DESS Informatique et Information dans les Réseaux de Soins,
Faculté de Médecine H. Warembourg
Université du Droit et de la Santé Lille II
promotion 2004-2005

Mémoire de stage

Emmanuel CHAZARD
Département de l'Information Médicale du CHRU de Lille
Chef de service : Professeur Régis Beuscart
Maîtres de stage : Docteur Cécile Grave, Monsieur François Delaby
Tuteur universitaire : Monsieur François Delaby

Remerciements

Monsieur le Professeur Régis Beuscart
Professeur de Biostatistiques et d'Informatique Médicale
Praticien Hospitalier – Chef de service du Département d'Information du CHRU de Lille
Directeur du Centre d'Etudes et de Recherche en Informatique Médicale
Officier dans l'Ordre des Palmes Académiques
Je vous remercie de m'avoir accueilli au DIM et de m'avoir autorisé et encouragé à suivre l'enseignement du DESS.

Madame le Docteur Cécile Grave
Praticien Hospitalier en Information et Informatique médicale
Je vous remercie de vos conseils avisés, guidés par l'expérience médicale.

Monsieur François Delaby
Ingénieur informaticien
Je vous remercie de votre bienveillant encadrement, votre disponibilité, vos réponses précises.

Je tiens également à remercier toute l'équipe du DIM pour son accueil chaleureux.

Je tiens enfin à remercier l'équipe enseignante du DESS IIRS, pour la richesse de son enseignement et la qualité du suivi pédagogique.

Table des matières

Vue d'ensemble du projet DIND

I. Le PMSI MCO en quelques mots

Le Programme de Médicalisation des Systèmes d'Information en court séjour (Médecine Chirurgie Obstétrique) consiste en un recueil systématique et obligatoire d'informations sur les séjours hospitaliers, publics et privés. Le MCO (court séjour médecine chirurgie obstétrique) est le principal domaine du PMSI. Il existe aussi un PMSI moyen séjour (SSR), et un PMSI hospitalisation à domicile (HAD), dont nous ne traitons pas ici.
Pour chaque patient, l'établissement recueille des informations d'état civil, des diagnostics médicaux, des actes médicaux, et d'autres informations.
Les diagnostics sont codés grâce à la CIM 10 (classification internationale des maladies).
Les actes médicaux sont codés grâce à la CCAM (classification commune des actes médicaux), ou anciennement le CdAM (catalogue des actes médicaux).

Toutes ces informations permettent de créer des fichiers de RUM (Résumé d'Unité Médicale), à chaque passage dans une unité médicale (exemple : « le couloir ouest 2° étage du service de neurologie »). Ces informations sont formatées grâce à une description de format qui change régulièrement. On a donc 1 ligne de texte formaté par passage dans une unité médicale, soit une ou plusieurs lignes par séjour de patient.
Lorsque le patient quitte l'établissement ou décède, la ou les lignes qui correspondent à son séjour sont agrégées et anonymisées, et constituent une seule ligne du fichier de RSA (Résumé de Sortie Anonymisé), dont le format est lui aussi décrit et change régulièrement. On n'a plus qu'une ligne par séjour.

Chaque RSA est produit par un programme de groupage, qui permet d'affecter les séjours dans des catégories plus ou moins fines nommées GHM (Groupe Homogène de Malade - exemple : 01C02Z Craniotomies, âge inférieur à 18 ans), affectées d'un tarif de base parfois modulable.

Le fichier de RSA est envoyé périodiquement au ministère. L'activité ainsi présentée permet d'allouer une dotation à l'établissement, dans le cadre de la T2A (Tarification A l'Activité). Les séjours sont valorisés un par un, et l'assurance maladie peut ainsi financer l'établissement. Il existe des modalités complémentaires de financement de l'établissement.

II. L'application DATIM du ministère

L'établissement transmet son fichier de RSA au ministère via une plate-forme électronique baptisée e-PMSI. Cette plate-forme contient entre autres l'application DATIM (Détection des ATypies de l'Information Médicale). DATIM est une application utilisée par le ministère pour détecter des atypies sur les fichiers de RSA des établissements hospitaliers. Le ministère a publié les 64 algorithmes de test. Voici deux exemples simples de tests :

- le test 23 compte les RSA pour lesquels le diagnostic est incompatible avec le sexe (exemple : accouchement chez un homme). Ces RSA sont alors visualisables, à ceci près qu'ils sont anonymes. Dans ce test, on voit bien qu'un RSA est à lui seul bon ou mauvais.
- Le test 4 calcule le pourcentage de décès pour chaque GHM, puis il compare ce pourcentage à la moyenne nationale. En cas d'atypie, une alerte est déclenchée. Dans ce test, on remarquera qu'aucun RSA n'est à lui seul bon ou mauvais, mais que c'est l'atypie statistique de l'ensemble du groupe qui est signalée.

La plupart des tests génèrent un score en unité arbitraire, ce score est d'autant plus important que l'atypie est avérée. Un score total est ainsi calculé. Un score total élevé est susceptible d'entraîner une procédure de contrôle administratif pour s'assurer :

1- que les RSA sont conformes aux séjours réels des patients
2- que les séjours réels des patients correspondent à des pratiques médicales ou administratives recommandées et honnêtes.

III. L'application DIND

A. La demande du stage

L'objet initial du stage était d'implémenter les algorithmes DATIM dans une application maison. Cette application devait être dotée de deux fonctionnalités supplémentaires :

1- faire perdre l'anonymat aux résumés atypiques
2- pouvoir, d'une manière ou d'une autre, cibler des cliniques ou CR (centres de responsabilité : sous-groupes de l'établissement) plus génératrices d'atypie que d'autres.

B. Notre application en bref

Nous avons baptisé cette application DIND, acronyme récursif signifiant Dind Is Not Datim, conformément au folklore informatique. Le principal de ce mémoire est cependant constitué par l'aide fournie avec DIND, qui vous est livrée en deuxième partie.

DIND est un moteur générique de tests sur des résumés PMSI. DIND agît en quatre phases :

1- une première phase de configuration simple consiste à choisir les fichiers à utiliser
2- puis DIND fusionne un fichier de RSG et un fichier de RSA pour réaliser des « résumés composites », non anonymes, qui contiennent à la fois toutes les informations agrégées et tarifaires du RSA, et toutes les informations des différents RSG (ou RUM) initiaux. Ces informations supplémentaires permettent une perte d'anonymat complète (numéro de RSS, de séjour, date de naissance, date d'entrée ou de sortie, code postal..) mais également apportent des informations sur le processus chronologique de prise en charge : passage dans les unités médicales, informations entrées à ce moment-là…
3- ensuite DIND applique le fichier de règles choisi (qu'il s'agisse des règles DATIM ou de tout autre jeu de règles)
4- enfin l'affichage des résultats de ces règles est simple et complet

DIND utilise en outre de nombreux fichiers de ressources, contenant entre autres les différentes nomenclatures nécessaires, le tout conçu avec le plus d'abstraction possible. Tout ce que vous souhaitez savoir précisément se trouve dans l'aide de DIND.

IV. Conclusion

Le sujet de stage initial est atteint : fournir au DIM (Département d'Information Médicale) un outil de détection d'atypies permettant d'une part de corriger ces atypies, d'autre part de préparer une intervention pédagogique. Au-delà, nous avons écrit des classes qui ne demandent qu'à être réutilisées dans d'autres applications traitant du PMSI. Plus encore, nous pensons que ces autres application devraient plutôt être des cas d'utilisation de DIND, des nouveaux jeux de règles, et non de nouvelles applications.
En perspectives, nous souhaitons pouvoir faire évoluer DIND dans les mois qui viennent :

- améliorer l'aspect directement opérationnel des résultats produits
- engager le programme sur des voies plus diversifiées.

Aide de DIND

Souvent, des programmes conçus en interne, même réussis, tombent en désuétude car les utilisateurs ne s'émancipent jamais du concepteur du programme. Afin d'éviter cet écueil, nous avons dès le début consacré un effort important à la rédaction de l'aide du programme. C'est la raison pour laquelle, dans ce mémoire, nous avons pris le parti de vous livrer l'aide en l'état, jugeant qu'il était toujours plus utile de mettre à jour l'aide qu'un mémoire ponctuel.

Avant de commencer, vous devriez examiner le paragraphe **Précisions diverses > Nommage des résumés PMSI**.

I. Introduction

A. Présentation du programme original DATIM

DATIM (**D**étection des **AT**ypies de l'**I**nformation **M**édicale) est une application web implantée sur la plate-forme e-PMSI. Elle analyse tous les fichiers de RSA télé-transmis par les établissements aux tutelles via cette plate-forme. Sur chaque fichier, DATIM recherche des "atypies" en appliquant 64 algorithmes de contrôle, et fournit finalement un score.
Cette analyse est effectuée avant validation par l'établissement. En revanche, le contrôleur n'a accès à cette information qu'après validation. L'établissement peut ainsi corriger le tir, à ceci près que les RSA sont anonymes et ne permettent donc de retrouver ni les RSS incriminés, ni les unités médicales génératrices de ces atypies.
Après validation, le résultat de l'analyse est susceptible de déclencher un contrôle administratif dans l'établissement, afin de déceler des anomalies de codage ou de prise en charge, voire d'infliger des sanctions financières. Les contrôleurs peuvent confronter les résultats de tous les établissements d'une même région.
Les sources de DATIM ne sont pas publiées. En revanche, les 64 algorithmes sont publiés sous forme papier (PDF téléchargeable sur le site www.atih.sante.fr).

B. Présentation de DIND du CHRU de Lille

DIND est un acronyme récursif signifiant *Dind Is Not Datim*, répondant partiellement à la question. En quelque sorte, DIND vous évitera d'être le dindon de la farce hospitalière.
DIND est un programme qui implémente les algorithmes DATIM dans un ensemble de scripts PHP réalisant une application destinée à être utilisée sur un poste client uniquement, mais bénéficiant ainsi de l'interface d'un navigateur web. Il propose en outre des fonctionnalités supplémentaires :

- le stockage des algorithmes de contrôle à l'extérieur du programme permet de mettre à jour facilement DIND, mais également d'utiliser des règles utilisateur pour d'autres études (ONCODIM, requêtes récurrentes...)
- le stockage séparé des descriptions de formats et des tables de nomenclatures (dans le format natif du kit de nomenclatures ATIH) permet des mises à jour aisées
- il est possible d'enrichir les résumés à l'aide de tables externes. Par exemple, cela permet de réaffecter aux RUM des informations de structure (service, clinique), ou encore un numéro de patient ou de dossier.
- l'originalité du système est de parcourir dans le même temps chaque RSA et ses RUM constitutifs (en fait ses RSG, voir urgemment le chapitre **Précisions diverses > Nommage des résumés PMSI**). Cela permet de réaffecter à un RSA un numéro de RSS et une liste d'unités médicales parcourrues. Cela peut permettre par exemple de

recoder certains RSA problématiques et de cibler des unités médicales génératrices d'atypie, et donc candidates à des interventions de formation par le DIM. Cela permet également d'étendre les requêtes DIND non seulement aux RSA complets, mais également aux RSG constitutifs.
- la gestion de projets et l'archivage des résultats de tests permettent une utilisation interactive et non linéaire. La convivialité de l'édition des résultats et des résumés sélectionnés permet une analyse agréable.

C. Réalisation

DIND a été programmé par Emmanuel CHAZARD dans le cadre de son stage de DESS IIRS (**I**nformatique et **I**nformation dans les **R**éseaux de **S**oins, Lille 2) réalisé au DIM du Professeur Régis BEUSCART au CHRU de Lille, sous le bienveillant encadrement de François DELABY et du Docteur Cécile GRAVE.

II. Les trois rôles dans DIND

A. Analyse

Afin de simplifier la documentation et l'accès aux fonctions, nous avons défini trois rôles.

1. Le rôle utilisateur :

Rôle : il applique un traitement tout prêt aux résumés.
Connaissances PMSI : c'est un connaisseur du PMSI "côté sens".
Connaissances informatiques : il sait ce qu'est un fichier, utiliser une souris.
Connaissance de DIND : il suit simplement les 4 étapes.

2. Le rôle rédacteur de règles :

Rôle : il rédige les règles. Toutefois, son travail n'est opérationnel que dans le contexte plus global d'un traitement. Cette intégration repose sur l'administrateur de DIND.
Connaissances PMSI : c'est un connaisseur du PMSI côté format des données.
Connaissances informatiques : c'est un programmeur débutant en PHP procédural.
Connaissance de DIND : il doit seulement savoir accéder aux variables des résumés et des tables de référence. Cet accès est largement documenté dans les annexes dynamiques de l'aide.

3. Le rôle administrateur :

Rôle : il met à jour DIND avec les nomenclatures. Mais surtout, il fournit un cadre à chaque type de traitement en préparant les fichiers de projet.
Connaissances PMSI : c'est un connaisseur du PMSI côté nomenclatures.
Connaissances informatiques : il doit savoir modifier un document XML.
Connaissance de DIND : il doit connaître précisément les ressources utilisées par DIND.

B. Synthèse

Un processus complet suit cet ordre :

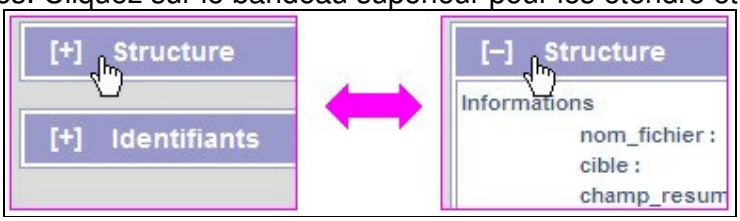
1. l'administrateur de DIND prépare un fichier de projet ou un modèle de fichier de projet, appelant toutes les ressources nécessaires au traitement.
2. en utilisant ce fichier de projet, le rédacteur de règles peut ensuite rédiger les règles.
3. en utilisant ce fichier de projet, l'utilisateur peut ensuite, très simplement, appliquer les règles à des fichiers de données.

III. Ce (et seulement ce) que vous devez savoir...

A. ... en tant qu'utilisateur

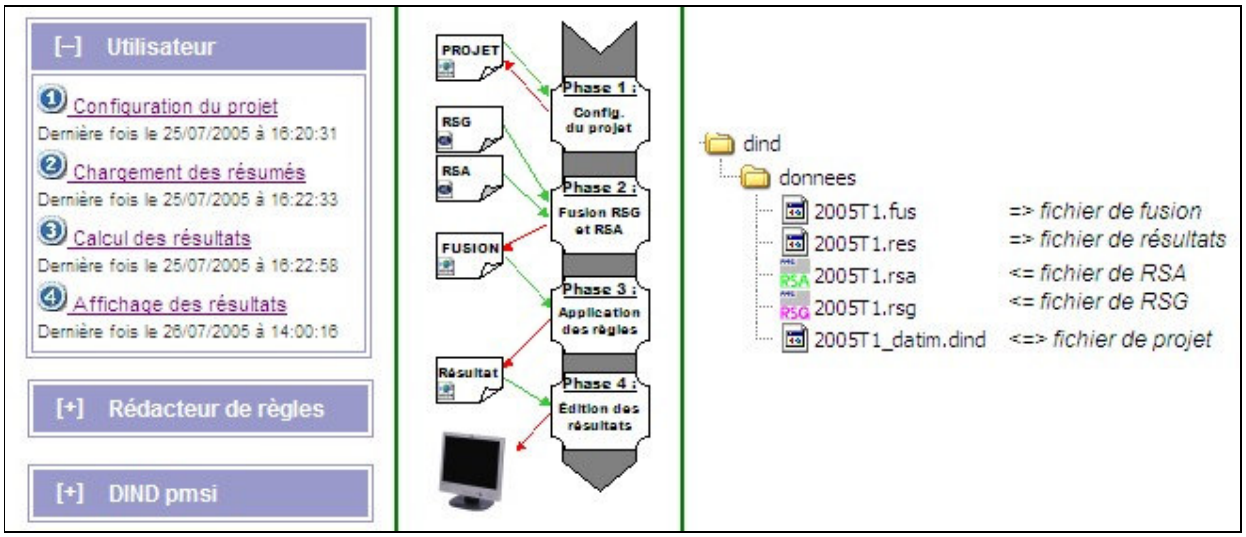
1. Généralités

DIND propose une interface agréable. Afin de rendre l'affichage accessible, DIND utilise des panneaux rétractables. Cliquez sur le bandeau supérieur pour les étendre et les réduire.



De plus, chaque fois que vous verrez l'icône dans un lien hypertexte, cela vous permettra d'ouvrir la même cible mais dans une fenêtre différente.
L'utilisation standard de DIND a été simplifiée au maximum. Pour qui souhaite exécuter un jeu de règles prédéfinies, par exemple appliquer les algorithmes DATIM, il suffit de suivre les quatre étapes présentées dans le sommaire.
L'enregistrement dans un fichier de projet permet ensuite de reprendre ou modifier l'analyse très simplement, à n'importe quelle étape. La visualisation graphique des résultats peut être différée, ou reprise sans relancer les calculs, à partir du même fichier de projet. Un traçage des opérations permet de voir l'état d'avancement du projet.
Observons le schéma ci-dessous :

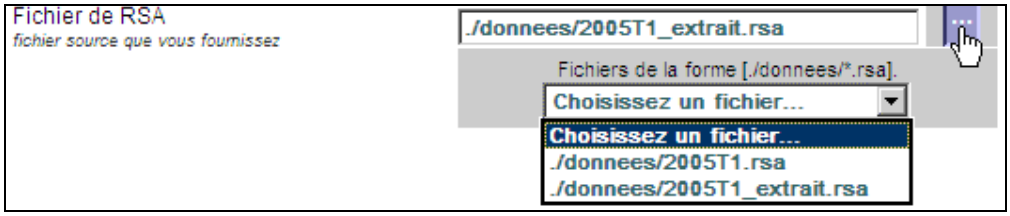
- à gauche, le sommaire donne le ton : c'est simple
- au milieu, les 4 étapes du traitement de DIND et les 5 fichiers utilisés
- à droite, les 5 fichiers utilisés



2. Détail des étapes

1- Configuration du projet

Cette étape vous permet de définir les adresses des fichiers requis. Vous n'avez a priori pas besoin de modifier les adresses de la partie "configuration du programme", mais seulement la partie "configuration du projet".
Pour une saisie automatique, pressez le bouton [...], la liste des fichiers existants compatibles vous sera automatiquement proposée. Pour créer un nouveau fichier, vous devrez utiliser la saisie manuelle.



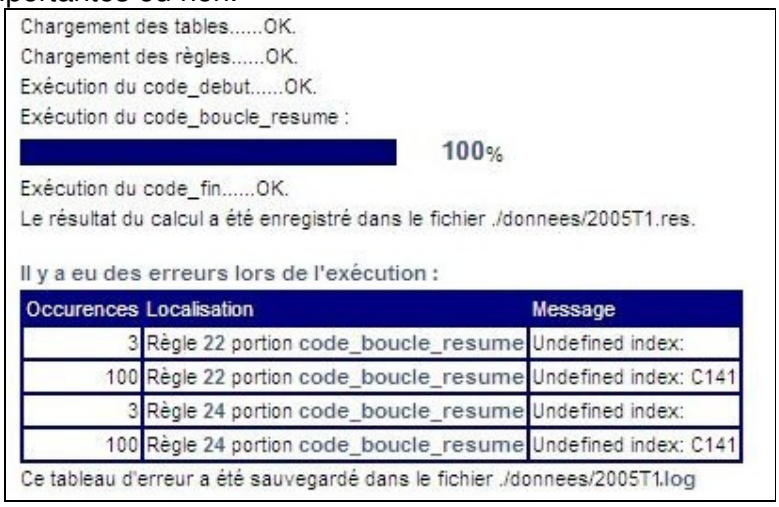
Pour une saisie manuelle, vous devez utiliser les conventions sur l'adressage relatif des fichiers : [./] désigne le répertoire courant, [../] désigne le répertoire parent. De la sorte, [./donnees/mon_rsa.rsa] représente le fichier [mon_rsa.rsa] situé dans le répertoire [donnees], à partir du répertoire du programme.

2- Chargement et fusion des résumés

Un simple clic suffit.
Le fichier de fusion est généré.

3- Calcul de la sortie DATIM

Un simple clic suffit.
Le processus de calcul est lancé et le résultat est enregistré.
Un rapport d'erreur peut apparaître comme ci-dessous. Ces erreurs peuvent provenir de la rédaction des règles autant que des données utilisées. Seul un rédacteur de règles pourra vous dire si elles sont importantes ou non.



4- Affichage de la sortie DATIM

Un simple clic suffit.
Le programme affiche le résultat enregistré à l'étape précédente. Le calcul n'est pas répété, ce qui vous permet de reprendre rapidement vos résultats.

3. Détail de la quatrième étape

La fenêtre de visualisation des résultats vous permet les actions suivantes pour chaque règle, en cliquant sur leurs intitulés :

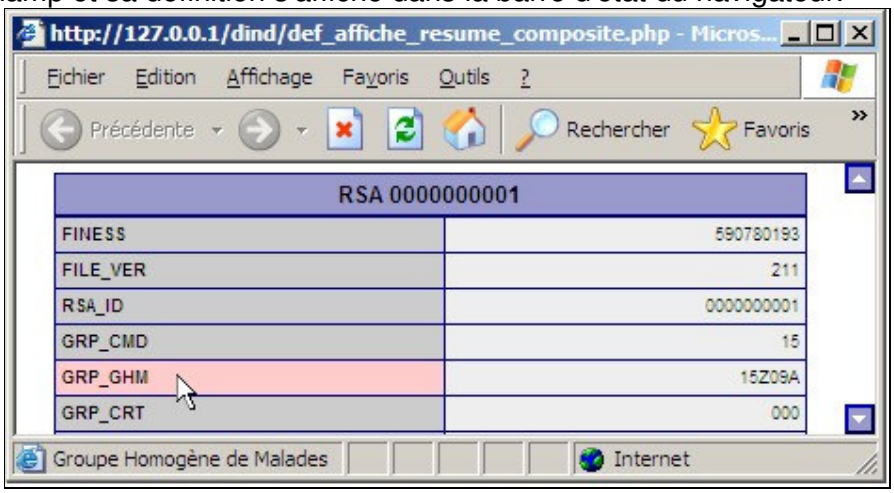
N°	Nom	Résumés référencés	Infos sup	Résultat	Score	Alerte
10	GHM atypiques : durée moyenne de séjour basse Définition, Règle complète		Voir Résultats	3	5	
11	séjours courts issus de transferts en court séjour Définition, Règle complète			0.01506024	0	
12	séjours courts issus de transferts en court séjour Définition, Règle complète	Voir 5 résumés		5		
13	séjours courts avec sortie transfert en court séjour Définition, Règle complète	Voir 21 résumés		0.04085603	1.1038	
14	séjours courts avec entrée domicile et sortie domicile Définition, Règle complète			0.07671781	3	
15	séjours courts avec entrée domicile et sortie domicile Définition, Règle complète	Voir 345 résumés		345		

- définition affiche une définition sommaire de la règle en popup
- règle complète affiche la règle complète, ainsi que son code source, avec coloration syntaxique et signalement des erreurs de syntaxe
- résumés référencés permet le cas échéant d'afficher les résumés suspects enregistrés par le rédacteur de la règle (*)
- infos sup permet le cas échéant d'afficher les résultats supplémentaires enregistrés par le rédacteur de la règle (*)

(*) toutes les règles ne permettent pas nécessairement d'enregistrer de tels élements. Consultez le détail de la règle.
Dans le cas des règles DATIM, la règle 99 calcule les scores totaux de l'établissement en additionnant certains scores de règles.

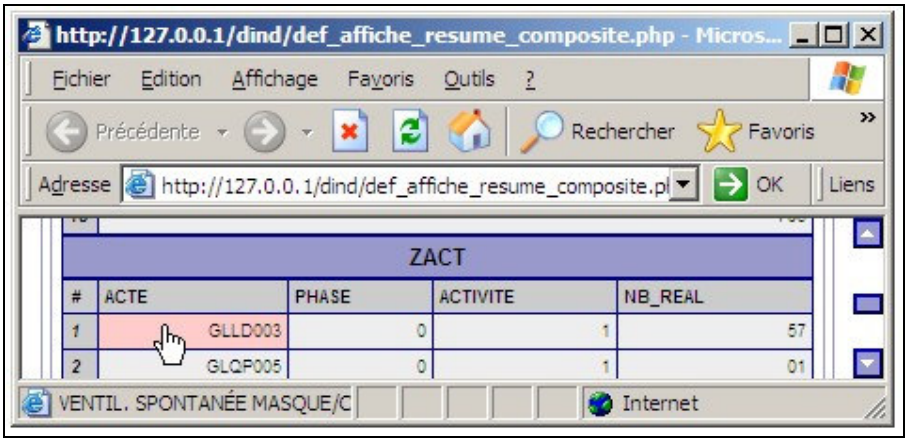
4. Détail de l'affichage des résumés

DIND permet un affichage détaillé des résumés, quelle que soit leur nature. Le terme de **résumé composite** désigne un RSA et le ou les RSG qui le constituent. En haut de page, il est possible de faire défiler les résumés un par un.
Les champs qui s'affichent se limitent à ceux définis par l'administrateur de votre projet. Survolez un champ et sa définition s'affiche dans la barre d'état du navigateur.

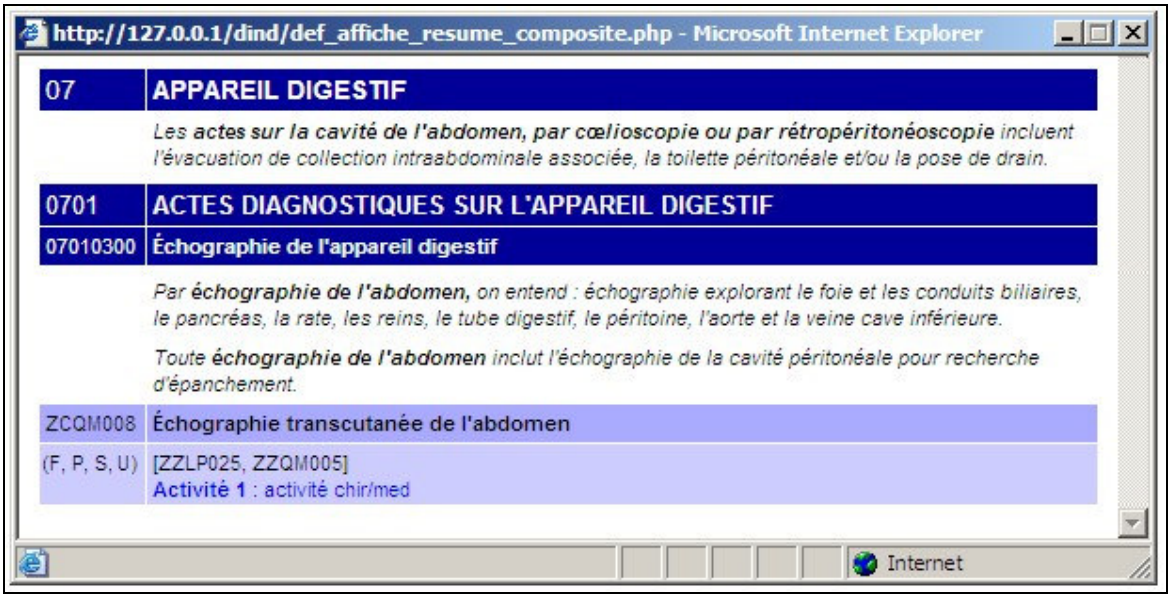


Certaines valeurs de champs sont elles aussi légendées. Elles sont faciles à reconnaître car elles se colorent et le curseur change lorsqu'on les survole. Une légende succincte s'affiche alors en bas. Lorsque vous cliquez dessus, une légende détaillée s'affiche.

Le même fonctionnement s'applique aux zones répétables, excepté que l'enchaînement est horizontal :



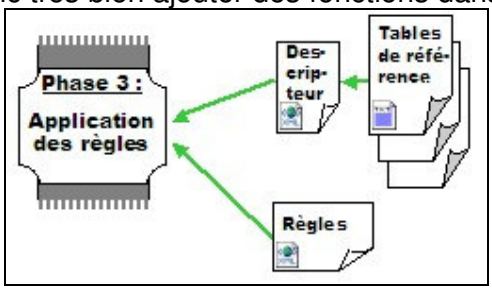
Dans le cas des actes CCAM, la légende détaillée diffère singulièrement et reprend la présentation des catalogues papier, avec la hiérarchie, les notes d'utilisation, les actes associables et les modificateurs :



B. ... en tant que rédacteur de règles

1. Généralités

L'intérêt de DIND est qu'il permet d'ajouter des règles utilisateur, à des fins de contrôle, de recodage, ou simplement pour délocaliser des requêtes sur les RSA/RSG.
Pour ajouter de nouvelles règles, il suffit soit d'augmenter le fichier de règles, soit de créer un nouveau fichier de règles suivant le même schéma, dans ce cas uniquement, il faut alors modifier le fichier de projet en conséquence, voire le fichier de projet par défaut.
DIND ne permet pas la saisie des règles. Vous devez saisir vos règles directement dans un fichier XML. En revanche, DIND vous propose un ensemble de pages dynamiques qui vous aideront à la rédaction des règles. Voir les annexes dynamiques de l'aide correspondantes.
Les règles sont écrites en script PHP. Toutes les fonctions internes au PHP peuvent donc être utilisées. L'utilisateur peut donc très bien ajouter des fonctions dans son code.



2. Déroulement de l'exécution

Chaque règle est constituée de 3 fragments de code. Les fragments de code sont exécutés dans l'ordre par une commande EVAL(). Sachez donc que vous manipulez des variables globales. Cela vous permet :

- d'utiliser des dépendances entre tests : un test Y utilise les résultats du test X
- de définir des fonctions utilisateur directement dans votre code
- d'utiliser toutes les fonctions PHP

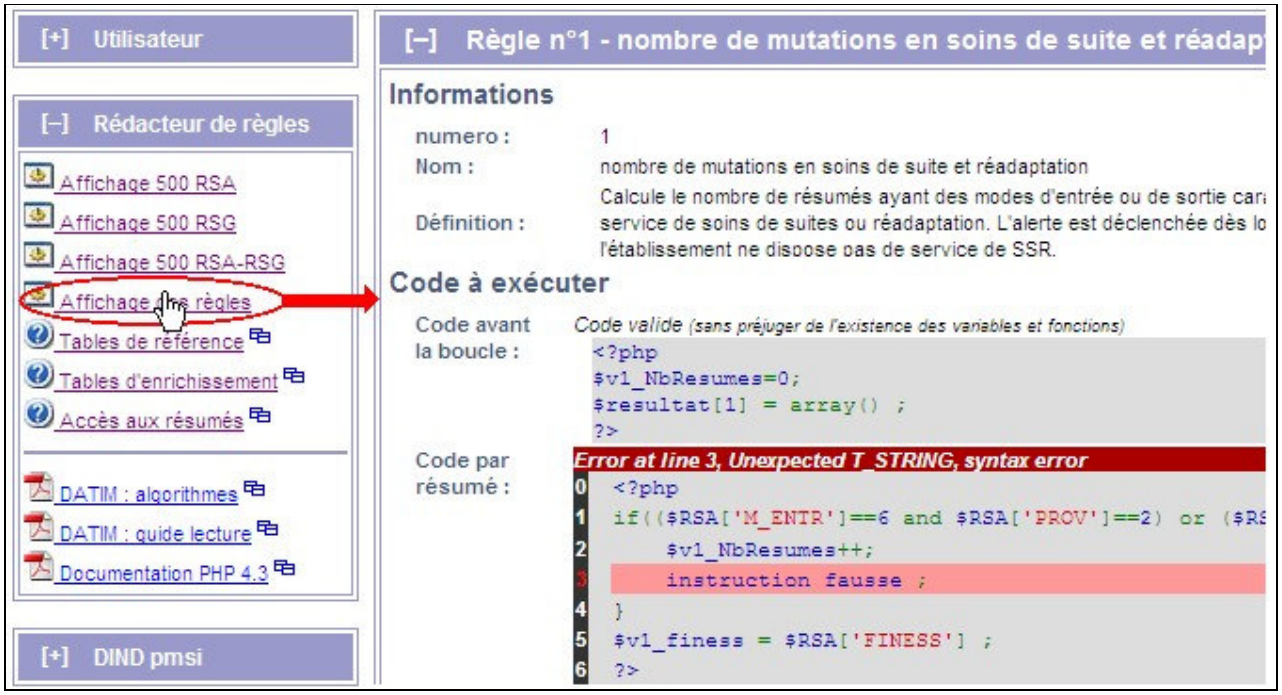
L'ordre d'exécution est le suivant :

1. tous les fragments *"code_debut"* sont exécutés en premier, dans l'ordre du fichier XML (et non dans l'ordre des numéros)
2. puis le programme réalise une itération sur chaque résumé, selon le niveau choisi. **Pour chaque résumé**, les fragments *"code_boucle_resume"* sont alors exécutés dans le même ordre. Logiquement, seuls ces fragments de code accèdent aux variables des résumés.
3. enfin tous les fragments *"code_fin"* sont exécutés, dans le même ordre

Le code des règles accède à certaines variables présentées plus bas.

3. Fichiers de règles

Un fichier XML contient un jeu de plusieurs règles. Ces fichiers sont stockées dans le dossier [./regles].
Pour mettre à jour les règles, il suffit de modifier un de ces fichiers. Si l'adresse change, il faut modifier en conséquence le fichier de gestion du projet, voire le fichier de projet par défaut.
Un module permet l'affichage des règles avec une coloration syntaxique. Les erreurs de code sont signalées.



Important : Le code PHP des règles est validé lors de l'affichage et avant exécution. La validation signifie que les erreurs de syntaxe du code sont signalées. En revanche, les erreurs liées à l'appel de fonctions non déclarées ou à l'invocation de variables non initialisées ne seront visibles que lors de l'exécution. Ces erreurs sont interceptées, agrégées et enregistrées dans un fichier de log, après affichage à l'écran. Voici quelques cas divers et fréquents où ces erreurs peuvent se manifester :

- erreur de règle : erreur de dénomination des variables ou fonctions
- erreur de règle : non initialisation d'un compteur
- code diagnostique incorrect absent des tables de référence, sans test if(isset($Table[$index])) {}
- utilisation d'un champ parfois non renseigné (diagnostic relié) du résumé dans une des tables de référence (l'index '' n'existe pas), sans test if($var !=''){}
- fichier de résumé incohérent : zone de diagnostic secondaire déclarée mais vide

Enfin, les erreurs sémantiques (calcul faux) ne peuvent évidemment pas être dépistées.
A savoir : les algorithmes diffusés sur papier par le ministère peuvent contenir des erreurs. Les erreurs suivantes sont particulièrement fréquentes :

- fermeture prématurée de boucle IF, erreurs d'imbrication des structures de contrôle
- un test annonce compter les résumés en erreur mais l'algorithme peut compter plusieurs fois un résumé contenant plusieurs erreurs
- pas toujours d'initialisation des variables
- erreurs de copier-coller dans le calcul des résultat/alerte/score : erreur sur le numéro du test, confusion entre le 5° et le 95° percentile...
- noms de champs erronés, casse variable
- pas de distinction, voire inversion, entre les valeurs de référence *de la catégorie d'établissement* et *toutes catégories confondues*

Tout cela pour dire que l'algorithme devra toujours être confronté à sa définition textuelle. Nous recommandons également au lecteur de se méfier des symboles | (pipe) qui désignent la fonction "valeur absolue" mais sont parfois peu visibles sur le document PDF.

4. Précautions d'écritures

Bien que DIND soit intégralement écrit en Orienté Objet, les règles ont été pensées en programmation procédurale afin de simplifier leur rédaction. Puisque les règles accèdent aux variables globales, nous **recommandons vivement** d'utiliser les conventions suivantes :

- les noms des variables scalaires commencent par $v
- les noms des variables tableaux commencent par $t
- les noms des variables objets (pointeurs) commencent par $o (lettre o)
- les noms des variables incluent le numéro du test qui les génère

Exemples fictifs :

- **$v12_nombre_resumes** désigne le nombre de résumés dans le test 12
- **$t12_effectifs_ghm** désigne un tableau d'effectifs dans le test 12

Pour savoir quels noms de variables vous ne devez absolument pas utiliser, consultez le code source du fichier [def_calcul.php].

5. Enregistrement des résultats

Les variables suivantes doivent contenir les variables de résultat, où **#** doit être remplacé par le numéro de la règle. Ceci n'est pas une recommandation mais une **obligation** :

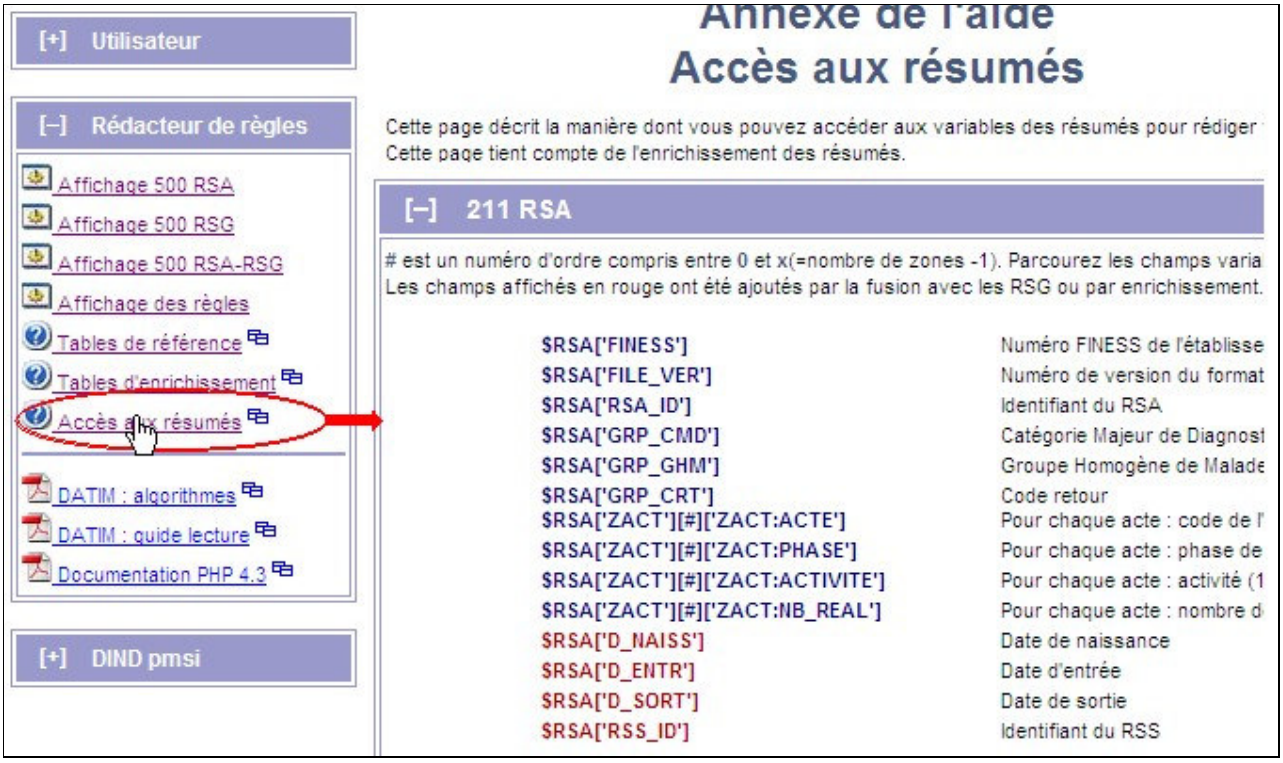
- **$resultat[#]['resultat']** contient le résultat de la règle (nombre)
- **$resultat[#]['alerte']** contient les valeurs 0 ou 1
- **$resultat[#]['score']** contient le score de la règle (nombre), qui incrémentera le score total
- **$resultat[#]['sup']** peut éventuellement contenir du texte libre qui sera porté à la connaissance de l'utilisateur. Vous pouvez y enregistrer des variables ou des tableaux en texte brut. DIND tentera alors d'afficher ce texte dans un tableau HTML. à cet effet, utilisez "\n" pour les sauts de ligne, et "\t" pour les tabulations
- **$resultat[#]['resume']** peut éventuellement contenir les résumés que vous souhaitez que l'utilisateur puisse visualiser dans la feuille de résultats. Pour ajouter facilement un résumé, ne modifiez pas directement $resultat[#]['resume'], mais invoquez simplement la procédure prédéfinie **edition_standard()** explicitée plus bas.

6. Variables accessibles

Vos règles peuvent accéder aux variables des résumés parcourrus, mais aussi à des tables de valeurs de référence.

a) Variables des résumés

Ces variables sont accessibles UNIQUEMENT dans le fragment de code "code_boucle_resume". Leur liste pourra être restreinte par l'administrateur, à des fins de simplification. La liste des champs disponibles est présentée dans l'annexe dynamique de l'aide correspondante, présentée ci-dessous. Des champs supplémentaires apparaissent en rouge, nous verrons plus tard qu'ils résultent de l'enrichissement de vos résumés. Leur accès est strictement identique aux champs natifs.



Par exemple, vous verrez que **$RSA['DUREE']** vous permet tout naturellement d'accéder à la durée du RSA.

```
<?php
// charge la durée du RSA
$duree = $RSA['DUREE'] ;
...
?>
```

En revanche, vous accèderez aux codes des actes réalisés sur le RSA en bouclant sur **$RSA['ZACT'][#]['ZACT:ACTE']** en remplaçant **#** par le numéro d'ordre de l'acte. Exemple :

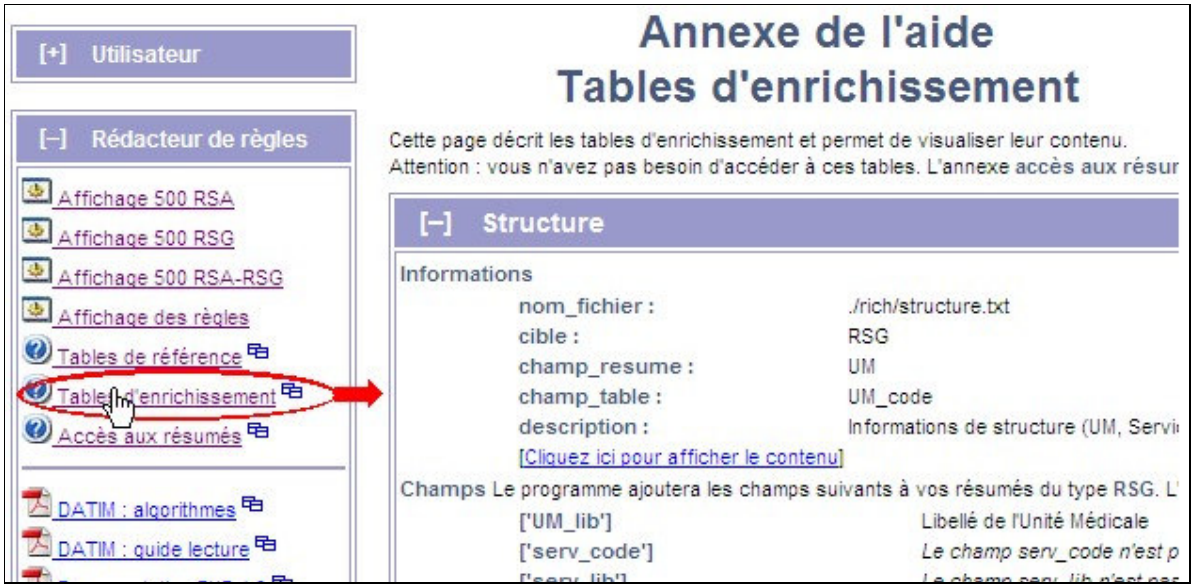
```
<?php
// examine tour à tour le code de chaque acte du RSA
for( $i=0;$i++;$i<count($RSA['ZACT']) {
   $code_acte = $RSA['ZACT'][$i]['ZACT:ACTE']
   ...
}
?>
```

b) Variables supplémentaires des résumés

Les résumés peuvent être enrichis par de nouvelles variables. Cet enrichissement peut provenir :

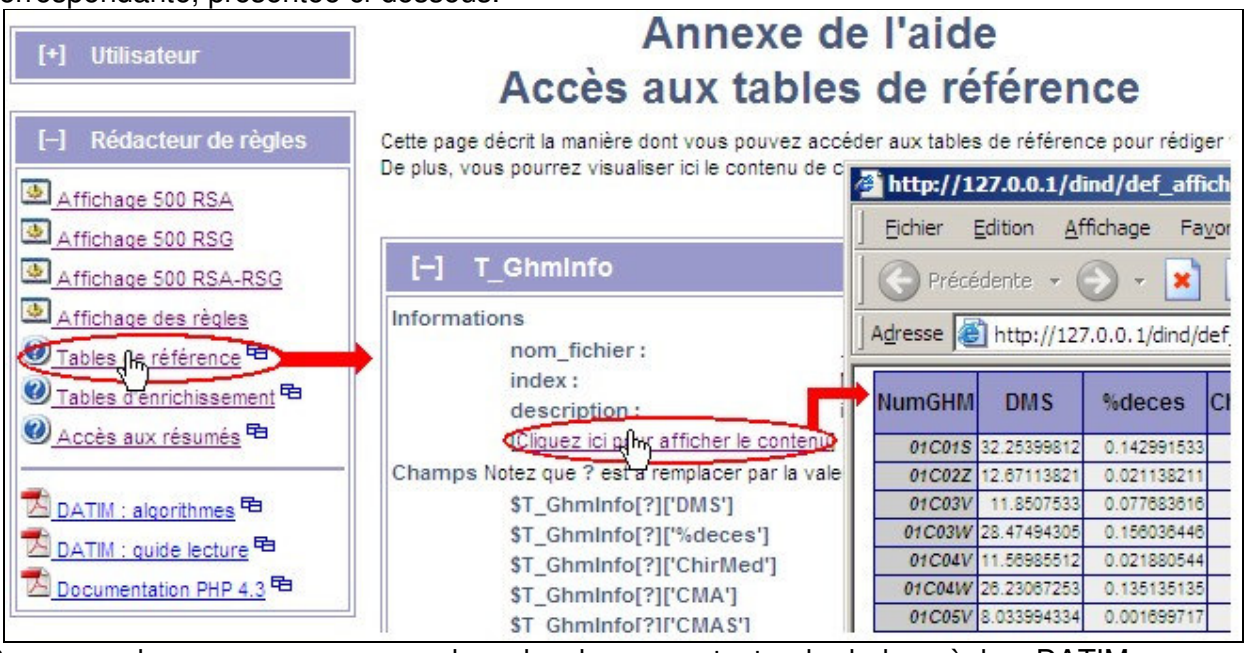
- de la fusion des RSA et des RSG (perte de l'anonymisation)
- de l'enrichissement actif des résumés, consulter pour ce faire un administrateur de DIND (*Tables d'enrichissement des résumés*)

Dans tous les cas, les tableaux $RSA et $RSG d'en trouvent simplement augmentés. Cette augmentation est documentée. Les tables d'enrichissement (structure et contenu) sont visibles à titre documentaire.



c) *Tables de valeurs de référence*

Ces variables sont accessibles à tout endroit.
Pour charger des tables, consulter un administrateur de DIND (*Tables de valeurs de référence*).
L'accès aux tables et leur contenu sont documentés dans l'annexe dynamique de l'aide correspondante, présentée ci-dessous.

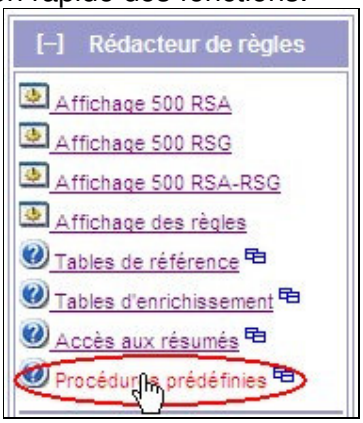


Par exemple, vous verrez que, dans le chargement standard des règles DATIM, on peut accéder au pourcentage de décès pour un GHM donné par le scalaire **$T_GhmInfo[?]['%deces']** où **?** doit être remplacé par le numéro du GHM. Exemple :

```
<?php
// récupère le GHM de notre RSA
$v99_mon_GHM = $RSA['GRPRSA_GHM'] ;
// retrouve le pourcentage de décès normal pour le GHM de notre RSA
$v99_pourcent_deces = $T_GhmInfo[$v99_mon_GHM]['%deces'] ;
...
?>
```

7. Fonctions et procédures prédéfinies

Certaines fonctions et procédures prédéfinies sont utilisables au moment de la rédaction des règles. **Ces fonctions ou procédures sont rigoureusement documentées dans une annexe de l'aide, selon les habitudes PHPDoc. Vous devriez surtout utiliser cette documentation.** Ce qui suit n'est qu'une présentation rapide des fonctions.



a) edition_standard()

L'invocation de cette procédure permet d'enregistrer dans le fichier de résultats une liste de résumés pathologiques.
Cette procédure ne peut être invoquée que dans **code_boucle_resume**.
Exemple :

```
<?php
if(  ... /* condition pathologique */ ) {
   $resultat[99]['resultat'] ++ ;  // incrémente le résultat du test 99
   edition_standard(99) ;          // enregistre le résumé courant pour le test 99
}
...
?>
```

b) edition_fichier()

Cette procédure permet d'enregistrer le résumé composite actuel dans un fichier externe. Elle permet également d'enregistrer des fichiers de RSA ou de RSG.
Cette procédure ne peut être invoquée que dans **code_boucle_resume**.
Exemple :

```
<?php
if(  ... /* condition de sélection */ ) {
   edition_fichier('./donnees/extrait1.fus') ;        // enregistre le résumé courant
   edition_fichier('./donnees/extrait1.rsa', 'RSA') ;  // enregistre le RSA du résumé courant
   edition_fichier('./donnees/extrait1.rsg', 'RSG') ;  // enregistre les RSG du résumé courant
}
...
?>
```

c) libelle()

Cette fonction retourne le libellé correspondant au code examiné.
Cette fonction peut être utilisée à tout moment.
Exemple :

```php
<?php
$v4_dp_code = $RSA['DP'] ;                    // voici le code du DP
$v4_dp_lib = libelle( $v4_dp_code , 'diag' ) ;    // voici le libellé du DP
...
foreach( $RSA['ZACT'] as $un_acte ) {
    $v4_acte_code = $un_acte['ZACT:ACTE'] ;          // voici le code de chaque acte
    $v4_acte_lib  = libelle( $v4_acte_code , 'acte' ) ; // voici le libellé de chaque acte
}
?>
```

d) passe_par()

Cette fonction permet de savoir rapidement si un résumé composite passe à un moment ou à un autre par une strucutre donnée (UM, service, Clinique). Elle peut s'avérer assez puissante grâce aux paramètres optionnels.
Cette fonction ne peut être invoquée que dans **code_boucle_resume**.
Exemple 1 :

```php
<?php
if(  passe_par(3870) ) {
   /* ici ce RSA est passé l'unité médicale 3870 */
}
...
?>
```

Exemple 2 :

```php
<?php
if(  passe_par('neuro', 'LIB_SERVICE', 1) ) {
   /* ici ce RSA est passé par un service (regroupement d'unités
      médicales) dont le nom contient 'neuro' */
}
...
?>
```

e) numero_rsg_principal()

Cette fonction renvoit le numéro d'ordre du RUM/RSG principal du résumé composite. Ce numéro est calculé selon les règles PMSI.
La désignation du RUM principal sert uniquement à affecter un Diagnostic Principal à l'ensemble du RSA. Il peut être utile de récupérer cette information afin de savoir quelle unité médicale ou quel service/clinique a codé le DP finalement retenu pour l'ensemble du RSA, surtout si ledit DP pose problème.

```php
<?php
$v0_numero = numero_rsg_principal() ;
$v0_um = $RSG[$v0_numero]['UM'] ;
// $v0_um est donc le numéro d'unité médicale du RSG principal.
$RSA['UM'] = $v0_um ;
// notez comme cette fourberie nous permet d'ajouter cette variable comme un véritable champ !
?>
```

8. Exemples de règles complètes

a) Exemple 1

Cette règle tout simple permet de sauvegarder dans un autre fichier tous les RSA dont au moins un RSG passe dans le service de Neurologie A. Cela paraît tout simple comme cela, mais ça ne l'était pas du tout car :

- les RSA ne contiennent aucune information de passage dans les unités médicales, seule la réaffectation des RSG a permis cela
- les RSG contiennent un numéro d'UM mais pas de notion de service, c'est une des tables d'enrichissement définies par l'administrateur qui a permis l'ajout de ce champ
- sans le savoir, le rédacteur de règles fait ici une boucle et teste un à un les RSG constifutifs du RSA.

Ce type de règle peut permettre par exemple de réaliser un extrait par clinique de l'activité de l'établissement, afin de calculer par les règles DATIM un score d'atypie par clinique.
Code avant la boucle :

```
<?php
// rien ici
?>
```

Code par résumé :

```
<?php
if(  passe_par('Neurologie A', 'lib_serv'  ) ) {
   edition_fichier('./donnees/neurologie_a_2005T1.fus' ) ;
}
?>
```

Code après la boucle :

```
<?php
// rien ici
?>
```

b) Exemple 2

Cette authentique règle DATIM examine tous les RSA présentant une mutation en SSR. La règle vérifie si l'établissement fait du SSR. Si tel n'est pas le cas, une alerte est déclenchée.
Code avant la boucle :

```
<?php
// simples initialisations
$v1_NbResumes=0;
$resultat[1] = array() ;
?>
```

Code par résumé :

```
<?php
// si le RSA entre par mutation depuis SSR
// ou si le RSA sort par mutation vers SSR
if(($RSA['M_ENTR']==6 and $RSA['PROV']==2) or ($RSA['M_SORT']==6 and $RSA['DEST']==2)) {
   // on incrémente alors un compteur
   $v1_NbResumes++;
```

```
    // et on mémorise ce RSA pour contrôle de visu (règle n°1)
    edition_standard(1) ;
}
// on récupère le numéro FINESS
$v1_finess = $RSA['FINESS'] ;
?>
```

Code après la boucle :

```
<?php
// pour la règle n°1, on décide que résultat = compteur
$resultat[1]['resultat']=$v1_NbResumes;
// si résultat>0 et que l'établissement ne fait pas de SSR (table T_Finess)
if( $T_Finess[$v1_finess]['SSR']==0 and $v1_NbResumes>0) {
    // on met une alerte
    $resultat[1]['alerte']=1 ;
} else {
    $resultat[1]['alerte']=0 ;
}
// le score est le produit de l'alerte par la pondération du test n°1 (table T_LibTest)
$resultat[1]['score']=$resultat[1]['alerte'] * $T_LibTest[1]['Ponderation'] ;
?>
```

c) *Exemple 3*

Cette règle s'apparente plus à une requête. Nous avons souhaité ici afficher dans un tableau tous les couples GHM-DP des RSA, leurs libellés, et leur nombre. Le tableau sera enregistré dans le champ SUP du tableau de résultats. Cela s'apparente à un simple *SELECT ghm, dp, count(*) FROM table_rsa GROUP BY ghm, dp* avec les libellés en plus.

Code avant la boucle :

```
<?php

$t1_couples = array() ;
// Ecrivons la première ligne du tableau dans le champ SUP
$resultat[1]['sup'] = "ghm\tghm\tdp\tdp\tnombre\n" ;
?>
```

Code par résumé :

```
<?php
// nous extrayons les codes GHM et DP
$v1_ghm_code = $RSA['GRPRSA_GHM'] ;
$v1_dp_code  = $RSA['DP'] ;
// ce qui suit prend 8 lignes car nous programmons proprement, mais seule la septième est obligatoire.
// Nous incrémentons un compteur.
if( !isset($t1_couples[$v1_ghm_code]) ) {
    $t1_couples[$v1_ghm_code] = array() ;
}
if( !isset($t1_couples[$v1_ghm_code][$v1_dp_code] ) ) {
    $t1_couples[$v1_ghm_code][$v1_dp_code] = 1 ;
```

```
} else  {
   $t1_couples[$v1_ghm_code][$v1_dp_code] ++ ;
}
?>
```

Code après la boucle :

```
<?php
// Nous parcourrons maintenant le tableau pour le mettre en forme.
foreach($t1_couples as $un_ghm => $un_tableau) {
   foreach($un_tableau as $un_dp => $nombre) {
      // écrivons chaque ligne du tableau
      // on en profite pour ajouter des libellés aux GHM et DP
      $resultat[1]['sup'] .= $un_ghm ."\t". libelle( $un_ghm, 'ghm') ."\t"
                  . $un_dp ."\t".  libelle( $un_dp, 'diag')  ."\t"
                  . $nombre ."\n" ;
   }
}
?>
```

d) Exemple 4

Cette règle totalement factice montre l'intérêt de requérir en même temps sur les RSA et sur les RSG. Dans cet exemple, nous examinons tous les RSA avec CMA. Nous parcourrons alors les RSG constitutifs, pour voir si une des unités médicales avait oublié de coder la CMA. La règle affiche alors un tableau avec un score par unité médicale. On saura alors désormais que les unités médicales arrivant en tête sont des cibles privilégiées de recodage des CMA ou bien d'efforts pédagogiques.

Code avant la boucle :

```
<?php
$t1_unites = array() ;
// Ecrivons la première ligne du tableau dans le champ SUP
$resultat[1]['sup'] = "UM_lib\tscore\n" ;
?>
```

Code par résumé :

```
<?php
$mon_ghm = $RSA['GRPRSA_GHM'] ;
// si ce GHM conprend une CMA
if (  $T_GhmInfo[$mon_ghm]['CMA'] == 1 ) {
   // alors regardons chaque RSG
   for( $i=0;$i<count($RSG);$i++) {
      $ce_RSG_contient_une_CMA = false ;
      // regardons chaque DAS de ce RSG
      for( $j=0;$j<count($RSG[$i]['ZDAS']);$j++ ) {
         $mon_das = $RSG[$i]['ZDAS'][$j]['ZDAS:DAS'] ;
         $mon_libelle_um = $RSG['LIB_UM'] ;
         // supposons que vous disposiez d'une table indexée sur la CIM10
         if ( $T_CIM10[$mon_das]['CMA']==1 ) {
```

```
            $ce_RSG_contient_une_CMA = true ;
        }
      // si ce RSG ne contient pas de CMA
      if ( $ce_RSG_contient_une_CMA == false ) {
        // alors on augmente le "tableau d'honneur"
        $t1_unites[$mon_libelle_um] ++ ;
      }
   }
}
?>
```

Code après la boucle :

```
<?php
// Nous parcourons maintenant le "tableau d'honneur"
foreach($t1_unites as $une_UM => $un_score ) {
   $resultat[1]['sup'] .= $une_UM ."\t". $un_score ."\n" ;
}
?>
```

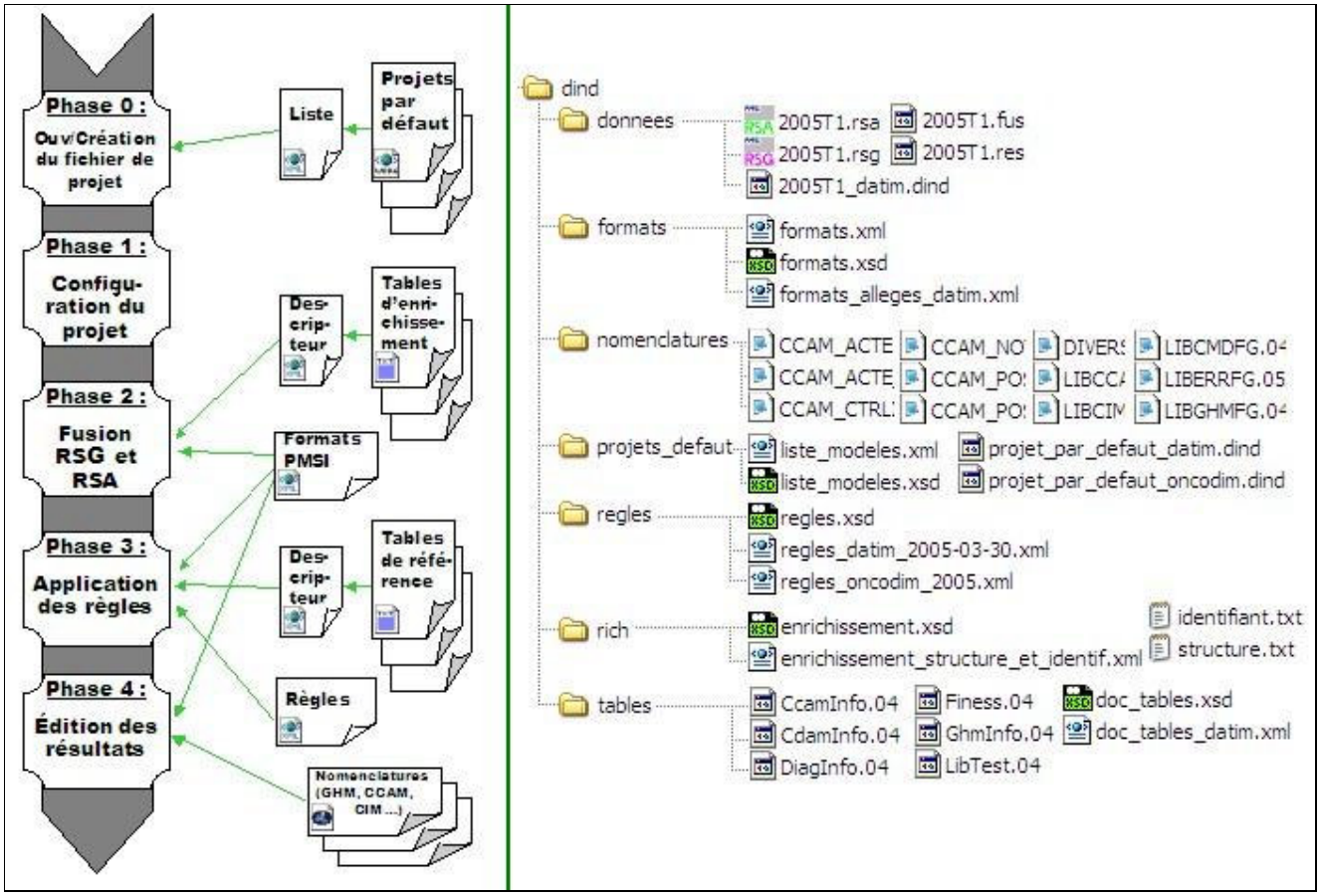
C.  ... en tant qu'administrateur

1. Généralités

L'administrateur met à jour DIND avec les nomenclatures. Mais surtout, il fournit un cadre à chaque type de traitement en préparant les fichiers de projet par défaut. Cela exige une connaissance détaillée du programme. L'administrateur appréciera toutefois le mot clef de l'implémentation de DIND : **abstraction**.

DIND ne fournit pas d'interface à l'administrateur. Son rôle s'exerce directement dans la gestion des fichiers.

Le schéma ci-dessous vous présente à gauche l'utilisation des ressources de DIND, à droite un exemple simplifié d'arborescence de fichiers de ces ressources. Chaque type de ressource est détaillé dans les chapitres suivants.



2. Les fichiers de projets par défaut

Les fichiers de projet contiennent toutes les références aux fichiers utilisés. Afin de faciliter la tâche de vos utilisateurs, vous aurez tout intérêt à leur fournir des fichiers de projets pré-configurés en fonction de la tâche.
Toutefois, vous constaterez que les préconfigurations peuvent varier significativement selon la tâche à accomplir (eu égard aux rubriques suivantes : tables de valeurs de référence, tables d'enrichissement, fichier de règles).
Pour ajouter un projet par défaut, ajoutez le fichier WDDX souhaité dans le dossier [./projets_defaut], puis ajoutez obligatoirement sa référence et sa description dans le fichier [./projets_defaut/liste_modeles.xml] qui suit le schéma [./projets_defaut/liste_modeles.xsd]. Très logiquement, ce fichier XML est le seul fichier dont l'adresse est inscrite en dur dans le code source de DIND. Il ne peut donc ni être déplacé, ni être renommé. Il est unique.
Important : La conception de ces projets par défaut est primordiale afin d'intégrer les fichiers de règles dans un processus de traitement. Effectivemenent, on peut supprimer des champs et des étapes dans le fichier de projet, ils ne seront alors pas proposés à l'utilisateur. Par exemple, si vous souhaitez que l'utilisateur puisse procéder à une requête simple sur les RSG, vous pourrez rendre invisibles la phase de fusion RSA-RSG ainsi que les enrichissements.
Vous noterez de plus que le rédacteur de règles ne peut travailler que sur un fichier de projet déjà configuré, ce qui signifie que le travail de l'administrateur de DIND est réalisé en amont.

3. Les tables de nomenclature

Ce sont les tables fournies par l'ATIH dans les "kits de nomenclatures". Ces tables décrivent les CMD, les GHM, les diagnostics CIM10, les actes CCAM, et d'autres informations.
Ces tables sont stockées dans le dossier [./nomenclatures].
Pour mettre à jour ces tables, il suffit d'ajouter les nouveaux fichiers dans le dossier, et de modifier en conséquence le fichier de gestion du projet, voire le fichier de projet par défaut.

En raison de leur hétérogénéité et de leur complexité (7 tables pour la CCAM), il n'existe pas de description abstraite des formats utilisés, il faut espérer qu'ils ne changent pas. Dans le cas contraire, la conception orientée objet de DIND rendra les modifications du code source aisées et très circonscrites.

4. La description des formats PMSI

Cette description est faite dans le fichier [./formats/formats.xml], qui suit le schéma [./formats/formats.xsd]. Il suffit d'augmenter ce fichier pour le tenir à jour.
Vous pouvez également créer un nouveau fichier suivant obligatoirement le même schéma. Ce fichier pourrait par exemple être une version diminuée se limitant aux versions et aux champs nécessaires, de manière à accélérer les traitements et alléger l'information délivrée au rédacteur de règles.
Dans ce cas seulement, vous devrez modifier en conséquence le fichier de gestion du projet, voire le fichier de projet par défaut.
Si vous souhaitez produire un fichier allégé, sachez que certains champs ne peuvent être supprimés :

1. **Champs indispensables à DIND :** 'RSS_ID', 'RSA_ID', 'FILE_VER', 'NB_RUM', 'NB_ZDAS', 'NB_ZDAD', 'NB_ZACT'
2. **Autres champs nécessaires à la fusion RSA-RSG :** 'GRP_GHM', 'SEXE', 'D_NAISS', 'AGE_AN', 'AGE_JOURS', 'MOIS_SORT', 'AN_SORT', 'DUREE', 'M_ENTR', 'M_SORT', 'D_ENTR', 'D_SORT' 'UM', 'SEANCES'
3. **Autres champs nécessaires aux règles DATIM (RSA) :** 'FINESS', 'GRP_GHM', 'GRP_CMD', 'GRP_CRT', 'GRPRSA_GHM', 'GRPRSA_CMD', 'GRPRSA_CRT', 'CP', 'PROV', 'DEST', 'POIDS', 'IGS2', 'NUM_GHS', 'NB_SUP_REA', 'NB_JOURS_EXH', 'EXB', 'NB_DIALYSES', 'NB_ACT_24Z05Z', 'NB_ACT_24Z06Z', 'NB_ACT_24Z07Z', 'TYPE_PO', 'DP', 'DR', 'ZDAS:DAS', 'ZACT:ACTE'

5. Les tables de valeurs de référence

Ce sont les tables utilisées dans les tests DATIM, dans leur état natif.
Ces tables sont stockées dans le dossier [./tables].
Leur liste et la documentation de leurs champs sont stockées dans le fichier [./tables/doc_tables.xml] qui suit le schéma [./tables/doc_tables.xsd].
Si vous souhaitez ajouter d'autres tables de référence, il suffit d'ajouter vos fichiers dans le format DATIM dans le répertoire [./tables]. Il faut obligatoirement augmenter d'autant le fichier [./tables/doc_tables.xml].
Si vous souhaitez écrire un nouveau fichier de listing des tables, il devra suivre le même schéma [./tables/doc_tables.xsd]. Dans ce cas uniquement, vous devrez modifier en conséquence le fichier de gestion du projet, voire le fichier de projet par défaut. L'intérêt de définir un nouveau fichier de listing des tables est de diminuer le temps de chargement : seules les tables listées sont chargées, il est donc souhaitable de ne charger que les tables nécessaires au jeu de règles utilisé.
Le format utilisé par DATIM répond aux normes suivantes :

- texte ascii
- les champs sont séparés par des tabulations (\t)
- les lignes sont séparées par des sauts de type Windows [char(13).char(10) ou bien "\r\n"]
- il n'y a pas de délimiteur de champ (du type guillemets simples ou doubles)
- la première ligne contient le nom des champs
- le premier champ est le champ index unique, chaque valeur peut donc être appelée par le scalaire *$T_nomTable[valeur_premier_champ][nom_champ]* , qui retourne la valeur du champ *nom_champ* pour la ligne dont le premier champ a pour valeur *valeur_premier_champ*.

- le séparateur décimal peut être le point comme la virgule. Toutes les virgules seront remplacées par des points.

Note impotante :
Il est possible que les tables fournies par le ministère soient entâchées d'erreur, car elles résultent probablement d'un *copier coller* depuis un tableur. Voici, à titre de mise en garde, les problèmes que nous avons rencontrés :

- le séparateur décimal est tantôt un point tantôt une virgule (géré par DIND)
- il existe de nombreuses colonnes ou lignes surnuméraires (géré par DIND)
- certains nombres sont transformés en notation scientifique : par exemple, 7,896E-5 doit donc être remplacé manuellement par 0,00007896 dans les tables.
- la table T_Finess n'existe pas, il convient donc de la créer manuellement avec pour seule entrée l'établissement étudié
- le champ libellé de la table T_GHM n'existe pas
- la table T_Resultats n'est pas publiée, il faut la demander à l'ATIH
- la table T_GHS n'existe pas, il faut la créer soi-même
- ...

6. Les tables d'enrichissement des résumés

Ces tables servent à ajouter des nouvelles variables à vos résumés.
Un exemple sera plus parlant : d'après le numéro d'Unité Médicale, vous pourrez importer automatiquement le libellé de l'UM, ainsi que les codes et libellés du service et de la clinique.
Pour ce faire, vous devrez fournir une table à plat, indexée sur le numéro d'UM.
De la même manière, vous pourrez rappatrier un numéro de dossier en fonction du numéro de RSS.
Ces tables sont stockées dans le dossier [./rich].
Par exemple, pour les règles DATIM, le fichier de liaison de ces tables aux résumés s'appelle [./rich/enrichissement_structure_et_identif.xml] et suit obligatoirement le schéma [./rich/enrichissement.xsd].
Si vous souhaitez ajouter d'autres tables d'enrichissement, il suffit d'ajouter vos tables dans le format DATIM dans le répertoire [./rich]. Il faut obligatoirement augmenter d'autant le fichier XML d'enrichissement.
Si vous souhaitez écrire un nouveau fichier d'enrichissement, il devra suivre le même schéma [./rich/enrichissement.xsd]. Dans ce cas uniquement, vous devrez modifier en conséquence le fichier de gestion du projet, voire le fichier de projet par défaut. L'intérêt de définir un nouveau fichier d'enrichissement est de diminuer le temps de chargement : seules les tables listées sont chargées, il est donc souhaitable de ne charger que les tables nécessaires au jeu de règles utilisé. De plus, le fichier d'enrichissement renseigne la "clef étrangère" d'enrichissement des résumés.
Le format utilisé pour les tables répond aux normes suivantes (identique aux tables de valeurs de référence) :

- texte ascii
- les champs sont séparés par des tabulations (\t)
- les lignes sont séparées par des sauts de type Windows [char(13).char(10) ou bien "\r\n"]
- il n'y a pas de délimiteur de champ (du type guillemets simples ou doubles)
- la première ligne contient le nom des champs
- le premier champ est le champ index unique, c'est donc lui qui détermine quelles valeurs seront utilisées pour enrichir le résumé.
- le séparateur décimal peut être le point comme la virgule. Toutes les virgules seront remplacées par des points.

Notez que les noms de champs déclarés dans les tables seront utilisés tels quels dans les résumés. Veillez à ne pas utiliser des noms de champs existant, cela aurait pour effet d'écraser les champs existants.
Notez enfin que l'enrichissement se fait dans cet ordre :

1. éventuellement enrichissement des RSG selon les tables d'enrichissement, dans l'ordre de déclaration
2. enrichissement des RSA par les RSG (dates précises, numéro de RSS...)
3. éventuellement enrichissement des résumés selon les tables d'enrichissement, dans l'ordre de déclaration

Cela signifie qu'un enrichissement peut s'appuyer sur un enrichissement précédent. A titre d'exemple, vous pourrez ajouter un numéro de dossier à un RSA sur la foi du numéro de RSS : ce numéro aura été ajouté par la fusion avec les RSG.
Pour des raisons multiples que nous ne détaillerons pas, pour l'utilisateur, nous utilisons les entrées du lexique du *fichier de formats PMSI*, et non les descrpitions renseignées plus haut. C'est la raison pour laquelle vous serez peut-être amené à ajouter vos nouveaux champs dans ce fichier, en respectant la casse.

IV. Précisions diverses

A. Nommage des résumés PMSI

La lecture des documentations ATIH (formats des résumés, notice du programme de groupage) nous a enseigné une dénomination univoque des résumés. Nous nous y sommes conformés mais jugeons utile de la préciser ici en raison des ambiguïtés :

Sigle DIND	Nom officiel	Exemple	Description	Appelation courante
RUM	résumé d'unité médicale	011	une ligne par RUM, ne contient pas les informations de groupage.	RUM
RSG	résumé de sortie standardisé groupé	111	une ligne par RUM, mais contient les informations de groupage (répétées toutes les lignes)	tantôt RUM, tantôt RSS
RDS	résumé de sortie	-	une ligne par séjour, ce fichier est fourni par le programme de groupage. Quasiment inconnu du public, c'est une version non anonyme du RSA. Son format est parfois documenté mais ne porte pas de numéro.	correspondrait au RSS mais est inconnu
RSA	résumé de sortie anonymisé	211	une ligne par séjour, résumé anonyme.	RSA

Dans le choix des sigles, nous n'avons pas utilisé le sigle RSS en raison de son ambiguïté. Nous ne prétendons pas ici imposer nos choix à autrui, mais simplement justifier la discipline que nous avons appliquée.

B. Evolutions souhaitées

Nous souhaitons à l'avenir ajouter les fonctionnalités suivantes :

- enrichir les règles DATIM de fonctionnalités supplémentaires de reporting
- étendre DIND à des types de requêtes différents (cf. paragaphe suivant)
- formulaire d'aide à la saisie des règles, à grand renfort de JavaScript
- intégration de modèles tarifaires dans les résumés
- préparation d'autres jeux de règles
- formulaires génériques de modification des fichiers XML depuis le schéma XSD

Nous souhaitons modifier les points suivants :

- lorsqu'un résumé est sélectionné par une règle, actuellement il est stocké dans le fichier de résultat, ce qui alourdit son chargement. Nous souhaitons externaliser ce stockage dans un fichier de sortie dédié.
- nous souhaitons abandonner le format WDDX en raison de sa piètre fiabilité, au profit d'un format XML traditionnel

Nous souhaitons corriger le bug suivant :

- en raison du format WDDX utilisé pour stocker les résumés et les résultats, et malgré l'utilisation de fonctions d'encodage UTF8, les tables d'enrichissement ne doivent pas contenir de caractères spéciaux.

C. La notion de traitement

Les traitements rendus possibles par DIND sont de plusieurs types. Actuellement, seul le premier type est implémenté. Toutefois, la structure du code a été pensée de manière à pouvoir implémenter les autres types, plus simples, à l'avenir. Le type de traitement devrait être fixé par l'administrateur en début de projet :

- **RSA-RSG enrichis** : un fichier de RSA et de RSG sont fusionnés puis enrichis par des tables optionnelles
- **RUM simples** : un fichier de RUM est directement traité, l'étape 2 devient inutile.
- **RUM enrichis** : un fichier de RUM est enrichi par des tables optionnelles
- **RSG simples** : un fichier de RSG est directement traité, l'étape 2 devient inutile.
- **RSG enrichis** : un fichier de RSG est enrichi par des tables optionnelles
- **RSA simples** : un fichier de RSA est directement traité, l'étape 2 devient inutile.
- **RSA enrichis** : un fichier de RSA est enrichi par des tables optionnelles. Ce cas est très théorique car, n'ayant ni numéro de RSS ni notion de structure, il semble difficile d'enrichir les RSA, mais cela reste possible (zone géographique, classification OAP...)

L'exploitation de fichiers de RDS pose problème car d'une part ils sont peu répandus, d'autre part le numéro du format n'est pas renseigné. Le numéro de version des RSG initiaux, lui, ne suit pas la règle habituelle (positions 9-10-11) mais est succeptible de changer selon les versions. Il faut donc connaître la version pour savoir où trouver le numéro de version ! C'est pour ces raisons que nous avions choisi de fusionner des RSA et des RSG au lieu d'utiliser directement les RDS.
La fusion RSA-RUM serait techniquement possible, mais qui possède le fichier de RUM possède aussi le fichier de RSG.

V. Aspects techniques

A. Langages

PHP5 : PHP est un langage de scripts habituellement interprétés sur un serveur web. PHP est un acronyme récursif signifiant *PHP: Hypertext Preprocessor*. DIND est intégralement programmé en PHP5 orienté objet. Les règles sont également programmées en PHP5. Toutefois, pour des raisons de simplicité, et en raison de la portée limitée des règles en général, leur contenu devrait être un code procédural simple compatible avec PHP4.3.0.
XML 1.0 : XML (*eXtended Marquee Language*) est le format de stockage universel "en vogue" au début du troisième millénaire. Dans le cas de DIND, tous les stockages d'informations et références sont réalisés à l'extérieur du programme. A l'exception des données préexistantes (nomenclatures) et des tables de références et d'enrichissement, tous ces stockages utilisent XML 1.0 et l'encodage ISO-8859-1.
Voir http://www.w3.org/XML.

XSD : XSD *XML Schema Description* est la nouvelle norme de description de documents XML, et la seule validée par le W3C (contrairement aux DTD). Tous les fichiers XML de DIND sont fournis avec un schéma XSD qui devrait permettre, à l'avenir, une modification par formulaire des fichiers XML.
Voir http://www.w3.org/XML/Schema.
WDDX : certains fichiers utilisent la norme WDDX *Web Distributed Data Exchange*, qui est une utilisation particulière de XML1.0, permettant les bijections immédiates entre tableaux associatifs et fichiers XML. WDDX est initialement conçu pour les services Web, permettant la persistance de structures complexes. Nous avons utilisé WDDX pour simplifier la programmation lorsqu'aucun des trois rôles de DIND n'avait à modifier ces fichiers. Nous rappelons que les chaînes stockées en WDDX doivent d'abord être encodées à l'aide de **utf8_encode()**, faute de quoi l'écriture sera possible mais la lecture ne le sera plus.
Voir http://www.openwddx.org.
Base de données : par soucis de portabilité, de simplicité d'utilisation, de mises à jour, et pour rester dans l'esprit initial du PMSI, nous n'utilisons pas de base de données. Les calculs utilisent directement les fichiers fournis. Certains traitements s'en trouvent ralentis, mais les délais restent acceptables et les traitements sont plus intelligibles pour l'utilisateur.

B. Plate-forme

La procédure qui suit est précisée à titre documentaire uniquement, car DIND est fourni avec une distribution déjà adaptée par nos soins d'EasyPHP. L'installation de DIND est donc évoquée au chapitre suivant.
Dans un premier temps nous avons simplement installé le célèbre easyphp1.8.0.1 pour Windows. Voir http://www.easyphp.org/telechargements.php3. Puis nous avons modifié easyphp pour utiliser PHP5 qui, contrairement à PHP4.3, est un véritable langage orienté objet. Pour procéder à la mise à jour, nous avons suivi la procédure suivante, que nous avons ensuite publiée sur le forum d'easyPHP car nombre de programmeurs semblaient bloqués :

1. installer easyphp1.8
2. supprimer le répertoire [easyphp1-8\php]
3. le remplacer par le contenu du ZIP [php-5.0.4-Win32.zip] téléchargé sur le site de PHP à l'adresse http://www.php.net/downloads.php#v5
4. chercher le fichier [easyphp1-8\php\php.ini-dist] et écraser avec lui le fichier [easyphp1-8\conf_files\php.ini]
5. dans le fichier [easyphp1-8\conf_files\httpd.conf] remplacer les PHP4 par des PHP5 ce qui donne en détail :
   - *LoadModule php4_module "C:/EasyPHP/php/php4apache.dll"* devient *LoadModule php5_module "C:/EasyPHP/php/php5apache.dll"*
   - *AddModule mod_php4.c* devient *AddModule mod_php5.c*
   - Ajouter index.php5 dans la ligne *DirectoryIndex index.html index.shtml index.wml index.pwml index.php index.php3 index.php4 index.php5*
   - Ajouter .php5 dans la ligne *AddType application/x-httpd-php .phtml .pwml .php3 .php4 .php5 .php .php2 .inc*
6. dans le nouveau fichier [easyphp1-8\conf_files\php.ini] remplacer *extension_dir = ".\"* par *extension_dir = "${path}\php\ext"*

Pour les besoins de DIND, nous avons modifié la configuration de base ainsi :

- Modifier la configuration de PHP [easyphp1-8\conf_files\php.ini] comme suit :
  - max_execution_time = 600 ; délai d'exécution
  - max_input_time = 60 ; délai de chargement des paramètres GET/POST
  - memory_limit = 32M ; mémoire
  - include_path = ".;${path}\php\pear" ; répertoires où chercher des fichiers à inclure

    - error_reporting = E_ALL ; nous affichons volontairement les notifictions
    - magic_quotes_gpc = On ; les entrées de formulaires sont échappées avec des slashes
    - register_globals = Off ; les variables GET et POST ne sont accessibles que par les tableaux
    - track_errors = on ; le dernier message d'erreur est enregistré dans $php_errormsg
- Modifier la configuration d'Apache [easyphp1-8\conf_files\httpd.conf] comme suit :
    - Timeout 600 ; délai d'exécution
    - DocumentRoot "${path}/www" ; répertoire de l'hôte local

Enfin nous avons étendu PHP avec certaines librairies PECL et PEAR
Voir http://pecl.php.net et http://pear.php.net.
Les packages suivants doivent être installés :

- XML Simple (par défaut)
- WDDX (par défaut)
- PHP_PARSER 1.0 de Greg Beaver et Alan Knowles, version du 12 juillet 2005 : ce module étant expérimental, le nom et le comportement de ses méthodes pourrait changer. Il est donc risqué de le mettre à jour. De plus, nous avons ajouté deux toutes petites modifications lignes 470 et 472 pour éviter l'affichage de warnings.

C. Installation de DIND, poste client

Pour installer DIND, Apache et PHP5, il suffit de copier tel quel le répertoire d'easyPHP modifié par nos soins
Pour exécuter DIND :

- exécuter easyPHP
- lancer Apache et PHP avec le bouton de gauche
- presser la touche F7 pour démarrer le navigateur sur http://localhost

Le poste doit être équipé d'Internet Explorer version 5 (ou supérieure). La compatibilité de notre code JavaScript avec d'autres navigateurs n'est pas garantie.
Précisons si besoin que **DIND n'est pas une application WEB**. DIND est destiné à être exécuté et visualisé sur un seul et unique poste, pour les raisons suivantes :

- EasyPHP n'est pas destiné à une utilisation web
    - le programme, conçu pour Windows, n'est pas jugé suffisamment stable pour un environement de production
    - la configuration par défaut n'accepte que les requêtes HTTP locales
- PHP5 n'est pas destiné à une utilisation web
    - cette version est encore expérimentale (même si par exemple l'hébergeur OVH permet déjà son utilisation en hébergement mutualisé)
- DIND n'est pas destiné à une utilisation web
    - DIND utilise des fichiers de données locaux et non distants, aucun *upload* n'est possible en l'état
    - la charge de travail est trop importante
    - les flux HTTP seraient trop nombreux (la bufferisation de sortie est désactivée pour les barres de progression)

Configuration requise :

- PC récent, processeur Pentium
- Microsoft Windows 98 ou supérieur

- Microsoft Internet Explorer 5 ou supérieur
- 180 Mo d'espace disque (incluant vos fichiers de données)
- 256 Mo de RAM

Notre environnement de test :

- PC portable, Intel Pentium Mobility 1.4 GHz, carte mère à 587 MHz
- Microsoft Windows XP édition familiale
- Microsoft Internet Explorer 6
- 240 Mo de RAM

En cas d'espace disque restreint, les dossiers suivants peuvent être supprimés : mysql, phpmyadmin, home.

D. Programmes utilisés pour le développement

- **EasyPHP 1.8 et ses composants, PHP 5 :** voir plus haut
- **PHP Designer 2005 :** éditeur de code PHP, HTML et JavaScript compatible avec PHP5, freeware pour Windows
- **XML Spy :** éditeur XML permettant entre autres le design automatique des schémas XSD, version d'évaluation pour Windows
- **phpDocumentor :** phpDocumentor est un package PEAR permettant de générer automatiquement une documentation des classes PHP, en utilisant une déclaration dans le code source comparable aux commentaires JAVADOC. Nous avons utilisé phpDocumentor en ligne de commande, mais il est habituellement recommandé d'utiliser docBuilder, une interface web conviviale. Nous avons utilisé l'élégant template HTML:frames:DOM/earthli écrit par Marco von Ballmoos.
- **Ramboost XP :** logiciel permettant une surveillance (et une vidange) du contenu de la RAM, très utile pour évaluer l'efficacité d'un forçage du ramasse-miettes
- **Divers logiciels de Windows :** bloc-notes, Paint, Internet Explorer, explorateur de fichiers

Quelques remarques médicales sur DIND

I. Deux exemples complets d'interprétation de test DATIM

A. Test 7 : pourcentage de séjours issus de transferts

1. Informations commentées

Les informations ci-dessous sont fournies avec l'algorithme du test. Nous ajoutons éventuellement des commentaires explicatifs, en italique.

Nom pourcentage de séjours issus de transferts

Définition Calcule le pourcentage de RSA dont le mode d'entrée est un transfert

On parle de transfert lorsque le patient provient d'un autre établissement. Ne pas confondre transfert (depuis l'extérieur) et mutation (entre deux unités médicales du même établissement)

Fonction repérage d'atypie

Type 1

Signifie que ce test retourne un score d'atypie qui sera ajouté à l'indice synthétique d'atypie Q1. Les tests de type 2 augmentent l'indice global de qualité Q2. Les tests de type 3 se contentent de signaler un par un des résumés suspects.

Valeur résultat pourcentage

Le résultat est un nombre compris entre 0 et 1, dont la valeur est la proportion de résumés dont le mode d'entrée est un transfert. Pour l'interpréter, il faut regarder l'item « dénominateur ».

Dénominateur total résumés hors CM24

Cet item nous enseigne que les résumés en CM24 n'ont pas été pris en compte, ni sur le numérateur, ni sur le dénominateur.

Alerte résultat >= 95 °p. de la valeur de référence par catégorie d'établissement

L'alerte vaut 1 seulement si le résultat atteint ou excède le 95°percentile du résultat obtenu par tous les établissements de la même catégorie. Sinon, elle vaut 0.

Pondération 3

Dans le calcul de l'indice global d'atypie Q1, le score obtenu par ce test sera triplé. Ce facteur est déjà inclus dans la formule ci-dessous. Ce signalement est donc purement informatif.

Score pondération * min(max(((valeur test – moyenne référence test)/écart type référence test),0),1)

Si la valeur du test dépasse la valeur moyenne des autres établissements, alors le score prend pour valeur ledit excès divisé par l'écart type, borné à 1 maximum, et enfin multiplié par la pondération. Sinon, le score vaut zéro. Plus l'atypie est forte, plus le score est élevé.

2. Résultats

Résultat = 0.06479

Notre établissement comporte donc 6.5% de séjours issus de transferts depuis d'autres établissements.

Score = 2.57125

Nous voici donc affecté d'un score (important) de 2.6, sachant que notre score total Q1 sera de 36.4 au total. Ce test constitue 7% de notre atypie.

Résumés sélectionnés : aucun

Ce test ne sélectionne aucun résumé, ce qui est visible dans le code source. Aucun résumé n'est suspect en soi, c'est l'atypie statistique du groupe qui est suspecte. Il nous serait possible d'ajouter dans notre règle une édition systématiques des résumés, une seule ligne suffit. Toutefois, en raison de leur nombre très élevé, le lecteur perdrait probablement beaucoup de temps.

3. Commentaires

Ce test cherche peut-être à identifier des établissements qui transfèreraient trop facilement des patients, afin de réduire artificiellement leur coût ou, pire, d'augmenter artificiellement leur tarification. Le test suivant identifie effectivement les établissements dont trop de séjours se

terminent par un transfert. A eux deux, ces tests visent à débusquer les flux suspects de patients.
On pourrait se dire que ce résultat est nécessairement suspect, puisque les valeurs de référence ont été recueillie sur d'autres CHU, donc a priori comparables en terme de recours ou référence. En réalité, la structure de l'offre de soins est variable selon les régions. La région Nord-Pas-de-Calais se caractérise par la très forte centralisation de l'offre de soins : le secteur privé est peu développé, et les centres hospitaliser périphériques assurent une activité « quotidienne ». Il n'est donc pas surprenant que de nombreux transferts aient lieu vers le CHU de Lille. En quelque sorte, cette atypie n'est pas due aux particularités de l'établissement, mais aux particularités des établissements voisins.
Il ne faut pas conclure pour autant que l'établissement pourrait être sanctionné pour ses particularités. Ce test décèle une atypie. Tout au plus cette atypie pourra-t-elle déclencher un contrôle. Mais si le contrôle révèle que les résumés correspondent bien aux dossiers réels et que les dossiers réels montrent une prise en charge loyale, il n'y a aucune raison de sanctionner l'établissement.

B. Tests 24-25 compatibilité âge / diagnostic

1. Informations commentées

Les informations ci-dessous sont fournies avec l'algorithme du test. Nous ajoutons éventuellement des commentaires explicatifs, en italique. Les commentaires sont ici moins détaillés, et supposent que vous avez lu l'exemple précédent.

Test 24

Nom	compatibilité âge / diagnostic
Définition	Calcule pourcentage de diagnostics présentant une incompatibilité avec l'âge. La liste des associations diagnostics / âge est donnée par tranche d'âge, dans la table DiagInfo, champs 0_27, 28_1, 1_9, 10_19, 20_65, 65etSup. *On voit ici que la table DiagInfo qui est fournie contient des informations utilisables par ailleurs*
Fonction	évaluation de la qualité du fichier
Type	2 *Ce test augmente l'indice global de qualité Q2.*
Valeur résultat	pourcentage *Fournit en retour le pourcentage (entre 0 et 1) de RSA contenant au moins 1 diagnostic incompatible avec l'âge*
Dénominateur	total résumés *Tous les résumés sont donc testés*
Alerte	résultat >= 95eme p. de la valeur de référence toutes catégories confondues *Dans ce test, l'ATIH estime que la catégorie de l'établissement ne devrait pas influencer le résultat.*
Pondération	1
Score	pondération * min(max(((valeur test – moyenne référence test)/écart type référence test),0),1)
Dépendances	25

Test 25

Nom	compatibilité âge / diagnostic
Définition	Calcule pourcentage de diagnostics présentant une incompatibilité avec l'âge. La liste des associations diagnostics / âge est donnée par tranche d'âge, dans la table DiagInfo, champs 0_27, 28_1, 1_9, 10_19, 20_65, 65etSup.
Fonction	amélioration de la qualité
Type	3
Valeur résultat	dénombrement de résumés
Dénominateur	sans objet
Alerte	sans objet
Pondération	sans objet

Score sans objet
Dépendances 24

Ici, on voit bien que le test 25 ne fait que compléter le test 24. Les résumés considérés comme pathologiques dans le test 24 sont comptabilisés et visualisables dans le test 25.

2. Résultats

Test 24 Résultat = 0.00143

0.14% des séjours de notre établissement présentent une incompatibilité âge-diagnostic.

Test 24 Score = 0

Ce résultat paraît non atypique car inférieur au 95°percentile

Test 25 Résumés : 56 résumés consultables

56 résumés sont cependant anormaux, il est possible de les visualiser

3. Commentaires

Ce test nous permet de mieux cerner le concept d'atypie :

- nous avions conclu plus haut qu'une atypie n'était pas forcément une anomalie
- nous voyons ici que, au contraire, l'absence d'atypie n'exclut pas une anomalie

Certes notre score Q2 n'est pas augmenté, néanmoins ces résumés sont ajoutés à la liste des résumés contrôlables Q3.

La lecture des résumés incriminés met en évidence deux erreurs de codage récurrentes :

1- lorsque le RSA concerne une femme qui accouche, il est fréquent qu'un diagnostic concernant le nouveau-né soit mentionné dans le RUM de la mère, alors qu'il devrait figurer uniquement dans le RUM du nouveau-né

2- des maladies anciennes (ex : anomalie de la puberté) ne devraient plus être mentionnées dans les DAS (diagnostic associé significatif) mais dans les DAD (diagnostic associé documentaire) si le patient est un adulte. Les DAD des RUM ne sont pas reportés dans le RSA et ne peuvent donc pas être testés.

Toutefois, ces anomalies de codage sont communes à tous les établissements, c'est pourquoi, bien que les résumés soient incorrects, le score reste inchangé.

II. Quelques cas d'utilisations génériques de DIND

DIND peut être utilisé par exemple dans les cas suivants :

1- DATIM (et NESTOR) : ensemble de règles de contrôle et de détection d'atypies. Exemples :
   - actes incompatibles avec un sexe (accouchement chez un homme)
   - pourcentage de décès trop élevé ou trop faible dans un GHM donné
   - diagnostics trop fréquents pour un sexe (cancer du sein chez un homme)
   - incohérences administratives (mutations en psychiatrie dans un établissement ne disposant pas d'autorisation en psychiatrie)
   - mauvais codage : codes diagnostics trop flous, codes interdits
   - erreurs de chaînage
   - …

2- ONCODIM : comptage d'après des tests complexes du nombre de séjours pour cancer, ou bien du nombre de patients cancéreux pour une année

3- Contrôle des prescriptions : rédaction de règles maison vérifiant les actes d'imagerie prescrits en fonction de la pathologie et des référentiels émis par les sociétés savantes

4- … autres requêtes complexes… l'aide de DIND contient quelques exemples de règles, code source à l'appui.

III. La demande de vues structure

Sans connaître le détail des algorithmes DATIM, nous souhaitions pouvoir produire un état pour l'établissement, mais aussi un état pour chaque clinique. La lecture approfondie des

algorithmes souleva alors un problème technique et un problème médical. Nous l'expliquerons à l'aide des deux approches possibles :

A. Première hypothèse : répartir l'erreur de chaque RSA

Il eût été logique de se dire que, si un RSA présente à lui seul une atypie, ce poids doit être réparti entre les différentes cliniques qu'il a traversées (le plus souvent une seule). Cette approche est tout à fait valide pour certains tests, dans lesquels chaque RSA est soit mauvais, soit bon. Mais, dans la plupart des cas, une règle DATIM ne détecte pas les bon ou les mauvais résumés, pris isolément, mais l'atypie statistique de l'ensemble des résumés. Si le taux de décès des RSA de tout un sous-groupe est trop élevé, comment en attribuer la responsabilité à telle ou telle clinique ?

B. Deuxième hypothèse : constituer des échantillons

L'autre approche consiste à créer des échantillons de RSA. Par exemple, travailler sur tous les RSA passant par la clinique d'obstétrique. Techniquement, cela signifie qu'un premier fichier de règles permet de constituer un fichier de fusion par clinique et qu'ensuite il faut appliquer les règles DATIM sur chacun de ces fichiers séparément. On obtient alors un fichier de résultats par clinique.
Il faut s'attendre alors à un problème d'interprétation médicale. Les atypies sont définies d'après les mesures effectuées sur des établissements réels dans leur globalité. Au niveau de la clinique, certaines alertes risquent de se déclencher alors qu'il n'y en avait pas pour l'établissement pris dans son ensemble. Par exemples :

- le taux de décès dans une clinique de réanimation sera anormalement élevé
- le taux de diagnostics incompatibles avec l'âge sera anormalement élevé dans une clinique d'obstétrique (codage fautif mais fréquent d'un diagnostic du nouveau-né dans le RUM chez la mère)
- le taux de diagnostics principaux en Z sera trop élevé dans un clinique de néphrologie (dialyses)

Bien qu'elle ne puisse apporter entière satisfaction, nous avons choisi de suivre cette seconde voie. Il suffit donc d'utiliser au préalable une règle afin de constituer des extraits de fichiers.

Quelques remarques informatiques sur DIND

I. Nos grands principes de programmation

A. Abstraction

L'abstraction est un maître mot dès qu'on considère qu'un programme n'a pas **un** état final, mais seulement **des** états fonctionnels stables. Concrètement, la démarche abstraite prépare d'une part les mises à jour des ressources du programme, et d'autre part les futures évolutions du programme, *sans les connaître*. Elle consiste à *débarquer* le plus de logique possible du code, en la décentralisant par exemple dans des fichiers XML. Il serait aberrant de lister les moments auxquels cette démarche s'est imposée, car nous la souhaitions permanente. Nous montrons toutefois un exemple parlant.

Un exemple parmi d'autres :

Une des demandes du sujet de stage était de réaffecter, sur la foi de l'unité médicale du RUM, un RSA à une ou plusieurs cliniques. Une approche simple aurait été de dire : « l'utilisateur fournit une table des cliniques, le programme ***sait*** comment l'utiliser et réaffecter le RSS ».
Notre démarche fut plus abstraite : nous avons défini le concept plus global d'enrichissement du résumé :

- L'enrichissement se fonde sur un nombre illimité de tables à plat en texte tabulé, dont le premier champ est un index.
- Un fichier XML unique et simple liste toutes les tables d'enrichissement à charger, et « explique au programme » comment utiliser ces tables pour enrichir chaque résumé.
- Ces fichiers, autant les tables (texte tabulé) que leur descripteur unique (XML), peuvent être modifiés facilement par un utilisateur, sans toucher aucunement au code.
- Chaque résumé (RSA/RUM/RSG), en plus de ses champs « natifs », peut alors disposer de champs « enrichis », accessibles exactement de la même manière.
- Le concept général d'enrichissement a pu être utilisé d'une autre manière, en enrichissant un RSA par des informations contenues dans ses RUM (RSG) constitutifs. Dans ce cas en revanche, le calcul reste inclus dans le code source.
- Les modalités d'accès aux variables des résumés sont exposées au rédacteur de règles par une annexe dynamique de l'aide. Cette annexe, à la lecture du fichier de formats XML et du fichier XML d'enrichissement, liste tous les champs disponibles, natifs ou enrichis, leur définition et leur modalité d'accès.
- Une autre annexe dynamique de l'aide présente à titre documentaire au rédacteur de règles le processus d'enrichissement et les tables utilisées, à la lecture du fichier XML d'enrichissement
- Les enrichissements peuvent s'appuyer les uns sur les autres en cascade.
- Voici quelles peuvent être les autres applications du concept d'enrichissement, sans écrire une seule ligne de code : rapatrier un numéro de dossier depuis le numéro de RSS, déduire le canton d'origine depuis le code géographique, appliquer une classification de type OAP depuis le GHM, retrouver le signe astrologique chinois…

L'abstraction a sans doute un prix pour le programmeur : notre application dépasse les 5000 lignes de code, sans compter les 3700 lignes du fichier de règles DATIM, les nombreuses tables de référence et les 1100 lignes d'aide HTML. En revanche, c'est un investissement pour l'avenir du programme : mises à jour, transferts de tâches.

B. Encapsulation

Ce maître mot de la programmation orientée objet n'est plus à présenter. Nous nous sommes efforcés de l'appliquer au mieux, gagnant en fiabilité, en sécurité, et en évolutivité du code.

Cette philosophie se prête particulièrement aux projets menés par ajouts successifs de fonctionnalités sur un produit rapidement fonctionnel mais rudimentaire au début.
Il est raisonnable de préparer l'évolutivité d'un produit, surtout dans un domaine versatile comme le PMSI MCO, et une discipline d'exigence croissante comme l'Information Médicale.

C. Utilisation

Nous avons été privilégiés dans notre parcours par le fait d'avoir été utilisateur avant d'être programmeur. Cela signifie que nous n'avons pas implémenté une fonctionnalité parce qu'elle nous était demandée, mais parce que, en tant qu'utilisateur, nous jugions nécessaire de l'ajouter, ou en tout cas de lui laisser « une petite place » pour l'insérer dans l'avenir.
La structure de DIND permet déjà, en l'état, une utilisation tout à fait différente de ce que veut être DATIM. DIND est un système de requêtes complexes.
L'aide de DIND (deuxième partie de ce mémoire) contient un paragraphe listant des évolutions futures : ce sont des vœux qui nous sont chers depuis le début et qui ont une place déjà réservée dans la structure actuelle de DIND, il ne reste plus qu'à…

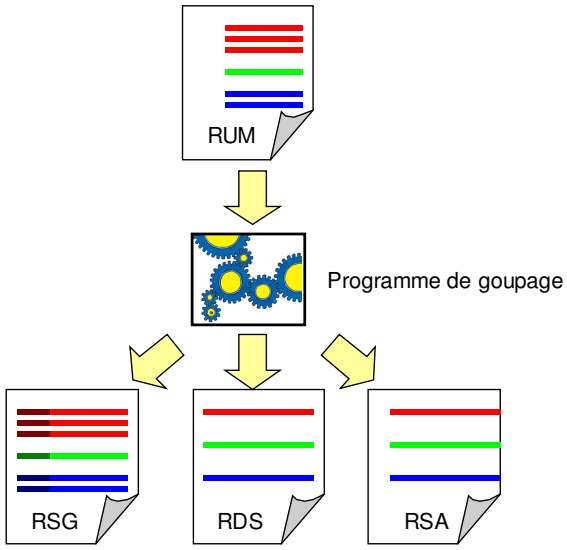
II. Pourquoi et comment fusionner des RSA et des RSG ?

A. Dans quel but ?

Ce tableau récapitulatif montre ce que peuvent s'apporter mutuellement le fichier de RSA et de RSG :

cause	Problème précis	**Fichier de RSA**	**Fichier de RSG**
Agrégation	*Une ligne par…*	séjour	Passage dans une UM
	Informations tarifaires	Tout ce qui est nécessaire	Quelques informations
	Unités médicales fréquentées	NON	OUI
Anonymisation	*Numéro de RSS, identification*	NON	OUI
	Dates d'entrée/sortie	Mois uniquement	OUI
	Date de naissance	Age uniquement	OUI
	Code postal	anonymisé	OUI

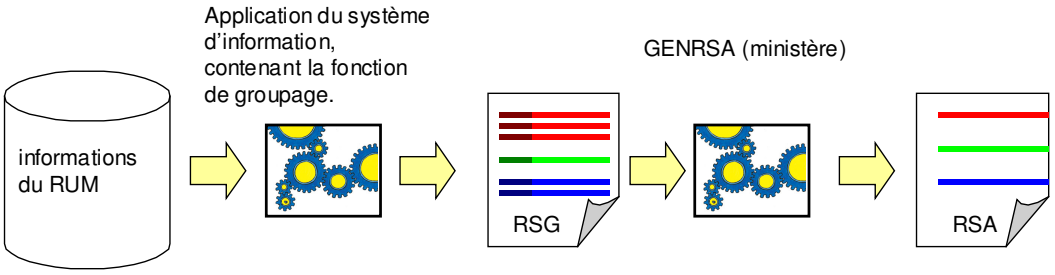
B. Sur quel fondement ?



Il nous paraît utile de re-préciser la genèse des fichiers PMSI. Les dénominations que nous avons utilisées sont justifiées plus bas, dans l'aide de DIND.

L'établissement fournit au programme de groupage un fichier de RUM, contenant les informations recueillies, mais pas les informations de groupage. Le numéro de RSS permet toutefois de savoir dans quelle mesure deux RUM constitutifs sont ou non inclus dans le même séjour de patient. A la lecture de ce fichier, le programme de groupage génère le fichier de RSG, qui est identique mais contient en plus quelques informations de groupage, comme le GHM du RSS. Il fournit également le RSA, et son équivalent non anonyme, le RDS. Ces fichiers ont perdu toute notion de structure interne à l'établissement, mais contiennent en plus des informations tarifaires.

Ce simple processus n'est en réalité pas le plus fréquent. Voici comment les choses se passent habituellement :



Les établissements disposent d'un système d'information qui génère directement le fichier de RSG car il contient une fonction de groupage, achetée par la société informatique au ministère. Ils doivent ensuite utiliser GENRSA, qui génère un fichier de RSA. Dans un tel processus, le fichier de RDS n'est jamais produit.

Quel que soit le processus, il est important de noter que les groupes de RUM d'un même séjour, et les RSA, sont présentés dans le même ordre. Effectivement, la notice du programme de groupage nous enseigne que, même en cas d'erreur, une ligne est toujours produite dans le fichier de RSA.

C. Par quel procédé ?

Partant de ce principe, nous avons fusionné dans leur ordre de présentation chaque RSA avec chaque groupe de RSG partageant le même numéro de RSS. Nous avons toutefois réalisé plusieurs contrôles, qui nous ont montré qu'il n'y avait pas d'erreur. Nous avons confronté les champs suivants :

Champs de chaque RSG du groupe	***Valeur reportée ou calculée de part et d'autre***	***Champs du RSA***
GHM et CMD	GHM et CMD	GHM et CMD
Sexe	Sexe	Sexe
Date de naissance, date de sortie	âge en ans, âge en jours (le jour de la sortie)	âge en ans, âge en jours
Date de sortie	Mois de sortie, année de sortie	Mois de sortie, année de sortie
Date d'entrée, date de sortie	durée	durée
Mode entrée, mode sortie, provenance, destination (de chaque RSG)	Mode entrée, mode sortie, provenance, destination (aux extrémités du séjour)	Mode entrée, mode sortie, provenance, destination (du RSA)
1	Nombre de RSG	Nombre de RUM
Nombre de séances	Nombre de séances	Nombre de séances

Sur les 40000 RSA de notre échantillon de test, 2 erreurs de fusion sont signalées :

1- la première concerne un patient âgé de 1 an et 0 jours, pour lequel GENRSA a calculé un âge de 0 ans et 365 jours
2- la deuxième concerne un groupe de RSG dont la structure est initialement aberrante

III. Modifications apportées à formats.xml

A. Introduction

Le DIM utilise actuellement une description XML des fichiers de résumés. Ce format a été défini intelligemment. La mise en place de ce fichier avait un caractère visionnaire, comparée à la philosophie générale des divers fichiers du PMSI. Toutefois, outre certaines incohérences, le format utilisé ne nous a pas semblé compatible avec les évolutions futures, prévisibles ou non, du PMSI.

B. L'ancien format

1. Extrait

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<formats>
	<format version="209">
		...
	</format>
	<format version="210">
		<typeFichier>RSA</typeFichier>
		<typeGH>GHS</typeGH>
		<typeAct>CCAM</typeAct>
		<fixe>
			<champ nom="FINESS">A9</champ>
			<champ nom="FILE_VER">A3</champ>
			<champ nom="INDEX">A10</champ>
			<champ nom="RSS_VER">A3</champ>
			<champ nom="GENRSA_VER">A3</champ>
			<champ nom="GRP_VER">A2</champ>
			<champ nom="GRP_CMD">A2</champ>
			<champ nom="GRP_GHM">A4</champ>
			...
			<champ nom="NB_DAS">A2</champ>
			<champ nom="NB_ACT">A2</champ>
			<champ nom="DAS"> </champ>
			<champ nom="ACTES"> </champ>
		</fixe>
		<variable start="123">
			<taille nom="T_DIAGS">6</taille>
			<taille nom="T_ACTES">8</taille>
			<taille nom="T_NB_ACT">2</taille>
		</variable>
	</format>
	<lexique>
		<commentaire nom="Partie Variable">Attention, les champs déclarés dans la
			partie variable ne doivent contenir qu'un espace.</commentaire>
		<champ nom="ACTES">L'ensemble des Actes du RUM ou RSS/RSA</champ>
		<champ nom="AGE_AN">Age en années</champ>
		<champ nom="AGE_JOURS">Age en jours</champ>
		<champ nom="AN_SORT">Année de sortie</champ>
		...
		<taille nom="T_ACTES">Taille d'un acte</taille>
		<taille nom="T_DIAGS">Taille d'un code diagnostic</taille>
		<taille nom="T_NB_ACT">Taille du nombre de fois qu'apparaît un acte</taille>
	</lexique>
</formats>
```

2. Inconvénients

- L'enregistrement de valeurs du type « A# », où # est un nombre, permettait un découpage très rapide en PERL à l'aide d'une fonction de hash. Toutefois il procure des inconvénients :
  - il n'est pas possible de définir un champ sur l'espace occupé par un autre (exemple : la CMD en v9 est constituée des 2 premiers caractères du GHM)
  - il est obligatoire de renseigner les espaces blancs, donc en dédoublant les champs éventuellement (UN et UN2)
  - il est obligatoire, en lisant une ligne, de charger tous les champs, même si un seul nous intéresse
- Les champs variables posent certains problèmes :

- il est obligatoire de renseigner les champs variables une première fois avec pour valeur un espace seul par convention, ce qui impose une trop grande dépendance entre format et code.
- à l'égard du champ « actes » par exemple, l'actuelle présentation n'est utilisable que lorsqu'on connaît déjà les champs à regrouper, par une documentation papier, soit ici T_ACTES et T_NB_ACT. La logique doit alors être embarquée en dur dans le code, il est impossible de s'adapter aux évolutions futures.
- à l'égard des champs variables en général, aucune référence n'est faite à la variable fixe qui renseigne le nombre de répétitions. Par exemple, comment savoir que le nombre de zones d'actes est fourni par la variable NB_ACT. De plus les champs désignent alors une simple longueur, ce qui, sans être faux, n'est pas cohérent avec le reste.
- ambiguïté sur « NB_ACT » qui désigne tantôt le nombre de zones d'actes et tantôt le nombre de réalisations de l'acte.

- Remarques plus générales :
  - manquent quelques informations sur le format
  - les balises « champ » et « taille » sont utilisées autant dans la description que dans le lexique, sans avoir le même sens ni la même structure
  - absence de schéma ou de DTD (qui d'ailleurs n'aurait pas pu être défini en l'état)
  - il existe certaines autres incohérences dans le nommage des champs

C. Notre proposition

Notre proposition s'inspire largement du format précédent, pour toutes ses qualités. En outre elle prétend, au prix d'une complexité accrue, résister aux évolutions futures, en restant plus abstraite.

1. Extrait

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<formats xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="formats.xsd">
       <format version="209" typeFichier="RSA" versionGHM="9" versionAct="CDAM"
              versionDiag="CIM10">
              ...
       </format>
       <format version="210" typeFichier="RSA" versionGHM="9" versionAct="CCAM"
              versionDiag="CIM10">
              <fixe>
                     <champ nom="FINESS"   debut="0"      taille="9"/>
                     <champ nom="FILE_VER" debut="9"      taille="3"/>
                     <champ nom="RSA_ID"   debut="12"     taille="10"/>
                     <champ nom="RSS_VER"  debut="22"     taille="3"/>
                     <champ nom="GENRSA_VER"      debut="25"     taille="3"/>
                     <champ nom="GRP_VER"  debut="28"     taille="2"/>
                     <champ nom="GRP_CMD"  debut="30"     taille="2"/>
                     <champ nom="GRP_GHM"  debut="30"     taille="6"/>
                     ...
                     <champ nom="NB_ZDAS"  debut="122"    taille="2"/>
                     <champ nom="NB_ZACT"  debut="124"    taille="2"/>
              </fixe>
              <variable debut="126">
                     <groupe nom="ZDAS"    taille="6"     champ_nombre="NB_ZDAS">
                            <champ nom="ZDAS:DAS" debut="0"      taille="6"/>
                     </groupe>
                     <groupe nom="ZACT"    taille="10"    champ_nombre="NB_ZACT">
                            <champ nom="ZACT:ACTE"       debut="0"      taille="7"/>
                            <champ nom="ZACT:PHASE"      debut="7"      taille="1"/>
                            <champ nom="ZACT:NB_REAL"    debut="8"      taille="2"/>
                     </groupe>
              </variable>
       </format>
       <lexique>
              <item type="groupe" nom="ZACT">L'ensemble des Actes du RUM ou RSS/RSA</item>
              <item type="groupe" nom="ZDAD">L'ensemble des Diagnostics Associés Documentaires
                     du RUM</item>
              <item type="groupe" nom="ZDAS">L'ensemble des Diagnostics Associés Significatifs
                     du RUM ou RSS/RSA</item>
              <item type="champ" nom="AGE_AN">Age en années</item>
              <item type="champ" nom="AGE_JOURS">Age en jours</item>
              <item type="champ" nom="AN_SORT">Année de sortie</item>
              ...
              <item type="champ" nom="ZACT:ACTE">Pour chaque acte : code de l'acte</item>
              <item type="champ" nom="ZACT:NB_REAL">Pour chaque acte : nombre de répétitions</item>
              <item type="champ" nom="ZACT:PHASE">Pour chaque acte : phase de réalisation</item>
              <item type="champ" nom="ZDAS:DAS">Pour chaque DAS : code du DAS</item>
```

```
        </lexique>
</formats>
```

2. Schéma XSD

Un schéma XSD (W3C) est fourni avec le fichier. Il n'est pas reproduit ici en raison de sa taille. Outre son exactitude structurelle, nous avons veillé à y inclure chaque fois que possible une énumération des valeurs possibles à l'aide des balises <xs:restriction base="xs:NMTOKENS"> et <xs:enumeration value="*valeur_possible*" />.
Le fichier formats.xsd est visible dans les sources de DIND. Il permet à l'administrateur de DIND de modifier les fichiers *.xml de formats sans risque d'erreur.

3. Apports

- Au sujet des champs en général, il est désormais possible de :
    - Définir deux champs sur les mêmes caractères (CMD et GHM)
    - Charger uniquement les champs souhaités, et donc créer un fichier de formats *.xml partiel, limitant l'utilisation de la mémoire
- Au sujet des champs variables :
    - Chaque zone est clairement définie comme un regroupement de champs
    - Lesdits champs peuvent être chargés séparément, tout comme les champs fixes
    - Le champ indiquant le nombre de répétitions de zones est identifié ; son nom peut, au besoin, n'avoir aucun lien lexical avec le nom de la zone.
    - La dénomination des champs dans les zones perd toute ambiguïté par un nommage univoque : "ZACT:ACTE" désigne le code de l'acte de chaque zone d'actes par exemple
    - Autant que possible, les noms des champs et des zones ont gagné en cohérence
- De manière générale :
    - Un schéma XSD est respecté
    - La logique est moins embarquée dans le code
    - Certaines ambiguïtés sont levées

La seule limite qui nous semble subsister est que l'ordre des groupes est l'ordre de déclaration dans le XML. Il aurait été possible d'ajouter un numéro d'ordre.

4. Passage au format 2005

Nous avons défini ce format avant la sortie des formats 2005 (011/111/211). Notre fichier XML a su « encaisser » les importantes modifications pourtant inattendues : l'inflation de la zone d'actes, l'ajout d'une zone d'unités médicales. A titre documentaire, nous fournissons un autre extrait de notre fichier :

```
...
	<format version="211" typeFichier="RSA" versionGHM="9" versionAct="CCAM" versionDiag="CIM10">
		<fixe>
			...
		</fixe>
		<variable debut="150">
			<groupe nom="ZUM"	taille="10"	champ_nombre="NB_RUM">
				<champ nom="ZUM:TYPE"		debut="0"	taille="2"/>
				<champ nom="ZUM:DUREE"		debut="2"	taille="3"/>
				<champ nom="ZUM:REA"		debut="5"	taille="1"/>
				<champ nom="ZUM:POST_DATE"	debut="6"	taille="1"/>
				<champ nom="ZUM:TYPE_FORCE"	debut="7"	taille="1"/>
				<champ nom="ZUM:POS_DP"		debut="8"	taille="2"/>
			</groupe>
			<groupe nom="ZDAS"	taille="6"	champ_nombre="NB_ZDAS">
				<champ nom="ZDAS:DAS" debut="0"	taille="6"/>
			</groupe>
			<groupe nom="ZACT"	taille="21"	champ_nombre="NB_ZACT">
				<champ nom="ZACT:DATE_REAL"	debut="0"	taille="3"/>
				<champ nom="ZACT:ACTE"		debut="3"	taille="7"/>
				<champ nom="ZACT:PHASE"		debut="10"	taille="1"/>
				<champ nom="ZACT:ACTIVITE"	debut="11"	taille="1"/>
				<champ nom="ZACT:EXT_DOC"	debut="12"	taille="1"/>
				<champ nom="ZACT:MODIF"		debut="13"	taille="4"/>
				<champ nom="ZACT:REMB_EX"	debut="17"	taille="1"/>
				<champ nom="ZACT:ASS_NP"	debut="18"	taille="1"/>
				<champ nom="ZACT:NB_REAL"	debut="19"	taille="2"/>
			</groupe>
		</variable>
	</format>
...
```

IV. Pourquoi travailler sur des fichiers texte ?

Ce choix fondamental mais radical a été fait dès le début. La réponse se formule différemment selon la catégorie de fichiers dont nous parlons.

A. Pour les fichiers utilisateur

1. Pour les fichiers de résumés

Les fichiers de résumés suivent un format unique sur tout le territoire. Pour aussi étrange que soit ce format, si notre application parle ce langage, elle sera comprise par tous les DIM de France. D'autre part, nous aurions pu prendre le parti d'interroger directement une base de donnée décisionnelle, du type Atalante, mais notre application aurait alors perdu sa portabilité.
Les fichiers de résumé nous ont permis de travailler sur le même support que la plate-forme e-PMSI, et donc de comparer les résultats au DATIM officiel.

2. Pour les fichiers de fusion et de résultats

La question s'est à nouveau posée pour les fichiers de sortie générés par le programme. Il est vrai que nous aurions gagné en rapidité en utilisant un stockage en base de données. La librairie SQLite aurait certainement facilité les choses, car elle permet de s'affranchir d'un SGBD, et de stocker toute une base dans un fichier unique. Deux arguments de poids se sont alors imposés :

a) Une vision utilisateur

Nous souhaitions par exemple qu'un utilisateur puisse reprendre par exemple un fichier de résultat pour le visualiser sur un PC portable dans une unité de soins. Cela nous a orientés vers une vision non linéaire de l'utilisation, qui a guidé notre conception du *fichier de projet*. L'individualisation claire d'un fichier de fusion et d'un fichier de résultat permet cela, tout en améliorant la compréhension que pourrait avoir l'utilisateur du programme (voir l'aide de DIND).

b) Une vision programmeur

La notion de base de données ne s'entend que dans un schéma relationnel, si possible normalisé. Ce schéma ne correspondait pas à la nature des traitements souhaités. Nos traitements utilisent tour à tour chaque RSA et les RSG qui les constituent, un peu comme dans une base objet. Nous nommons cet ensemble un résumé composite. La manipulation directe d'un fichier de fusion, résumé composite par résumé composite, répondait à cette attente.

B. Pour les fichiers de ressources

La réponse sera plus lapidaire :

- les fichiers disponibles suivent des formats texte déjà +/- normalisés, il fallait conserver leur format natif
- l'administrateur de DIND doit pouvoir modifier ces ressources à l'aide d'un simple gestionnaire de fichier, chaque modification doit prendre effet immédiatement
- l'écriture de descripteurs XML répond intelligemment aux problèmes

V. Pourquoi PHP5 ?

A. Pourquoi programmer DIND en PHP ?

1. Pourquoi une interface web

- L'interface WEB permet, via HTML, les formulaires et JavaScript, de produire une interaction puissante entre le programme et son utilisateur.
- La plupart des utilisateurs savent déjà naviguer sur un site internet.
- L'affichage des pages est illimité vers le bas, ce qui correspond bien à la quantité de données à traiter.
- Certaines contraintes du HTML/HTTP peuvent être contournées simplement : barres de progression en temps réel, time out, nécessité d'un serveur web
- Le développement est très rapide, on peut se contenter d'améliorer l'interface dans un second temps seulement, la sortie HTML est très pratique pour le débogage
- Le programme original DATIM est une application web dont l'interface est en HTML

Toutefois, il faut reconnaître que l'absence de persistance des processus en mémoire vive entre deux requêtes http est un handicap notable, mais il nous a poussés vers une bonne POO.

2. Pourquoi un langage de script

- Le PMSI est entouré d'une longue culture de programmation en PERL par les informaticiens de DIM. Or l'application doit pouvoir évoluer simplement.
- Comme nous l'expliquons plus bas, il a fallu que les rédacteurs de règles puissent utiliser un langage de script. Il semblait simple d'utiliser alors la fonction eval() dans le même langage que celui des règles.

Toutefois, il est certain que l'exécution d'un script est plus longue que celle d'un fichier compilé ou pré-compilé.

3. Pourquoi PHP et non PERL

- PHP est un peu plus simple que PERL, ce que les rédacteurs de règles apprécieront. A titre d'exemple, les scalaires, les listes, les tableaux associatifs, et les structures complexes, s'appèlent tous avec un symbole $ et non tantôt $ tantôt @ tantôt %. Ce simple détail facilite grandement la vie du programmeur débutant.
- La portée des variables est plus saine : sauf spécification contraire, les variables ne sont pas globales. Les procédures n'existent pas.
- A l'inverse de PERL, PHP5 bénéficie d'un véritable modèle objet.

- La richesse des librairies disponibles est comparable.
- Les expressions rationnelles dans PHP ne sont pas directement intégrées dans le langage contrairement à PERL, mais existent en tant que fonctions bien individualisées. Si on souhaite les utiliser, le code est alors un peu plus long, mais plus intelligible et reproductible par les non initiés.
- Grâce à EasyPHP, l'ensemble constitué par le programme et son environnement d'exécution peuvent être simplement copiés sur un PC, sans réelle installation, ce qui est un avantage notable dans le contexte de postes bloqués, comme c'est le cas au CHU.

B. Pourquoi PHP5 et non PHP4 ?

L'arrivée de PHP5 marque un tournant dans l'évolution de PHP. Outre les détails de la configuration par défaut et des librairies disponibles, PHP5 se distingue par un véritable modèle Objet. Loin de l'esquisse objet de PHP4, le modèle objet de PHP5 est comparable à celui des « grands » langages objet tels Java. Bref, PHP5 est à la fois un langage de script, facile à mettre en œuvre, et à la fois un véritable langage objet, permettant si on le souhaite une programmation propre, abstraite, modulaire, évolutive.

On y retrouve toutes les caractéristiques classiques :
- héritage, redéfinition, composition, méthodes et champs statiques, finaux, classes abstraites, implémentation, exceptions, réflectivité…
- accès public, privé, protégé
- typage (optionnel) des objets
- constructeurs et destructeurs

En revanche PHP se distingue par :
- absence de type de retour des méthodes : cette souplesse pourrait être source d'erreurs. Pourtant, c'est souvent un avantage car une méthode peut à la fois retourner false en cas d'erreur, et un objet en cas de réussite. L'exemple ci-dessous illustre cette idée :

```
<?php
$mon_fichier = new fichier_rsa('./donnees/rsa.rsa') ;
while( $un_resume = $monFichier->parcourt() ) {
       /* je peux traiter sans inquiétude $un_resume  */
       /* si le fichier est terminé ou en cas d'erreur, la boucle s'arrête toute seule.  */
       /* le test de fin du fichier est donc géré par la méthode et non par la boucle */
}
?>
```

- absence de typage primitif obligatoire : c'est un avantage pour les calculs numériques, qui restent exacts. C'est un avantage pour les fichiers du PMSI, où même les nombres sont en texte. Cela peut générer des erreurs avec les string, mais d'une part les cast explicites sont possibles, d'autre part l'opérateur de concaténation est le point et non le plus, levant ainsi certaines ambiguïtés.
- absence de surdéfinition : c'est au contraire un inconvénient. Il peut en revanche être pallié à l'intérieur de la méthode, mais on perd en clarté et en souplesse. En effet, certains arguments de méthode peuvent être déclarés comme facultatifs, le typage des arguments est facultatif, et rien n'interdit de tester le type des arguments à l'intérieur de la méthode, à l'aide de la réflectivité.

C. Pourquoi programmer les règles en PHP ?

Dans un premier temps, nous avons naïvement souhaité dresser une typologie très précise des règles DATIM, afin de les stocker directement dans un formalisme XML. Les choses semblaient possibles. Mais, dès la règle numéro 4, nous déchantâmes. Nous livrons son algorithme en annexe 5.

Il nous est apparu trop compliqué d'inventer un langage pour un tâche telle que créer un tableau associatif (ici T_GHM), puis l'incrémenter, puis le parcourir. Le poursuite de la lecture des règles nous a prouvé que leur polymorphisme était tel que la tâche semblait insurmontable.

D'autre part, l'expérience prouve que :

- les diverses expériences de nouveaux langages sont des échecs car les utilisateurs non programmeurs renoncent rapidement
- ces nouveaux langages sont à la fois une grosse charge de travail pour le programmeur, et à la fois très peu puissants
- au final ce sont toujours les informaticiens qui doivent écrire les règles, alors à quoi bon leur faire apprendre un nouveau langage souvent éloigné des langages classiques ?

Nous nous sommes alors orientés vers un langage de script facile à apprendre. Le PHP, langage très couramment parlé et très proche du PERL (en plus simple), semblait tout indiqué.

VI. Notes et essais personnels sur les destructeurs de PHP5

A. Premier essai

Afin de délester la mémoire vive, nous avons jugé utile d'utiliser des destructeurs et de forcer leur usage. Le test suivant mérite toute attention :

1. Code source

```
<?php
class ma_classe {
        protected $mot = "" ;
        public function __construct() {
                print "<p>*** construction</p>" ;
        }
        public function action($mot) {
                print "<p>Mot valait [".$this->mot ."] et vaudra [$mot].</p>" ;
                $this->mot = $mot ;
        }
        public function __destruct() {
                print "<p>*** destruction</p>" ;
        }
}

class ma_classe2 extends ma_classe {
        public function __destruct() {
                parent::__destruct() ;
        }
}


$mon_objet = new ma_classe() ;
$mon_objet->action("bonjour") ;
$mon_objet->__destruct() ;
$mon_objet->action("adieu") ;

print "<p>Coucou !</p>" ;

?>
```

2. Sortie

```
*** construction
Mot valait [] et vaudra [bonjour].
*** destruction
Mot valait [bonjour] et vaudra [adieu].
Coucou !
*** destruction
```

3. Conclusions

1- la méthode __destruct() se contente de contenir les instructions de clôture. Comme le montre l'exemple, la destruction d'un objet... ne le détruit pas. Si on force la destruction, en réalité, l'objet persiste. Il faut donc définir à la main la méthode __destruct() et lui donner un ensemble d'instructions : mise à *null* des champs de type primitifs, destruction des champs qui sont des objets.

2- la méthode __destruct() est appelée à la fin, c'est à dire non pas lors de la dernière utilisation de l'objet, mais après tous les processus. Cela n'allège donc la mémoire vive que dans le cas critique de l'activation du garbage collector / ramasse-miettes.

3- même si l'objet a déjà été détruit, la méthode __destruct() est tout de même appelée en fin de script.

De ces deux premières conclusions nous est apparue la nécessité de définir des méthodes __destruct() et de les appeler manuellement au moment opportun. On notera toutefois que, contrairement aux autres méthodes, ni le constructeur, ni le destructeur ne sont hérités. Il est nécessaire de les créer à chaque fois comme dans l'exemple de ma_classe2.
La troisième conclusion est lourde de sens, nous l'examinons au test suivant.

B. Deuxième essai

Soit une classe contenant un champ qui est lui-même un objet.

1. Code source

```
<?php
class ma_classe {
        protected $objet_contenu ;

        public function __construct() {
                $this->objet_contenu = new ma_classe2() ;
        }
        public function __destruct() {
                print "<p>debut destruction...." ;
                $this->objet_contenu->__destruct() ;
                $this->objet_contenu = null ;
                print "....fin destruction</p>" ;
        }
}

class ma_classe2 {
        // un objet quelconque, ce qui suit est strictement inutile
        // mais nous rassurera sur l'origine de l'erreur
        public function __destruct() {
                // rien de spécial
        }
}

$mon_objet = new ma_classe() ;
$mon_objet->__destruct() ;

print "<p>Coucou !</p>" ;
?>
```

2. Sortie

```
debut destruction........fin destruction
Coucou !
debut destruction....
Fatal error: Call to a member function __destruct() on a non-object in essai_destructeurs.php on line 10
```

erreur fatale : appel de la méthode __destruct() sur un membre non-objet

3. Conclusions

L'erreur a ici été de vouloir détruire l'objet, puis « supprimer » son pointeur, ou plus exactement rompre le lien entre le pointeur et l'objet. Ceci crée une erreur fatale (ici après les traitements, donc sans gravité) car le pointeur $objet_contenu ne désigne plus un objet mais *null*, l'objet référencé ne peut donc plus être détruit.
On peut donc détruire les objets, annuler les champs, mais pas annuler les pointeurs. Rappelons que l'annulation d'un pointeur sans destruction de son objet serait sans aucun intérêt, si ce n'est l'activation du ramasse-miettes.
Autre remarque, un objet pouvant constituer un champ de plusieurs autres objets, la destruction aura en fait lien *au moins* deux fois.

C. Mise en pratique

1. Premier essai sans ramasse-miettes

La deuxième phase de DIND charge tous les RSA, tous les RSG, les compare et les fusionne, puis écrit un fichier de fusion. Sur notre fichier de test se trouvent approximativement 40 000 RSA et 60 000 RSG, qu'il faut découper en de nombreux champs pour les comparer et les fusionner.

Dans un premier temps, craignant que les RSG ne fussent pas tous dans un ordre cohérent, nous avons réalisé ces étapes successivement. Concrètement, cela signifie que chaque objet était désigné par un pointeur unique du type :

```
<?php
/* boucle RSA */ {
        $tableau_rsa[]=new rsa(...);
        /* ... traitements ... */
}
?>
```

L'effet fut catastrophique. La RAM était saturée dès le début et, au bout du timeout de 10 minutes, seulement la moitié de la tâche était réalisée.

2. Ramasse-miettes spontané

Dans un deuxième temps, partant de l'hypothèse forte (mais vérifiée) que tous les RSG de même numéro étaient à la suite, nous avons réalisé ces opérations à la volée, RSA par RSA, en écrasant les pointeurs à chaque fois, du type :

```
<?php
/* boucle RSA */ {
        $mon_rsa=new rsa(...);
        /* ... traitements ... */
}
?>
```

Le temps d'exécution passa à 4 minutes 30 secondes, sans saturation de la RAM. On peut légitimement penser que cette amélioration provient de l'entrée en action du garbage collector / ramasse-miettes.

3. Nettoyage maison

Dans un troisième temps, après avoir décrit précisément les destructeurs dans les classes, nous avons simplement ordonné la destruction de chaque résumé après son utilisation, ainsi :

```
<?php
/* boucle */ {
        $mon_rsa=new rsa(...);
        /* ... traitements ... */
        $mon_rsa->__destruct() ;
}
?>
```

Le temps d'exécution passa de 4'10" à 2'25".

Cet exemple montre bien la puissance du garbage collector / ramasse-miettes. Il montre également combien il peut parfois être utile de ne pas attendre son entrée en scène, en définissant et en invoquant soi-même les méthodes __destruct().

Rappelons que faire dans ces destructeurs :

- annuler les champs primitifs
- détruire les champs objets
- ne pas annuler les champs objets (ou plus précisément les pointeurs).

Annexe 1 : sujet de stage initial

I. Présentation de DATIM

DATIM est une application web implantée sur la plate-forme e-PMSI. Elle analyse tous les fichiers de RSA télé-transmis par les établissements aux tutelles via cette plate-forme. Sur chaque fichier, DATIM recherche des « atypies » en appliquant 64 algorithmes de contrôle, et fournit finalement un score.
Cette analyse est effectuée avant validation par l'établissement. En revanche, le contrôleur n'a accès à cette information qu'après validation. L'établissement peut ainsi corriger le tir, à ceci près que les RSA sont anonymes. Après validation, le résultat de l'analyse est susceptible de déclencher un contrôle administratif dans l'établissement, afin de déceler des anomalies de codage ou de prise en charge, voire d'infliger des sanctions financières. Les contrôleurs peuvent confronter les résultats de tous les établissements.
Les sources de DATIM ne sont pas publiées. En revanche, les 64 algorithmes sont publiés sous forme papier sur le site www.atih.sante.fr.

II. Implémentation de DATIM

L'objet du stage est d'implémenter les algorithmes de DATIM. Toutefois, certaines contraintes supplémentaires sont posées, afin d'étendre le champ du programme au-delà de ce que fait déjà la version en ligne du ministère.
Contraintes d'utilisation :

- permettre un stockage séparé des algorithmes afin de :
  - faciliter les mises à jour ultérieures
  - pouvoir définir des règles utilisateur supplémentaires (afin par exemple de détecter des dossiers candidats à un recodage par le DIM)
- permettre un stockage séparé des formats de fichiers PMSI (ces formats sont très changeants)
- permettre le stockage et l'utilisation d'une information de structure, afin d'agréger des données par Clinique (actuellement il s'agît d'unités médicales)
- permettre de retrouver d'une manière ou d'une autre le numéro des RSS « atypiques » dans deux buts :
  - pouvoir recoder certains RSS problématiques
  - cibler des unités médicales génératrices d'atypie, et donc candidates à des interventions de formation par le DIM
- afficher des vues synthétiques à l'échelle du CHU, à l'échelle de la Clinique, à l'échelle du test.

Annexe 2 : sigles utilisés

En italique figurent les sigles propres à ce document, donc non officiels.

ATIH	Agence Technique de l'Information Hospitalière
CCAM	Classification Commune des Actes Médicaux
CdAM	Catalogue des Actes Médicaux
CIM10	Classification Internationale des Maladies, 10° révision
CHRU	Centre Hospitalier Régional Universitaire
CHU	Centre Hospitalier Universitaire
CM	Catégorie Majeure (regroupement de GHM)
CMD	Catégorie Majeur de Diagnostic (= la plupart des CM)
CR	Centre de Responsabilité (=Clinique)
DAD	Diagnostic Associé Documentaire (information du RUM)
DAS	Diagnostic Associé Significatif (information du RUM et du RSA)
DATIM	Détection des ATypies de l'Information Médicale
DIM	Département de l'Information Médicale
DIND	*Dind Is Not Datim*
DMI	Dispositifs Médicaux Implantables
ENC	Echelle Nationale des Coûts
ENT	Echelle Nationale des Tarifs
EXB	Séjour extrêmement court (minoration)
EXH	Séjour extrêmement long (supplément journalier)
GHM	Groupe Homogène de Malade
GHS	Groupe Homogène de Séjour
HAD	Hospitalisation A Domicile
MCO	Médecine Chirurgie Obstétrique (= Court séjour)
PHP	PHP : Hypertext Preprocessor
PMSI	Programme de Médicalisation des Systèmes d'Information
RDS	*Résumé De Sortie*
RUM	Résumé d'Unité Médicale
RSS	Résumé de Sortie Standardisé
RSA	Résumé de Sortie Anonymisé
RSG	*Résumé de Sortie standardisé Groupé*
SSR	Soins de Suite et de Réadaptation (=moyen séjour)
T2A	Tarification A l'Activité
UM	Unité Médicale

Annexe 3 : autres tâches réalisées en stage

I. Journées Francophones d'Informatique Médicale

Organisation matérielle des stands de la DSIH (direction du système d'information hospitalier : démonstration des applications hospitalières) et des stands des industriels.

II. Représentation graphique des données d'activité

- recherche bibliographique sur les nouveaux modes de représentation graphique
- implémentation des algorithmes TREEMAP « Cluster » et « Squarified » dans une classe PHP5 permettant l'édition à la volée de graphiques au format PNG, intégrable dans les tableaux de bord par une macro Excel, ou au format SVG
- présentation en réunion de service

III. Passage à la version 1de la CCAM

- recherche de codes en deuxième intention (demandes internes au DIM)
- centralisation, rédaction et suivi des questions posées par le DIM aux tutelles sur les forums
- validation de la mise en forme et du contenu des catalogues des services mis en ligne sur Otalia.

IV. Financements MIGAC MERRI

- Identification et quantification des ressources consacrées à la Recherche dans l'activité des services hospitaliers.
- Préparation de la déclaration nationale par les CHU d'indicateurs communs, en réponse aux palmarès publiés par la presse.

V. Hospitalisation à domicile (HAD)

- veille bibliographique, rédaction et mises à jour d'un manuel synthétique de l'HAD, formation interne au service
- participation aux réunions préparatoires à la mise en place de l'HAD au CHU

Annexe 4 : diffusion des algorithmes DATIM (ministère)

Contenu de la page http://www.atih.sante.fr/?id=0002400068FF :

DATIM, outil d'aide à l'analyse de la qualité des informations PMSI dans les fichiers de RSA, est en ligne sur e-PMSI.
Ce logiciel s'adresse aux ARH, notamment à l'Unité de Coordination Régionale chargée d'organiser les contrôles externes ainsi qu'aux établissements désireux d'enrichir leur démarche d'amélioration de la qualité.
Les droits requis pour accéder aux résultats à travers la plate forme e-PMSI sont, le rôle "contrôleur" pour les médecins contrôleur et le rôle "lecteur" pour l'établissement.
Le traitement DATIM est effectué à la suite de MAT2A lors de chaque commande de traitement du fichier de RSA par l'établissement. Les résultats sont consultables immédiatement par le rôle "lecteur". En revanche, le rôle "contrôleur" ne peut pas visualiser les résultats de l'établissement tant que ce dernier n'a pas procédé à la validation de son fichier de RSA.
Exceptionnellement, les résultats DATIM relatifs aux résumés de l'année 2004 sont simultanément accessibles aux établissements et aux contrôleurs.
A ce jour, seuls les établissements "publics" sont traités. Les établissements "privés" seront intégrés d'ici quelques semaines.
Diverses fonctionnalités supplémentaires, dont le module de préparation du contrôle à l'usage des médecins contrôleurs, sont prévues et seront prochainement mises en service.

Documentation

Guide le lecture et d'interprétation
Présentation générale de DATIM, définition littérale des tests, éléments d'interprétation et modalités de calcul des scores.
GuideLectureDatim041.pdf http://www.atih.sante.fr/openfile.php?id=1027

Algorithmes détaillés des tests
Définition de chaque test sous forme algorithmique.
AlgorithmesDatim041.pdf http://www.atih.sante.fr/openfile.php?id=1017

Tables de références
Caractéristiques de diagnostics, d'actes et de GHM utilisées dans les tests.

DocTablesRef041.doc	http://www.atih.sante.fr/openfile.php?id=1018
GhmInfo.041	http://www.atih.sante.fr/openfile.php?id=1019
DiagInfo.041	http://www.atih.sante.fr/openfile.php?id=1020
CdamInfo.041	http://www.atih.sante.fr/openfile.php?id=1021
CcamInfo.041	http://www.atih.sante.fr/openfile.php?id=1022
LibTest.041	http://www.atih.sante.fr/openfile.php?id=1023

Annexe 5 : exemple de la règle n°4 de DATIM

Capture d'écran d'une partie du document PDF http://www.atih.sante.fr/openfile.php?id=1017

Test 4 : pourcentage de décès hors norme

Définition :

Calcule le nombre de GHM hors CM 24 et 90, comprenant plus de trois décès et dont le pourcentage de décès est dix fois plus grand ou dix fois moindre que la moyenne nationale de référence.

Fonction	: évaluation de la qualité du fichier
Type	: 2
Valeur résultat	: nombre de GHM
Dénominateur	: sans objet
Alerte	: oui / non (oui si nombre de GHM > 0, 0 sinon)
Pondération	: 1
Score	: pondération * alerte (oui=1, non = 0)

Algorithme :

- Variables :

NumTest	: Numérique, Entier	//Numéro du test
NbMiniDeces	: Numérique, Entier	//Nombre minimal de décès par GHM
BorneInf	: Numérique, Réel	//% national de décès / 10
BorneSup	: Numérique, Réel	//% national de décès * 10
RatioDeces	: Numérique, Réel	//% de décès pour le GHM observé
GhmTraite	: Numérique, Entier	//Numéro du GHM en cours de traitement
Resultat	: Numérique, Entier	//Nombre de GHM atypiques

- Fichiers :

F_RSA : Fichier de RSA

- Tables :

T_GHM (NumGHM, CMD, Nbdeces_GHM, effectif_GHM)

//Table GHM établissement : Numéro du GHM, CMD, Nombre de mode sortie 9 par GHM, effectif du GHM

- Initialisation :

NumTest	← 4
NbMiniDeces	← 3
Resultat	← 0

- Calcul table

  Créer T_GHM(NumGHM, CMD, Nbdeces_GHM, effectif_GHM) ← F_RSA (R_Ghm_GenRSA, R_CMD_GenRSA, R_Md_S)

  Avec :

  Nbdeces_GHM : *nombre de mode de sortie = 9 pour un NumGHM donné*
  effectif_GHM : *effectif pour un NumGHM donné*

- Traitement

  Pour tous T_GHM (NumGhm) faire :

```
GhmTraite    ← T_GHM (NumGhm)
BorneInf     ← T_GhmInfo(%deces,[GhmTraite=NumGhm]) / 10
BorneSup     ← T_GhmInfo(%deces,[GhmTraite=NumGhm]) * 10
RatioDeces   ← T_GHM(Nbdeces,[GhmTraite=NumGhm])/
               T_GHM(effectif_GHM,[GhmTraite=NumGhm])

Si   T_GHM(Nbdeces,[GhmTraite=NumGhm]) > NbMiniDeces
  ET
    ((RatioDeces  < BorneInf) OU (RatioDeces  > BorneSup))
  ET
    T_GHM(CMD,[GhmTraite=NumGhm]) ∉  {24,90}
      Alors
        Resultat ← Resultat + 1
        Editer
          GhmTraite;
          T_GHMInfo (LibelleGHM,[GhmTraite=NumGhm]);
          T_GHM(effectif_GHM,[GhmTraite=NumGhm])
          RatioDeces
          BorneInf
          BorneSup
        FinEditer

    FinSi {T _GHM(Nbdeces,[GhmTraite=NumGhm]) > NbMiniDeces }

FinPour {Pour tous T_GHM (NumGhm) faire :}
```

- Calcul résultat

  T_ResulDatim(Valeur,[Test=NumTest])← Resultat

- Calcul de l'alerte

  Si Resultat > 0 Alors T_ResulDatim(Alerte, [Test=NumTest]) ← 1

- Calcul du score

  T_ResulDatim(Score, [Test=NumTest]) ← T_ResulDatim(alerte, [Test=NumTest]) * T_LibTest(ponderation,[test=NumTest])