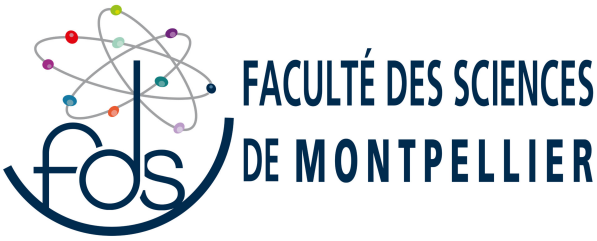


Master 2 : Informatique
Génie Logiciel
2023

---

**HAI919I - DM**
**Création d'un langage textuel permettant la spécification d'un emploi du temps**

---

**Etudiant:**
BENGOUFA Ryan
CRISTA Allan
MEKHNACHE Toudherth
ROUQUAIROL Lucas

December 30, 2023

# Contents

# 1 Introduction

Dans le cadre de notre Master en Informatique, nous avons entrepris le développement d'un langage spécifique au domaine (DSL) pour la modélisation d'emplois du temps, en utilisant le framework Xtext. Ce projet, intégré à l'unité d'enseignement "Ingénierie Dirigée par les Modèles" (IDM), vise à résoudre les complexités de la planification académique en automatisant et en rationalisant le processus de création d'emplois du temps. En combinant théorie et pratique, nous avons développé un outil qui simplifie la gestion des contraintes liées aux horaires, salles, et des enseignants. Ce rapport détaille notre approche méthodologique, de la conception du métamodèle à l'implémentation et la validation du système, illustrant l'application efficace de la modélisation dirigée par les modèles dans un contexte réel.

## 1.1 Objectifs du projet

Avant de commencer le développement, nous avons cherché à définir des objectifs spécifiques :

- Concevoir un métamodèle qui capture les concepts clés et les contraintes associés à la planification d'emploi du temps.
- Implémenter une grammaire Xtext qui facilite la saisie et la modification des emplois du temps.
- Générer un éditeur de texte riche en fonctionnalités qui offre une coloration syntaxique, de l'autocomplétion, et des validations de modèle en temps réel.
- Produire une documentation (rapport) qui détaille l'utilisation et les capacités du DSL créé.

# 2 Approche méthodologique et conception

## 2.1 Analyse des besoins

Pour représenter notre emploie du temps, nous avons choisis de nous concentrer sur un planning à la semaine type collège. C'est à dire, des créneaux d'une heure répartis sur les jours de la semaine sauf le samedi et dimanche, et avec une pause entre 12h et 14h.

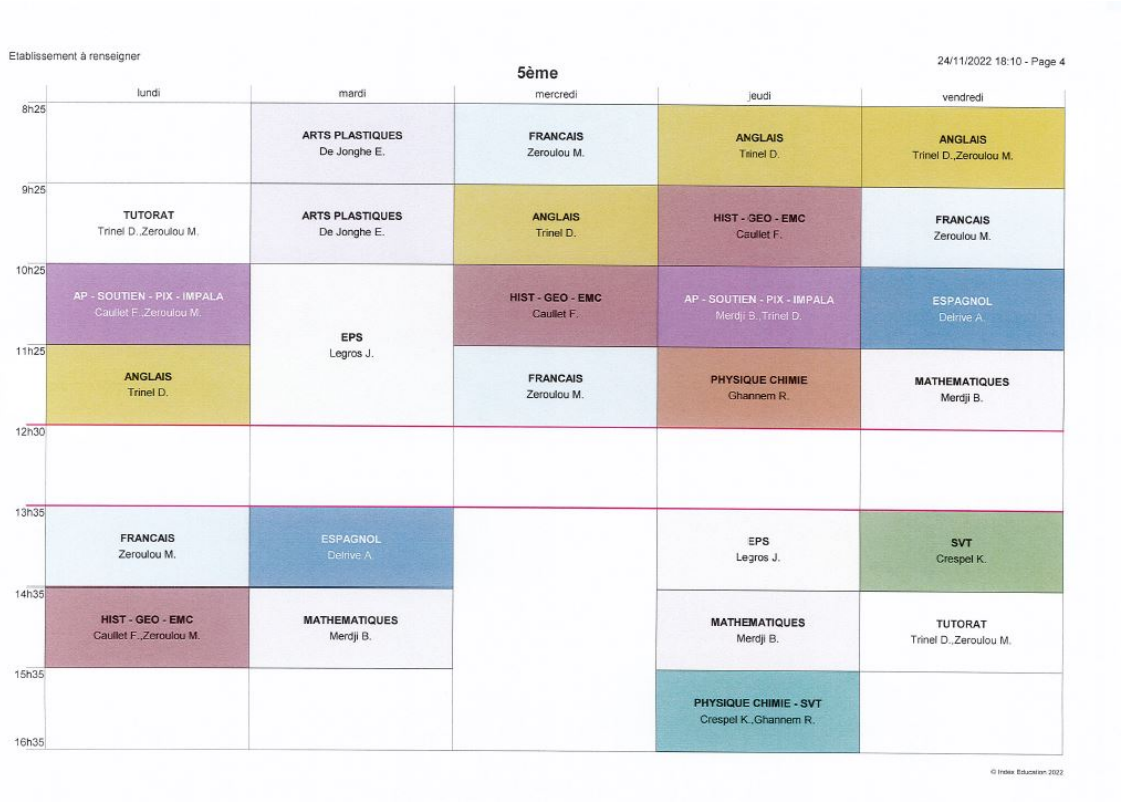


Figure 1: Exemple de notre type d'emploi du temps

Pour des soucis de simplicité, nous avons décidé que les créneaux ferait tous 1h. Ceci nous permet de ne définir qu'une date de début et de ne pas calculer de débordement pour la pause ou la fin de journée. Aussi dans notre cas, si deux cours ne commencent pas en même temps, ils ne se chevauchent forcément pas. Il est bien sur possible de mettre deux fois le même créneau pour simuler un cours de deux heures.

Les jours férié ne se comportent pas de la même manière que le weekend. En effet, un jour férié ne contient pas une liste de créneaux mais à la place, un nom d'évènement associé. Pour les jours du weekend, il sont traité comme les autres jours à la différence que leur liste de créneau doit rester vide.

La conception de notre système d'emploi du temps, développé avec Xtext, repose sur une architecture clairement définie et un métamodèle détaillé. Nous avons également intégré des mécanismes de validation robustes pour assurer la cohérence et la pertinence des emplois du temps générés.

## 2.2 Métamodèle détaillé (UML)

Notre métamodèle, conçu à l'aide de la modélisation UML, définit la structure et les relations entre les différentes entités de notre DSL (Domain-Specific Language). Les principales entités

sont :
**Créneau Horaire** : Représente un intervalle de temps dédié à une activité académique spécifique.
**Jour** : Définit les jours de la semaine et les jours fériés, chacun avec ses particularités.
**Enseignant** : Détaille les informations sur les enseignants, y compris leurs disponibilités et leurs spécialisations.
**UE (Unité d'Enseignement)** : Décrit les cours, leur volume horaire, et les exigences associées.

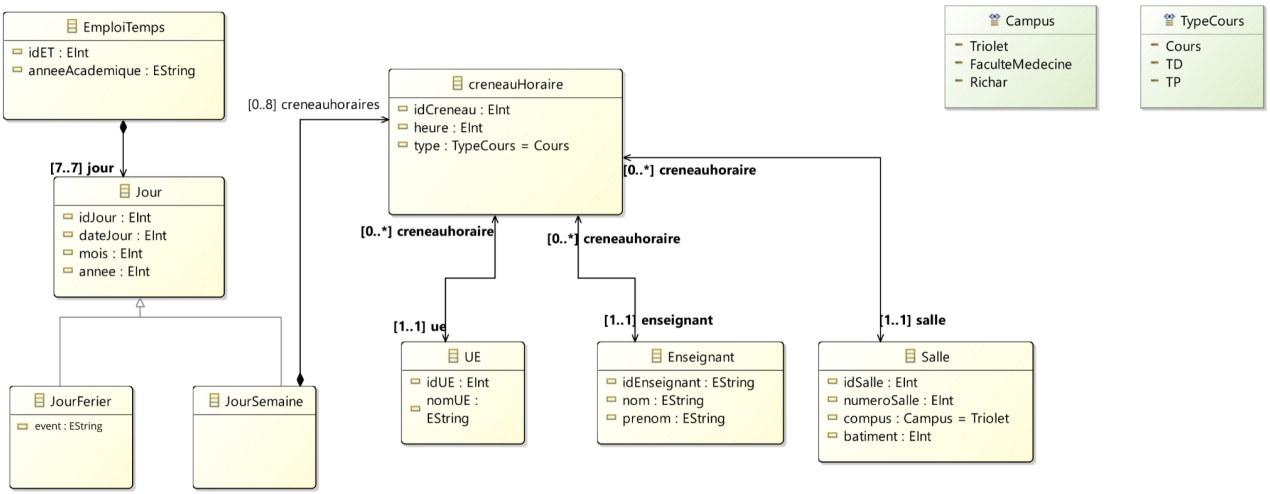


Figure 2: Diagramme UML du métamodèle

Ce métamodèle UML sert de base pour la création de notre grammaire Xtext et pour la génération automatique de l'éditeur de texte.

## 2.3 Validation et contraintes métier

La validité des emplois du temps est assurée par un ensemble de règles et de contraintes métier:
**Contraintes de Temps** : Nous vérifions que les créneaux horaires ne se chevauchent pas et respectent les heures de travail.
**Contraintes Enseignants** : Il est impératif que les enseignants ne soient pas assignés à plusieurs cours simultanément.
**Gestion des Jours Fériés** : Les jours fériés sont traités différemment, sans créneaux de cours mais avec des événements associés.
**Respect du Volume Horaire des UE** : Chaque emploi du temps doit respecter le volume horaire défini pour chaque UE.

Ces mécanismes de validation sont essentiels pour garantir que les emplois du temps générés sont à la fois pratiques et conformes à nos exigences académiques.

# 3 Implémentation avec Xtext

## 3.1 Définition de Xtext

Xtext est un framework pour le développement de langages de programmation et de langages spécifiques à un domaine (DSL - Domain-Specific Language). Il s'appuie sur une grammaire générée par ANTLR et sur le framework de modélisation EMF. L'avantage principal de Xtext réside dans la génération automatique d'un IDE pour le langage à implémenter.

Xtext couvre tous les aspects d'un IDE moderne, incluant le parseur, le compilateur, l'interpréteur, et une intégration complète dans l'environnement de développement Eclipse, permettant de fournir les fonctionnalités essentielles d'un IDE classique.

## 3.2 Développement avec Xtext

La grammaire Xtext que nous avons développée vise à faciliter la modélisation des emplois du temps dans un contexte académique, permettant une représentation structurée de divers éléments tels que les salles, les enseignants, les UE, et les emplois du temps eux-mêmes.

- **Définition de la grammaire :** La grammaire débute avec la déclaration de la racine *Model*. Cette règle est conçue pour encapsuler tous les éléments du fichier, permettant l'inclusion de divers types d'entités comme *Salle*, *Enseignant*, *UE*, et *EmploiTemps*. La structure flexible de cette règle permet d'intégrer une ou plusieurs instances de ces entités dans n'importe quel ordre.

```
Model:
    (includes+=Includes)* (salles+=Salle | enseignants+=Enseignant | ues+=UE)*
    (emplois+=EmploiTemps*) ;
```

- **Déclaration des Includes** La règle *Includes* facilite l'inclusion de fichiers externes avec l'extension *.emp*, renforçant la modularité et la réutilisation des données.

```
Includes:
    '%include' '<'importURI=INCLUDE'>;';
```

- **Définition des Entités Principales**

1. **Emploi du temps** La règle *EmploiTemps* est au cœur de notre grammaire, définissant la structure principale d'un emploi du temps académique. Elle encapsule plusieurs éléments clés.

```
EmploiTemps:
    'EmploiDeTemps' idET=ID':'
      ('AnneeAcademique'':' anneeAcademique=STRING';' &
      'Jours'':' jour+=Jour*';')
    'End.'
;
```

2. **Jour** La règle *Jour* permet de distinguer les jours ouvrables (*JourSemaine*) des jours fériés (*JourFerie*), une distinction cruciale pour une représentation précise de la semaine académique.

```
Jour:
    JourFerie | JourSemaine;
```

3. **Créneau Horaire** La règle *CreneauHoraire* détaille chaque créneau horaire dans un emploi du temps, fournissant des informations essentielles pour la gestion horaire.

```
CreneauHoraire:
    'CreneauHoraire'idCreneau=ID':'
        ('Heure'':' heure=INT';'&
        'Type'':' type=TypeCours';'&
        'UE'':' ue=[UE]';'&
        'Enseignant'':' enseignant=[Enseignant]';'&
        'Salle'':' salle=[Salle]';')
    'End.';
```

L'heure associée à chaque créneau détermine précisément le moment de la journée où se déroule le cours ou l'activité. Il est à noter que l'heure est définie automatiquement pour une durée standard d'une heure par cours. Cette standardisation aide à maintenir une uniformité dans la planification des cours et des activités.

- **Définitions des Énumérations** Les énumérations *Campus*, *TypeCours*, et *NameJours* offrent des listes prédéfinies de valeurs pour certains attributs, assurant une cohérence et une validation des données.

```
enum TypeCours:
    COURS='Cours' | TD='TD' | TP='TP';
```

Figure 3: Énumérations pour les attributs *Campus*, *TypeCours*, et *NameJours*

- **Reconnaissance des Fichiers Inclus** La règle terminale *INCLUDE* est utilisée pour reconnaître les références aux fichiers externes avec l'extension *.emp*.

```
terminal INCLUDE: ID('.emp');
```

Figure 4: Règle terminale *INCLUDE* pour la reconnaissance de fichiers externes

## 3.3 Génération et fonctionnalités de l'éditeur

Dans cette sous-section, nous examinons comment Xtext génère automatiquement un éditeur riche en fonctionnalités pour le langage spécifié par la grammaire Xtext.

### 3.3.1 Génération Automatique de l'Éditeur

Xtext automatise plusieurs aspects du développement d'un éditeur pour un langage spécifique. La définition d'une grammaire dans Xtext entraîne la génération d'un éditeur de texte intégré à l'IDE Eclipse, incluant :

- **Analyse Syntaxique et Sémantique :** Xtext construit un parseur qui analyse le code écrit, reconnaît sa structure selon la grammaire, et construit un arbre syntaxique abstrait (AST).
- **Coloration Syntaxique :** L'éditeur met en évidence la syntaxe, améliorant ainsi la lisibilité et la compréhension du code.
- **Validation et Correction d'Erreur :** Le système intégré détecte et corrige les erreurs syntaxiques et sémantiques.

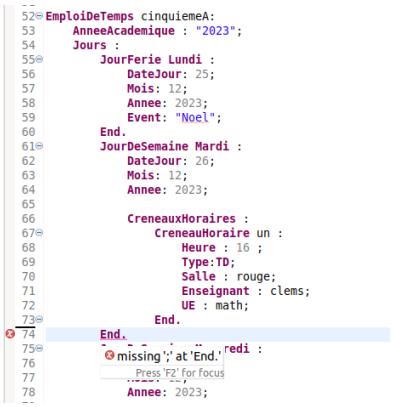


Figure 5: Test montrant une syntaxe d'AST colorée avec une erreur

### 3.3.2 Fonctionnalités Avancées de l'Éditeur

Outre les fonctionnalités de base, l'éditeur généré par Xtext offre des options avancées qui enrichissent l'expérience de développement, telles que :

- **Complétion de Code :** Suggestions de complétion basées sur la grammaire et le contexte.
- **Navigation et Liens :** Facilite la navigation dans le code avec des liens vers les définitions et les références.

- **Refactorisation :** Outils pour modifier le code existant en respectant les contraintes de la grammaire.
- **Support Multi-fichiers :** L'éditeur permet de travailler avec des modèles répartis sur plusieurs fichiers.

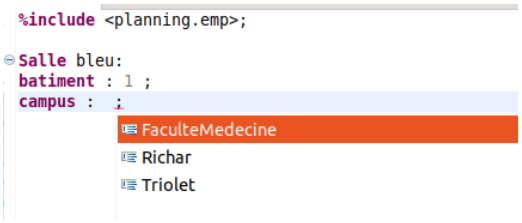


Figure 6: Fonctionnalités avancées de l'éditeur Xtext

## 3.4 Validation et contraintes métier

Pour gérer les contraintes de notre emploi du temps, nous allons utiliser une classe de validator. Nous allons trier nos méthodes par entité déclaré dans la grammaire. Nous avons déclaré une Map pour stocker les créneaux horaires par enseignant et par heure, et une liste de jours férié. Cette liste est fixe (final).

### 3.4.1 Contraintes sur les emploi du temps

Nos emplois du temps définissent une semaine, nous utilisons donc une énumération pour représenter les jours de la semaine. L'ID de chaque jour sera un élément unique de cette énumération.

```
// 7 JOURS DANS UN EMPLOI DU TEMPS
@Check
public void checkNombreJoursDansEmploiDuTemps(EmploiTemps emploiDuTemps) {
    if (emploiDuTemps.getJour().size() != 7) {
        error("Un emploi du temps doit être composé de 7 jours",
              EmpPackage.Literals.EMPLOI_TEMPS__ID_ET);
    }
}
```

Figure 7: Un emploi du temps est composé de 7 jours

### 3.4.2 Contraintes sur les enseignants

Les enseignants ne peuvent pas participer à deux cours différents en même temps. On utilise la Map qu'on a créé pour gérer les horaires de cours de chaque enseignants et ainsi rendre une erreur ou pas.

```
// UN ENSEIGNANT N'EST PAS SUR 2 CRENEAUX EN MÊME TEMPS
@Check
public void checkEnseignantsUniquesPourCreneaux(Model model) {
    System.out.println("Début de la vérification des enseignants uniques pour les créneaux");

    // Réinitialiser la map pour chaque exécution de validation
    creneauxParEnseignantEtHeure.clear();

    // Parcourir tous les emplois du temps dans le modèle
    for (EmploiTemps emploiDuTemps : model.getEmplois()) {
        for (Jour jour : emploiDuTemps.getJour()) {
            if (jour instanceof JourSemaine) {
                for (CreneauHoraire creneau : ((JourSemaine) jour).getCreneauHoraire()) {
                    Enseignant enseignant = creneau.getEnseignant();
                    int heure = creneau.getHeure();

                    // Vérifier si l'enseignant est déjà assigné à cette heure dans un autre emploi du temps
                    if (creneauxParEnseignantEtHeure.containsKey(enseignant) && creneauxParEnseignantEtHeure.get(enseignant).containsKey(heure)) {
                        error("Un enseignant ne peut pas être dans deux salles différentes à la même heure",
                                creneau,
                                EmpPackage.Literals.CRENEAU_HORAIRE__ID_CRENEAU);
                    } else {
                        // Ajouter le créneau à la map
                        creneauxParEnseignantEtHeure.computeIfAbsent(enseignant, k -> new HashMap<>()).put(heure, emploiDuTemps);
                    }
                }
            }
        }
    }

    System.out.println("Fin de la vérification des enseignants uniques pour les créneaux");
}
```

### 3.4.3 Contraintes sur les jours

Comme présenté précédemment, un jour peut être un jour de la semaine ou un jour férié, nous avons donc créé des validators en fonction de la date du jour. Il faut faire attention de gérer le traitement sur les jours de la semaine uniquement car les jour férié n'ont pas une liste de créneaux.

```
// UN JOUR DE LA SEMAINE N'EST PAS FERIE
@Check
public void checkJourNonFerie(JourSemaine jour) {
    String date = jour.getDateJour() + "-" + jour.getMois();
    if (JOURS_FERIES.contains(date)) {
        error("Ce jour est un jour férié : " + date,
              EmpPackage.Literals.JOUR__ID_JOUR);
    }
}

// SAMEDI ET DIMACHE PAS DE CRENEAUX
@Check
public void checkWeekendSansCreneaux(EmploiTemps emploiDuTemps) {
    EList<Jour> jours = emploiDuTemps.getJour();

    for (Jour jour : jours) {
        if (jour instanceof JourSemaine) {
            NameJours idJour = ((JourSemaine) jour).getIdJour();
            EList<CreneauHoraire> creneaux = ((JourSemaine) jour).getCreneauHoraire();

            if ((idJour == NameJours.SAMEDI || idJour == NameJours.DIMANCHE) && !creneaux.isEmpty()) {
                error("Les créneaux doivent être vides pour les samedis et dimanches",
                        jour,
                        EmpPackage.Literals.JOUR__ID_JOUR);
            }
        }
    }
}

// IL N'EXISTE PAS 2 FOIS LE MEME JOUR DANS LA SEMAINE
@Check
public void checkJoursUniquesDansEmploiDuTemps(EmploiTemps emploiDuTemps) {
    EnumSet<NameJours> joursUniques = EnumSet.noneOf(NameJours.class);
    EList<Jour> jours = emploiDuTemps.getJour();

    for (Jour jour : jours) {
        if (jour instanceof JourSemaine) {
            NameJours idJour = ((JourSemaine) jour).getIdJour();
            if (!joursUniques.add(idJour)) {
                error("La semaine ne peut pas avoir deux fois le même jour",
                        jour,
                        EmpPackage.Literals.JOUR__ID_JOUR);
            }
        }
    }
}
```

Figure 8: Un jour de la semaine n'est pas férié / Samedi et Dimanche ne contiennent pas de créneaux / Chaque jour est unique

```
// LES DATES SONT VALIDENT
@Check
public void checkValiditeDate(JourSemaine jour) {
    try {
        String dateString = jour.getDateJour() + "-" + jour.getMois() + "-" + jour.getAnnee();
        LocalDate.parse(dateString, DateTimeFormatter.ofPattern("dd-MM-yyyy"));
    } catch (DateTimeParseException e1) {
        try {
            String dateString = jour.getDateJour() + "-" + jour.getMois() + "-" + jour.getAnnee();
            LocalDate.parse(dateString, DateTimeFormatter.ofPattern("d-M-yyyy"));
        } catch (DateTimeParseException e2) {
            error("Date invalide : " + jour.getDateJour() + "-" + jour.getMois() + "-" + jour.getAnnee(),
                  EmpPackage.Literals.JOUR__ID_JOUR);
        }
    }
}

// PAS DEUX DATES DUPLIQUES
@Check
public void checkDatesUniquesDansEmploiDuTemps(EmploiTemps emploiDuTemps) {
    Set<String> datesUniques = new HashSet<>();
    EList<Jour> jours = emploiDuTemps.getJour();

    for (Jour jour : jours) {
        if (jour instanceof JourSemaine) {
            JourSemaine jourSemaine = (JourSemaine) jour;
            String date = jourSemaine.getDateJour() + "-" + jourSemaine.getMois() + "-" + jourSemaine.getAnnee();

            if (!datesUniques.add(date)) {
                error("Il ne peut pas y avoir deux jours de la semaine avec la même date: " + date,
                        jour,
                        EmpPackage.Literals.JOUR__DATE_JOUR);
            }
        }
    }
}
```

Figure 9: Les dates des jours sont valident et ne se dupliquent pas

### 3.4.4 Contraintes sur les créneaux

Les créneaux ne doivent pas dépasser les heures de travail d'une journée et doivent rester vide pour la pause de midi. Il suffit simplement d'établir des conditions sur les heures respectives de ces créneaux.

```
// JAMAIS 2 CRENAUX A LA MEME HEURE DANS LA MEME JOURNEE
@Check
public void checkCreneauxUniquesParJour(EmploiTemps emploiDuTemps) {
    EList<Jour> jours = emploiDuTemps.getJour();

    for (Jour jour : jours) {
        if (jour instanceof JourSemaine) {
            Set<Integer> heuresCreneaux = new HashSet<>();
            EList<CreneauHoraire> creneaux = ((JourSemaine) jour).getCreneauHoraire();

            for (CreneauHoraire creneau : creneaux) {
                if (!heuresCreneaux.add(creneau.getHeure())) {
                    error("Il ne peut pas y avoir deux créneaux à la même heure dans une même journée",
                              creneau, // l'objet spécifique du créneau horaire en double
                              EmpPackage.Literals.CRENEAU_HORAIRE__HEURE // l'attribut spécifique où l'erreur doit être affichée
                         );

                }
            }
        }
    }
}

// IMPOSSIBLE D'AJOUTER UN COUR ENTRE 12h ET 14h
@Check
public void checkHeureCreneau(CreneauHoraire creneau) {
    if (creneau.getHeure() == 12 || creneau.getHeure() == 13) {
        error("Impossible d'ajouter un créneaux de cours entre 12h et 14h", EmpPackage.Literals.CRENEAU_HORAIRE__HEURE);
    }
}

// IMPOSSIBLE D'AJOUTER UN COUR AVANT 8h ET APRES 17h
@Check
public void checkHeureCreneauDansPlageValide(CreneauHoraire creneau) {
    int heure = creneau.getHeure();
    if (heure < 8 || heure > 17) {
        error("Les cours commencent à 8h et finnissent à 18h",
              EmpPackage.Literals.CRENEAU_HORAIRE__HEURE);
    }
}
```

Figure 10: Les créneaux ne se superposent pas et les horaires des créneaux sont compris entre 8h et 17h avec une pause entre 12h et 14h

### 3.4.5 Contraintes sur les UE

Chaque UE à un volume horaire que chaque emploi du temps doit respecter. En fonction du nombre d'heure total d'un UE dans une semaine, nous pouvons afficher dans l'erreur si il manque des heures ou si il y en a trop. Nous modélisons aussi le fait qu'il ne peut pas y avoir plus de 2h du même cours dans une journée.

```
// LES VOLUMES HORAIRES DES UE SONT RESPECTE
@Check
public void checkVolumeHoraireUEParEmploiDuTemps(Model model) {
    System.out.println("Début de la vérification du volume horaire des UEs par emploi du temps");

    // Parcourir tous les emplois du temps dans le modèle
    for (EmploiTemps emploiDuTemps : model.getEmplois()) {
        // Créer une map pour suivre le total des heures pour chaque UE dans cet emploi du temps
        Map<UE, Integer> totalHeuresParUE = new HashMap<>();

        for (Jour jour : emploiDuTemps.getJour()) {
            if (jour instanceof JourSemaine) {
                for (CreneauHoraire creneau : ((JourSemaine) jour).getCreneauHoraire()) {
                    UE ue = creneau.getUe();
                    totalHeuresParUE.put(ue, totalHeuresParUE.getOrDefault(ue, 0) + 1);
                }
            }
        }

        // Vérifier si le total des heures pour chaque UE correspond à son volume horaire
        for (Map.Entry<UE, Integer> entry : totalHeuresParUE.entrySet()) {
            UE ue = entry.getKey();
            int totalHeures = entry.getValue();

            if (totalHeures < ue.getVolumeHoraire()) {
                error("Pas assez d'heures pour l'UE " + ue.getNomUE() + " dans l'emploi du temps " + emploiDuTemps.getIdET() +
                      ". Heures assignées : " + totalHeures + ", Volume horaire requis : " + ue.getVolumeHoraire(),
                      ue,
                      EmpPackage.Literals.UE__VOLUME_HORAIRE);
            } else if (totalHeures > ue.getVolumeHoraire()) {
                error("Trop d'heures pour l'UE " + ue.getNomUE() + " dans l'emploi du temps " + emploiDuTemps.getIdET() +
                      ". Heures assignées : " + totalHeures + ", Volume horaire requis : " + ue.getVolumeHoraire(),
                      ue,
                      EmpPackage.Literals.UE__VOLUME_HORAIRE);
            }
        }
    }

    System.out.println("Fin de la vérification du volume horaire des UEs par emploi du temps");
}
```

Figure 11: Les volumes horaires des UE sont respecté

```java
// PAS PLUS DE 2h DE LA MEME UE
@Check
public void checkMaximumDeuxCreneauxParUEParJour(EmploiTemps emploiDuTemps) {
    EList<Jour> jours = emploiDuTemps.getJour();

    for (Jour jour : jours) {
        if (jour instanceof JourSemaine) {
            Map<UE, Integer> compteUE = new HashMap<>();
            EList<CreneauHoraire> creneaux = ((JourSemaine) jour).getCreneauHoraire();

            for (CreneauHoraire creneau : creneaux) {
                UE ue = creneau.getUe();
                if (compteUE.containsKey(ue)) {
                    compteUE.put(ue, compteUE.get(ue) + 1);
                } else {
                    compteUE.put(ue, 1);
                }

                if (compteUE.get(ue) > 2) {
                    error("Il ne peut pas y avoir plus de deux créneaux de la même UE dans une journée",
                            creneau,
                            EmpPackage.Literals.CRENEAU_HORAIRE__ID_CRENEAU);
                }
            }
        }
    }
}
```

Figure 12: Une UE n'apparaît pas plus de deux fois dans la même journée

## 3.5 Exemples d'emplois du temps générés

Cliquez **ici** pour accéder à la vidéo de présentation. Activez le son pour entendre les explications.

```
%include <planning.emp>;
%include <salles.emp>;
%include <enseignants.emp>;
%include <ues.emp>;

//////////////////////
/////// SALLE ////////
//////////////////////

Salle triolet_36_203:
batiment : 36 ;
campus : Triolet ;
numero : 203 ;
End.

Salle triolet_36_205:
batiment : 36 ;
campus : Triolet ;
numero : 205 ;
End.

Salle triolet_36_404:
batiment : 36 ;
campus : Triolet ;
numero : 404 ;
End.
```

```
27
28  Salle medecine_3_3:
29  batiment : 3 ;
30  campus : FaculteMedecine ;
31  numero : 3 ;
32  End.
33
34  ////////////////////////
35  /////// ENSEIGNANT ///////
36  ////////////////////////
37
38  Enseignant tony_stark :
39  nom : "Stark";
40  prenom: "Tony";
41  End.
42
43  Enseignant eric_dupont :
44  nom : "Durand";
45  prenom: " ric ";
46  End.
47
48  Enseignant xavier_dupont :
49  nom : "Dupont";
50  prenom: "Xavier";
51  End.
52
53  Enseignant john_doe :
54  nom : "Doe";
55  prenom: "John";
56  End.
57
58
59  ////////////////////
60  /////// UE ////////
61  ////////////////////
62
63  UE mathematique:
64  nom : "Mathematiques";
65  volumeHoraire: 4;
66  End.
67
68  UE anglais:
69  nom : "Anglais";
70  volumeHoraire : 5;
71  End.
72
73  UE informatique:
74  nom : "Informatique";
75  volumeHoraire : 2;
76  End.
77
```

```
UE francais:
nom : "Francais";
volumeHoraire : 5;
End.

//////////////////////
/////// EDT ////////
//////////////////////

EmploiDeTemps cinquieme_A:
    AnneeAcademique : "2024";
    Jours :
        JourDeSemaine Lundi :
            DateJour: 05;
            Mois: 05;
            Annee: 2024;

            CreneauxHoraires :
                CreneauHoraire cours1:
                    Enseignant: eric_dupont;
                    Heure: 8;
                    Salle: triolet_36_203;
                    Type: Cours;
                    UE: francais;
                End.
                CreneauHoraire cours2:
                    Enseignant: xavier_dupont;
                    Heure: 9;
                    Salle: triolet_36_205;
                    Type: Cours;
                    UE: mathematique;
                End.
                CreneauHoraire cours3:
                    Enseignant: xavier_dupont;
                    Heure: 10;
                    Salle: triolet_36_205;
                    Type: Cours;
                    UE: mathematique;
                End.
                ;
        End.
        JourDeSemaine Mardi :
            DateJour: 06;
            Mois: 05;
            Annee: 2024;

            CreneauxHoraires :
                CreneauHoraire cours1:
                    Enseignant: eric_dupont;
                    Heure: 8;
                    Salle: triolet_36_203;
```

```
                    Type: Cours;
                    UE: francais;
                End.
                CreneauHoraire cours1:
                    Enseignant: eric_dupont;
                    Heure: 9;
                    Salle: triolet_36_205;
                    Type: Cours;
                    UE: francais;
                End.;
        End.
        JourDeSemaine Mercredi :
            DateJour: 07;
            Mois: 05;
            Annee: 2024;

            CreneauxHoraires :
                CreneauHoraire cours1:
                    Enseignant: eric_dupont;
                    Heure: 8;
                    Salle: triolet_36_404;
                    Type: Cours;
                    UE: francais;
                End.
                CreneauHoraire cours2:
                    Enseignant: tony_stark;
                    Heure: 9;
                    Salle: triolet_36_205;
                    Type: Cours;
                    UE: informatique;
                End.
                CreneauHoraire cours2:
                    Enseignant: tony_stark;
                    Heure: 10;
                    Salle: triolet_36_205;
                    Type: Cours;
                    UE: informatique;
                End.
            ;
        End.
        JourFerie Jeudi :
            DateJour: 08;
            Mois: 05;
            Annee: 2024;

            Event: "1945";
        End.
        JourDeSemaine Vendredi :
            DateJour: 09;
            Mois: 05;
            Annee: 2024;
```

```
180
181             CreneauxHoraires :
182                 CreneauHoraire cours1:
183                     Enseignant: xavier_dupont;
184                     Heure: 14;
185                     Salle: triolet_36_205;
186                     Type: Cours;
187                     UE: mathematique;
188                 End.
189                 CreneauHoraire cours2:
190                     Enseignant: xavier_dupont;
191                     Heure: 15;
192                     Salle: triolet_36_205;
193                     Type: Cours;
194                     UE: mathematique;
195                 End.
196                 CreneauHoraire cours3:
197                     Enseignant: eric_dupont;
198                     Heure: 8;
199                     Salle: triolet_36_404;
200                     Type: Cours;
201                     UE: francais;
202                 End.
203             ;
204         End.
205         JourDeSemaine Samedi :
206             DateJour: 10;
207             Mois: 05;
208             Annee: 2024;
209 
210             CreneauxHoraires :;
211         End.
212         JourDeSemaine Dimanche :
213             DateJour: 11;
214             Mois: 05;
215             Annee: 2024;
216 
217             CreneauxHoraires :;
218         End.
219     ;
220 End.
```

# 4 Conclusion et perspectives

## 4.1 Conclusion générale

En s'appuyant sur la puissance et la flexibilité de Xtext, nous avons conçu un métamodèle robuste et une grammaire intuitive pour représenter notre emploi du temps et ses contraintes.

L'éditeur généré automatiquement par Xtext offre une expérience utilisateur agréable, avec des fonctionnalités telles que la coloration syntaxique, l'autocomplétion, et des validations en temps réel, facilitant ainsi la tâche de potentiels utilisateurs.

Les contraintes métier intégrées, telles que la gestion des créneaux horaires et des jours fériés, assurent que les emplois du temps générés sont non seulement pratiques mais aussi conformes à nos exigences.

## 4.2 Perspectives et améliorations futures

Pour améliorer davantage le projet de création d'emplois du temps académique avec Xtext, plusieurs axes d'amélioration peuvent être envisagés :

- **Extension du Métamodèle** : Élargir le métamodèle pour inclure des éléments supplémentaires, comme la prise en compte de semaines de travail irrégulières (semaine A / B), ou l'ajout de périodes spéciales (examens, vacances).
- **Amélioration de l'Interface Utilisateur** : Développer une interface graphique plus intuitive pour faciliter la saisie et la modification des emplois du temps, en particulier pour les utilisateurs moins familiarisés avec les langages de modélisation.
- **Personnalisation et Flexibilité** : Permettre une plus grande personnalisation des emplois du temps, avec des options pour différentes durées de cours, des pauses variables, ou des formats de semaine alternatifs.
- **Tests et Assurance Qualité** : Création de processus de test, en incluant des tests automatisés et des scénarios d'utilisation réels, pour assurer la fiabilité et la robustesse du système.