Chapitre III

Projet Cabinet Infirmier

NB : Tous les documents nécessaires à la réalisation de ce projet vous sont fournis sous forme de documents téléchargeables sur la plate-forme.

1 Le cabinet infirmier

1.1 Présentation générale

Le cabinet infirmier *Soins à Grenoble* souhaite organiser les déplacements de ses infirmier(e)s. Il contacte pour cela la société *L3M Agency* pour décider avec elle d'un logiciel capable d'organiser et d'optimiser ses déplacements. Le cabinet emploie aujourd'hui une secrétaire médicale et 3 infirmières, mais est susceptible de s'agrandir. Chaque jour, une infirmière fait le tour de ses patients pour des soins. Chaque jour, la secrétaire médicale entre la liste des patients à visiter ainsi que leurs coordonnées et toute information utile aux soins. Elle affecte ensuite les patients à chacune des infirmières. Lorsqu'une infirmière s'identifie sur l'application, elle obtient la liste des patients qu'elle doit visiter dans la journée, ordonnés de façon à optimiser son trajet quotidien. Elle peut également obtenir la facture correspondant à chaque patient.

1.2 Organisation d'une application exploitant ces données

L'application qui pourrait être développée se présenterait comme un Web Service. Le Web Service permettra aux soignants de se connecter à distance à leur serveur métier pour obtenir divers services, comme la facturation client, et renseignements, comme la liste des patients à visiter ou le trajet optimal pour le visites. Des méthodes d'optimisation d'itinéraire telles que celles qui sont développées en Programmation avec Contraintes et Recherche Opérationnelle (PRCO) et une interface (telle que vous la développeriez en IHM), pouvant se connecter au serveur SOAP, pourraient être utilisés dans le cadre de ce projet, lors de sa finalisation, mais ceci est facultatif.

1.3 Scénario d'utilisation d'une application exploitant ces données

Deux types de clients peuvent s'adresser au serveur: le/la secrétaire médical(e) ou bien un(e) infirmièr(e). L'action du/de la secrétaire médical(e) sera intégralement prise en charge en IHM via le système d'interface. Le serveur métier, lui, va intervenir pour la gestion des requêtes des infirmièr(e)s afin d'obtenir le trajet de la tournée de la journée:

- le serveur reçoit une requête d'un client "infirmière" (via l'IHM)
- le serveur métier fourni la réponse attendue: il lui transmet la requête et le contenu du fichier XML, ou une page HTML
- le module du serveur métier (que vous pourriez développer en FDD-XML) construit une requête à emettre vers GoogleMapAPI pour obtenir les matrices de distances des adresses des patients.

- Le serveur envoie cette requête à GoogleMap API et récupère la réponse sous forme de matrices de distances dans un fichier XML
- le module du serveur métier lit les informations obtenues et appelle les fonctions d'optimisation développées en Recherche Opérationnelle pour calculer le meilleur trajet.
- le serveur métier renvoie ce résultat dans la page résultat de l'application cliente (l'IHM).

2 Objectifs du projet

Dans ce projet, vous allez :

- modéliser la partie données d'un Cabinet Infirmier (UML et XMLSchema)
- développer un programme permettant de vérifier la validité des données XML par rapport au schéma que vous aurez défini
- créer des transformations XSLT permettant d'extraire des connaissances spécifiques (par exemple les données d'un patient particulier) ou de présenter des données sous forme de pages HTML (comme par exemple la liste des interventions d'une infirmière)
- apprendre à utiliser les parseurs et XPath pour extraire des connaissances depuis le fichier XMLSchema
- apprendre à sérialiser ces données
- développer un mini serveur Rest (HTTP) pour accéder à ces données

3 Le document XML

L'application proposée doit stocker les informations des patients et du cabinet. Vous allez donc créer un langage XML pour le stockage de ces informations, c'est à dire modéliser le cabinet XML avec un nouveau Schema XML, à partir des connaissances contenues dans ce document XML.

Il a été convenu que le même document XML stockerait les informations du cabinet (son nom, son adresse et les identifiants des infirmières) ainsi que les données des patients.

3.1 Données du cabinet

Un première partie de ce document XML est représentée dans la figure III .1

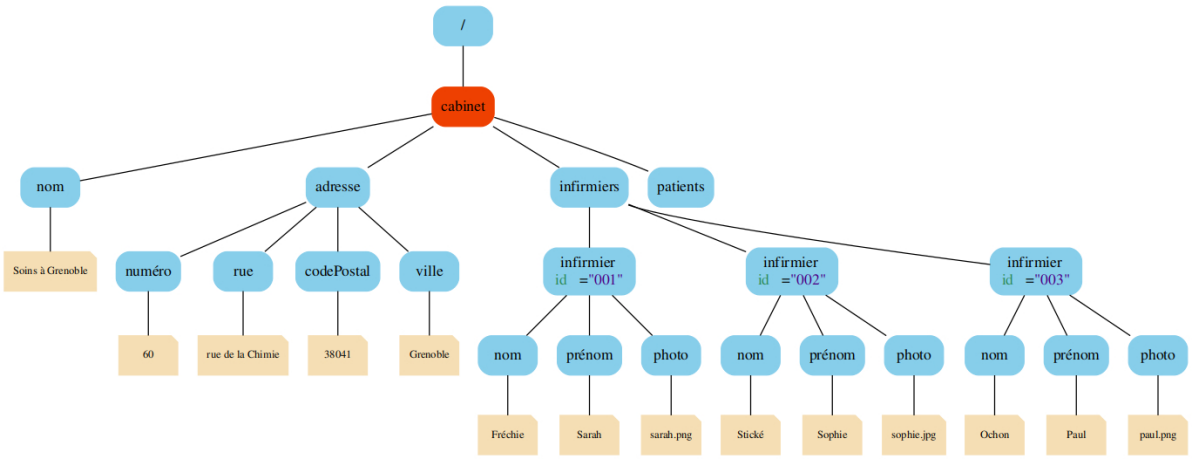

Figure III .1: **Arbre d'instance du cabinet infirmier (données du cabinet)**.

3.2 Données des patients

Après de longues discussions avec le cabinet infirmier et la rédaction du cahier des charges, vous en avez déduit que les informations à rentrer pour chaque patient devait être:

Des données administratives :

- son nom
- son prénom
- sa date de naissance
- son numéro de sécurité sociale
- son adresse précise

Ce document doit aussi comporter, pour chaque patient, et pour chaque visite, sa date, le(s) soin(s) à effectuer par l'infirmière ainsi que son(leur) code NGAP.

Enfin, le(a) secrétaire médical(e) affecte pour chaque patient un(e) infirmièr(e). Cette personne sera représentée par son identifiant (à déterminer).

Un exemple de patient avec toutes les informations précédentes est représenté dans la figure III .2

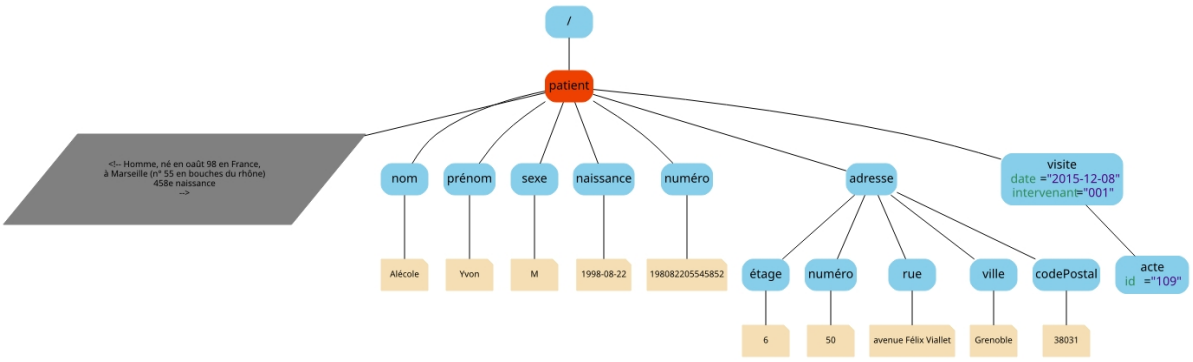

Figure III .2: **Données d'un patient (complément de l'arbre d'instance du cabinet infirmier)**.

3.3 L'adresse

Notez sur le document précédent, que pour que l'adresse du patient soit utilisable par la suite, il faudra bien identifier :

- l'étage (si besoin)
- le numéro de la rue (s'il existe)
- le nom de la rue
- le code postal (un nombre à 5 chiffres)
- la ville

3.4 Actes infirmier et nomenclature NGAP

Revenons à présent sur l'attribut `id` du noeud `acte`. Il correspond à l'identifiant NGAP. Voici un document XML type contenant les actes infirmier et codes NGAP :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ngap
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns='http://www.univ-grenoble-alpes.fr/l3miage/actes'
    xsi:schemaLocation='http://www.univ-grenoble-alpes.fr/l3miage/actes ../
        schema/actes.xsd'>
    <types>
        <type id="pi">prélèvements et injections</type>
        <type id="pc">pansements courants</type>
        <type id="pl">
            pansements lourds et complexes nécessitant des condition
            d'asepsie rigoueuse
        </type>
        <type id="sd">
            Soins infirmiers à domicile pour un patient, quel que
            soit son âge, en situation de dépendance temporaire ou permanante.
        </type>
    </types>
    <actes>
        <acte id="101" type="pi" clé="AMI" coef="1.5">
            Prélèvement par ponction veineuse directe
        </acte>
        <acte id="102" type="pi" clé="AMI" coef="5">
            Saignée
        </acte>
        <acte id="103" type="pi" clé="AMI" coef="1">
            Prélèvement aseptique cutané ou de sécrétions muqueuses, prélè
                vement
            de selles ou d'urine pour examens sytologiques, bactériologiques,
                mycologiques.
        </acte>
        <!--  ... et plein d'autres actes -->
    </actes>
</ngap>
```

La page suivante explique ce qu'est la nomenclature NGAP pour les soins infirmiers et est à lire attentivement.

Chaque acte est désigné par un intitulé, une lettre-clé et un coefficient.

La lettre-clé est un ensemble de 3 lettres parmi les suivantes:

- AMI pour (Acte Médico-Infirmier)
- AIS (Acte de Soins Infirmier)
- DI (Démarche de soins Infirmiers)

Le coefficient est un chiffre, qui servira de coefficient multiplicateur de la valeur de la lettre-clé pour obtenir la valeur de l'acte. Pour faire simple: Valeur de l'acte = (lettre-clé)xcoefficient.

Vous trouverez la liste complète sur le site de la sécurité sociale, mais nous nous restreindrons à une toute petite partie des actes. Un document plus restreint et plus lisible vous est fourni. Il stocke au format XML les codes NGAP que nous utiliserons dans ce projet. Il sera notamment très utile lorsqu'il vous sera demandé de calculer le coût de chaque visite.

3.5 Numéro de Sécurité Sociale

Si l'on se reporte à Wikipedia, le **numéro de sécurité sociale** en France (nom usuel), ou numéro d'inscription au répertoire des personnes physiques (abrégé en NIRPP ou plus simplement NIR) est un code alphanumérique servant à identifier une personne dans le répertoire national d'identification des personnes physiques (RNIPP) géré par l'Insee. C'est un numéro « signifiant » (c'est-à-dire non aléatoire)

composé de 13 chiffres, suivi d'une clé de contrôle de 2 chiffres.

Pour définir le format du numéro de sécurité sociale, veuillez vous reporter à la documentation Wikipédia. (On pourra éventuellement aussi utiliser ce site).

4 Réalisation - Plan

Les étapes de la réalisation vous sont ici présentées dans l'ordre attendu :

- modélisation UML du cabinet
- modélisation XML Schema du cabinet
- instanciation XML du cabinet
- modélisation de la fiche patient
- transformations XSLT (HTML et XML) du cabinet et du patient
- renforcement des schémas par des contraintes de cohérence
- validation et appels de feuilles de transformation en C#
- usage des parsers (DOM / XmlReader)
- sérialisation
- serveur HTTP

5 Réalisation du projet - 1ere partie (modélisation)

Les étapes de la réalisation vous sont ici présentées dans l'ordre attendu :

- modélisation UML
- génération d'une instance XML
- validation du document
- usage :
  - transformations XSLT
  -

5.1 Découverte du projet

Lisez attentivement l'énoncé... oui celui que vous avez survolé au dessus ! Prenez le temps de bien regarder les arbres d'instance fournis.

5.2 Modélisation UML - Diagramme papier et PlantUML

Dans cette partie, vous allez modéliser sous forme de diagrammes UML le document. Cela requiert une bonne analyse des arbres d'instance.
Vous reviendrez plusieurs fois sur ce travail au cours des différentes séances de TP.
Astuce : pour progresser plus facilement, il est suggéré de modéliser d'abord seulement des sous parties de l'arbre, par exemple une adresse ou un infirmier.

5.2.1 Papier.

Réalisez scrupuleusement toutes ces étapes :

- Commencez par identifier à partir des 2 arbres d'instance précédents les types dont vous avez besoin : listez les.
- Identifiez quels types seront des restrictions (des types simples restreints) et comment.
- Dessinez sur papier un schéma UML contenant tous les types et leurs relations (composition, ...)
- N'oubliez pas d'englober ces types dans un namespace. Le nom du vocabulaire défini pour la base de données du cabinet est `http://www.univ-grenoble-alpes.fr/l3miage/medical`.

5.2.2 UML - PlantUML

Vous allez maintenant implémenter votre diagramme dans le langage de script `PlantUML`. Vous trouverez toute la documentation nécessaire sur ce site, en particulier ici : documentation pour les diagrammes de classe.

Pour tester vos scripts et générer vos diagrammes sous forme d'images, vous devez les soummettre dans le service en ligne PlantUML.

Même remarque que précédemment, progressez doucement par petites implémentations.

5.3 Modélisation XMLSchema

Vous allez écrire le schéma XML implementant le diagramme UML que vous aurez mis au point. Il s'agira de créer un fichier unique nommé (très exactement ; attention au respect de la casse et de l'orthographe) `cabinet.xsd`. Le XMLSchema devra être le plus restrictif possible.

Pour rappel, le nom du vocabulaire défini pour la base de données du cabinet est `http://www.univ-grenoble-alpes.fr/l`

Pensez à vérifier que :

- votre document XML (le schema que vous développez est un document XML) est conforme (voir section 5.5)
- votre document XML (le schema que vous développez est un document XML) est valide par rapport au schéma du vocabulaire `http://www.w3.org/2001/XMLSchema` (voir section 5.6)

5.4 Document XML

5.4.1 Création du document XML

Créez un document xml nommé `cabinet.xml` (très exactement ; attention au respect de la casse et de l'orthographe) et qui représente l'arbre précédent. Ce document devra contenir les données données dans les arbres d'instance.

Idéalement, à ce stade, vous aurez développé préalablement votre Schéma XML... mais il est évident que ce ne sera pas abouti. Vous pouvez donc soit générer l'instance XML à partir du Schema (void chapitre II), soit écrire votre document XML à la main. Dans tous les cas il vous faudra rentrer les données à la main et ... à la fin il faudra que ça marche !

Pensez à vérifier que :

- votre document XML est conforme (voir section 5.5)
- votre document XML est valide par rapport au schéma décrit dans le `cabinet.xsd` (voir section 5.6)

5.4.2 Modifications du document XML

On souhaite ajouter les patients suivant dans le document XML, à la suite du premier patient dans le noeud patients et sur le même modèle.

- Premier patient :
  - Nom: Orouge
  - Prénom: Elvire
  - Elvire est née le 08 mars 82 en France, à Lyon (n° 23 dans le rhône), il s'agissait de la 52e naissance dans cette ville
  - Adresse: Rond-Point de la Croix de Vie 38700 La Tronche.
  - Visite: La visite devra avoir lieu le 08/12/2015 pour un Pansement de brûlure étendue. L'infirmière qui interviendra n'a pas encore été décidée.
- Deuxième patient :
  - Nom: Pien
  - Prénom: Oscare
  - Oscare est née le 25 mars 75 en France, à Chambéry (n° 65 dans la Savoie) Il s'agissait de la 692e naissance.
  - Adresse: rue Casimir Brenier 38000 Grenoble.
  - Visite: La visite devra avoir lieu le 08/12/2015 pour une ponction veineuse directe le et une injection intraveineuse directe isolée d'anthalgiques.
- Troisième patient :
  - Nom: Kapoëtla
  - Prénom: Xavier
  - Xavier est un petit garçon de 4 ans, né le 02 août 2011 en France, à Bordeaux (n° 63 dans la Gironde). Il s'agit de la 35e naissance.
  - Adresse: 25 rue des Martyrs 38042 Grenoble.
  - Visite: La visite devra avoir lieu le 08/12/2015 pour recevoir l'injection de plusieurs allergènes afin d'être désensibilisé.

Ajoutez ces informations. L'objectif ici est que vous compreniez que votre travail va consister non seulement à programmer, mais à vous interesser aux données pour lesquelles vous réalisez ces programmes.

5.5 Conformité de l'instance XML et du XMLSchema

Vous procéderez de 2 manières différentes selon votre avancement dans le projet.

- En premier lieu, vous pouvez utiliser l'outils de vérification de conformité intégré à votre IDE (voir chapitre II)
- Le plus tôt possible, cherchez à utiliser un programme C# pour réaliser cette opération. Tout est décrit dans le cours dans le chapitre sur les parsers.

5.6 Validation de l'instance XML vis-à-vis du XMLSchema

Vous procéderez de 2 manières différentes selon votre avancement dans le projet.

- En premier lieu, vous pouvez utiliser l'outils de validation intégré à votre IDE (voir chapitre II)
- Le plus tôt possible, cherchez à utiliser un programme C# pour réaliser cette opération. Tout est décrit dans le cours dans le chapitre sur les parsers.

5.7 Contraindre davantage : clefs d'existence et d'unicité

Modifiez le Schema XML du centre de soin de façon à ce que les identifiants des médecins soient uniques.

Modifiez le Schema XML du centre de soin de façon à ce que les identifiants des médecins auquels se réfèrent les patients existent.

6 Réalisation du projet - 2ème partie (Transformations XSLT)

6.1 HTML - Le CV des programmeurs

L'objectif ici est une prise en main de HTML, un langage à balise permettant de présenter des documents. Vous avez besoin de savoir écrire du HTML pour ensuite savoir comment transformer vos documents XML en pages HTML.
Dans cet exercice, vous devez tout développer à la main (pas d'outils de génération / design de pages web). Vous n'êtes pas autorisés à utiliser autre chose que HTML et CSS.

Réaliser 2 pages XHTML (une par membre du binôme) qui montrent les CVs des webmasters (vous). Il doit s'agir de votre véritable CV. Ce travail doit être fait consciencieusement. La soin apporté à la page et au contenu sera prise en compte dans l'évaluation du projet.

Ces pages devront contenir chacune au minimum:

- une image (votre photo, par exemple)
- une table (le récapitulatif de votre cursus universitaire par exemple)
- une liste numérotée ou non (par exemple la liste de vos hobbies)
- un lien relatif (vers la page de l'autre webmaster)
- un lien URL vers un site web extérieur

Pour vous aider, vous devez vous appuyer sur la documentation donnée dans le site W3Schools dédié à HTML.

6.2 Transformations XSLT

Dans cette partie, nous allons voir comment automatiser la génération de pages HTML contenant des données extraites de documents XML à l'aide de transformations XSLT. Nous verrons que l'on peut aussi extraire des données pour générer d'autres types de documents, en particulier des documents XML.

6.2.1 La page de l'infirmière

a) ***Contexte.*** Votre application va permettre à un(e) assistant(e) médical(e) d'entrer dans le fichier XML les données du cabinet, des patients et des infirmières.

Dans un deuxième temps, lorsqu'une infirmière s'identifiera sur votre application la requête sera interceptée par un serveur métier (UE ASR) qui appellera une méthode d'optimisation (UE PCRO) vous permettant d'ordonner la liste des visites des patients pour optimiser le trajet de l'infirmière.

Une fois les patients et visites ordonnées pour chaque infirmière, il faudra renvoyer cette information à l'infirmière qui s'est identifiée.

Cet exercice a pour but d'écrire une feuille XSLT qui transforme le document XML contenant les visites triées en une page html qui indique à l'infirmière combien elle a de patients à visiter, et pour chaque patient:

- Son nom
- Son adresse
- La liste des soins à effectuer
- Un bouton qui permettra d'afficher la facture correspondante

Pour afficher une page dédiée à une infirmière en particulier, il faudra passer à la feuille de style un paramètre global. Pour cela, il faut utiliser la ligne suivante, juste avant la définition de la première template (i.e. la template qui match "/").

On pourra choisir une valeur par défaut dans ce paramètre (`001` dans l'exemple ci-dessous) que l'on pourra changer pour tester d'autres infirmières.

```
<xsl:param name="destinedId">001</xsl:param>
```

b) ***1ere étape : Une première page simple.*** Dans un premier temps, on souhaite que la page n'affiche qu'une phrase du type:
`Bonjour Annie,`

`Aujourd'hui, vous avez 5 patients`

C'est-à-dire que la feuille doit

- Sélectionner le prénom de l'infirmière étant donné son identifiant (donné par le paramètre global de la feuille)
- Compter le nombre de patients pour cette infirmière

Ecrire une feuille XSLT qui, appliquée à votre document XML donne une page HTML contenant les informations ci-dessus.

c) ***2ème étape : inclure la liste des patients et des soins.*** A la suite de la phrase d'accueil, on souhaite lister pour chaque patient à visiter (et dans l'ordre de visite), son nom, son adresse correctement mise en forme et la liste des soins à effectuer. Pour la liste des soins à effectuer, vous devez aller piocher les noms dans le fichier actes.xml (qui vous a déjà été fourni). Pour piocher des informations dans un autre document XML que le document cible de la transformation, on peut par exemple utiliser une variable contenant les noeuds ngap du document actes.xml (placé dans le même répertoire) comme suit:

```
<xsl:variable name="actes" select="document('actes.xml', /)/act:ngap"/>
```

Ecrire les templates nécessaires.

d) ***3ème étape : ajouter un bouton `Facture`.*** Nous allons ajouter, pour chaque patient un bouton Facture qui ouvrira une nouvelle fenêtre avec la facture correspondante.

Ajouter après la liste des soins de chaque patient un bouton HTML qui a pour texte Facture. La documentation du bouton HTML se trouve ici.

Pour générer le bouton via la transformation XSLT, vous devez créer un `xsl:element` nommé `button` et ayant pour texte Facture. Cet élément contiendra un `xsl:attribute` nommé `onclick` comme suit :

```
<xsl:attribute name="onclick">
   openFacture('<xsl:value-of select="??"/>',
               '<xsl:value-of select="??"/>',
               '<xsl:value-of select="??"/>')
</xsl:attribute>
```

Remarque : une autre façon de procéder consiste à créer un élément `input` doté d'attributs `type`, `onclick` et `value` comme expliqué dans la doc HTML.

L'action de ce bouton appelera le script JS `openFacture` donné ci-dessous :

```
function openFacture(prenom, nom, actes) {
       var width  = 500;
       var height = 300;
       if (window.innerWidth) {
           var left = (window.innerWidth-width)/2;
           var top = (window.innerHeight-height)/2;
       } else {
           var left = (document.body.clientWidth-width)/2;
           var top = (document.body.clientHeight-height)/2;
```

```
        }
        var factureWindow = window.open('','facture','menubar=yes, scrollbars=
            yes, top='+top+', left='+left+', width='+width+', height='+height
            +'');
        factureText = "Facture pour : " + prenom + " " + nom;
        factureWindow.document.write(factureText);
    }
```

Ce script devra être intégré dans l'entête du fichier HTML comme expliqué dans la doc.

Lors de l'appel de se script (dans le bouton), les valeurs de nom, prénom et actes seront transmis en paramètres.

Faites en sorte d'appliquer une feuille CSS : Créez une feuille de style CSS pour votre page web html ou complétez celle que vous avez pour votre application, puis ajoutez dans la feuille XSLT l'élément qui permettra à votre feuille html de charger votre feuille de style.

e) ***4ème étape : amélioration de la facture.*** Dans cette partie, nous allons utiliser l'API DOM en Javascript pour afficher la facture d'une visite.
Avant de continuer, vérifiez que l'arborescence de votre projet est la suivante. Tous les fichiers xml, xsd, ... sont dans un dossier data, comme suit :

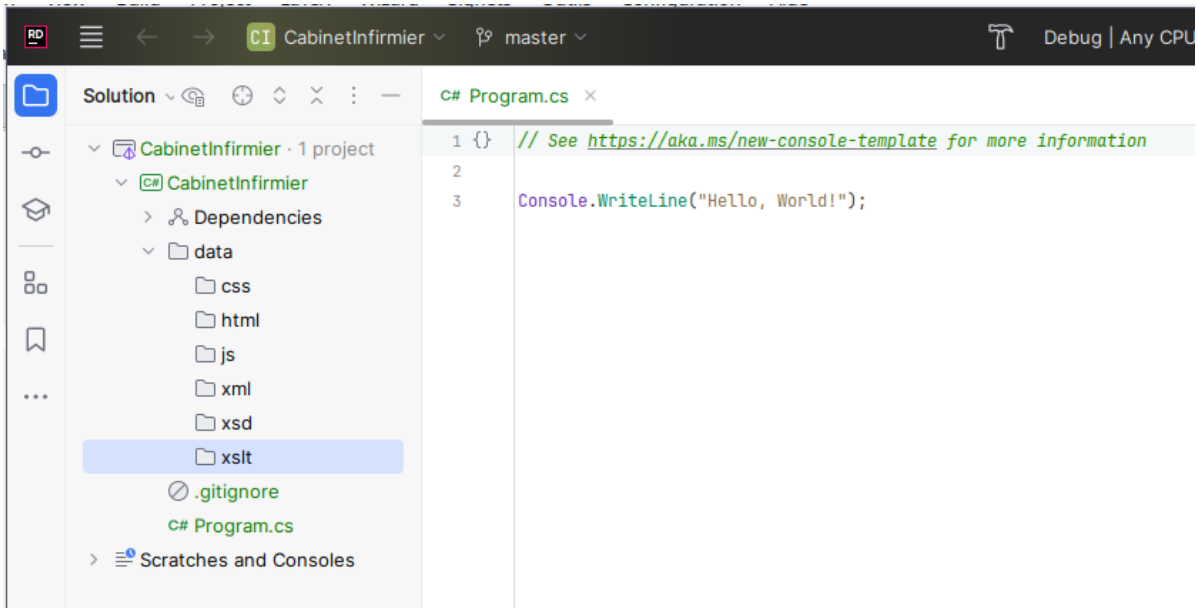

Figure III .3: **Arborescence du projet)**.

Dans le dossier `js`, ajoutez le fichier `facture.js` fourni.

Modifiez la feuille xslt de façon à ce qu'elle appelle ce script comme suit :

```
<script type="text/javascript" src="js/facture.js"></script>
```

Vérifiez son bon fonctionnement.

Modifiez le script pour qu'au lieu d'avoir `var factureText = "Facture pour : " + prenom + " " + nom;` , on ait `var factureText = afficherFacture(prenom, nom, actes);`
Vérifiez son bon fonctionnement.

[Facultatif - à ne faire qu'à la fin du projet] Modifiez/complétez `facture.js` pour que vous ayez une facture correctement mise en page qui affiche:

- l'identité du patient

- son adresse
- son numéro de sécurité sociale
- un tableau qui récapitule les actes, leurs codes et tarif
- la dernière ligne du tableau qui affiche la somme totale à régler

6.2.2 La fiche patient

a) Contexte. Votre application va permettre à un(e) patient(e) de récupérer les informations sur les différents rendez vous qu'il va avoir à partir du fichier XML contenant les données du cabinet, des patients et des infirmières.

Lorsqu'un patient fait cette demande, **un nouveau fichier XML** est généré qui contient les informations le concernant (état civil, adresse, téléphone ...), ce qui lui permettra de vérifier ces informations et d'envoyer une requête de modification le cas échéant. Ce fichier contiendra également la liste des visites médicales le concernant avec toutes les informations écrites en clair, à savoir la date (les visites étant triées par date), le libellé exact de l'acte médical, et le nom de l'infirmier qui viendra pour chaque visite. Vous devez obtenir ce type de document :

```
<?xml version="1.0" encoding="UTF-8"?>
<patient>
    <nom>Pourferlavésel</nom>
    <prénom>Vladimir</prénom>
    <sexe>M</sexe>
    <naissance>1942-12-30</naissance>
    <numéroSS>198082205545842</numéroSS>
    <adresse>
        <rue>Les morets</rue>
        <codePostal>38112</codePostal>
        <ville>Méaudre</ville>
    </adresse>
    <visite date="2017-12-11">
        <intervenant>
            <nom>Fréchie</nom>
            <prénom>Sarah</prénom>
        </intervenant>
        <acte>Ablation de fils ou d'agrafes, plus de dix, y compris le
            pansement éventuel.</acte>
    </visite>
    <visite date="2017-12-08">
        <intervenant>
            <nom>Fréchie</nom>
            <prénom>Sarah</prénom>
        </intervenant>
        <acte>Séance hebdomadaire de surveillance clinique infirmière et de pré
            vention par séance d'une demi-heure.</acte>
    </visite>
</patient>
```

Une fois ce document XML généré et stocké jusqu'à la fin des interventions, un document HTML sera généré grâce à une deuxième feuille de transformation XSLT et renvoyé par le WebService.

b) Identification du patient Pour obtenir ces informations, le patient s'identifiera grâce à son nom. Lorsque le patient se sera connecté au WebService avec son nom. Cette valeur (le nom) sera transmise comme paramètre à la feuille XSLT permettant d'extraire et transformer son dossier en un nouveau document XML.

Dans la première feuille de transformation XLST, comme pour l'infirmière, il faut utiliser la ligne suivante, juste avant la définition de la première template (i.e. la template qui match "/").

On pourra choisir une valeur par défaut dans ce paramètre (Le patient `Alécole` est demandé dans l'exemple ci-dessous) que l'on pourra changer pour tester d'autres patients.

```
1 <xsl:param name="destinedName">Alécole</xsl:param>
```

c) Opérations effectuées sur la fiche patient au cours des TP

- **Première feuille XSLT** :
  - *(Réalisé plus tard)* Fournir au WebService un nom (et un prénom) de patient. Dans l'immédiat, fixer un patient particulier comme indiqué ci-dessus.
  - *(TP d'aujourd'hui)* Modéliser le patient en UML et XML Schema.
  - *(TP d'aujourd'hui)* Ecrire une 1ere feuille XSLT qui utitilise le nom du patient (ex: Pourferlavésel) pour transformer `cabinet.xml` en un nouveau fichier xml nommé `NOMPATIENT.xml` (ex: `Pourferlavésel.xml`) comme demandé. Fixez le nom du fichier. Celui-ci sera plus tard généré par le programme C#.
  - *(TP d'aujourd'hui)* Appliquer la tranformation pour générer le fichier XML patient
- **Deuxième feuille XSLT** :
  (TP d'aujourd'hui) Réaliser une deuxième feuille XSLT qui transforme `NOMPATIENT.xml` en une page html `NOMPATIENT.html` présentant les renseignements voulus.

Pour tester cette feuille vous pouvez modifier votre document XML et affecter des patients à différentes infirmières. Vous pouvez tester divers patients.

7 Réalisation du projet - 3ème partie (Fonctionnalisation)

Dans cette partie, nous allons fonctionnaliser nos données en C#.

Deux remarques :

- Faites bien attention à l'arborescence (précisée à la 4ème étape de la page de l'infirmière). Depuis votre programme, si par exemple un fichier `cabinet.xml` se trouve dans le dossier `data/xml`, le chemin que vous devrez préciser sera `"./data/xml/cabinet.xml"`
- Précisez à la solution dotnet que les fichiers doivent être copiés : par défaut, lors de la construction (build) de la solution (projet) par JetBrains, les fichiers de données (xml, xsd, images ...) ne sont pas copiés dans le dossier `bin/Debug/netX.0/`. Pour qu'ils puissent être copier et que ces fichiers soient disponibles (par exemple `bin/Debug/net8.0/data/xml/cabinet.xml`), il faut le préciser. Pour cela, dans l'arborescence de fichier de JetBrains à gauche (File System), faites un clic droit sur le fichier à ajouter, sélectionnez `Properties`, puis dans la section `Editable`, et dans `Copy to output directory` sélectionnez `Copy always`. (voir par exemple la figure ci-dessous)

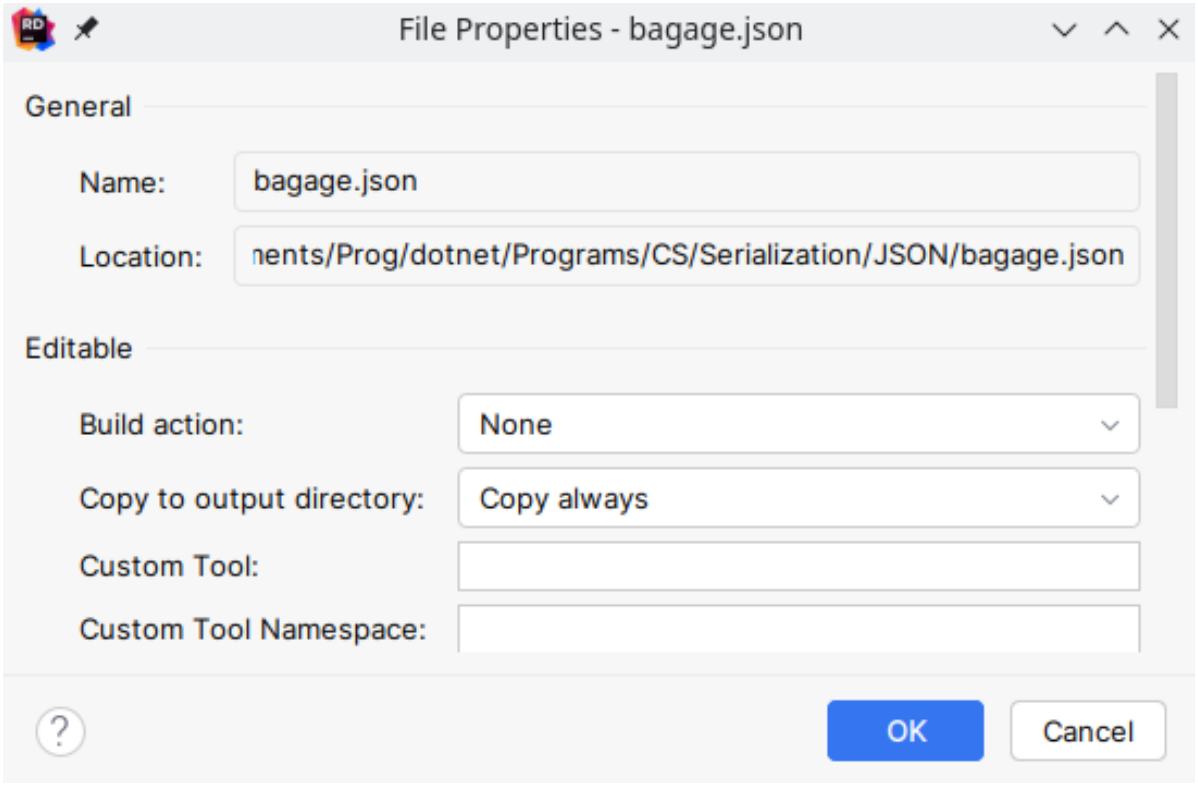

Figure III .4: **Copie d'un fichier pendant la construction).**

7.1 Validation de document XML en C#

Créez une classe statique `XMLUtils`. Cette classe contiendra des méthodes pour travailler sur des documents XML.

Ajoutez-y les méthodes méthodes statique suivantes :

```
1    public static async Task ValidateXmlFileAsync(string schemaNamespace,
         string xsdFilePath, string xmlFilePath) {
2        var settings = new XmlReaderSettings();
3        settings.Schemas.Add(schemaNamespace, xsdFilePath);
4        settings.ValidationType = ValidationType.Schema;
5        Console.WriteLine("Nombre de schemas utilisés dans la validation : "+
             settings.Schemas.Count);
```

```
        settings.ValidationEventHandler += ValidationCallBack;
        var readItems = XmlReader.Create(xmlFilePath, settings);
        while (readItems.Read()) { }
    }


    private static void ValidationCallBack(object? sender, ValidationEventArgs
        e) {
        if (e.Severity.Equals(XmlSeverityType.Warning)) {
            Console.Write("WARNING: ");
            Console.WriteLine(e.Message);
        }
        else if (e.Severity.Equals(XmlSeverityType.Error)) {
            Console.Write("ERROR: ");
            Console.WriteLine(e.Message);
        }
    }
```

Ces méthodes permettent de tester la validation de vos documents XML avec un Schema XML. Remarquez qu'il est indispensable de préciser le namespace. Dans votre programme, vous pouvez appeller cette fonction de validation comme suit :

```
XMLUtils.ValidateXmlFileAsync("http://www.univ-grenoble-alpes.fr/l3miage/
    medical", "./data/xml/cabinet.xml", "./data/xsd/cabinet.xsd");
```

Testez ce programme. Faites de même pour les autres documents et schémas.

Vous penserez plus tard à la possibilité d'inclure cette validation dans vos fonctions (en la modifiant éventuellement).

7.2 Appel de feuilles de transformation XSLT en C#

Dans la classe `XMLUtils` ajoutez la méthode statique suivante :

```
public static void XslTransform(string xmlFilePath, string xsltFilePath, string
     htmlFilePath) {
        XPathDocument xpathDoc = new XPathDocument(xmlFilePath) ;
        XslCompiledTransform xslt = new XslCompiledTransform();
        xslt.Load(xsltFilePath);
        XmlTextWriter htmlWriter = new XmlTextWriter(htmlFilePath,null);
        xslt.Transform(xpathDoc,null,htmlWriter);
    }
```

Utilisez cette fonction pour appliquer les transformations XSLT que vous avez écrites précédemment aux instances XML cibles.

Modifiez votre programme pour qu'il fournisse en paramètre l'identifiant de l'infirmier (ou du patient) à la feuille XSLT. Pour faire cela, suivez la documentation C# ici.

7.3 Parseurs DOM et XmlReader

Ecrivez une classe `Cabinet`. Ecrivez également une classe `Adresse`. Pour l'instant, votre classe `Adresse` ne déclare et instancie qu'un nom et une adresse (pas d'infirmiers ni de patients).

Ces mêmes classes serviront à pratiquer les parsers `XmlReader` et `DOM`, mais aussi, une fois complétées pour correspondre au XML Schema, à pratiquer la sérialisation.

7.3.1 Récupération d'informations à la volée avec XmlReader.

Référez vous au cours pour ajoutez à votre classe `Cabinet` une méthode `void AnalyseGlobale(string filepath)` qui :

- parse un fichier avec un XmlReader

- détecte le type de noeud et l'utilise comme switch
- Affiche un message quand on entre dans le document
- Affiche un message quand on entre dans un élement ; ce message doit afficher le nom de l'élément et le nombre d'attributs qu'il contient
- Affiche un message quand on sort d'un élément
- Affiche un message quand on rencontre du texte (le texte doit être affiché)
- Affiche un message quand on rencontre un attribut (en affichant le nom et le contenu de l'attribut).

Une fois cette fonction testée, sur ce même principe, écrivez une autre fonction qui permet de récupérer le texte d'éléments particuliers (par exemple tous les noms, ou tous les noms des infirmiers). Idéalement, votre recherche doit être passée en paramètre de la fonction.

Créez une autre fonction utilisant le XmlReader pour compter combien d'actes différents devront être effectués, tous patients confondus.

7.3.2 Vérification de présence de valeurs particulières avec DOM.

Apprenez à récupérer des informations particulières en utilisant DOM et XPath en vous inspirant de l'exemple ci-dessous.

```
1 DOM2Xpath bagagesDOM = new DOM2Xpath("./XML/bagagerieHF.xml");
2 String myXpathExpression = "//bg:bagage[bg:description='skis']/@idPassager";
3 XmlNodeList nlBagagesDOM = bagagesDOM.getXPath("bg", "http://www.timc.fr/
      nicolas.glade/bagages", myXpathExpression);
4 foreach (XmlNode node in nlBagagesDOM)
5         Console.WriteLine(node.InnerText);
```

Remarque : que dans votre document vos éléments soient préfixés ou non, vous devez ici préciser un préfixe (même s'il ne correspond pas). Un défaut de l'API `DOM2XPath` oblige à procéder ainsi.

Ecrivez une fonction qui, appliquant un chemin XPath à un document XML, renvoie ainsi une NodeList.

Utilisez ce programme pour vérifier que votre document contient :

- 3 infirmiers
- 4 patients
- une adresse complète pour le cabinet
- une adresse complète pour chaque patient
- qu'un numéro de sécurité sociale est valide par rapport aux informations fournies (date de naissance, et sexe, mais aussi vérifier à l'intérieur du numéro de sécu si la clef est valide).
- que l'ensemble des numéros de sécurité sociale sont valides par rapport aux informations fournies

Une façon de procéder est de créer autant de fonctions que de points à vérifier (en passant les valeurs requises en paramètres), par exemple une méthode `int count(string elementName)` ou une méthode `bool hasAdresse(string elementName)`.

Tout marche bien ? Parfait ! au moins comme ça vous savez comment les profs peuvent évaluer automatiquement vos travaux et pourquoi les noms des éléments et attributs, ainsi que le schéma doivent être strictement conformes au cahier des charges.

7.3.3 Modification de l'arbre DOM et de l'instance XML.

Utilisez DOM pour ajouter un infirmier dont l'id sera 005. Il s'appelle Némard Jean. Vous écrirez une fonction générique d'ajout d'infirmiers qui prend comme paramètre le nom et le prénom de l'infirmier. le nom de sa photo est généré automatiquement (prenom.png) et son identifiant également (ce doit être un incrément par rapport au plus grand id connu de la liste d'infirmiers).

Sur le même principe :

- Utilisez DOM pour ajouter un patient (nom, prénom, adresse, NSS, ...adresse ...). Un patient nouvellement créé n'a aucune visite
- Utilisez DOM pour ajouter des visites à un patient. Une visite doit être affectée à un intervenant.

Ajoutez le patient Mme Niskotch Nicole, dont vous fixerez une date de naissance, lieu de naissance ... pour avoir un NSS valide, à laquelle vous donnerez une adresse. Ajoutez lui ensuite une visite de votre choix avec l'intervenant de votre choix.

7.4 Sérialisation

7.4.1 Sérialisations simples

a) Adresse. Dans un premier temps, écrire un document XML ne contenant qu'une adresse (dont la racine de l'instance XML est une adresse <=> modifiez le schéma XML de façon à permettre la création d'une telle instance). En utilisant ce que vous avez vu en TP sur les XmlSerializer, écrire un programme permettant de sérialiser cette adresse avec un objet de type `Adresse`.

b) Infirmier. Procédez de la même manière pour un `infirmier` en créeant une classe sérialisable `Infirmier`.

c) Modification des classes Adresse et Infirmier. Les champs de ces 2 classes (ex: numéro, rue, codePostal, ...) peuvent être amenés à être modifiés depuis le programme C#, par exemple via un formulaire. Cela signifie que, si aucune précaution n'est prise, une personne mal informée ou mal intentionnée pourrait saisir n'importe quelle valeur (par exemple un code postal abérant comme `38ex79` ou un numéro de rue négatif). En utilisant des propriétés C#, vous pouvez via le setter (`set`) contraindre, en C#, les valeurs à des expressions (un code postal doit correspondre à une expression régulière particulière) ou valeurs spécifiques (un numéro de rue doit être positif et non nul). Modifiez les classes Adresse et Infirmier en leur ajoutant des propriétés (qui deviennent les attributs de classe sérialisés) de telle manière que leurs setters contraignent les valeurs possibles, en accord avec le schéma que vous avez écrit.

d) Questions. Vérifiez que vous savez répondre aux 2 questions suivantes :

- Comment modifier ces classes de telles manière qu'elles ne soient plus modifiables (non mutables) ? Créez une variation immuable des classes `Adresse` et `Infirmier` que vous nommerez `AdresseRO` et `InfirmierRO` (RO pour read only).
- Comment, sans rajouter de propriétés, pourriez vous vous assurer, *a posteriori*, qu'une valeur abérante (non conforme au XML Schema) saisie côté programme C# et sérialisée, puisse quand même être détectée ? Implémentez le !

7.4.2 Sérialisation d'une liste d'infrimiers

Ecrivez une classe sérialisable `Infirmiers` de telle manière qu'elle contienne une liste d'infirmiers. Créez également une instance XML ne contenant que la liste des infirmiers avec pour racine `infirmiers`. Vérifiez que vous êtes bien capables de désérialiser-sérialiser.

7.4.3 Sérialisation du Cabinet

Tout est dans le titre ! Ecrivez toutes les classes manquantes de manière à pouvoir sérialiser/désérialiser le cabinet entier (classe `Cabinet` déjà créée). Faites en sorte de contraindre comme il faut les attributs.

7.4.4 Utilisation de la sérialisation du Cabinet

Utilisez votre programme C# pour désérialiser, ajouter un patient depuis C# (instancier un patient et le rajouter à la liste des patients), et une visite à un patient existant (instancier une visite et l'ajouter à un patient), et sérialisez dans un nouveau fichier `cabinet_modif.xml`.