Situation professionnelle n°2

CONTEXTE GALAXY SWISS-BOURDIN

APPLICATION EN C#

GESTION DES ETATS DES FICHES DE FRAIS

Table des matières

1. RAPPEL DU CONTEXTE & SPECIFICATIONS

1.1. Rappel du contexte

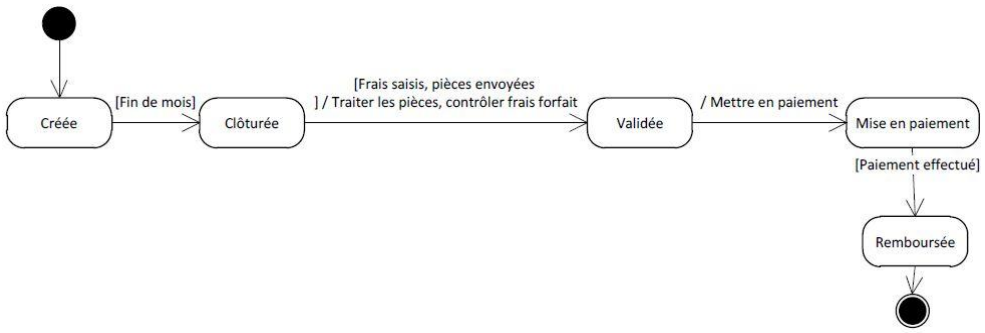
Galaxy Swiss-Bourdin (dit GSB) est un laboratoire de fabrication de médicaments. Afin de faire connaître ses produits aux praticiens de la médecine, l'entreprise emploie des visiteurs médicaux. Ces derniers sont chargés de promouvoir les produits de l'entreprise aux médecins : ils ne peuvent pas vendre quoique ce soit, simplement inciter les médecins à prescrire les médicaments Galaxy Swiss-Bourdin à leurs patients.

Les visiteurs sont répartis sur tout le territoire français. Leur activité engendre des frais, que les visiteurs doivent avancer eux-mêmes. Ces frais sont ensuite remboursés par l'entreprise GSB. Pour être remboursés, les visiteurs doivent compléter, chaque mois, des fiches de frais sur le site de l'application de gestion de frais, qui renseignent l'état de leurs dépenses. Les comptables contrôlent, valident et mettent en paiement les fiches des visiteurs sur cette même application Web.

Les fiches de frais passent par plusieurs états, qui représentent l'évolution de leur traitement :

- **Saisie en cours** : le visiteur remplit sa fiche de frais ;
-**Clôturée** : le visiteur a fini de remplir sa fiche de frais, celle-ci est disponible pour les comptables qui peuvent la contrôler et la valider. Les fiches sont clôturées entre le 1er et le 10 du mois suivant.
-**Validée et mise en paiement** : la fiche de frais a été validée par un comptable et les remboursements peuvent être effectués pour le visiteur. Les fiches sont mises en paiement le 20 du mois suivant.
-**Remboursée** : la fiche de frais a été remboursée.

La vie d'une fiche de frais



1.2. Spécifications

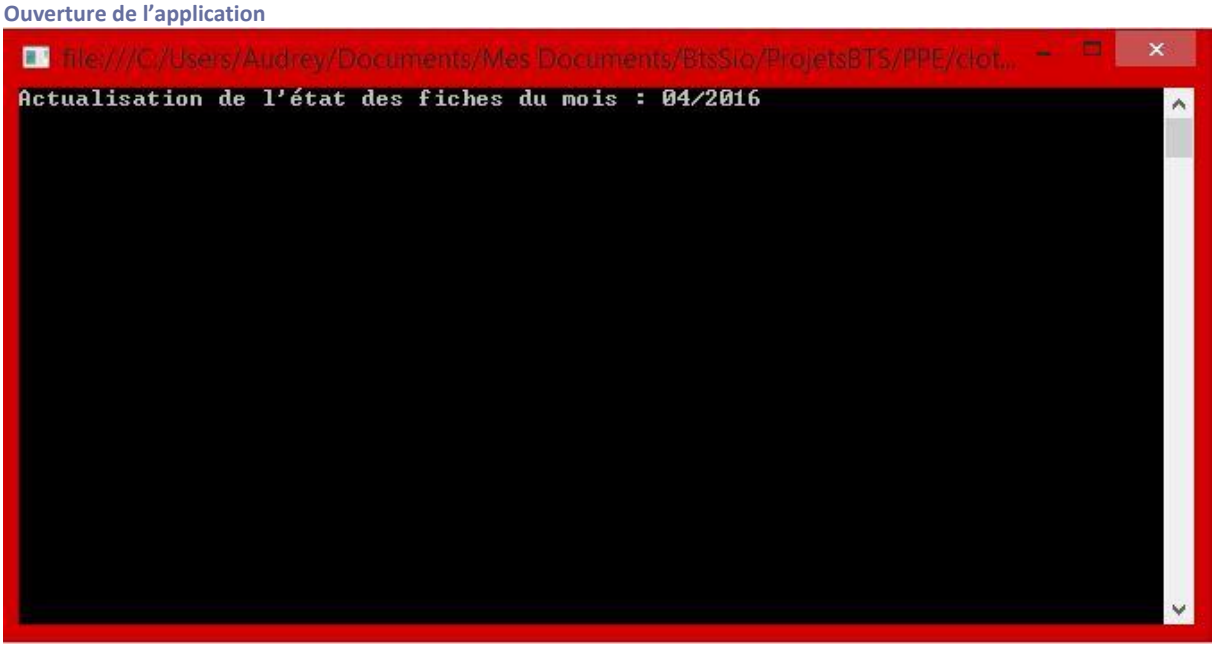
Une application C# doit être développée afin de remplir deux objectifs :

- Clôturer automatiquement les fiches du mois précédent dont la saisie est encore en cours afin de permettre aux comptables de les contrôler.
- Mettre en paiement automatiquement les fiches du mois précédent qui ont été clôturées à partir du vingtième jour du mois actuel.

2. PRINCIPE DE L'APPLICATION

L'application n'affiche qu'une console qui indique le démarrage et le mois qui est concerné par le changement d'état des fiches (donc le mois précédant le mois actuel).

Ouverture de l'application



Le temps où l'application reste ouverte, elle met à jour régulièrement et automatiquement les états des fiches de frais.

2.1. Les fiches de frais voient leur état passer à « Clôturée »

SI ET SEULEMENT SI
- leur mois est égal au mois précédant le mois en cours
ET
-leur état initial était « En cours de saisie ».
AINSI
- Aucune fiche de frais mise en paiement ou remboursée ne risque de passer à nouveau clôturée.
-Aucune fiche de frais dont la saisie est en cours pour le mois en cours ne peut être clôturée.
-Bien que les fiches soient clôturées entre le 1 et le 10 du mois, il semblait inutile que ce soit ici une condition, les résultats étant les mêmes lors d'une ouverture entre le 1 et le 10 du mois. En revanche, il me semblait plus intéressant de laisser le script s'appliquer même après ces dates, car en cas de retard du service comptabilité, certaines fiches en cours de saisie n'auraient jamais été clôturées.

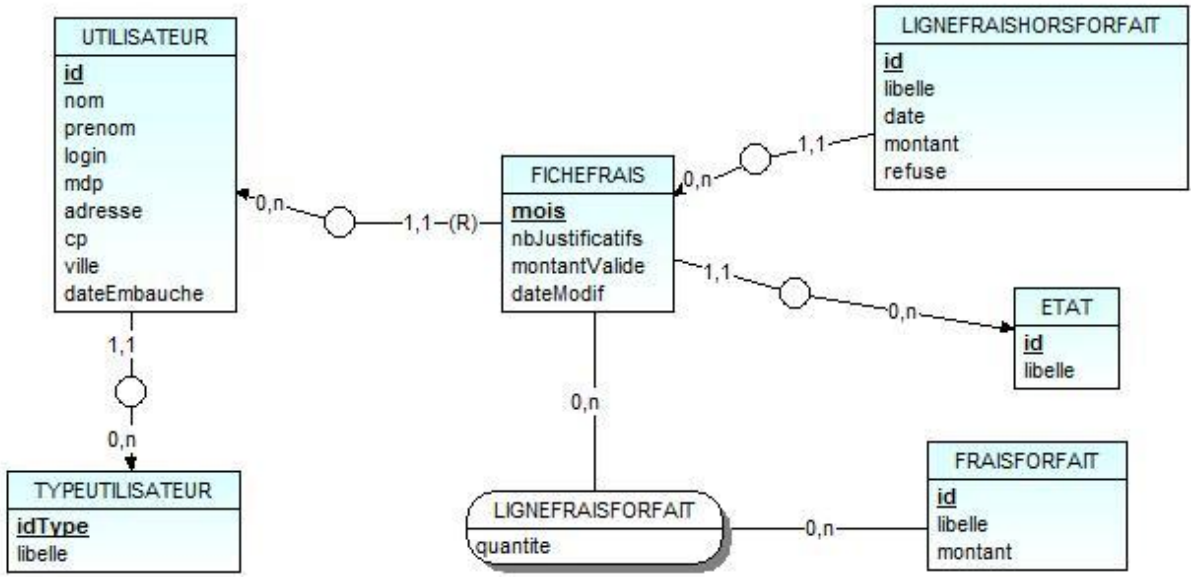
2.2. Les fiches de frais voient leur état passer à « Validée et mise en paiement »

SI ET SEULEMENT SI
-leur mois est égal au mois précédant le mois en cours.
ET
-leur état initial était « Clôturée ».
ET
-le numéro du jour actuel est égal ou supérieur à 20.
AINSI
-Aucune fiche de frais n'est mise en paiement avant le 20 du mois.
-Aucune fiche non clôturée n'est mise en paiement.
-Aucune fiche ne correspondant pas au mois précédent du mois en cours ne peut être mise en paiement.

3. BASE DE DONNEES

La base de données est la même que celle utilisée pour l'application de gestion des frais en PHP (situation professionnelle n°1).

SCD de la base de données GSB_frais

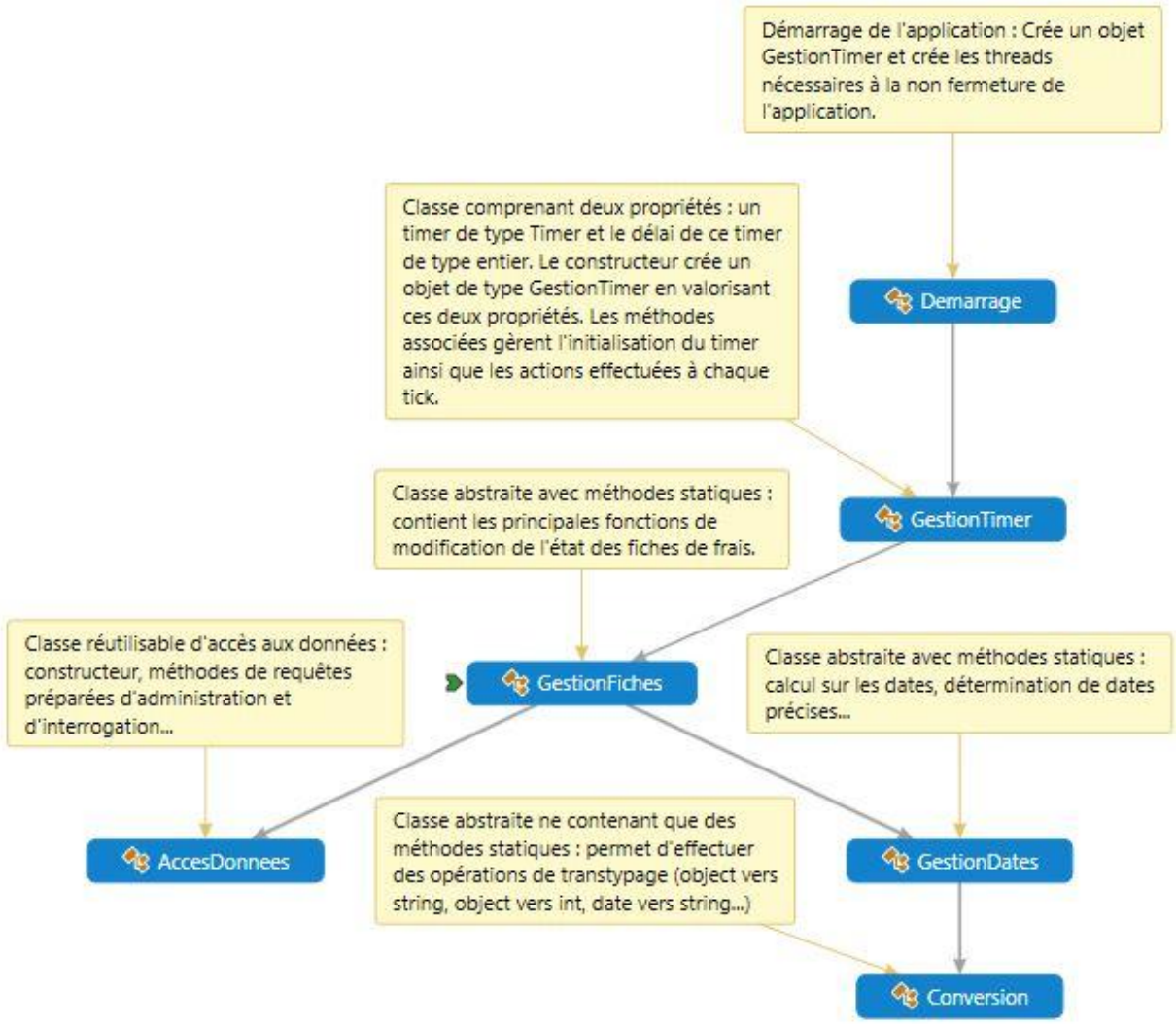


4. ARCHITECTURE DE L'APPLICATION

4.1. Architecture générale

L'application a été développée en programmation objet (C#). Elle se compose de plusieurs classes, chacune remplissant un rôle bien précis. Le code a été optimisé de façon à éviter le plus possible les répétitions.

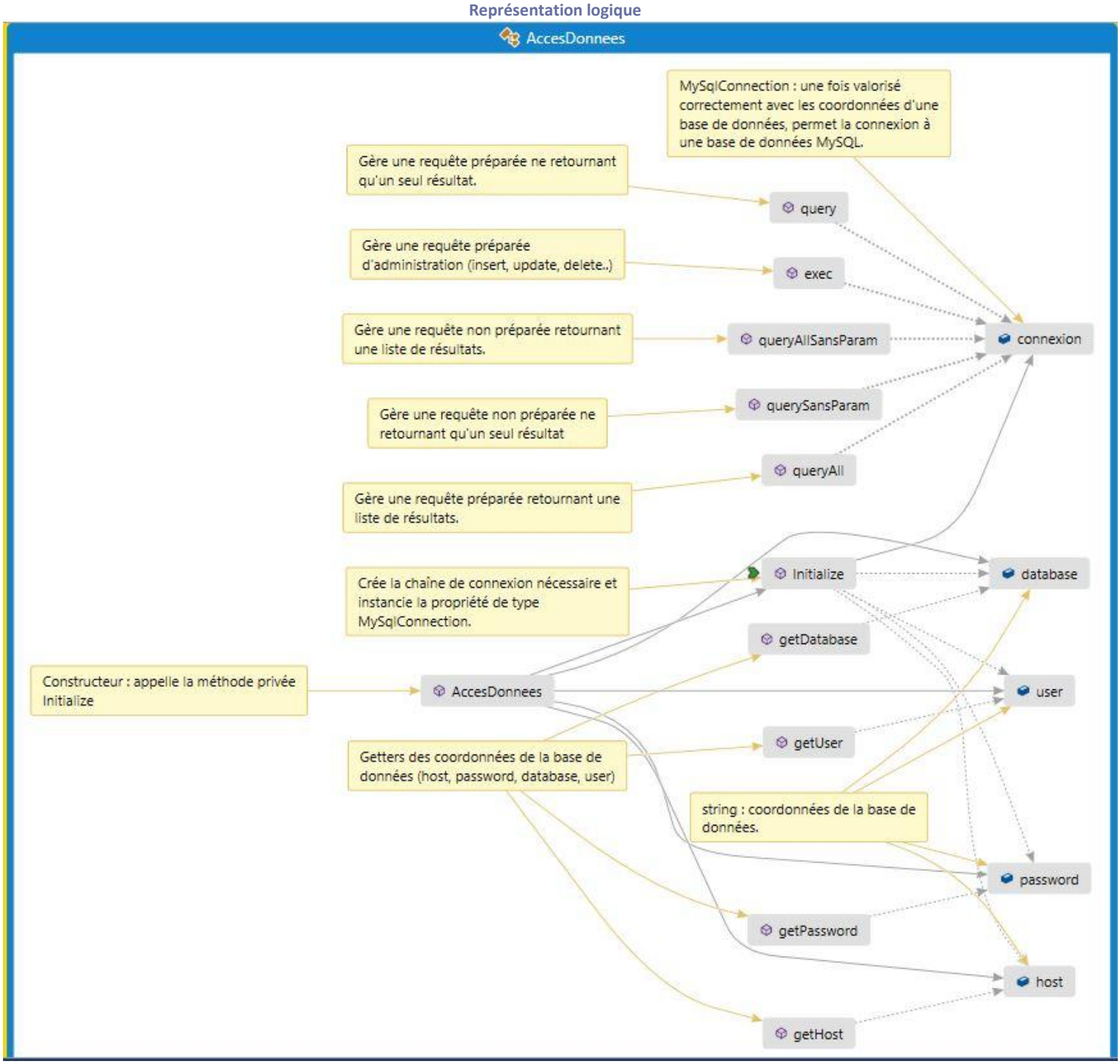
Diagramme explicatif des classes



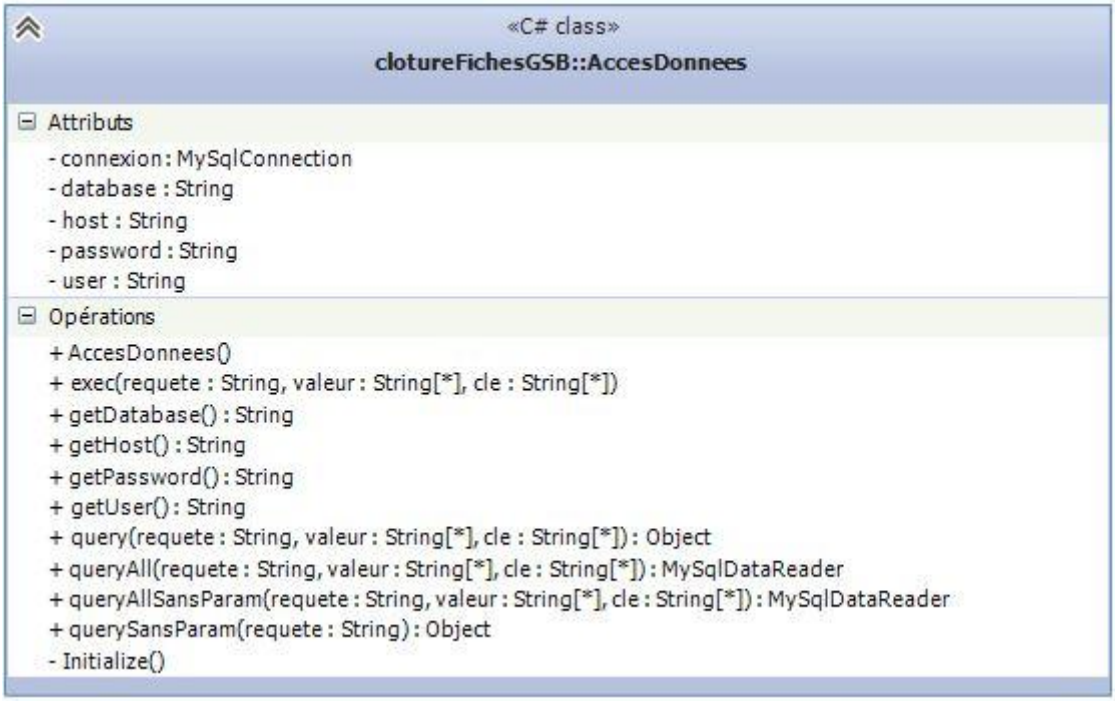
4.2. Classe AccesDonnees

Cette classe se charge de permettre un accès à une base de données MySQL. Elle est totalement réutilisable. Elle comprend les fonctionnalités essentielles de connexion, de requêtes d'interrogation pour un ou plusieurs résultats, préparées ou non ainsi que les requêtes d'administration préparées.

Représentation logique



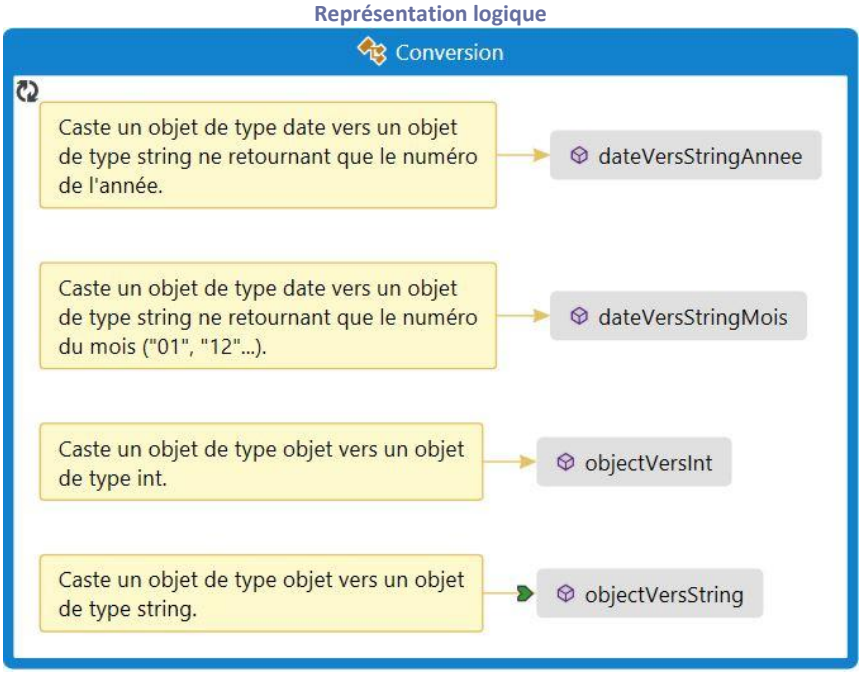
Représentation technique



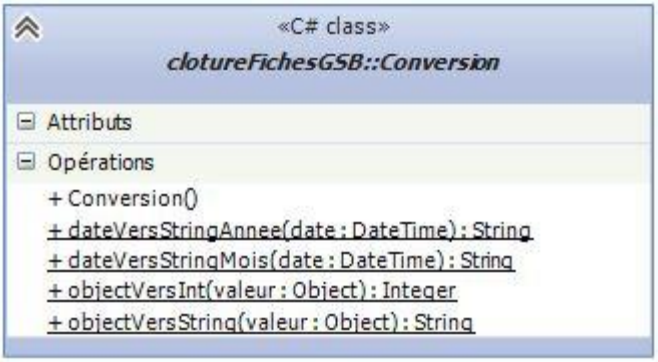
4.3. Classe Conversion

Cette classe est abstraite et ne contient que des méthodes statiques. Ses méthodes permettent de caster un type d'objet vers un autre type d'objet.

Représentation logique



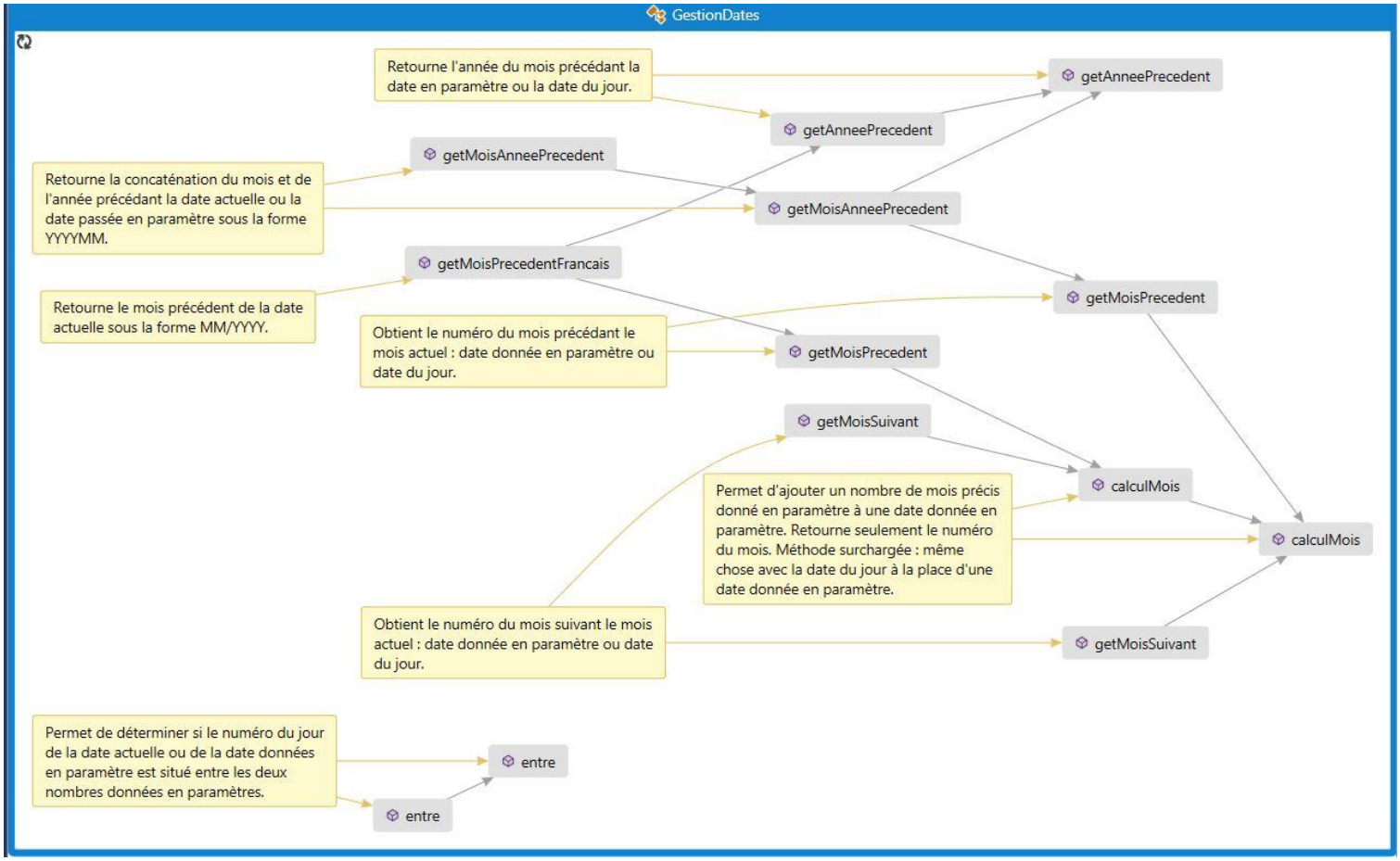
Représentation technique



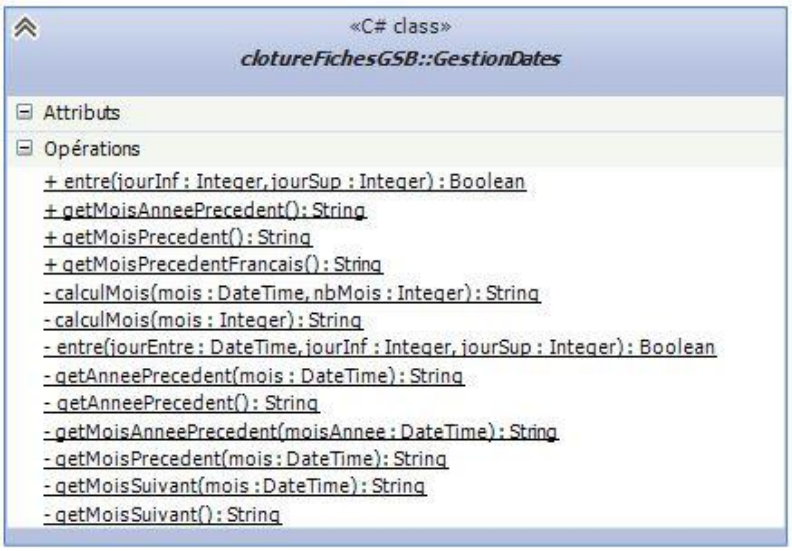
4.4. Classe GestionDates

Cette classe est abstraite et ne comprend que des méthodes statiques. Ces méthodes permettent des opérations sur les dates : calculs, récupération de parties spécifiques de la date, situer une date…

Représentation logique



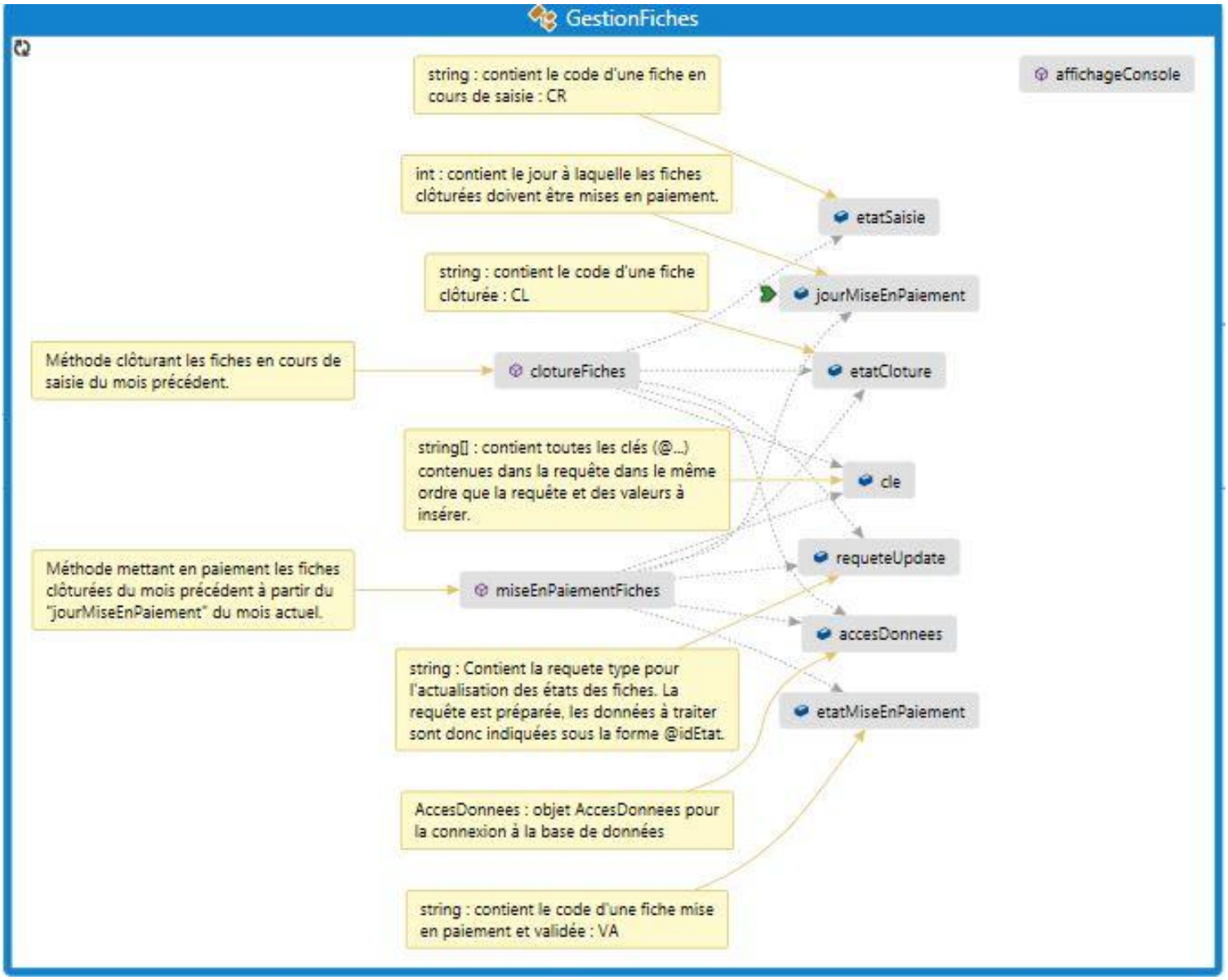
Représentation technique



4.5. Classe GestionFiches

Cette classe est abstraite et ne contient que des méthodes statiques. Elle possède les deux méthodes permettant d'actualiser l'état des fiches de frais en base de données. Ces méthodes utilisent les méthodes statiques des classes GestionDates et AccesDonnees afin de créer les requêtes permettant cette mise à jour.

Représentation logique

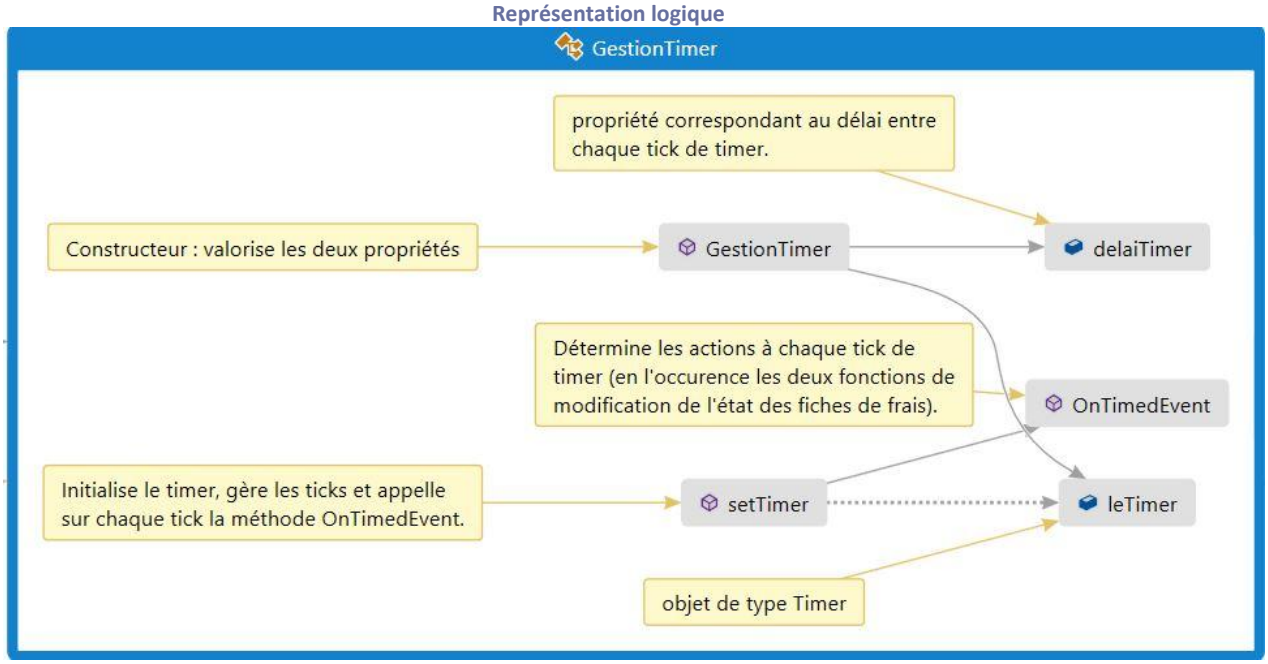


Représentation technique

4.6. Classe GestionTimer

Cette classe comprend tous les éléments relatifs à la répétitivité de l'application. Elle permet de définir un timer avec un délai de répétition et de gérer les évènements relatifs à ce timer.

Représentation logique



Représentation technique

4.7. Démarrage de l'application : Classe démarrage

Cette classe est statique et comprend une seule et unique méthode : Main(). Cette méthode est celle qui s'exécute au lancement de l'application. Elle crée et lance le timer, puis s'assure que l'application reste correctement ouverte.

5. TESTS UNITAIRES

Les tests unitaires ont été réalisés sur les classes GestionDates, Conversion et AccesDonnees grâce à l'utilitaire NUnit. A cet effet, les classes équivalentes suffixées de « Tests » ont été créées.

Exemple de classe de test : classe GestionDatesTests

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using NUnit.Framework;
using clotureFichesGSB;

namespace NUnit
{
    [TestFixture]
    0 références
    public class GestionDatesTests
    {
        /// <summary>
        /// Test mois précédent avec la date du jour
        /// </summary>
        [Test]
        0 références
        public void getMoisPrecedentTest1()
        {
            Assert.AreEqual("04", clotureFichesGSB.GestionDates.getMoisPrecedent());
        }
```

A leur exécution, les tests génèrent un fichier XML comprenant le rapport complet de ces tests (ce rapport est disponible dans la documentation du dossier de la situation professionnelle).

Exemple de rapport de test avec NUnit

```
<results>
  <test-suite type="TestFixture" name="AccesDonneesTests" executed="True" result="Success" success="True" time="0.165" asserts="0">
    <results>
      <test-case name="NUnit.AccesDonneesTests.getDatabase" executed="True" result="Success" success="True" time="0.157" asserts="1" />
      <test-case name="NUnit.AccesDonneesTests.getHost" executed="True" result="Success" success="True" time="0.000" asserts="1" />
      <test-case name="NUnit.AccesDonneesTests.getPassword" executed="True" result="Success" success="True" time="0.000" asserts="1" />
      <test-case name="NUnit.AccesDonneesTests.getUser" executed="True" result="Success" success="True" time="0.000" asserts="1" />
    </results>
  </test-suite>
  <test-suite type="TestFixture" name="ConversionTests" executed="True" result="Success" success="True" time="0.003" asserts="0">
    <results>
      <test-case name="NUnit.ConversionTests.dateVersStringAnnee" executed="True" result="Success" success="True" time="0.001" asserts="1" />
      <test-case name="NUnit.ConversionTests.dateVersStringMois" executed="True" result="Success" success="True" time="0.000" asserts="1" />
    </results>
  </test-suite>
  <test-suite type="TestFixture" name="GestionDatesTests" executed="True" result="Success" success="True" time="0.013" asserts="0">
    <results>
      <test-case name="NUnit.GestionDatesTests.entreTest1" executed="True" result="Success" success="True" time="0.001" asserts="1" />
      <test-case name="NUnit.GestionDatesTests.entreTest2" executed="True" result="Success" success="True" time="0.001" asserts="1" />
      <test-case name="NUnit.GestionDatesTests.entreTest3" executed="True" result="Success" success="True" time="0.000" asserts="1" />
      <test-case name="NUnit.GestionDatesTests.entreTest4" executed="True" result="Success" success="True" time="0.001" asserts="1" />
      <test-case name="NUnit.GestionDatesTests.getAnneePrecedentTest1" executed="True" result="Success" success="True" time="0.000" asserts="1" />
      <test-case name="NUnit.GestionDatesTests.getAnneePrecedentTest2" executed="True" result="Success" success="True" time="0.001" asserts="1" />
      <test-case name="NUnit.GestionDatesTests.getAnneePrecedentTest3" executed="True" result="Success" success="True" time="0.000" asserts="1" />
      <test-case name="NUnit.GestionDatesTests.getMoisAnneePrecedentTest1" executed="True" result="Success" success="True" time="0.001" asserts="1" />
      <test-case name="NUnit.GestionDatesTests.getMoisAnneePrecedentTest2" executed="True" result="Success" success="True" time="0.000" asserts="1" />
      <test-case name="NUnit.GestionDatesTests.getMoisAnneePrecedentTest3" executed="True" result="Success" success="True" time="0.000" asserts="1" />
      <test-case name="NUnit.GestionDatesTests.getMoisPrecedentTest1" executed="True" result="Success" success="True" time="0.000" asserts="1" />
      <test-case name="NUnit.GestionDatesTests.getMoisPrecedentTest2" executed="True" result="Success" success="True" time="0.000" asserts="1" />
      <test-case name="NUnit.GestionDatesTests.getMoisPrecedentTest3" executed="True" result="Success" success="True" time="0.000" asserts="1" />
      <test-case name="NUnit.GestionDatesTests.getMoisSuivantTest1" executed="True" result="Success" success="True" time="0.000" asserts="1" />
      <test-case name="NUnit.GestionDatesTests.getMoisSuivantTest2" executed="True" result="Success" success="True" time="0.000" asserts="1" />
      <test-case name="NUnit.GestionDatesTests.getMoisSuivantTest3" executed="True" result="Success" success="True" time="0.000" asserts="1" />
    </results>
```

6. DOCUMENTATION

La documentation est générée automatiquement par Visual Studio dans le projet, sous forme de fichier XML. La documentation de l'application ainsi que celle des tests ont été générées (disponibles dans la documentation du dossier de la situation professionnelle).

Exemple de fichier XML de documentation

```xml
<?xml version="1.0"?>
<doc>
    <assembly>
        <name>clotureFichesGSB</name>
    </assembly>
    <members>
        <member name="M:clotureFichesGSB.AccesDonnees.#ctor">
            <summary>
            Constructeur de la classe AccesDonnees : crée un objet d'accès à la base de données
            </summary>
        </member>
        <member name="M:clotureFichesGSB.AccesDonnees.Initialize">
            <summary>
             Méthode pour initialiser la connexion.
            </summary>
        </member>
        <member name="M:clotureFichesGSB.AccesDonnees.getHost">
            <summary>
            Get Serveur.
            </summary>
            <returns></returns>
        </member>
        <member name="M:clotureFichesGSB.AccesDonnees.getPassword">
            <summary>
            Get Mot de passe.
            </summary>
            <returns></returns>
        </member>
        <member name="M:clotureFichesGSB.AccesDonnees.getUser">
            <summary>
            Get Utilisateur.
            </summary>
            <returns></returns>
        </member>
        <member name="M:clotureFichesGSB.AccesDonnees.getDatabase">
```

7. OUTILS UTILISES

- Langages utilisés : **C#** ;

- IDE : **Visual C# Express** (application et tests)**, Visual Studio 2013 Ultimate** (diagramme et documentation) ;
- SGBD : **MySQL** ;
- Environnement de développement : **Wamp Server** ;

- Outil de gestion de versions : **Git** et **Github** ;
- Outil de tests unitaires : **NUnit** ;
- Outil de génération de documentation : **Outil intégré à Visual Studio**;
- Ouverture des fichiers XML : **Sublime Text 2** ;

- Création des SCD pour la documentation : **Win Design**.