignant déjà indisponible durant cette période');
  END IF;
END unavailability_constraints;

-------------------------------------------------

CREATE OR REPLACE TRIGGER attachment_site_constraint
BEFORE INSERT OR UPDATE OF attachment_site ON T_TEACHER
FOR EACH ROW
BEGIN
  IF :new.attachment_site <> 'Extérieur' AND
     :new.attachment_site <> 'Departement informatique' THEN
     RAISE_APPLICATION_ERROR(-20030, 'site de rattachement invalide');
  END IF;
END attachment_site_constraint;
```

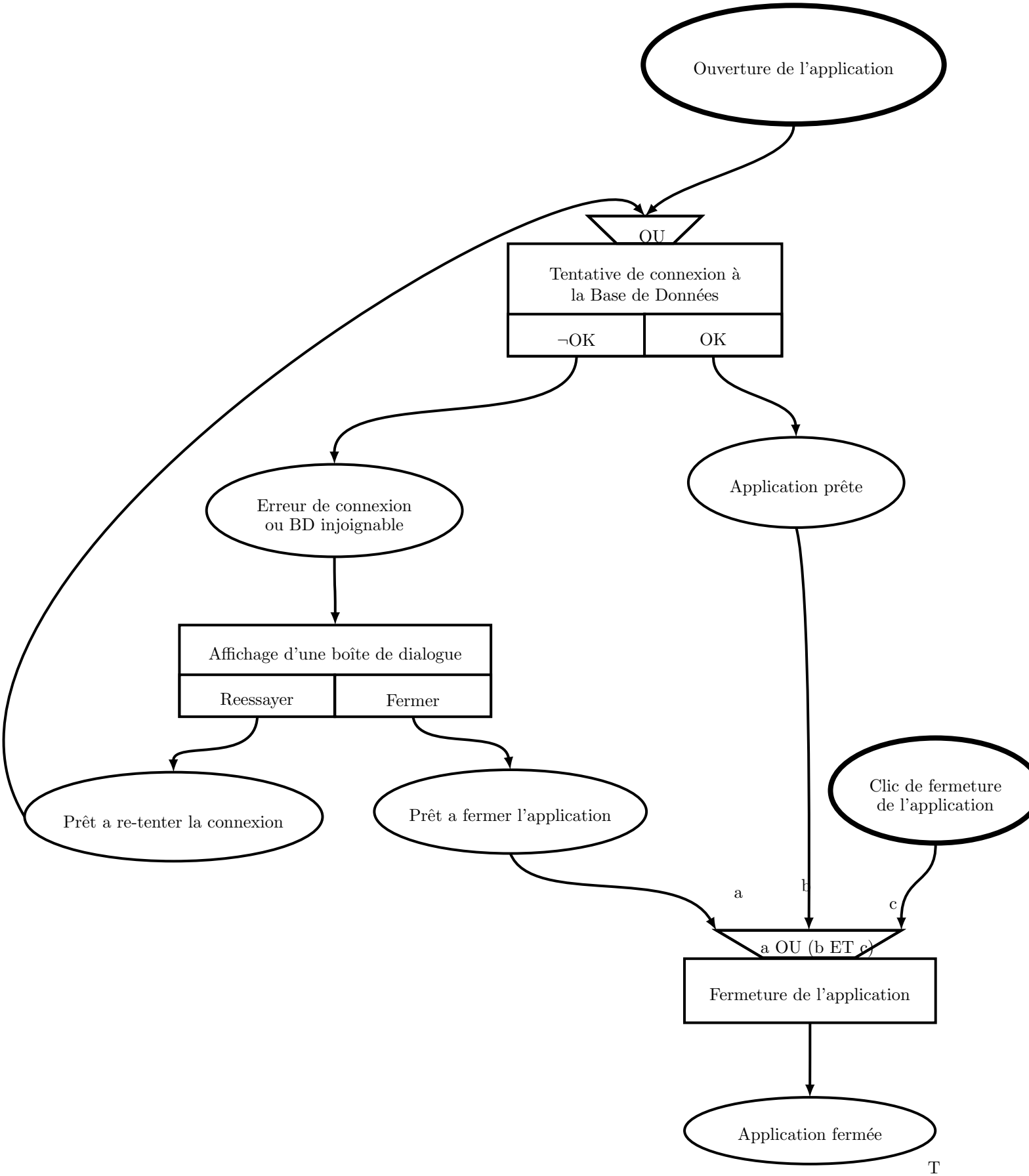


Figure 9 – MCT (1)

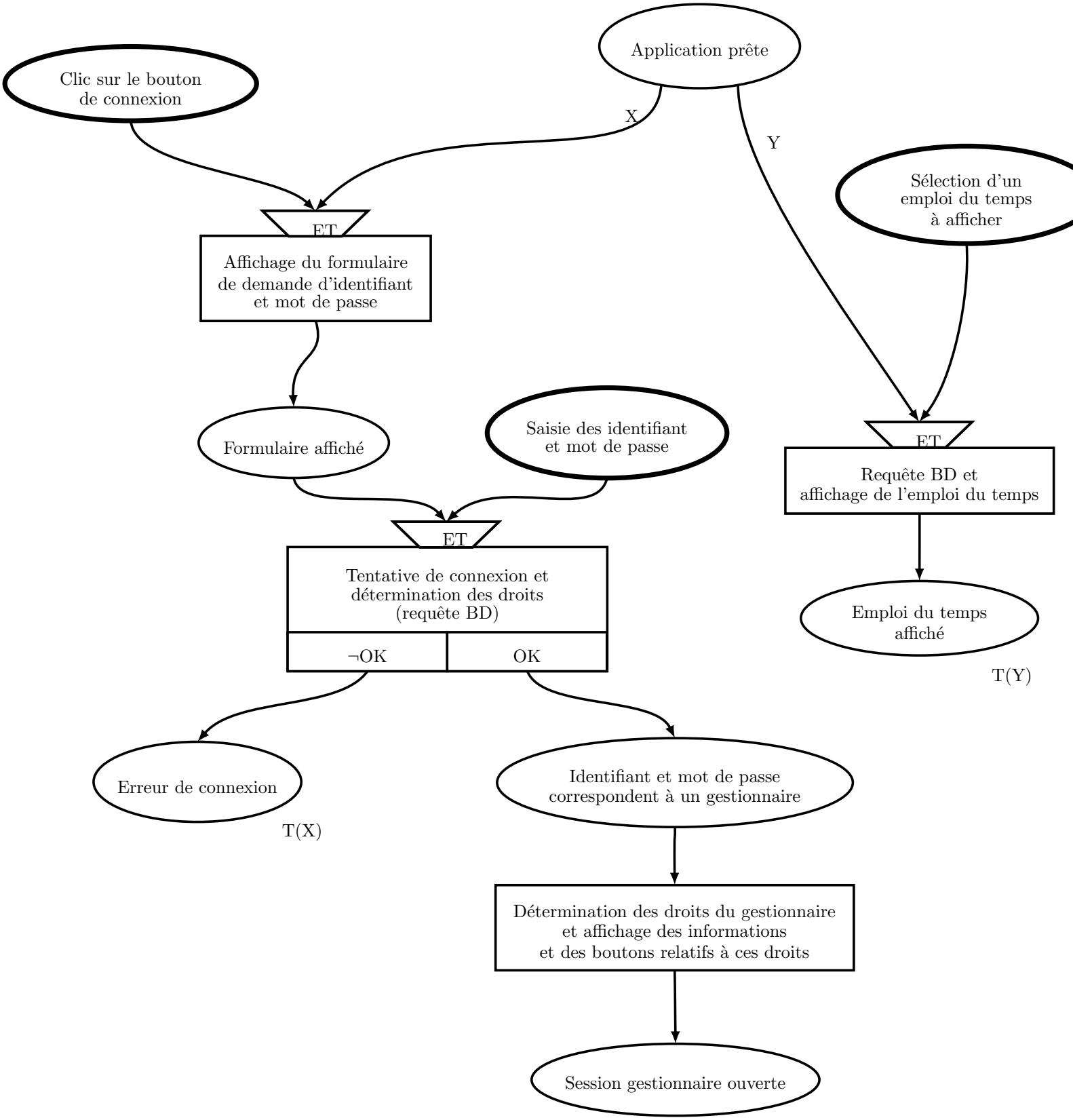


FIGURE 10 – MCT (2)

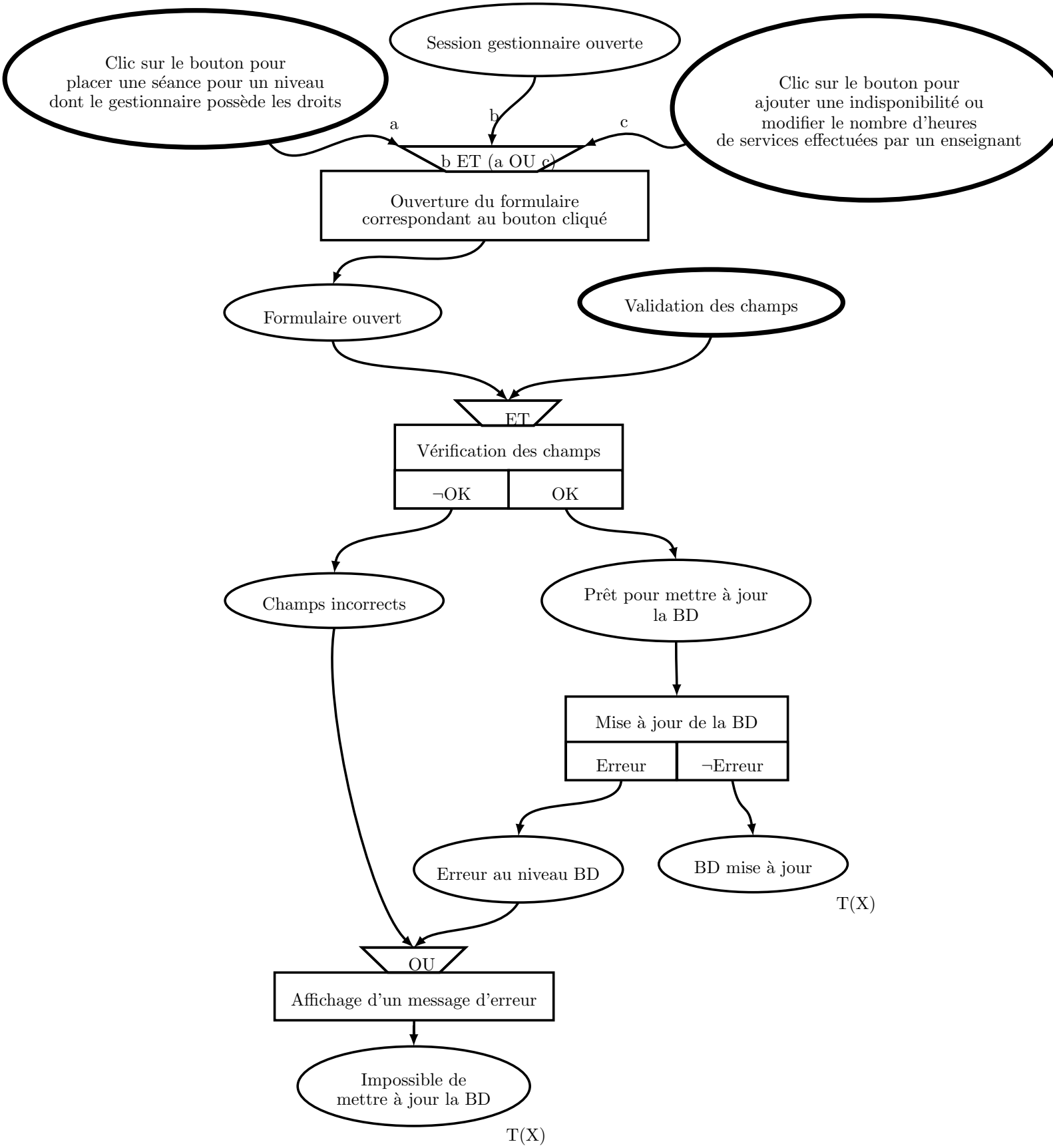


FIGURE 11 – MCT (3)