Gestion XML/XSLT de données d'interaction protéine-protéine

Rapport de projet

CEZANNE Emmanuel CROS Sébastien EVEN Claire
JOLIBERT Nicolas MARQUES Sandrine
ROUMEGOUS Christophe SABLAYROLLES Patrick
THOMAS-NELSON René

ENSEIRB, 2ème Année Informatique, 23 mai 2003

Résumé

Dans le cadre du projet Proteomics Standards Initiative (PSI) qui a pour but de mettre en place un nouveau format de fichier XML pour des données d'interaction protéine-protéine, nous avons développé plusieurs outils associé à ce format. Ces outils se présente sous la forme de fichiers de transformation XSLT à appliquer au fichiers XML à traiter.
Les outils développés sont :

- Des outils permettant de convertir un fichier PSI sous sa forme canonique (sans redondance) en un fichier PSI sous forme déroulé (sans référence) et inversement. A ces outils on à associé deux autres outils de test pour vérifier si un fichier XML PSI est bien sous forme canonique ou déroulé.
- Un outil de tranformation de fichier PSI canonique en fichier de type CSV, pour être traiter dans des tableurs ou autres logiciels n'acceptant pas directement les fichiers XML.
- Un outils de conversion des fichiers XML d'intération protéine-protéine de la banque de donnée D.I.P. au format PSI canonique. On a associé à cet outil un outils de génération de rapport qui décrit les problème de conversion.
- Un outil de visualisation des intérations d'un fichier PSI canonique sous forme de page HTML.

Table des matières

Chapitre 1

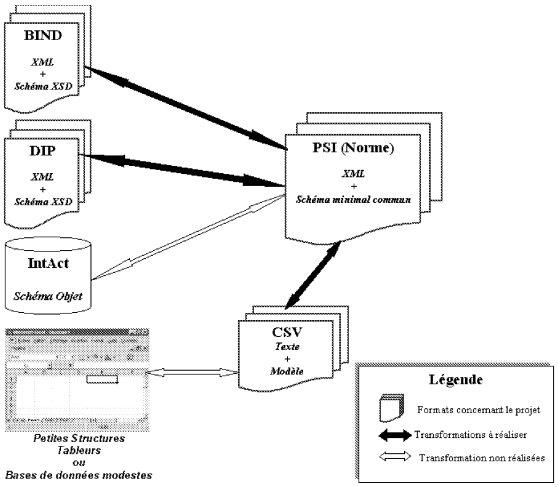
Cahier des charges

1.1 Contexte du Projet

Le groupe Proteomics Standards Initiative (PSI) développe, sous l'égide de la HUPO, des normes internationales pour l'échange et la publication de données sur les protéomes d'espèces vivantes notamment l'Homme. Le groupe PSI organise ses activités autour de deux thèmes principaux : les données de spectrométrie de masse, et les données expérimentales d'interaction protéine-protéine.

Ces formats et les outils associés vont permettre aux banques de données existantes (BIND, DIP, IntAct...), aux constructeurs d'appareils et aux scientifiques (institutionnels et industriels) de partager et gérer les téra-octets de données actuellement dispersées dans les laboratoires et centres de recherche.

C'est dans ce contexte qu'intervient notre projet, dont les objectifs sont résumés par le schéma ci-dessous :



1.2 Objectifs Généraux

Lors du dernier meeting du " Proteomics Standards Initiative " à l'Institut Européen de Bio informatique (Hinxton, UK), il a été défini un format de fichier XML pour l'échange d'informations relatives aux expériences d'interaction entre protéines, le format PSI, sur lequel nous allons nous baser pour cette étude. Ce format permet d'enregistrer les données sous deux formes : PSI " Canonique " et PSI " Déroulé ".

Dans ce cadre, notre travail consistera à réaliser des outils de conversion entres les différents formats de fichiers.

La première conversion se fera entre les deux formes de format PSI, permettant de transformer un fichier au format PSI Canonique en PSI Déroulé et réciproquement.

On s'attachera également à la transformation de fichiers au format DIP (Database of Interacting Proteins : http ://dip.doe-mbi.ucla.edu/) au format PSI " Canonique ", on envisagera ensuite la transformation identique pour le format BIND (Biomolecular Interaction Network Database : http ://www.bind.ca).

Il apparait aussi nécessaire de spécifier un nouveau format de fichier de type CSV (Comma Separated Value) exploitable au niveau de petites structures qui utilisent des outils de traitements de données simples. Une fois ce format défini, on implémentera un outil de conversion permettant de passer du format PSI Canonique au format CSV qui aura été défini.

1.3 Travail à réaliser

1.3.1 Conversion de fichiers entre les formats PSI Canonique et PSI Déroulé

Les applications réalisant ces objectifs sont réèlement indispensables pour le client et ne sauraient ne pas être livrées.

Besoins fonctionnels

On développera un outil réalisant la conversion de fichiers entre les formats PSI Canonique et PSI Déroulé en 2 modules principaux : le premier réalisant la conversion du format PSI Canonique vers le format PSI Déroulé et le second réalisant la conversion réciproque.

Besoins non fonctionnels

Cet outil suppose que le fichier fourni en entrée est conforme à la xsd de PSI (schéma XML) sous forme canonique.

1.4 Algorithme

La transformation sera valide si :

- On a remplacé chaque référence dans les interactions par la description correspondante de la liste de définitions. En pratique, un élément *experimentRef* doit être remplacé par un élément *experimentDescription*, *availabilityRef* par *availabilityDescription* et *interactorRef* par *proteinInteractor*. La correspondance se fait entre l'attribut *ref* des références et l'attribut *id* des descritions.
- Si on a vidé les listes situées en en-tête. En fait on doit pas recopier dans le fichier de sortie le contenu des listes : *availabilityList, experimentList* et *interactorList*.
- On a recopié tous les autres éléments et attributs à l'identique.

En résultat doit être un fichier au format PSI déroulé.

Pour valider les transformations entre PSI Canonique et PSI Déroulé. Il sera possible d'áppliquer de manière circulaire, une conversion, puis l'autre plusieurs fois de suite. Ce mode de test nous permettra de vérifier la stabilité et la cohérence de nos transformations.

1.4.1 Conversion de fichiers entre les formats DIP ET PSI

Seule la transformation du format DIP vers le format PSI a été envisagée.

Besoins fonctionnels

Pour rapprocher ces deux formats, on réalisera préalablement une étude des informations contenues dans ces fichiers à partir de leur DTD et schéma XML. On implémentera ensuite un outil réalisant la conversion du format DIP vers le format PSI Canonique.

Besoins non fonctionnels

Cet outil suppose que le fichier fourni en entrée est conforme à la xsd de PSI (scéma XML) sous forme déroulé.

La transformation sera valide si :

- On a remplacé toutes les descriptions au niveau des intéractions par des références. Les élements *experimentDescription*, *availibilityDescription*, *proteinInteractor* doivent être remplacés respectivement par les références (*experimentRef, availabilityRef* et *interactorRef*). L'attribut *ref* de chaque référence correspond à l'attribut *id* de chaque descrition.
- On a remplit les listes de descriptions *availabilityList, experimentList* et *interactorList*. Pour chaque liste :
  - on lit les descritions au niveau des intéractions.
  - on ajoute cette définition si elle n'est pas déjà dans la liste. Concrètement, seule la première définition (dans l'ordre du fichier) sera conservée.
  - trier les descriptions par l'attribut *id*.
- On a recopié tous les autres éléments et attributs à l'identique.

En résultat doit être un fichier au format PSI canonique.

Les données stockées dans le fichier de sortie sont les mêmes que dans le ficher initial.

1.4.2 Création d'un format de type CSV

Besoins fonctionnels

Le format de fichier CSV n'étant pas défini, on réalisera dans un premier temps une étude de la structure qui sera utilisée pour stocker les données de façon efficace dans ce format. Ce format décrira uniquement des interactions binaires entre protéines. Pour établir cette nouvelle structure, on s'appuiera sur les informations contenues dans les fichiers issus de BIND et DIP dans ce type d'interaction. On donnera alors une définition précise de la structure retenue.

Besoins non fonctionnels

Ce format devra contenir toutes les informations contenu dans un fichiers PSI. Le format CSV présentera l'information stockée dans un tableau -format tableur-.
Ce format utilisera le separateur **;** pour séparer les colonnes.

1.4.3 Conversion de fichiers entre les formats PSI et CSV

Dans un premier tant seuls l'affichage grâce au format CSV de fichiers de la base PSI sont envisagés. Selon létat d'avancement et les délais, la possibilité de rentrer de nouveaux fichiers dans la base PSI grâce à une saisie sur un fichier CSV pourra être envisagée.

Besoins fonctionnels

On développera un outil réalisant la conversion du format PSI Canonique de type XML vers le format défini précédemment de type CSV. Cet outil proposera deux options de transformation au choix :

- création d'un fichier de type CSV " synthétique " contenant les informations les plus pratiques.
- création d'un fichier de type CSV " complet " contenant toutes les informations du fichiers PSI de départ.

Besoins non fonctionnels

Le fichiers CSV de sortie doit de composer de 5 tableaux :
Availability List : tableau des *availability*.
Experiment List : tableau des *experimentDescription*.
Interactor List : tableau des *proteinInteractor*.
Interaction List : tableau des intéractions avec un participant par ligne et une référence vers l'intéraction à laquelle il participe.
Feature List : tableau des *featureDescription* avec une référence à la protéine concernée et à l'intéraction où il est utilisé.
Les noms de colonnes du fichiers CSV de sortie doivent être sans ambiguité.

1.4.4 Caractéristiques communes aux différents outils de conversion

- Vérification des données Il sera généré à chaque conversion un rapport indiquant les éventuelles erreurs et pertes d'information. Ce rapport pourra

se présenter sous la forme d'un fichier texte ou HTML.
- Cohérence des données en entrée Dans tous les cas de conversion, le fichier introduit en entrée devra respecter la DTD ou le schéma XML relatif à son format pour que la conversion soit faite.
- Jeu de tests Une fois que l'outil de conversion sera implémenté, on évaluera la correction des transformations en appliquant un jeu de tests significatifs. Les résultats seront utilisés pour valider l'outil développé.
- Développement par les tests Tout notre développement sera orienté tests. Chaque application sera validée par l'application de jeux de tests. Il ne sera pas fourni de preuve de la correction des algorithmes utilisés.
  Il pourra être interessant de vérifier si les fichiers XML produits par nos transformations sont conformes aux schémas XSD qu'ils sont censés vérifier.
- Choix technologiques Les choix technologiques pressentis sont les suivants :
  - Les outils de conversion seront réalisés à partir de feuilles de style en XSLT.
  - Un choix d'un processeur XSLT unique pour l'ensemble des développements.
  - Un ou plusieurs validateurs de fichiers XML par rapport à des schémas XSD.

1.4.5 Manuels et Rapports

A la fin de l'étude, il sera fourni au client le document suivant :
- Manuel Utilisateur : décrivant le fonctionnement des produits développés
- Manuel de Maintenance : donnant les détails d'implémentation, les choix algorithmiques et techniques

1.5 Evolutions possibles

Les points suivants ne font pas partie des priorités du client, ils pourront être traités, si le temps le permet et feront l'objet d'un avenant à ce cahier des charges.

1.5.1 Conversion de fichiers entre les formats BIND et PSI

On pourra développer un outil réalisant la conversion d'un fichier au format BIND vers le format PSI Canonique.

1.5.2 Conversion réciproque entre le format PSI et les formats DIP, BIND et CSV

On pourra développer les outils de transformation réciproque à celles détaillées ci-dessus. Ces outils se présenteraient sous une forme de module similaire à celui de la transformation directe.

1.5.3 Site Internet de présentation et de diffusion des outils

- 3.6.1 Les outils de conversion Le site présentera rapidement les outils de conversion réalisés et en détaillera le fonctionnement. Il permettra également le téléchargement de ces outils.
- Les formats de fichier Le site indiquera la structure des fichiers PSI utilisés ou indiquera des liens vers ces définitions. Le site présentera la structure des fichiers de type CSV et en donnera la définition précise.

Le site Internet sera implémenté en HTML et PHP et pourra également interagir avec une base de données MySQL.

1.5.4 Web Service

Dans un premier temps on pourra mettre en place un système simple, qui à partir d'une page du site permettra le téléchargement d'un fichier, lui appliquera la transformation désirée et retournera le fichier résultat.

On pourra encore envisager de faire évoluer ce système vers un véritable Web Service en implémentant une interface de communication en SOAP.

1.6 Lexique

XML : eXtensible Markup Language Format permettant de stocker des données dans un fichier de façon structurée. XML a été mis au point par le XML Working Group sous l'égide du World Wide Web Consortium (W3C) en 1996. Depuis le 10 février 1998, les spécifications XML 1.0 ont été reconnues comme recommandations par le W3C, ce qui en fait un langage reconnu. (Tous les documents liés à la norme XML sont consultables et téléchargeables sur le site Internet du W3C, http ://www.w3c.org/XML/).

XSL : eXtensible StyleSheet Language Un langage de feuilles de style extensible développé spécialement pour XML permettant de transformer la structure des éléments XML.

XSLT : eXtensible StyleSheet Language Transformation Langage de transformation des données permettant de transformer la structure des éléments XML. Il s'agit d'une recommandation W3C du 16 novembre 1999.

DTD : Document Type Definition C'est une grammaire permettant de vérifier la conformité du document XML. La norme XML n'impose pas l'utilisation d'une DTD pour un document XML, mais elle impose par contre le respect exact des règles de base de la norme XML.

XSD : XML Schema Definition Fichier XML donnant la définition d'un format de fichier XML.

BIND : Biomolecular Interaction Network Database Site Internet : http ://www.bind.ca
BIND possède un format de fichiers XML propre décrit dans une DTD. Ce format est très complet : il a été crée pour définir de manière exhaustive les interactions entre les protéines.

DIP : Database of Interacting Proteins Site Internet : http ://dip.doe-mbi.ucla.edu/
DIP possède un format de fichiers XML propre décrit par un schéma XSD. C'est un format permettant de définir les interactions binaires entre deux sites.

IntAct : IntAct est un projet interactions protéines- protéines. La base de données IntAct est actuellement en cours de développement. IntAct est fondée sur une architecture objet, les insertions des éléments dans la base sont faites par sérialisation des objets.

Fichiers de type CSV : Comma Separated Value Fichier texte dont les données sont séparées par un séparateur prédéfini (virgule, point-virgule, tabulation...). Ce type de fichier résulte souvent d'exportation de données à partir de tableurs ou de base de données.

1.7 Echéancier prévisionnel

En termes de délais, voici une première évaluation du temps nécessaire détaillée phase par phase :

- 1 : Conversion de fichiers entre les formats PSI Canonique et PSI Déroulé : 8 jours.
- 2 : Conversion de fichiers entre les formats DIP et PSI : 8 jours.
- 3 : Création d'un format de type CSV : 2 jours.
- 4 : Conversion de fichiers entre les formats PSI et CSV : 2 jours.
- 5 : Site internet de présentation et de diffusion des outils : 3 jours.

Soit un total prévu : 23 jours

Les éléments du projet dont nous disposons ne nous permettent pas une évaluation précise pour le moment, cet échéancier est donc sujet à modifications.

Chapitre 2

Analyse et Spécifications

Le travail qui nous a eté demandé consiste donc à réaliser des outils permettant de transformer un document XML source en un document XML résultat sans perte d'informations. Le fichier résultat doit respecter la XSD de PSI. Une XSD étant une descriptiuon formelle de la structure d'un document XML. C'est à dire les balises autorisées, les types qui leur sont associés ainsi que leur nombre d'occurences à l'intérieur d'un élément donné.
Il nous été aussi demandé de réaliser un outil permettant de transformer un document PSI source en document CSV (document texte) dont le format était à définir.
Pour réaliser ces outils, notre choix s'est immédiatemment porté sur le langange XSL. Il s'agit d'un langage permettant de transformer un document XML (ou texte) source en un document XML (ou texte) résultat. C'est à dire un langage correspondant exactement à nos besoins.

Notre travail a donc été divisé en trois grande parties correspondant à un ensemble de transformations à effectuer.

Dans une première partie, nous exposerons donc l'analyse en ce qui concerne la transformation PSI CANONIQUE vers PSI DEROULE ainsi que la trnsformation réciproque.
Dans une deuxième partie, nous traiterons la transformation DIP vers PSI.
Enfin, nous presenterons la transformation PSI vers CSV.

2.1 Transformations PSI canonique et PSI déroulé

Dans cette partie, nous allons décrire tout d'abord le format PSI dans ses formes CANONIQUE et DÉROULÉ.
Par la suite, nous decrirons comment nous avons réalisé la transformation PSI CANONIQUE vers PSI DÉROULÉ et reciproquement.

2.1.1 Proteomics Standards Initiative (PSI)

PSI est un groupe internationnal qui travaille sur les échanges de données d'interaction protéine-protéine. Ce groupe à définit un format de fichier XML quie est celui utilisé par nos outils. Les données d'interaction sont manipulées différents organismes de taille variable. Certain de ces organismes possède des banques de données qui traitent un nombre important de données, d'autres possèdent des structures plus modeste.

A l'heure actuelle, chacun de ces acteurs stocke ses données dans un format qui leur est propre, ceci explique donc la nécessité de cette norme et des outils des transformation associés.

Nous allons donc définir cette norme. Une définiton précise des champs est fournie dans l'annexe 1. Nous essaierons ici de donner une vue plus générale du PSI.

La norme PSI est définie par un schéma XML, appelé XSD (**X**ML **S**chéma

Définition). Ce schéma définit le format des fichiers XML contenant les données d'interaction. Il donne les éléments (balises) valides pour un fichier XML, le type de contenu et les attributs qui leur sont associés.

Les concepteurs du format PSI ont défini de formes de fichiers PSI (PSI canonique et PSI deroulé).La première définit un fichier PSI sans redondance d'informations dit canonique. La deuxième définit un fichier PSI avec redondance d'informations dit déroulé.

Plus précisément, un fichier PSI présente les interactions concernant un ensemble de proteines. Pour cela, il doit contenir les informations concernant les expériences qui ont été effectuées sur ces protéines pour mettre en évidence les interactions, les différentes protéines y participant et les droits et licences qui leur sont associés. Ces éléments (protéines, expériences,droits et licences) sont mis en relation dans la définiton d'interactions (voir figure 2.1 et 2.2).

Dans sa forme canonique, le fichier PSI défini en entête la liste des expériences réalisées, la liste des protéines interagissant, ainsi que la liste des droits et licences. Chaque interaction fait ensuite référence á un ou plusieurs élément de ces listes en utilisant son identificateur. Dans ce mode, à l'interieur d'une interaction il n'y a donc que des références vers ces objets (voir figure 2.1).

Dans se forme déroulée, les listes d'entête sont vides et les interactions contiennent la description complète des experiences, des interacteurs, ainsi que des droits et licences. Il n'y a donc aucune référence dans les interactions. Ces informations sont donc dupliquées autant que nécéssaire (voir figure 2.2). Toutes les informations disponibles sont accessible au niveau de l'interaction.

Remarque : Un fichier contenant un mélange de ces deux définitions sera valide en ce qui concerne le schéma (XSD) PSI mais entraînera des pertes lors de l'exécution des transformations. En effet, il devra être considéré à un instant donné comme faisant partie de l'un de ces formats. En effet, s'il est considéré canonique, toute description dans une interaction sera conservée. En revanche, s'il est considéré comme déroulé, tous les élements présents dans les listes d'entête seront ignorés.

Nous présentons dans la suite les outils permettant de passer de l'un des formats à l'autre.

2.1.2 Transformation de PSI canonique vers PSI déroulé

Pour cette transformation, nous avons créé un fichier de transformation appelé *MIF_unfold.xsl*. C'est une feuille de style permettant de passer d'un format XML à un autre.

Ce fichier permet à partir d'un format PSI CANONIQUE d'aboutir à un format PSI DÉROULÉ, c'est à dire que l'on modifie les champs suivants :

- Remplacer chaque référence dans les interactions par la description correspondante de la liste de définitions. En pratique, un élément *experimentRef* sera remplacé par un élément *experimentDescription*, *availabilityRef* par *availabilityDescription* et *interactorRef* par *proteinInteractor*. La cor-

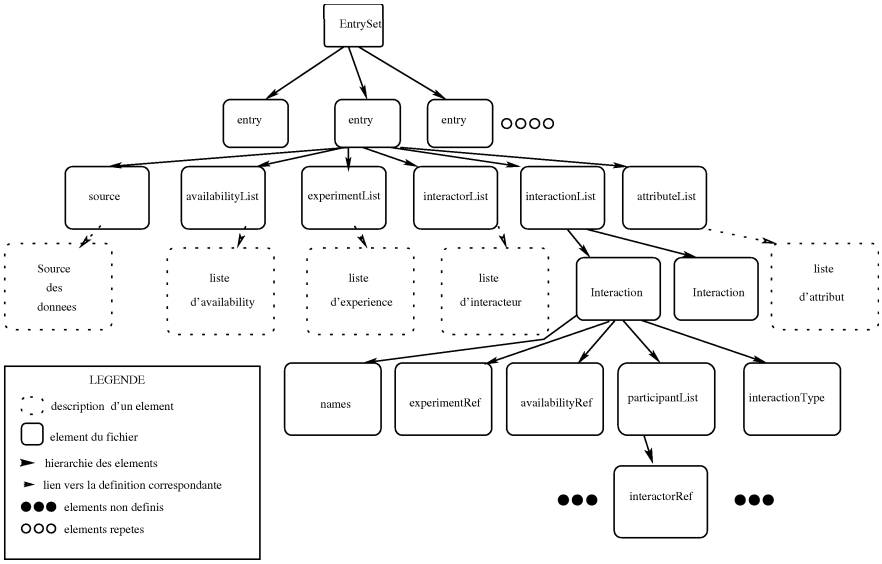


FIG. 2.1 – Structure simplifiée de PSI canonique

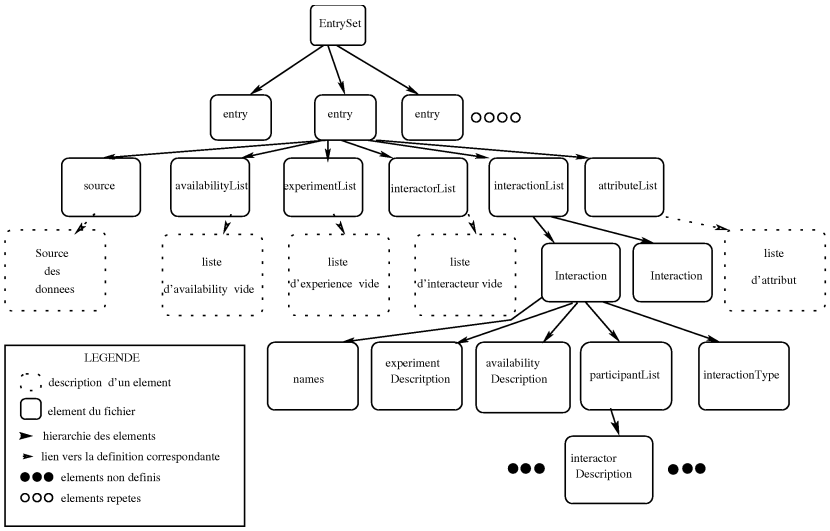


FIG. 2.2 – Structure simplifiée de PSI déroulé

respondance se fait entre l'attribut *ref* des références et l'attribut *id* des descritions.
- Vider les listes situées en en-tête. En fait on ne recopie pas dans le fichier de sortie le contenu des listes : *availabilityList, experimentList* et *interactorList.*
- Recopier tous les autres éléments et attributs à l'identique.

2.1.3 Transformation de PSI déroulé vers PSI canonique

Cette transformation a été un peu plus délicate. Pour ceci nous avons créé un fichier de transformation appelé *MIF_fold.xsl.*

Pour cette transformation, il a fallu :
- Remplacer toutes les descriptions au niveau des interactions par des références. Les élements *experimentDescription, availibilityDescription, proteinInteractor* sont remplacés respectivement par les références (*experimentRef, availabilityRef* et *interactorRef*). L'attribut *ref* de chaque référence correspond à l'attribut *id* de chaque descrition.
- Remplir les listes de descriptions *availabilityList, experimentList* et *interactorList.* Pour chaque liste :
  - on lit les descritions au niveau des interactions.
  - on ajoute cette définition si elle n'est pas déjà dans la liste. Concrètement, seule la première définition (dans l'ordre du fichier) sera conservée.
  - trier les descriptions par l'attribut *id.*
- Recopier tous les autres éléments et attributs à l'identique.

Mais ceci a posé quelques problèmes :
- Nous ne gardons dans les listes que le premier élément experimentDescription (resp. availabilityDescription et proteinInteractor) rencontré ayant un *id* unique. En effet on ne garde dans les listes qu'un représentant de chaque des cription et celui-ci sera toujours le premier rencontré.
- Si jamais le fichier de départ était mal formé, c'est à dire qu'il existerait plusieurs descriptions différentes pour le même *id*, il y aurait un problème lorsque l'on ajouterait une description dans une des listes. En effet on ne garde qu'une description pour chaque *id.* Donc nous avons choisi de ne garder que la première description rencontrée pour chaque *id* et de faire un fichier de test qui permet à l'utilisateur s'il le souhaite uniquement de vérifier qu'il n'y a pas de problèmes de ce genre avant d'utiliser le fichier de transformation.
- Il a fallu trouver une astuce (qui sera expliquée dans la partie conception et mise en oeuvre) permettant de traiter tout d'abord le remplissage des listes, puis la transformation des descriptions en références. Car nous avions un problème étant donné que la feuille de style ne prenait pas en compte les modifications au cours du traitement pour un traitement ultérieur, nous avions donc écrit deux feuilles de style que nous devions appliquer l'une après l'autre.

2.1.4 Tester si un fichier XML est CANONIQUE

On a crée un fichier de test appelé *MIF_ normalisedFormTester.xsl* qui permet de savoir si un fichier PSI est bien CANONIQUE. Ce test nous renvoie un fichier HTML qui indique les erreurs que le fichier peut contenir.

Dans cette partie, nous allons décrire l'outil permettant de vérifier si un fichier est canonique.
Il génère un fichier HTML contenant les erreurs eventuelles.
Cet outil a pour but de vérifier les points suivant :

- Vérifier si toutes les références sont valides. Une référence sera invalide si il ne correspond aucune description dont l'attribut *id* et égal à sont attribut *ref*.
- Vérifier s'il n'y a pas de description d'éléments au niveau des interactions. On recherche la présence des champs (*experimentDescription, availabilityDescription, proteinInteractor* au niveau d'une interaction.

2.1.5 Tester si un fichier XML est DÉROULÉ

On a crée un fichier de test appelé *MIF_ denormalisedFormTester.xsl* qui permet de savoir si un fichier PSI est bien DÉROULÉ. Ce test nous renvoie un fichier HTML qui indique les erreurs que le fichier peut contenir.

Pour cela on teste plusieurs points :

- On teste tout d'abord s'il y a des références au lieu de descriptions dans les intéractions.
  Si une telle erreur est détectée, elle sera indiquée dans la partie *References not allowed in the interaction elements.*
- On teste également s'il y a des descriptions dans les listes (experimentList, interactorList et availabilityList), car en DÉROULÉ il ne doit rien y avoir dans les listes.
  Quand une telle erreur est détectée, elle est indiquée dans la partie *Definitions in the lists of the top not allowed.*
- On vérifie que toutes les descriptions ayant un même *id* aient un contenu identique (cela ne teste que le contenu "écrit", mais pas les attributs qui pouraient se trouver à l'intérieur). Ceci est utile car lorsque l'on applique *MIF_FOLD.xsl*, on ne place dans les listes que la première description rencontrée ayant un certain *id*, il peut donc y avoir perte d'informations et c'est intéressant pour l'utilisateur de savoir quelle description sera conservée et quelles descriptions pouraient être perdues.
  Lorsqu'une telle erreur est rencontrée, elle est indiquée dans la partie *Unvailable descriptions.*

2.2 Description de l'outil de visualisation PSI

Il sagit d'un outil qui à partir d'un fichier PSI canonique retourne un fichier au format HTML qui présente les intéractions de façon synthétique.
La phase d'analyse de cet outil à été faites par Henning Hermjakob (acteur du projet PSI) qui nous a donné une version bien avancée de cet outil. Notre travail a consisté reprendre l'implémentation et à refaire la mise en forme de se document de façon à présenter le mieux possible les résultats.

2.3 Contenu du fichier

- On affiche d'abord les sources et les licences du fichiers.
- On construit un tableau dont chaque ligne est un participant à une intéraction. Le tableau présente les données : *interactorRef/@ref* or *proteinInteractor/names/shortLabel*, *participant/role*, *participant/isTaggedProtein*, *participant/isOverexpressedProtein*, *interactionType*.
- Pour plus de lisibilité les lignes sont regroupées par intéraction et on alterne la couleur d'affichage d'un partiant à chaque linge.

2.4 Transformation DIP vers PSI

2.4.1 Implémentation du format XML de DIP

Les fichiers DIP sont censés vérifier le schéma `XIN.xsd`. On peut représenter la structure de ces fichiers sous forme graphique.

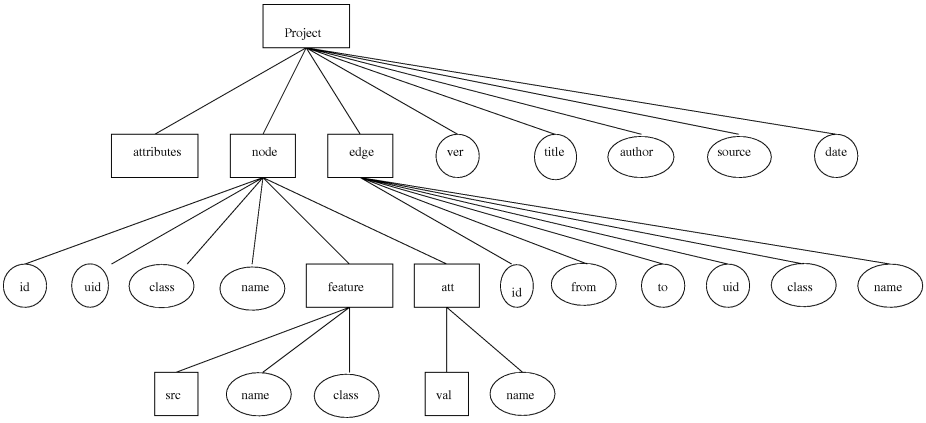


FIG. 2.3 – Structure du fichier DIP initial

Dans ce graphique, les rectangles représentent des éléments et les ellipses des attributs.

Balise project

La balise *project* est la racine du fichier XML. Elle a pour enfant des balises :

attributes Définition des attributs utilisés dans la suite du fichier

node Définition des protéines (interactor)

edge Définitions des interactions

Ses attributs sont :

title contient le titre du fichier

author contient l'auteur du fichier

source contient la source du fichier

date contient la date du fichier

ver contient la version du fichier

Balise attributes

La balise *attributes* défini les attributs *name* possibles pour les balises *att* dans les *node* et les *edge*. On ne sait traiter que les fichiers dont la balise *attributes* est :

```
<attributes>
 <edgeAtt name="class" type="text">
 </edgeAtt>
```

```
  <edgeAtt name="name" type="text">
  </edgeAtt>
  <nodeAtt name="descr" type="text">
  </nodeAtt>
  <nodeAtt name="shn" type="text">
  </nodeAtt>
  <nodeAtt name="organism" type="text">
  </nodeAtt>
  <nodeAtt name="taxon" type="text">
  </nodeAtt>
 </attributes>
```

Balise node

La balise *node* correspond aux *interactor* du fichier au format `MIFn`. Elle contient des balises :

att contient des informations sur la protéine

feature contient des références vers les autres bases de données.

Ses attributs sont :

id identifiant de la protéine dans le fichier

uid identifiant global

name nom de la protéine

class vaut "protein"

Tous ces attributs sont requis.

Balise node/att

La balise a un attribut*name* qui vaut :

shn représente le short name de la protéine.

descr représente une description de la protéine.

taxon représente le taxon de la protéine

organism représente l'organisme de la protéine.

Cette balise contient un élément *val* où est stockée la valeur de l'attribut.

node/feature

La balise a un attribut *class* qui vaut "cref". Elle correspond alors à une référence vers d'autres bases de données.
Elle a aussi un attribut *name* Cette balise contient un élément *src* où est stockée la valeur de cette référence.

Balise edge

La balise *edge* correspond aux *interaction* du fichier au format MIFn. Elle contient des balises :

att On ne sait pas à quoi correspond cette balise.

feature Décrit les expériences identifiant l'interaction.

Ses attributs requis sont :

id contient l'identifiant de l'interaction dans le fichier.

from contient une référence vers une protéine du fichier.

to contient une référence vers une protéine du fichier.

Ses attributs non requis sont :

uid contient un identifiant global.

class vaut "link"

name contient le nom le nom de l'interaction.

Balise edge/feature

Cet élément contient une référence vers une expérience. Elle a un attribut *uid*. Le nom de la base de donnée est stocké dans une balise *src* et le nom de l´ expérience dans la balise*val*

2.4.2 Problèmes rencontrés

Les fichiers DIP sous forme XML ne vérifient pas le schéma `XIN.xsd`. Par exemple l'attribut *name* est requis dans les éléments *feature* est souvent absent. Ainsi on matche l'attribut *uid* à la place bien qu'on ne soit pas sûr de sa présence. Les outils développés sont donc conçus autour des fichiers téléchargés et prennent moins en compte le fichier `XIN.xsd`.

2.5 Analyse et mise en oeuvre de la transformation de fichier PSI en fichier CSV

Le format PSI est un format XML particulier destiné à standardiser les formats XML de stockage de données tels que DIP par exemple. Ceci étant pris en compte, pour faciliter la consultation des données des fichiers au format PSI, il peut être intéressant de transformer ces fichiers en des fichiers avec un format aisément lisible par tout utilisateur du format PSI. Le format "Coma Separated Value" (CSV) semble remplir pleinement ce rôle puisqu'il est lisible par n'importe quel logiciel de tableur.

Dans cette section, nous alons présenter les étapes de l'élaboration de l'outil destiné à transformer des fichiers au format PSI canonique en des fichiers au format CSV.

2.5.1 Définition du format de sortie

structure globale

Comme nous l'avons déjà souligné, le format PSI est un format XML particulier. De ce fait, comme tout format XML, il possède une arborescence spécifique définie dans son schéma XSD. Or, un des buts de notre outil est de présenter de façon lisible l'ensemble des données contenues dans un fichier PSI. Par conséquent, notre première préoccupation dans ce projet a été de définir une structure pour les fichiers CSV générés par l'outil.

La solution la plus naturelle est de faire en sorte que les informations des fichiers CSV soient structurées d'une façon la plus proche possible de l'arborescence PSI. Or, en étudiant de près cette arborescence, on s'aperçoit que l'on peut décomposer un fichier PSI en quatre grande "parties" :

- availabilityList
- experimentList
- interactorList
- interactionList

Dès lors, il apparait logique de générer, dans notre fichier CSV, quatre tableaux, chacun contenant les données relatives à l'une des parties ci-dessus. Ainsi, on pourrait envisager de consacrer une colonne par champ correspondant à la valeur d'un noeud pour chaque "partie".

interactionList : le cas particulier

En théorie, la structure précédente parait naturelle. Cependant, le schéma XSD de PSI est conçu de telle sorte que le sous-arbre `interactionList` contient plusieurs `participants` et plusieurs `features` par interaction. Ceci représente donc un problème puisque cela implique que le tableau correspondant dans le fichier CSV pourrait avoir un nombre variable de colonnes. Ce serait alors la lisibilité qui en patirait ce qui irait à l'encontre du but initialement fixé.

Par conséquent, nous avons décidé de créer un tableau `interactionList` comportant, non plus une ligne par interaction, mais une ligne par participant à l'interaction. On répète alors les informations des interactions mais on gagnera nettement en clarté du fichier résultat. Ajouté à cela, on crée un tableau spécifique pour les `features`(nommé `featureList`) contenant une ligne par `feature` en indiquant, pour chacun de ces derniers, une référence vers l'interaction à laquelle il se rapporte.

Ainsi, on obtient un fichier CSV totalement formaté et donc plus lisible. En outre, ce formatage permet de faciliter l'analyse des fichiers CSV dans l'eventualité où on voudrait élaborer un outil permettant la transformation de fichier CSV en fichiers PSI. Pour de plus amples détails sur le format CSV retenu, nous invitons le lecteur à se reporter à la section *Conception* relative à cet outil dans la suite de ce rapport.

Chapitre 3

Conception

3.1 Transformations PSI canonique et PSI déroulé

3.1.1 Transformation de PSI canonique vers PSI déroulé

Nous avons vu dans la partie analyse et spécification les différentes opérations à effectuer sur le fichier de départ. Pour ce faire, l'implémentation n'est pas très compliquée, elle se fait en plusieurs opérations :

- Il faut tout d'abord remplacer les références par des descriptions. Expliquons comment on procède pour les experimentRef, cela se passe ensuite de la même manière pour les availabilityRef et les interactorRef.
  Pour les experimentRef :
  ```
  <xsl:template match="psi:experimentRef">
    <xsl:copy-of select="/psi:entrySet/psi:entry/psi:experimentList/
  psi:experimentDescription[@id=current()/@ref]"/>
  </xsl:template>
  ```
  On se place tout d'abord sur le bon template, là où on a une modification à effectuer, puis on applique un <xsl :copy-of select=...> dans lequel on selectionne le contenu que l'on veut mettre à la place de l'experimentRef.
- Ensuite, il faut vider le contenu des listes. Expliquons comment on procède pour experimentList, cela se passe ensuite de la même manière pour availabilityList et interactorList. Pour experimentList :
  ```
  <xsl:template match="psi:experimentList">
      <xsl:copy/>
  </xsl:template>
  ```
  On se place sur le bon template, puis on effectue un <xsl :copy/> qui permet de rendre le contenu de la liste vide tout en gardant les balises de la liste (<experimentList/>).
- Enfin, on recopie tous les autres noeuds tels quels :
  ```
  <xsl:template match="/ | @* | node()">
    <!-- we make a shallow copy... -->
    <xsl:copy>
      <!-- and continue -->
      <xsl:apply-templates select="@* | node()"/>
    </xsl:copy>
  </xsl:template>
  ```

3.1.2 Transformation de PSI déroulé vers PSI canonique

Nous avons utilisé des clés pour remédier au problème de la superposition de deux feuilles de style.
Prenons l'exemple avec le replacement dans la liste experimentList des experimentDescription, puis la transformation de ces descriptions en références. (Cela se passe de la même manière pour les availability et les proteinInteractor).

- Tout d'abord, on initialise une clé :
  ```
  <xsl:key name="exp-ids" match="//psi:experimentDescription" use="@id"/>
  ```
  Celle-ci contient tous les experimentDescription rencontrés et ils sont identifiés par leur *id*.
- Ensuite, on se place au niveau de l'experimentList, et on compare chaque experimentDecription avec les éléments de la clé grace à la fonction *generate-*

id. Cette fonction permet de créer pour chaque noeud un numéro unique.

```
<xsl:template match="psi:experimentList" >
  <xsl:element name="experimentList">
    <xsl:for-each select="//psi:experimentDescription[generate-id(.)=
                          generate-id(key('exp-ids', @id)[1])]">
      <xsl:sort select="@id"/>
      <xsl:copy-of select="."/>
    </xsl:for-each>
  </xsl:element>
</xsl:template>
```

A chaque fois que l'on rentre dans la boucle <xsl :for-each ...>, on copie la description trouvée dans la liste en la triant par rapport à l'id.
Donc après cette opération, on a bien dans les listes toutes les descriptions triées par rapport à leurs id et on n'a qu'un représentant par id.

– Enfin, on remplace toutes les descriptions dans les interactions en références.

```
<xsl:template match="psi:interactionList//psi:experimentDescription" >
  <xsl:element name="experimentRef">
    <xsl:attribute name="ref">
    <xsl:value-of select="@id"/>
  </xsl:attribute>
</xsl:element>
</xsl:template>
```

On se place sur les experimentDescription qui se trouvent dans interactionList, et on crée une balise <experimentRef> qui a comme attribut un ref dont la valeur est celle de l'id qui décrivait précédemment la description.

A la fin il ne reste plus qu'à recopier tous les noeuds et leur contenu :

```
<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="@* | node()" >
  <xsl:copy>
    <xsl:apply-templates select="@* | node()"/>
  </xsl:copy>
</xsl:template>
```

3.1.3 Teste si un fichier XML est CANONIQUE

Ce fichier de test renvoie un fichier HTML qui indique les erreurs(s'il y en a) rencontrées dans un fichier CANONIQUE (cf partie analyse et spécification).

On vérifie si toutes les références sont valides. Une référence sera invalide si elle ne correspond à aucune description dont l'attribut *id* est égal à son attribut *ref*.
On vérifie s'il n'y a pas de description d'éléments au niveau des interactions. On recherche la présence des champs (*experimentDescription, availabilityDescription, proteinInteractor* au niveau d'une interaction.

Le resultat est un rapport au format HTML qui décrit toutes les erreurs.

3.1.4 Teste si un fichier XML est DÉROULÉ

Ce fichier de test renvoie un fichier HTML qui indique les erreurs(s'il y en a) rencontrées dans un fichier DÉROULÉ (cf partie analyse et spécification).

On vérifie s'il n'y a pas de références d'éléments au niveau des interactions. On recherche la présence des champs (*experimentRef, availabilityRef, InteractorRef.*
On vérifie que les listes d'en-tête sont vides. On recherche la présence des champs (*experimentDescription, availabilityDescription, proteinInteractor* dans les listes *availabilityList, experimentList* et *interactorList.*
On vérifie que tous les éléments ayant un même identificateur ont le même contenu.

Le resultat est un rapport au format HTML qui décrit toutes les erreurs.

3.2 Description de l'outil de visualisation PSI

3.3 Contenu du fichier

3.3.1 Sources

Dans cette partie du fichier on affiche le noms des sources du fichier à partir du champ *source/names/shortLabel*.

3.3.2 Availability List

Dans cette partie du fichier on affiche les licences à partir du fichier à partir du champ *avaibility*.

3.3.3 Interaction List

On construit un tableau dont chaque ligne est un participant à une intéraction. Le tableau présente les données : *interactorRef/@ref* or *proteinInteractor/names/shortLabel*, *participant/role*, *participant/isTaggedProtein*, *participant/isOverexpressedProtein*, *interactionType*.

Pour plus de lisibilité les lignes sont regroupées par intéraction et on alterne la couleur d'affichage d'un partiant à chaque linge.

3.4 Transformation DIP vers PSI

Pour construire le fichier résultat PSI au format canonique, on construit l'arbre résultat et on appelle les templates pour sélectionnner les informations du fichier source.

3.4.1 Feuille de style de transformation

On présente ici les différentes templates de la transformation `DIP_TO_MIFn`. La première règle matche la racine. Elle traite la balise *project* et crée les balises *entrySet* et *entry*. Elle crée les attributs de *entrySet* : *level* initialisé à 1 et *version* initialisé à 12 puisque le fichier résultat vérifie la version 1.12 du schéma `MIF.xsd`. Elle appelle les autres templates sur le reste du fichier DIP.

Template project

La seconde règle matche la balise *project*.Cette règle gère les attributs de la balise *project* et génère la *source* dans le fichier PSI. Elle appelle d'autres règles sur les attributs *ver*, *author*, *source* et *date* afin de les recopier dans la *source* du fichier destination. Cela permet de ne créer des balises *attribute* que lorsque c'est nécessaire.
Ensuite cette règle crée les balises *interactorList* et *interactionList* puis appelle les règles sur les *node* et les *edge*.

Template node

Cette règle ne matche que les *node* dont l'attribut *class* vaut *protein*. Elle génère l'identifiant de la protéine pour le fichier PSI, crée la balise *names* et y recopie les *shn* et *descr* si ils existent. Elle s'occupe des *feature* grâce à une autre template et les recopie dans les *xref* de la protéine dans le fichier PSI. Ensuite elle crée un élément organism pour la protéine dans le fichier PSI si les attributs *taxon* et *organism* sont présents dans le fichier DIP.

Template feature

Cette règle matche les balises *feature* dont l'attribut *class* vaut *cref*. Ces balises correspondent à des *secondaryRef* dans le fichier PSI.

Template edge

Cette règle ne matche que les *edge* dont l'attribut *class* vaut*link*. Ces balises correspondent à des interactions. Elle crée un élément *names* lorsque c'est possible. Ensuite elle crée la balise *participantList* et crée les deux *interactorRef* des *proteinParticipant* correspondant aux références des attributs *from* et *to* de la balise *edge*. Étant donné que l'on n'a aucune information sur les expériences, on préfère ne pas créer d'*experimentLists*, plutôt que d'inventer une *interactionDetection* vide, ce qui malheureusement ne vérifie pas le schéma défini dans le fichier `MIF.xsd`.

3.4.2 Feuille de style de validation

Cette feuille de style *DIP_TO_MIFn_report* matche toutes les balises qui n'ont pas été traitées par la transformation *DIP_TO_MIFn* et les écrit de façon structurée dans un rapport au format HTML. Ainsi toute information est traitée soit par la transformation *DIP_TO_MIFn* soit par la validation *DIP_TO_MIFn_report*. La feuille de style de validation utilise un parcours semblable à celui de la transformation décrit précédemment.

3.5 Description de l'outil de transformation PSI vers CSV

Comme nous l'avons déjà précisé précédemment, un de nos buts était de réaliser un outil permettant de transformer un fichier XML au format PSI canonique en un fichier au format CSV. La solution la plus naturelle et sans doute, dans notre cas, la plus performante était de réaliser cet outil sous la forme d'une feuille de style au format XSLT.

Nous allons tout d'abord décrire la structure du fichier CSV obtenu après transformation du fichier PSI initial avant de présenter l'outil XSLT en lui-même.

3.5.1 Structure du fichier cible

Le format CSV

Le fichier résultat de l'application de notre outil au fichier XML est un fichier de type Coma Separated Value (CSV). Ce format décrit une structure de tableur, c'est à dire qu'il s'agit d'un fichier texte dans lequel les données sont alignées en colonne. Le saut d'une colonne à une autre est représenté par le caractère "point-virgule" et le retour à la ligne est représenté par le caractère "retour chariot". Le fichier résultat fournit en fait 5 sous-tableaux les uns en dessous des autres.

La structure du fichier résultat

Le fichier résultat récapitule les données du fichier PSI sous forme de 5 tableaux :

- Un tableau intitulé **[_Availability List_]** qui liste les licences et droits relatifs aux données contenues dans les fichiers. Les données de ce tableau sont celles contenues dans les noeuds de l'arborescence PSI de racine le noeud `availabilityList`. Il en est de même pour l'ensemble des cinq tableaux.

- Un tableau intitulé **[_Experiment List_]** qui liste les experiences et leurs caracteristiques. Les premiers champs *ID, shortlabel* et *fullname* permettent une identification rapide de chaque experience.

- Un tableau intitulé **[_Interactor List_]** qui liste les interacteurs et leurs particularités. Les premiers champs *ID, shortlabel* et *fullname* permettent une identification rapide de chaque interacteur.

- Un tableau intitulé **[_Interaction List_]** qui liste les interactions et leurs propriétés. Il est constitué d'une ligne par participant. Le premier champ, *No_participant*, indique un numero de participant par interaction (généré par la feuille de style) et le second, *interaction_id*, l'ID de l'interaction.

- Un tableau intitulé **[_ Feature List _]** qui liste les descriptions des dispositifs d'experimentation. Le premier champ, *interaction_ id*, indique l'ID de l'interaction correspondant à ces descriptions.

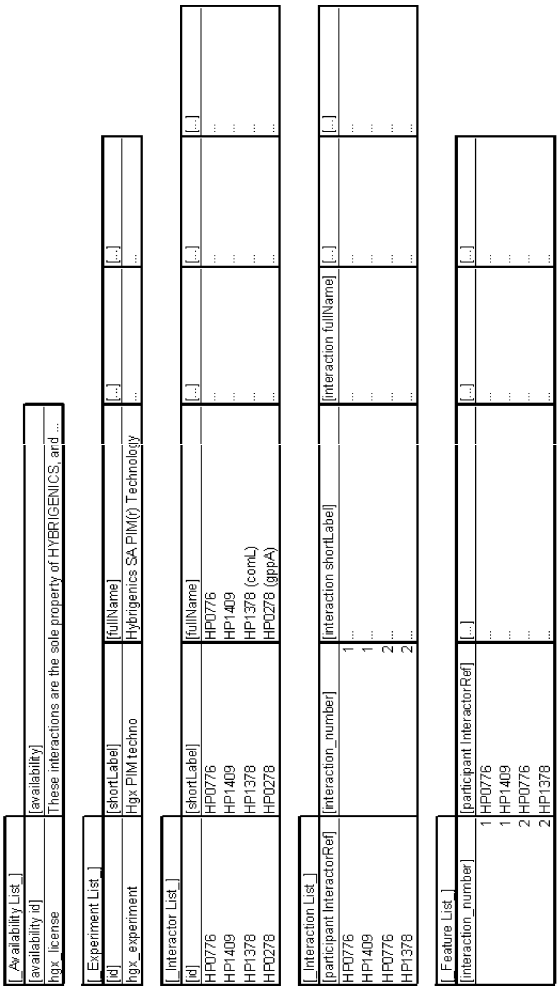


FIG. 3.1 – Structure du fichier CSV final

3.5.2 Structure de l'outil lui-même

Structure commune à tous les tableaux

A chaque tableau est associé un `template` particulier. Cependant, ces derniers ont une structure commune. En effet, chaque `template` débute par le codage en dur des en-têtes de colonnes du tableau associé. Cela consiste en l'écriture de chaque champ du tableau entre crochets, chaque en-tête étant séparé par un point-virgule. De cette manière, on définit la seconde ligne du tableau avec autant de colonnes qu'il n'y a de champs correspondant dans l'arborescence PSI.

En outre, chaque `template` doit permettre de gérer le fait qu'un tableau peut être constitué de plusieurs lignes. De ce fait, lors de l'appel à un `template` dans le `template` "général", on se place au noeud correspondant dans l'arborescence de PSI (par exemple, l'appel *<xsl :apply-templates select="availabilityList"/>* se place au niveau du noeud */entryset/entry/availabilityList* de l'arborescence). Ainsi, si on garde l'exemple, chaque ligne du tableau `availabilityList` va contenir une occurence de `availability`. Dès lors, pour traiter les informations de chaque `availability`, on fait appel à la fonction `xsl :for-each` qui initialise une boucle permettant de traiter les mêmes informations pour chaque ligne et de générer ces dernières les unes en dessous des autres.

En bref, chaque `template` contient la définition en dur des en-têtes de colonnes du tableau qui lui est associé et une boucle permettant d'inscrire les informations du tableau (provenant du fichier xml source) ligne par ligne.

Le tableau availabilityList

Ce tableau a une structure strictement identique à celle décrite ci-dessus. Par ailleurs, chaque noeud ayant pour père un noeud `availability` a sa valeur affichée dans la colonne correspondante du tableau via un appel à la fonction `xsl :value-of`.

Le tableau experimentList

Ce tableau a une structure strictement identique à la structure commune décrite précédemment. Par ailleurs, chaque noeud ayant pour père un noeud `experimentDescription` a sa valeur affichée dans la colonne correspondante du tableau via un appel à la fonction `xsl :value-of`.

Le tableau interactorList

Ce tableau a une structure strictement identique à la structure commune décrite précédemment. Par ailleurs, chaque noeud ayant pour père un noeud `interactor` a sa valeur affichée dans la colonne correspondante du tableau via un appel à la fonction `xsl :value-of`.

Le tableau interactionList

Ce tableau n'est pas construit de la même manière que les autres. En effet, il a la particularité suivante : au lieu de consacrer une ligne par interaction (comme on a dédié une ligne par `interactor` dans le tableau `interactorList` par exemple), on associe une ligne à chaque participant de l'interaction.

Plus en détails, étant donné que chaque interaction peut faire intervenir plusieurs participants, il apparait plus performant de consacrer une ligne par participant d'une interaction tout en stockant un champ de référence unique pour chaque interaction étudiée. Ceci facilite la construction du tableau qui, sinon, aurait un nombre de colonnes variable dépendant du nombre de participants.

Ceci étant dit, cette solution d'implémentation introduit de nouveaux problèmes. En effet, pour chaque participant, on va recopier les informations liées à son interaction dans le tableau. Par conséquent, outre la redondance d'informations occasionnée qui n'est pas gênante dans notre cas, on va être obligé de rechercher toutes les informations d'une interaction à chaque fois que l'on renseigne les données d'un de ses participants.

Pour palier à cette recherche inutile, nous avons déclaré un certain nombre de variables destinées chacune à stocker une information d'une interaction. Ainsi, lors de l'appel au `template`, on va traiter chaque interaction l'une à la suite de l'autre. Pour une interaction, on stocke ses données "propres" dans ces variables et, pour chacun de ses participants, on remplit une ligne du tableau en recherchant les données du participant lui-même et en faisant appel directement aux variables (c'est là qu'on économise sur la recherche) pour remplir les données liée à l'interaction. Dès que l'on a traité tous les participants d'une interaction, on stocke dans les mêmes variables les données relatives à l'interaction suivante dans la liste.

Ainsi, à chaque fois que l'on traite un participant, on n'est pas obligé de rechercher les données de l'interaction qui lui est associée, il suffit de faire appel aux variables dans lesquelles ces données sont stockées.

Enfin, comme nous l'avons souligné plus tôt, nous avons décidé de différencier les interactions les unes des autres en créant, via la commande *<xsl :value-of select="position()"/>* , un identifiant unique s'incrémentant automatiquement pour chacune d'entre elles. Ainsi, la première colonne de notre tableau contient l'identifiant permettant de rendre unique chaque participant, tandis que la seconde contient un identifiant permettant de rendre unique chaque interaction.

Le tableau featureList

D'après l'arborescence PSI, il peut y avoir plusieurs `feature` par interaction. De même que précédemment, cela pose un problème puisque le tableau `interactionList` peut alors avoir un nombre variable de colonnes. C'est pourquoi nous avons créé ce tableau.

Sa structure est identique à la structure générale décrite précédemment. Cependant, pour bien remettre chaque `feature` dans son contexte, il faut ajouter une colonne à ce tableau contenant une référence (un identifiant) à l'interaction à laquelle il se rapporte. Or, ceci entraine un problème, à savoir que les références en question doivent être identiques à celles générées dans le tableau `interactionList` décrit dans la section précédente pour que l'utilisateur final puisse se repérer.

Le problème est que, ces deux tableaux étant générés dans des `template` différents, les variables décrites précédemment n'ont pas une portée suffisante pour être réutilisées. Comment faire pour avoir une correspondance entre les identifiants des deux tableaux ?

La solution consiste à se replacer au niveau du noeud `interactionList`, ce qui nous permet de recommencer à numéroter chaque interaction de la même façon que précédemment et de gérer les `feature` au fur et à mesure. Ceci pose alors un nouveau problème au niveau du `template` "général", à savoir que pour générer les deux tableaux `interactionList` et `featureList`, on serait alors amenés à appeler le même `template` (au niveau du noeud `interactionList`) ce qui entrainerait une ambigüité. C'est pourquoi nous avons fait usage de l'attribut `mode` de la balise `template` qui nous permet de nous placer au même niveau de l'arborescence pour chacun des deux `template` tout en effectuant des traitements différents pour chacun d'eux.

Notre problème se trouve alors résolu puisque nos deux tableaux sont générés et que les identifiants des interactions correspondent dans les deux tableaux.

Chapitre 4

Bilan et perspectives

Au terme de ce projet, les objectifs fixés dans le cahier des charges ont tous été atteints. Il a en plus été réalisé des outils de tests et de visualisation des fichiers PSI.
Tous les outils réaliés pour effectuer les tranformations ont été testés sur un ensemble de fichiers fournis soit par le client, soit par les organismes concernés (DIP par exemple), et ont fourni un résultat satisfaisant les contraintes de validité définies dans le cahier des charges.
On peut cependant envisager d'étendre le nombre de transformations réalisées aux transformations de la figure 1.1 non encore réalisées.
C'est à dire :

- PSI vers DIP
- BIND vers PSI et réciproque
- CSV vers PSI

Chapitre 5

Bibliographie

- **http ://psidev.sourceforge.net/** Le site de PSI fournit toutes les informations sur la **P**roteomics **S**tandards **I**nitiative, plus particulièrement sur la XSD de PSI. Il est possible d'avoir accès aux mailing lists ce qui nous a permis d'être au courant des modifications de la norme PSI.

- **http ://dip.doe-mbi.ucla.edu/** Le site de DIP nous a fourni les exemples qui ont été utilisés pour tester nos outils, ainsi que la norme DIP.

- **http ://www.w3.org/Style/XSL/** Ce site donne la spécification de XSL, les éléments que l'on peut utiliser et leur fonctionnalités.

- **http ://www.w3.org/XML/Schema** Ce site donne la définition des schemas XML.

- **http ://www.enseirb.fr/ thomas-n/PFA/** Ce site a été réalisé par un membre de notre équipe pour permettre une communication plus efficace. Il contient un forum de discussion où étaient déposées les remarques, les compte-rendus de réunions avec le client, ainsi que la répartition des tâches associées à chaque étape du projet.

Chapitre 6

Glossaire

- **CSV** : Coma Separated Value, ou CSV, est un format de données. Un fichier de ce type contient des valeurs dans une table représentée par une série de lignes de caractères organisée de telle manière que chaque colonne est séparée de la colonne suivante par un point-virgule (ou une virgule, ou une tabulation...) et que chaque retour chariot désigne la fin d'une ligne.

- **DIP** : Database of Interacting Proteins, ou DIP désigne une base de données d'interactions protéine-protéine.

- **Processeur XSLT** : Le processeur XSLT, composant logiciel chargé de la transformation du fichier XML, crée un structure logique arborescente à partir du document XML et lui fait subir des transformations selon les `templates` contenus dans la feuille XSL pour produire un arbre résultat représentant la structure d'un nouveau document au format XML, CSV, HTML...

- **PSI** : Proteomic Standard Initiative, ou PSI, désigne un format de données. Ce format a été développé afin de standardiser les format de données relatives aux interactions protéine-protéine. Il existe deux variantes de ce format nommées PSI canonique et PSI déroulé.

- **Template** : Chaque `template` définit des traitements à effectuer sur un élément (noeud ou feuille) de l'arborescence XML.

- **XIN** : Le format XIN est un format XML très flexible qui permet de décrire des graphes. La base de donnée DIP est une application particulière de ce format.

- **XSD** : XML Schema Definition. Fichier XML donnant la définition d'un format de fichier XML

- **XML** : eXtensible Markup Language, ou XML, est un méta-langage extensible dérivé de SGML permettant de structurer des données.

- **XSLT/feuille de style** : eXtensible Stylesheet Language Transformation, ou XSLT, est un langage permettant de transformer la structure des fichiers au format XML. Ainsi, un document XML peut être représenté comme une structure arborescente. XSLT permet alors de transformer les documents XML à l'aide de feuilles de style contenant des règles appelées `template`.