TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

AE	Application Entity
AET	Application Entity Title
API	A Programming Interface
CMS	Cryptographic Message Syntax
DICOM	Digital Imaging and Communications in Medicine
HTML	HyperText Markup Langage
LATIS	Laboratoire d'Analyse et de Traitement de l'Information et des Signaux
PDU	Protocol Data Unit
SOP	Service Object-Pair
SCP	Service Class Provider
SCU	Service Class User
TCP	Transport Control Protocol
UID	Unique IDentifier
VR	Value Representation

INTRODUCTION

Le traitement et l'analyse des images médicales permettent d'améliorer la qualité de ces images afin d'identifier des caractéristiques et offrir un outil supplémentaire au spécialiste pour l'aider à poser un diagnostic. Les algorithmes développés en recherche pour améliorer la qualité des images doivent être testés avec des données provenant de patients. Ces données médicales existent, elles sont abondantes et leur distribution est facilitée parce qu'elle est faite de façon électronique. Toutefois, avant d'utiliser ces informations en recherche, il faut tenir compte du droit du patient à la confidentialité et à la protection de la vie privée. En effet, ces données médicales contiennent de l'information nominative (comme le nom, l'adresse, le numéro de patient, etc) permettant d'identifier le patient auquel elles appartiennent. Bien que ces informations soient nécessaires (pour ne pas dire essentielles) en institution pour garantir qu'un dossier médical appartienne précisément à un patient donné plutôt qu'à un autre, elles constituent une menace importante à la vie privée du patient lorsqu'elles sont exportées là où elles n'ont pas lieu d'être normalement, c'est à dire en dehors des murs de l'hôpital. Les institutions hospitalières sont soumises à des règles strictes qui garantissent, dans une certaine mesure, le droit du patient à la confidentialité et la protection de la vie privée. Même avec l'autorisation (nécessaire) du patient pour exporter les données, celles-ci doivent être mises sous une forme telle qu'elles ne représentent plus une atteinte aux droits du patient.

Cela dit, il existe des principes, techniques et outils afin de rendre un ensemble de données médicales « exportables » vers un milieu non-institutionnel. Ces techniques de base sont ce que nous nous permettons d'appeler la « désidentification » et « l'anonymisation ». La première technique, celle de désidentification, consiste à retirer toutes les informations nominatives pouvant mener à l'identification du patient en éliminant l'information ou en la remplaçant par des données factices. Cette technique, bien que nécessaire, ne garantit pas la confidentialité de l'information. En effet, dans un de ses travaux, L. Sweeney a prouvé qu'il était possible d'identifier précisément 87% des individus de la population états-unienne avec l'aide de seulement trois renseignements : le sexe, la date de naissance et le code postal [30]. Il est donc évident que d'appliquer la désidentification seulement n'est pas suffisant pour assurer la confidentialité du

patient. La seconde technique, complémentaire à la première, est l'anonymisation de l'information. Cette technique consiste à prendre l'information du patient et à la rendre ambigüe. Il s'agit, bien souvent, de faire en sorte que l'information d'un individu soit indistinguable de l'information des autres et que la probabilité de l'identifier avec précision soit gardée en dessous d'un seuil donné. Cela se fait concrètement par une généralisation de l'information où les valeurs numériques sont exprimées en termes d'intervalles (p. ex., [20-25] pour représenter 21) et les valeurs plus catégoriques sont exprimées à l'aide d'ensembles (p. ex., {Canada, É.-U., Mexique} pour Canada) ou de valeur représentant cet ensemble (p. ex., Amérique du nord). L'application d'un algorithme d'anonymisation implique une perte de précision par rapport aux données originales. Cette perte de qualité est quantifiable et des équations permettant de l'estimer fournissent une indication en ce sens. Cependant, il n'est pas du ressort de ce mémoire d'aborder les métriques relatives à la qualité de l'information découlant d'un algorithme d'anonymisation. Ces techniques d'anonymisation sont couramment utilisées et fonctionnent relativement bien pour des ensembles de données statiques. Leur application sur des ensembles de données évoluant dans le temps représente un défi supplémentaire. Tel est le problème que nous étudions dans ce mémoire de maîtrise.

Un patient est généralement suivi en clinique sur une longue période de temps. La progression de la maladie et l'évolution du diagnostic représentent, si elle n'est pas inestimable, une information au moins supplémentaire afin de poser un meilleur diagnostic. Les séries temporelles (« Time series ») qui comportent plusieurs images en série sont communément utilisées de nos jours. Pourtant, peu d'assertions basées sur l'évolution temporelle de la maladie ne sont effectuées automatiquement. Nous croyons qu'il s'agit là d'une erreur et que l'évolution du diagnostic et de la maladie elle-même constituent une information clinique importante aux fins de la recherche. C'est là qu'entre en jeu ce que nous appelons la « pseudonymisation ». Cette troisième et dernière technique consiste à prendre un numéro d'identification unique appartenant au patient (p. ex., le patient ID) à partir duquel on génère un nouvel identificateur. On conserve ensuite le lien avec le numéro original afin d'assurer la pérénité de l'information dans le temps. En effet, une fois les données du patient désidentifiées et anonymisées, il n'est plus

possible de mettre à jour l'information du patient en milieu de recherche. Cette technique est utilisée afin de mettre à jour l'information du patient de façon périodique. Le pseudonyme sert à camoufler le véritable identificateur du patient et permet à l'organisme détenteur de l'information de mettre à jour l'information périodiquement sans que la confidentialité du patient ne soit compromise. Le pseudonyme peut être réversible ou irréversible dépendant que l'on veuille, ou non, retracer l'identité du patient (ayant obtenu son consentement bien entendu). La pseudonymisation réversible peut être implémentée grâce à une table de correspondance (« mapping ») entre l'identificateur réel et le pseudonyme ou par un algorithme de chiffrement dont on connaît la clé (p. ex., AES). La pseudonymisation irréversible est implémentée grâce à une fonction de hachage (p. ex., SHA-1).

Le standard DICOM (pour « Digital Imaging and Communications in Medicine ») [23] est développé conjointement par le « American College of Radiology » (ACR) et le « National Electrical Manufacturers Association » (NEMA). Il sagit d'un standard décrivant, entre autres, la manière de transférer et de sauvegarder une image médicale. Les données que nous tentons d'anonymiser sont des données radiologiques en format DICOM. Comme la majeure partie de l'imagerie médicale actuellement disponible est sous le format DICOM, la procédure de désidentification que nous proposons suit les recommandations énoncées dans ce standard. Le standard DICOM a publié récemment un supplément sur la sécurité concernant la confidentialité au niveau de l'attribut des ensembles de données DICOM en incluant une partie sur la désidentification (le supplément 55). Ce supplément décrit les extensions qui sont apportées au standard original pour nous permettre de faire partiellement le type de mise à jour incrémentale qui fait l'objet de ce mémoire.

La nature même des données dans le format DICOM pose plusieurs défis importants à la réalistion de ce projet. Premièrement, le fait que les données DICOM soient représentées sous la forme de fichiers et non sous une forme tabulaire contrevient grandement aux techniques d'anonymisation qui sont actuellement en usage. Deuxièmement, la présence de UIDs uniques

pour représenter l'instance de la classe SOP[1], les études et les séries qui réfèrent à l'instance originale ajoute un niveau de complexité supplémentaire pour l'implémentation de l'opération de pseudonymisation et cela n'est que sommairement pris en considération dans le supplément DICOM. Troisièmement, la présence des données originales chiffrées à l'intérieur même des instances, tel que décrit par le supplément, pose un problème de sécurité à l'intérieur même du fichier image.

La technologie Java est certainement un choix judicieux pour réaliser l'implémentation de l'application dans le cadre de ce projet. Java est une plate-forme intégrant des aspects reliés à la sécurité informatique ayant connu un succès commercial [11]. Son *Java Cryptography Extension* (JCE) intégré depuis la version 1.4 du *Java 2 Software Development Kit* (J2SDK) fournit les algorithmes nécessaires à la signature numérique, au chiffrement et à la création ainsi qu'à la gestion des clés de chiffrement. Les algorithmes de chiffrement sont utilisés en conjonction avec les clés de chiffrement dans le cadre de ce projet pour prendre un identificateur en texte clair et en générer un pseudonyme et obtenir ainsi un pseudonyme réversible. Les algorithmes de hachage cryptographique (SHA-1, MD5) peuvent servir à générer des pseudonymes irréversibles. La plate-forme Java est également très prisée par les développeurs de cadres d'applications (« application framework ») DICOM. En effet, il existe une vaste gamme d'outils pour lire des fichiers DICOM, les recevoir, les envoyer, manipuler leur contenu. Les plus connus sont dcm4che, Dicom4J, JDCM et PixelMed pour ne nommer que les plus populaires. C'est à partir des cadres JDCM et PixelMed que nous avons conçu notre système d'anonymisation DICOM.

La contribution, présentée dans ce mémoire, consiste à prendre les outils précédents, à en faire l'analyse, à cerner les contraintes et à trouver le meilleur compromis approprié dans le but de concevoir et d'implémenter une solution logicielle permettant d'offrir une banque d'images médicales DICOM destinée à la recherche. Les informations nominatives du patient seront classifiées, certaines d'entre elles seront effacées, d'autres verront leur contenu modifié selon

[1] Une classe SOP (« Service-Object Pair Class ») consiste en l'union entre les définitions d'objets d'information (« Information Object Definition ») et l'ensemble des services qui peuvent s'y appliquer (« DICOM Message Service Element [DIMSE] Service Group »). Voir [23] pour plus de détails

une politique de désidentification clairement établie. Les images seront désidentifiées de manière à assurer la confidentialité des patients présents dans la banque d'images. Les données cliniques et les données de recherche seront séparées mais évolueront au même rythme à mesure que de nouvelles images arriveront des modalités d'acquisition. Dans le but d'assurer une mise à jour continue des données, la notion d'identificateur unique sera prise en compte à l'intérieur de la politique de désidentification et une procédure spéciale de pseudonymisation sera appliquée qui pourra être, soit réversible, soit irréversible tout dépendant des règles stipulées dans la politique de désidentification. Grâce à ce lien, les données désidentifiées relatives à un individu seront répertoriées sous l'individu correspondant du côté de la recherche. Les fonctionnalités de chiffrement disponibles à l'intérieur du langage seront exploitées de manière à obtenir une solution sécuritaire pour générer des pseudonymes nous permettant de faire le lien entre le monde réel et celui de la recherche.

Ce mémoire se divise en tois chapitres. Le premier chapitre traite des concepts théoriques qui entrent dans l'analyse et la conception du processus d'anonymisation qui a lieu par la suite. Ce chapitre relate les points saillants de la littérature scientifiques qui sont directement reliés à notre problème. On aborde les concepts relatifs aux opérations de désidentification, d'anonymisation (généralisation des données) et de pseudonymisation qui sont exploités dans la suite du mémoire. Le second chapitre consiste en l'analyse et la conception du système et sert à circonscrir d'avantage le problème en vue de l'élaboration de la solution finale. Le troisième et dernier chapitre qui traite de la mise en œuvre, présente une avenue de solution fonctionnelle de manière à répondre aux exigences dictées par les objectifs du projet. Ce mémoire comporte également 2 annexes pour servir d'appui à la partie traitant de l'implémentation. Le premier annexe dresse une liste assez complète des patrons de conception qui furent mis à profit dans l'élaboration du logiciel Anonym 1.0. Chaque patron y est décrit avec le niveau de détail nécessaire à la compréhension du fonctionnement général du patron et du rôle spécifique qu'il remplit à l'intérieur de l'application. L'annexe III contient le guide d'exploitation qui aide à la compréhension du logiciel.

CHAPITRE 1

REVUE DE LA LITTÉRATURE

Notre objectif consiste à produire une banque d'images médicales provenant du secteur clinique pour la rendre disponible à la recherche. Cette banque d'images doit obligatoirement être conçue pour protéger la confidentialité des patients qui auront préalablement donné leur autorisation pour l'exportation de ces informations. Pour y arriver, l'information de chaque image contenue dans la banque doit être altérée de telle sorte qu'il soit impossible de retracer le patient auquel l'image appartient. Toutefois, ces modifications doivent se faire en conservant une qualité optimale des données. De plus, la banque d'images doit offrir des mécanismes pour permettre sa mise à jour périodique de façon incrémentale grâce à l'ajout successif de nouvelles images. Pour cela, il faut prendre en considération non seulement l'anonymisation des images elles-mêmes, mais aussi celle des images déjà présentes dans la banque au moment de l'insertion de la nouvelle image. Les opérations qu'il est possible d'effectuer sur un attribut d'une table qu'on cherche à anonymiser sont de l'enlever, le vider ou de changer son contenu.

Ce chapitre présente les techniques de désidentification, d'anonymisation et de pseudonymisation qui sont mises en œuvre dans ce mémoire. Les techniques de désidentification et d'anonymisation sont utiles pour la protection de la confidentialité des patients. La technique de pseudonymisation s'applique à la mise à jour incrémentale des bases de données anonymisées. Chaque technique est définie puis analysée sommairement.

1.1 La désidentification

La première étape que doit subir un ensemble de données qu'on veut rendre disponible à la recherche est la désidentification. La désidentification consiste à enlever tous les identificateurs explicites pouvant mener à l'identification directe du patient [3]. Le nom du patient, son adresse ou son numéro de patient sont des exemples d'identificateurs explicites. Les opérations qu'il est possible d'effectuer pour désidentifier un champs sont d'enlever ce champs, le vider (le remplir

de blancs) ou de modifier son contenu pour une valeur qui ne mènera pas à l'identification du patient.

Une image DICOM comporte une panoplie d'attributs provenant de différents modules. Tous les attributs ne mènent pas nécessairement à l'identification du patient ; leurs valeurs n'étant pertinentes que pour l'équipement ayant servi à faire l'acquisition de l'image. Le supplément 55 de DICOM propose un mécanisme de protection des attributs dans n'importe quelle instance d'image DICOM (SOP instance) [22]. Ce supplément présente une façon de sécuriser un fichier DICOM « à la pièce » (c.-à-d. chaque attribut individuellement) ce qui fait que l'on peut manipuler sans problème l'image avec une application qui n'est pas au fait du niveau de sécurité qui lui a été appliqué. Il est également possible de continuer à utiliser les fonctionnalités de bas niveau du standard tels que les services de messages et les protocoles employés lors des transferts sur le réseau, du stockage et lors d'échange de média contenant des objets d'information.

Le supplément 55 dresse une liste des attributs contenus dans la plupart des objets d'information (IOD) qui doivent être protégés pour fournir un niveau minimal de confidentialité au patient. Le tableau I.1 reprend cette liste. Ce tableau contient le nom, l'étiquette (servant d'identificateur unique à l'attribut), le type et une brève description des attributs à protéger. La signification de la colonne type est expliquée plus loin dans le texte.

La désidentification présente trois alternatives : retirer l'attribut de l'objet DICOM, changer sa valeur ou vider l'attribut de son contenu (le remplir de blancs). Dans le cas du retrait, l'attribut ne fait plus partie de l'objet DICOM une fois l'opération de désidentification terminée. C'est la meilleure alternative dans la mesure où, puisque l'attribut du patient est enlevé, il ne peut contribuer à faire augmenter le risque de voir une personne malveillante déduire l'identité du patient. Cette technique a cependant le défaut de diminuer considérablement la qualité de l'information de l'objet DICOM ; la valeur de l'attribut utile à un adversaire pour déterminer l'identité du patient, peut aussi s'avérer pertinente à des fins de recherche. Son absence constitue souvent une perte grave au niveau de l'information utile. De plus, l'absence d'un attribut

peut compromettre la validité de l'objet d'information DICOM en fonction des règles énoncées dans le standard. La deuxième technique, la modification, consiste à donner à un attribut DICOM une valeur factice. Il s'agit d'une techniques où l'attribut est conservé à l'intérieur de l'objet d'information DICOM tout en respectant certaines règles de conformité avec le standard. La troisième et dernière alternative est un cas spécial d'application des deux premières techniques. Elle consiste à changer la valeur de l'attribut par des blancs. Elle s'apparente à la technique du retrait mais sans éliminer l'attribut. Même si cette approche ne règle pas le problème de qualité, cette technique a cependant le mérite de maintenir la validité des objets d'information DICOM une fois l'opération de désidentification effectuée.

Au moment d'effectuer une opération de désidentification qui modifie, vide ou enlève un attribut ou une valeur d'attribut DICOM, il est important de prendre en considération si un élément de donnée est obligatoire ou facultatif. La désignation de type, présentée à la section C.1.2.3 du document PS 3.3 du standard DICOM [23], indique si un élément de donnée est requis ou non dans un ensemble de données et, si sa présence est requise, quelle importance prend la valeur qu'il contient. Le tableau 1.1 liste les différents types qui sont utilisés dans DICOM.

Tableau 1.1 Les types d'élément de donnée DICOM

Type	**Description**
1	Requis avec une valeur.
1C	Requis avec une valeur, conditionnel.
2	Requis avec ou sans valeur.
2C	Requis avec ou sans valeur, conditionnel.
3	Optionnel.

Un autre problème qui complique considérablement le processus de désidentification en DICOM est la notion de valeur de représentation (« Value Representation ») des éléments de donnée. En effet, chaque élément de donnée DICOM possède une valeur de représentation (VR) qui décrit le type, le format et la longueur des données que l'élément peut accepter. Le dictionnaire des éléments avec leur VR respective se trouve dans le document PS 3.6 du standard [23]. La liste des VRs disponibles est dans le document PS 3.5 du standard [23]. Toutes

les VRs pour représenter les différents types de donnée allant de l'entité d'application (« Application Entity [AE] ») au texte illimité (« Unlimited Text [UT] ») en passant par le nom des personnes (« Person Name [PN] »), y sont scrupuleusement décrits. La désidentification doit obligatoirement se conformer à cet ensemble de règles pour produire une image DICOM qui soit conforme au standard. Cette standardisation n'aide en rien pour la désidentification des images DICOM.

Tel que mentionné précédemment, une des opérations qu'il est exigé d'effectuer pour désidentifier un objet de données consiste à retirer tous les identificateurs uniques pouvant mener à la réidentification du patient comme, par exemple, son numéro de patient. En plus des identificateurs qui sont propres à l'institution (comme le « Patient ID [0010,0020] »), DICOM introduit également des identificateurs uniques (« Unique identifier [UID] ») qui sont propres au standard.

Les UIDs DICOM offrent la possibilité d'identifier de façon spécifique un large éventail d'objets. Il garantissent le caractère unique des objets à travers plusieurs pays, sites, fournisseurs et types d'équipements. Ils permettent de distinguer les objets DICOM les uns des autres en dehors de toutes considérations sémantiques. Pour donner un exemple, la même valeur de UID ne peut être utilisée pour identifier une étude (« Study Instance UID ») et une série (« Series Instance UID ») à l'intérieur de cette étude ou à l'intérieur d'une étude différente. NEMA est l'organisme responsable de la définition et de l'enregistrement des UIDs DICOM.

Les UIDs sont omniprésents en DICOM et servent à identifier de façon unique toutes sortes d'objets. S'ils ne mènent pas tous nécessairement à l'identification directe du patient, plusieurs UIDs, s'il ne sont pas convenablement désidentifiés, offrent, de par leur unicité, une voie toute privilégiée à l'identification du patient auquel l'image appartient. Leur retrait inconditionnel du fichier DICOM n'est cependant pas une solution, les relations découlant des UIDs constituent une information très précieuse en DICOM que l'on désire sans aucun doute conserver. Si les UIDs ne peuvent rester inchangés pour des raisons évidentes de sécurité de l'information du

patient, il faut modifier leur valeur en respectant le format des UIDs et l'intégrité référentielle qu'ils servent à établir entre les objets.

La détermination des UIDs en DICOM est fondée sur la forme numérique du standard ISO 8824 traitant de l'identification des objets. Chaque UID est composé d'une portion <org root> et d'un <suffix> pour donner la forme suivante :

UID = <org root>.<suffix>

La partie <org root> de l'UID identifie une organisation (c.-à-d. un manufacturier, un laboratoire de recherche, NEMA, etc.) et est composé d'un ensemble de nombres tous séparés par des points tel que décrit dans ISO 8824. La portion <suffix> de l'UID est également composée d'un ensemble de composants numériques séparés par des points et doit être unique dans le champ d'application de <org root>.

La question des UIDs vient ajouter une contrainte supplémentaire à l'opération de désidentification des objets d'information DICOM. En effet, chaque objet disposant d'une panoplie d'identificateurs uniques, leur retrait inconditionnel ne peut avoir lieu sans compromettre considérablement l'intégrité des objets de donnée. Il faudra donc élaborer une stratégie basée sur la pseudonymisation afin de protéger la confidentialité du patient. La pseudonymisation est traitée un peu plus loin dans ce chapitre.

1.2 L'anonymisation

Le retrait des identificateurs directs par la désidentification ne peut garantir à lui seul la confidentialité du patient. Il est possible en prenant les champs restants, en les combinant entre eux et en les comparant avec des données accessibles publiquement, d'identifier précisément un individu. On donne à ces ensembles de champs le nom de quasi-identificateur (« quasi-identifier »). L'anonymisation des données est un processus complexe qui consiste à généraliser la valeur de ces champs quasi-identificateurs afin de rassembler les enregistrements en groupes dans lesquels les enregistrements sont indiscernables les uns par rapport aux autres.

Cette section présente sommairement les modèles k-anonymat et ℓ-diversité qui sont actuellement en usage en anonymisation. On traite du risque d'inférence dans les ensembles de données incrémentaux qu'on illustre à l'aide d'un exemple. Ensuite, le risque d'inférence est présenté de manière plus formelle. On y définit, entre autre, des canaux d'inférence entre classes d'équivalence compatibles ; une classe d'équivalence étant un groupe d'enregistrements anonymisés ne pouvant être distingués entre eux. On présente ensuite une technique d'anonymisation pour les ensembles de données incrémentaux basée sur une métrique de qualité des données. Il est également question d'une manière de prévenir les canaux d'inférence pouvant être appropriée à notre problème [3].

Une technique qui s'avère appropriée à la protection des données biométriques est le modèle k-anonymat (en anglais, « k-anonymity »). Cette technique consiste à faire en sorte qu'il soit impossible de distinguer n'importe quel élément de l'ensemble de données d'au moins $(k-1)$ autres éléments dans l'ensemble [30]. De cette façon, la probabilité d'identifier un individu en particulier dans l'ensemble de données est gardée inférieure à $1/k$.

Un autre modèle intéressant est celui connu sous le nom de ℓ-diversité (« ℓ-diversity ») qui prend en considération qu'un ensemble de données privées contient des attributs sensibles (« sensitive attributes ») qui ne peuvent être modifiés. Un attribut sensible est dévoilé lorsque sa valeur peut être associée à un individu avec certitude. Pour prévenir de telles inférences, le modèle ℓ-diversité suppose que chaque groupe d'enregistrements indicernables contient au moins ℓ valeurs distinctes d'attribut sensible. Ainsi, le risque de voir une valeur d'attribut sensible être dévoilée pour un patient donné est gardé inférieur à $1/\ell$.

Ces techniques fonctionnent bien avec des ensembles de données statiques où les données à anonymiser est disponible en entier. Elles ne s'appliquent pas aussi bien aux ensembles de données dynamiques [3]. Or, comme nous le savons, il existe un réel besoin pour de l'information actualisée et colligée au jour le jour. Par exemple, supposons qu'un hôpital veuille rendre disponible un ensemble de données destiné à la recherche médicale. L'institution devra rendre l'information anonyme de manière à protéger la confidentialité des patients. Cette opération se

résumera à quelque chose d'assez simple pour la version initiale des données, mais les choses se compliqueront sitôt que de nouveaux enregistrements seront ajoutés à l'ensemble de données.

La première approche pourrait être d'anonymiser et de publier les nouveaux enregistrements périodiquement. Les chercheurs pourraient étudier chacune des versions indépendemment ou fusionner les différents ensembles en un seul pour en faciliter l'analyse. Bien que relativement simple, cette approche a le défaut d'offrir des ensembles de données de qualité moindre. Le problème vient du fait que si de petits ensembles de données sont anonymisés indépendemment, les enregistrements doivent subir des modifications plus importantes que s'ils étaient anonymisés tous ensembles.

La seconde approche pourrait être d'anonymiser les données globalement à chaque fois que de nouvelles données s'ajoutent à l'ensemble original. Bien que cela puisse être réalisé avec les techniques existantes, cette approche comporte deux inconvénients majeurs qui ne peuvent être négligés. Le premier inconvénient est que cela représente une quantité importante d'opérations redondantes puisqu'il faut refaire l'anonymization du même ensemble de données à chaque nouvel ajout à l'ensemble de base. Un autre inconvénient, plus grave celui-là, est que même si les ensembles de données publiés sont anonymes de façon individuelle (ils sont anonymes par rapport à k-anonymat et ℓ-diversité), ils sont par contre vulnérables aux attaques par inférence s'ils sont observés collectivement. L'exemple suivant illustre une attaque par inférence.

1.2.1 Exemple d'inférence

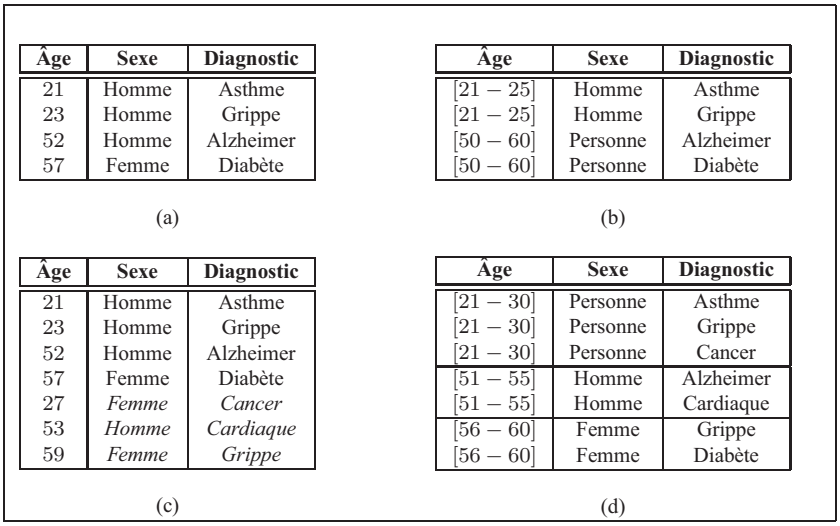
Supposons qu'un hôpital possède un ensemble de données tel que celui présenté à la figure 1.1(a) et qu'il veuille le rendre disponible à la recherche. Il décide d'apporter les modifications à l'ensemble de données de manière à ce qu'il soit impossible de distinguer chacun des items de l'ensemble d'au moins un autre item. L'ensemble est considéré 2-diverse dans ce cas et le résultat est présenté à la figure 1.1(b). Par exemple, si un attaquant savait que l'information de Tom, un jeune homme de 21 ans était dans l'ensemble, il ne pourrait prédire le diagnostic de la maladie dont il souffre avec une probabilité supérieure à $1/\ell$ où $\ell = 2$. Dans

ce cas, tout ce qu'il pourrait apprendre est que Tom souffre d'asthme ou de la grippe. Plus tard, l'ensemble de données est mis à jour et de nouveaux items sont ajoutés avec comme résultat la table présentée à la figure 1.1(c). Les nouvelles données sont présentées en italique dans la table. L'hôpital publie donc une nouvelle version de sa table 2-diverse et cette version est présentée à la figure 1.1(d). La vie privée de Tom est toujours protégée dans cette nouvelle version, mais ce n'est pas le cas pour les autres. Voici deux exemples d'attaque par inférence tirés de Byun et al. [3] :

Exemple 1 : le premier exemple d'inférence suppose que l'attaquant sait que Alice, qui est dans la fin vingtaine, vient d'être admise à l'hôpital. Il est certain que Alice ne se trouve pas dans le premier ensemble de données [figure 1.1(b)], mais qu'elle est dans le nouvel ensemble anonymisé [figure 1.1(d)]. À l'aide du nouvel ensemble de données, il peut apprendre que Alice souffre d'une des trois maladies asthme, grippe, cancer. Cependant, par inférence, il peut facilement déterminer que Alice ne souffre ni d'asthme ni de la grippe. Il est assuré que Alice souffre bien du cancer.

Exemple 2 : l'attaquant sait que Bob a 52 ans et qu'il se fait traiter dans cet hôpital depuis longtemps. Il est donc certain que l'information de Bob se trouve dans les deux versions de l'ensemble anonymisé. Il étudie le premier ensemble pour apprendre que Bob souffre d'alzheimer ou de diabète. En observant le deuxième ensemble de données, la personne malveillante découvre que Bob est soit cardiaque ou alzheimer. Il en conclut donc que Bob est atteint d'alzheimer. Noter que trois autres items dans l'ensemble de données sont vulnérables à de telles attaques par inférence.

L'idée maîtresse derrière l'approche de Byun et al. dans son article [3] est qu'on peut anonymiser efficacement un ensemble de données en insérant des items dans un ensemble déjà anonymisé. Cela implique que les nouveaux items ainsi que ceux qui se trouvent déjà dans l'ensemble anonymisés doivent être modifiés pour satisfaire aux exigences de l'anonymisation (c.-à-d. k-anonymat et ℓ-diversité). Cela doit se faire avec le souci de conserver une certaine qualité de l'information (« data quality ») et sans ouvrir la porte à des inférences indésirables.



Âge	Sexe	Diagnostic
21	Homme	Asthme
23	Homme	Grippe
52	Homme	Alzheimer
57	Femme	Diabète

(a)

Âge	Sexe	Diagnostic
[21 – 25]	Homme	Asthme
[21 – 25]	Homme	Grippe
[50 – 60]	Personne	Alzheimer
[50 – 60]	Personne	Diabète

(b)

Âge	Sexe	Diagnostic
21	Homme	Asthme
23	Homme	Grippe
52	Homme	Alzheimer
57	Femme	Diabète
27	*Femme*	*Cancer*
53	*Homme*	*Cardiaque*
59	*Femme*	*Grippe*

(c)

Âge	Sexe	Diagnostic
[21 – 30]	Personne	Asthme
[21 – 30]	Personne	Grippe
[21 – 30]	Personne	Cancer
[51 – 55]	Homme	Alzheimer
[51 – 55]	Homme	Cardiaque
[56 – 60]	Femme	Grippe
[56 – 60]	Femme	Diabète

(d)

Figure 1.1 (a) Ensemble de données initial - (b) ensemble de données initial anonymisé en 2-diverse - (c) nouvel ensemble de données mis à jour - (d) nouvel ensemble anonymisé en 2-diverse.
Tiré de Byun et al. (2006, p. 2)

Le modèle k-anonymat tient compte d'une organisation des données en table. Chaque table étant composée de lignes d'information comportant des attributs dont les valeurs proviennent de différents domaines. La première opération à réaliser consiste à retirer tous les attributs tels que le nom ou le numéro de patient. Même si cela constitue une opération importante et essentielle à l'anonymisation des données, l'effacement des identificateurs directs n'est malheureusement pas suffisant pour garantir au patient la confidentialité. En effet, les attributs restants même si, pris individuellement, ne peuvent mener à l'identification du patient, peuvent, en les combinant, mener au dévoilement de l'identité du patient. C'est une constatation qu'a fait Sweeney dans son article [30] où elle déclare que 87% de la population états-unienne peut être identifée en combinant la date d'anniversaire, le sexe et le code postal (« zip code »). Cette combinaison d'attributs, appelée « quasi-identificateur », permet à un adversaire d'établir clairement l'identité des gens.

Le quasi-identificateur d'une table T, dénoté Q_T, est un ensemble d'attributs pris dans T qui, utilisés conjointement, peuvent mener à l'identification d'un individu avec une probabilité égale à 1. L'objectif principal de la méthode k-anonymat est de transformer une table de

manière à ce que personne ne puisse établir de lien entre la table T et un individu avec une probalité inférieure à $1/k$.

On dit qu'une table T est k-anonyme en rapport avec un quasi-identificateur Q_T si et seulement si, pour tout enregistrement r dans T, il existe au moins $(k-1)$ autres enregistrements dans T qui ne peuvent être distingués de r par rapport à Q_T.

Par l'application du modèle k-anonymat, on s'assure que même si un adversaire sait que l'information d'un individu se trouve à l'intérieur d'une table T et qu'il connaît la valeur des attributs particuliers lui permettant d'identifier précisément un individu, il ne peut savoir qu'un enregistrement de T appartient à cet individu avec une probabilité supérieure à $1/k$.

Le modèle k-anonymat est appliqué en généralisant les valeurs numériques d'un domaine par des intervalles (p. ex., $[12-19]$) et les valeurs catégoriques par un ensemble (p. ex., $\{homme, femme\}$) ou une valeur représentant cet ensemble (p. ex., $Personne$). Chaque groupe d'enregistrements d'un ensemble de données qu'on ne peut distinguer les uns des autres s'appelle une classe d'équivalence (« equivalence class »). Dans un ensemble qui est considéré comme 2-anonyme les classes d'équivalence possèdent au moins 2 éléments chacun.

Les ensembles de données possèdent des attributs qu'on dit « sensibles » qui ne font pas partie du quasi-identificateur, mais qui jouent leur rôle dans la protection de la confidentialité des patients. Par exemple, dans l'ensemble de données de la figure 1.1(c), l'attribut $Diagnostic$ est considéré sensible. Le modèle k-anonyme ne tient pas compte de cette menace où il est possible d'inférer certaines valeurs à des attributs sans qu'il soit nécessaire d'avoir à identifier à qui appartiennent les enregistrements. Par exemple, prenons une classe d'équivalence dont la valeur d'attribut sensible est la même pour tous les membres de la classe. Bien qu'aucun de ces enregistrements ne puissent être identifiés précisément, il n'en demeure pas moins que la valeur de leur attribut sensible puisse être déterminée avec une probabilité de 1. La méthode ℓ-diversité vient pallier ce défaut du modèle k-anonymat [19]. Une table T est dite ℓ-diverse si les enregistrements dans chaque classe d'équivalence contient au moins ℓ valeurs d'attributs sensibles. Là encore, comme le modèle ℓ-diversité garantit que chaque classe d'équivalence

contient au moins ℓ valeurs distinctes d'attributs sensibles, le risque d'identifier à qui appartient un attribut est gardé inférieur à $1/\ell$.

1.2.2 Les canaux d'inférence

Supposons une table T avec un quasi-identificateur Q_T et un attribut sensible S_T. Seule la version ℓ-diverse de T, dénoté $\hat{T}$, est rendue disponible au public. De nouveaux enregistrements sont ajoutés périodiquement à T et $\hat{T}$ est mise à jour et publiée régulièrement pour tenir compte des changements apportés à T. C'est la mise à jour incrémentale des données. Les utilisateurs (y compris des adversaires) ont en leur possession une série de tables $T_0, T_1, \ldots, T_n$ anonymisées à différents moments dans le temps où $|\hat{T}_i| < |\hat{T}_j|$ pour $i < j$. Les tables étant chacune ℓ-diverse, la probabilité de déterminer qu'un enregistrement est associé spécifiquement à un individu est maintenu sous $1/\ell$. C'est lorsqu'on compare les changements qui existent entre les différentes tables anonymisées que l'on peut augmenter cette probabilité, on parle alors qu'il existe un canal d'inférence entre les tables. On dit qu'il existe un canal d'inférence entre deux tables $\hat{T}_i$ et $\hat{T}_j$ tirées de la même table privée T si, en comparant ces deux tables, on augmente la probabilité de dévoiler l'identité d'un individu avec une probabilité supérieur à $1/\ell$. Un canal d'inférence entre une table $\hat{T}_i$ et $\hat{T}_j$ est dénoté $\hat{T}_i \rightleftharpoons \hat{T}_j$. Il est de la responsabilité du fournisseur de données (« data provider ») de garantir qu'il n'existe aucun canal d'inférence entre la dernière table publiée et toutes les versions précédentes de la table.

L'adversaire, tel que nous le conçevons, possède un ensemble de tables $\hat{T}_0, \hat{T}_1, \ldots, \hat{T}_n$, où $\hat{T}_i$ est une table publiée au temps i. Il est raisonable de croire que l'attaquant connaît la liste des individus inclus dans chaque table ainsi que leur quasi-identificateur respectif. Cependant, comme elles respectent le modèle ℓ-diversité, si les tables sont prises individuellement, la probabilité qu'un individu avec un certain quasi-identificateur possède un attribut sensible est limité à $1/\ell$; $P(S_t = s|Q_T = q) \leq 1/\ell$. Le but de l'attaquant est d'augenter cette probabilité en comparant les tables entre elles.

Pour tenter dévoiler les attributs sensibles de Tom dont le quasi-identificateur est q, l'adversaire dispose de deux types de comparaison : 1) une comparaison où la table $\hat{T}_i$ ne contient pas

Tom et où la table $\hat{T}_j$ le contient [cette situation est représentée par $\delta(\neg\hat{T}_i, \hat{T}_j)$], et 2) une comparaison où les deux tables $\hat{T}_i$ et $\hat{T}_j$ contiennent Tom [représentée par $\delta(\hat{T}_i, \hat{T}_j)$]. Dans les deux cas $i < j$. L'attaquant doit trouver, dans T_i, une classe d'équivalence e_i où $q \subseteq e_i[Q_T]$. Dans le cas de $\delta(\neg\hat{T}_i, \hat{T}_j)$, l'adversaire sait pertinemment que T_i ne contient pas Tom ce qui peut lui permettre d'éliminer ces enregistrements de T_j. Dans le cas de $\delta(\hat{T}_i, \hat{T}_j)$, l'adversaire sait qu'un des enregistrements dans e_i est celui de Tom et, bien qu'il lui soit impossible de déterminer précisément quels sont ses attributs sensibles de Tom (parce que e_i contient un nombre ℓ d'enregistrements indiscernables soit par leur quasi-identificateur ou leurs valeurs d'attributs sensibles), cela peut représenter une information précieuse lorsqu'on examine la table $\hat{T}_j$.

Une fois qu'il a obtenu e_i, l'adversaire doit trouver les classes d'équivalence qui pourraient contenir Tom à l'intérieur de $\hat{T}_j$. Cette opération consiste à rechercher toutes les classes d'équivalence dites « compatibles » à e_i.

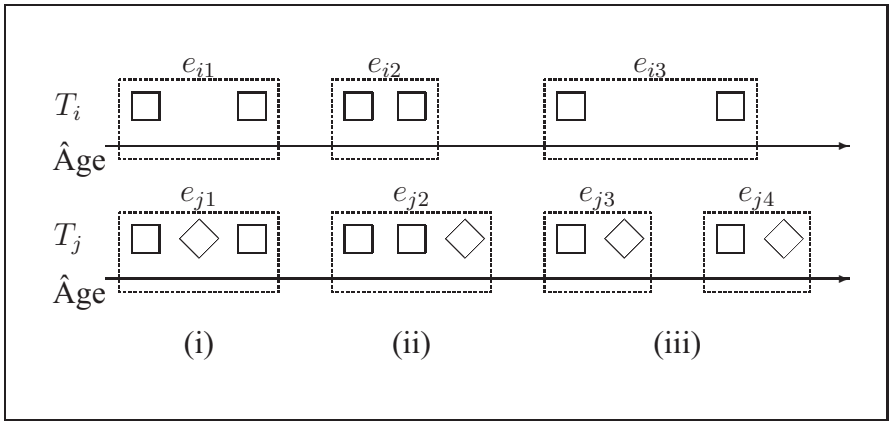


Figure 1.2 Classes d'équivalence compatibles.
Tiré de Byun et al. (2006, p. 7)

Byun et al. présentent la compatibilité entre les classes d'équivalence à l'aide d'un exemple [3]. La compatiblité entre deux classes d'équivalence implique qu'il y a des enregistrements présents dans les 2 classes d'équivalence. La figure 1.2 illustre ce qu'est la compatibilité entre les classes d'équivalence. Dans cet exemple, les deux tables T_i et T_j sont représentées par rapport au quasi-identificateur « Âge ». Nous posons l'hypothèse que les éléments sont dans l'ordre total (« Total Order of the elements »). La table T_i contient 6 enregistrements représentés

par des « □ » et sa version 2-diverse $\hat{T}i$ consiste en 3 classes d'équivalence e_{i1}, e_{i2} et e_{i3}. La table T_j contient quatre enregistrements supplémentaires représentés par des « ◊ » et dont la version 2-diverse $\hat{T}_j$ est séparée en 4 classes d'équivalence distinctes e_{j1}, e_{j2}, e_{j3} et e_{j4}. En considérant les tables $\hat{T}_i$ et $\hat{T}_j$, nous pouvons tirer les conclusions suivantes quant à leur compatiblité :

a. e_{j1} est identique à e_{i1}, c'est à dire que les valeurs de Q[1] d'une classe d'équivalence à l'autre sont égales [figure 1.2 (i)], on le dénote par $e_{i1} \cong e_{j1}$.

b. e_{j2} inclut e_{i2}, c.-à-d. que les valeurs de Q de e_{i2} sont inclues dans celles de e_{j2}, on dit que les valeurs de Q de e_{j2} sont plus généralisées que celles de e_{i2} [figure 1.2 (ii)] et on le dénote par $e_{i2} \prec e_{j2}$.

c. e_{j3} chevauche e_{i3}, c'est à dire que les valeurs de Q se recoupent pour engendrer de nouvelles classes d'équivalence [figure 1.2 (iii)]. On le dénote $e_{i3} \precdot\succ e_{j3}$ et $e_{i3} \precdot\succ e_{j4}$.

Il existe donc trois cas de compatibilité possibles entre deux classes d'équivalence, examinons le risque d'inférence pour les deux situations $\delta(\neg\hat{T}_i, \hat{T}_j)$ et $\delta(\hat{T}_i, \hat{T}_j)$ vues précédemment. En supposant que $e_i[S]$ et $e_j[S]$ soient les projections qui conserve les enregistrements dupliqués[2] des valeurs d'attributs sensibles dans les classes d'équivalence e_i et e_j respectivement.

Le tableau de la figure 1.3 résume les combinaisons pouvant mener à des canaux d'inférence. Voici quelques explications concernant chacune des parties du tableau :

	$e_i \cong e_j$	$e_i \prec e_j$	$e_i \precdot\succ e_{j1}$ et $e_i \precdot\succ e_{j2}$
$\delta(\neg\hat{T}_i, \hat{T}_j)$	$e_j[S] \setminus e_i[S]$	$e_j[S] \setminus e_i[S]$	$((e_{j1}[S] \cup e_{j2}[S]) \setminus e_i[S]) \cap e_{jk}[S], k = 1, 2$
$\delta(\hat{T}_i, \hat{T}_j)$	$\emptyset$	$\emptyset$	$e_i[S] \cap e_{jk}[S], k = 1, 2$

Figure 1.3 Sommaire des ensembles permettant l'inférence.
Tiré de Byun et al. (2006, p. 8)

a. $e_i \cong e_j$ ou $e_i \prec e_j$: dans ces cas, l'adversaire sait que tous les enregistrements dans e_i sont également dans e_j. Disons que $e_i[S]$ et $e_j[S]$ représentent les projections de valeurs

[1] Supposons $Q = \{q_1, q_2 \ldots, q_m\}$ un ensemble d'attributs formant un quasi-identificateur.
[2] Par conséquent, toutes les opérations $\cap$, $\cup$ et $\setminus$ sont effectuées sur des multi-ensembles [1]

d'attributs sensibles dans e_i et e_j respectivement. Les valeurs d'attributs sensibles pour un individu en particulier (Tom) sont représentées par s_T.

(a) $\delta(\neg\hat{T}_i, \hat{T}_j)$: l'adversaire sait que les valeurs d'attributs sensibles de Tom ne sont pas dans $e_i[S]$, mais sont dans $e_j[S]$. En d'autres mots, $s_T \notin e_i[S]$ et $s_T \in e_j[S]$. Comme il sait que toutes les valeurs dans $e_i[S]$ doivent également se retrouver dans $e_j[S]$, il peut conclure que $s_T \in (e_j[S] \setminus e_i[S])$. Par conséquent, l'attaquant peut inférer s_T avec une probabilité supérieure à $1/\ell$ si $(e_j[S] \setminus e_i[S])$ contient moins que ℓ valeurs distinctes.

(b) $\delta(\hat{T}_i, \hat{T}_j)$: dans ce cas $s_T \in e_i[S]$ et $s_T \in e_j[S]$. Cependant, comme les deux ensembles sont ℓ-diverse, l'adversaire ne peut obtenir plus d'information sur s_T.

b. $e_i \nsim e_{j1}$ et $e_i \nsim e_{j2}$[3] : dans ce cas, l'adversaire sait que les enregistrements dans e_i peuvent se trouver dans e_{j1} ou e_{j2}. Connaissant le quasi-identificateur de Tom (q), il peut facilement déterminer lequel des deux e_{j1} et e_{j2} contient l'enregistrement de Tom. Supposons que e_{j1} contient l'enregistrement de Tom. Disons que $e_i[S]$, $e_{j1}[S]$ et $e_{j2}[S]$ sont les projections de valeurs d'attributs sensibles dans e_i, e_{j1} et e_{j2} respectivement.

(a) $\delta(\neg\hat{T}_i, \hat{T}_j)$: l'attaquant sait que $s_T \notin e_i[S]$, $s_T \notin e_{j2}[S]$ et $s_T \in e_{j1}[S]$. Comme tous les enregistrements de e_i ne se sont pas nécessairement inclus dans e_{j1}, il ne peut conclure à $s_T \in (e_{j1}[S] \setminus e_i[S])$ comme dans le cas précédent. Il peut néanmoins prétendre que l'enregistrement de Tom est dans $e_{j1} \cup e_{j2}$, mais ne se trouve pas dans e_i. Il en déduit que $s_T \in (e_{j1}[S] \cup e_{j2}[S]) \setminus e_i[S]$. Comme l'enregistrement de Tom doit être dans e_{j1}, l'adversaire conclut que $s_T \in ((e_{j1}[S] \cup e_{j2}[S]) \setminus e_i[S]) \cap e_{j1}[S]$. Par conséquent, si l'ensemble qui en résulte ne contient pas au moins ℓ valeurs d'attributs sensibles distincts, l'adversaire peut inférer s_T avec une probabilité qui est supérieure à $1/\ell$.

(b) $\delta(\hat{T}_i, \hat{T}_j)$: l'adversaire sait que les attributs sensibles de Tom apparaissent dans $e_i[S]$ et $e_{j1}[S]$. À partir de ce fait, nous pouvons conclure que $s_T \in (e_i[S] \cap e_{j1}[S])$.

[3] Il est possible que $\hat{T}_j$ contienne plus que deux classes d'équivalence compatibles à e_i. Cependant, il est question de seulement deux classes pour des raisons de simplicité.

L'attaquant peut donc inférer s_T avec une probabilité supérieur à $1/\ell$ si $(e_i[S] \cap e_{j1}[S])$ contient moins de ℓ valeurs différente.

L'anonymisation des données est considérée comme un cas spécial d'optimisation où l'information doit subir le moins de modification afin d'assurer la meilleure qualité de données possible, alors que les constraintes d'anonymisation (en termes de k-anonymat et de ℓ-diversité) doivent être maintenues au maximum. Sans s'attarder sur les détails de l'algorithme de ℓ-diversité de Byun et al., disons qu'il se divise en 2 étapes dont la première consiste en le partitionnement des enregistrements selon un espace d-dimensionnel où d est le nombre d'attributs dans le quasi-identificateur. Les enregistrements sont ensuite modifiés de manière à partager tous la même valeur de quasi-identificateur dans une même classe d'équivalence. La solution de Byun et al. pour la mise à jour incrémentale des ensembles de données anonymisés tient compte d'une métrique de la qualité de l'information qui calcule la distorsion des données basée sur l'expansion de chaque classe d'équivalence.

La figure 1.4 (i) présente un exemple d'un ensemble de données dont les enregistrements sont représentés par des points sur un plan à 2 dimensions en fonction du quasi-identificateur $\{Age, Poids\}$. L'ensemble comporte 2 classes d'équivalence 3-diverse e_1 et e_2 représentées à l'aide de 2 régions traçée en ligne pointillée. Tous les enregistrements d'une même classe d'équivalence ont la même valeur pour les attributs du quasi-identificateurs. Par exemple, tous les enregistrements contenus dans e_1 partage la même valeur de quasi-identificateur $\langle [a_1 - a_2], [w_1 - w_2] \rangle$. La distorsion des données (« data distortion ») peut être évaluée en mesurant la taille des régions couverte par ces classes d'équivalence. Basé sur cette idée, on définit une nouvelle métrique de la qualité de l'information (« Information Loss metric ») (IL).

La mérique IL se définit comme suit :

Disons $e = \{r_1, \ldots, r_n\}$, une classe d'équivalence où $Q_T = \{a_1, \ldots, a_m\}$ est le quasi-identificateur. La quantité de distorsion en généralisant e, dénotée $IL(e)$, est définie par :

$IL(e) = |e| \times \sum_{j=1,\ldots,m} \frac{|G_j|}{|D_j|}$

où $|e|$ est le nombre d'enregistrements dans e, $|D_j|$ représente la taille du domaine de l'attribut a_j. $|G_j|$ représente l'ampleur de la généralisation pour l'attribut a_i (c.-à-d. la longueur de l'intervalle contenant toutes les valeurs de l'attribut a_j dans e).

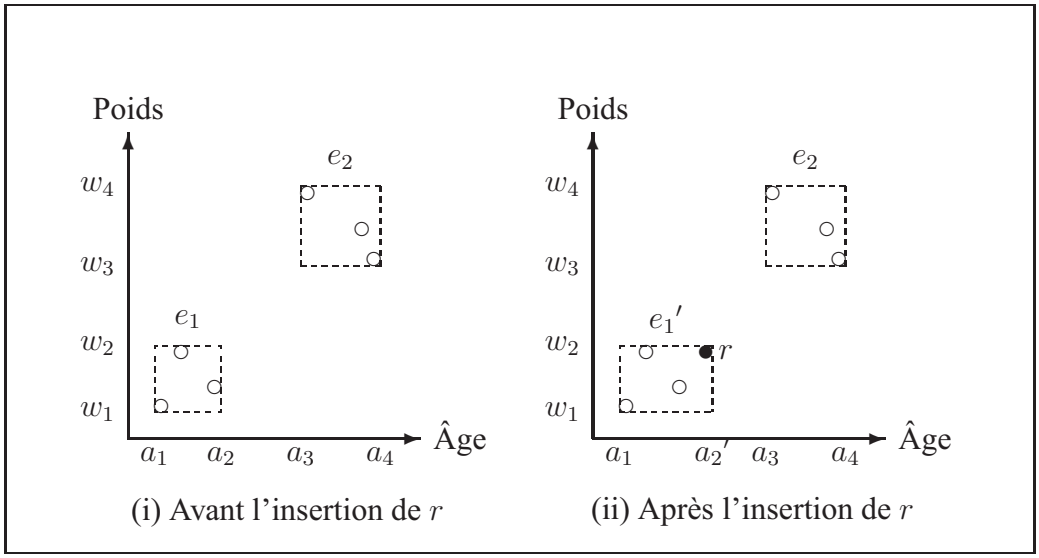


Figure 1.4 Distorsion des données causée par la généralisation.
Tiré de Byun et al. (2006, p.10)

L'objectif est de produire un nouvel ensemble de données à jour en insérant de nouveaux enregistrements dans un ensemble de données déjà anonymisé. Supposons qu'il existe une table anonymisée $\hat{T}$ qui est une version ℓ-diverse d'une table privée T. Suposons que, plus tard dans le temps, un nouvel ensemble d'enregistrements $R = r_1, \ldots, r_1$ sont insérés dans T. On dénote cette nouvelle version de T par T'. On peut penser intuitivement qu'une nouvelle version ℓ-diverse $\hat{T}'$ puisse être générée en insérant un à un les enregistrements de R dans $\hat{T}$. Pour cela, il faut respecter les exigences suivantes : 1) $\hat{T}'$ doit être ℓ-diverse, 2) la qualité des données doit être conservée autant que possible et 3) cela ne doit pas engendrer de nouveaux canaux d'inférence.

On décrit un algorithme d'insertion qui rencontre les deux premières exigences. L'idée consiste à insérer le nouvel enregistrement dans la classe d'équivalence la plus « rapprochée » de façon à réduire le plus possible les changements dûs à la généralisation. Pour illustrer ce concept, retournons à la figure 1.4 où (i) représente six enregistrements partitionnés en deux classes

d'équivalence 3-diverse et (ii) montre les mêmes ensembles révisés une fois l'insertion d'un nouvel enregistrement r complété. L'enregistrement r est inséré dans e_1 donnant comme résultat la nouvelle classe d'équivalence $e_1{}'$. La perte d'information du nouvel ensemble de données est augmentée de $IL(e_i{}') - IL(e_i)$. Cependant, si r avait été inséré dans e_2 la perte d'information aurait été supérieure. Basé sur cette idée, on élabore un algorithme d'insertion garantissant une qualité optimale des données.

a. **(Ajout)** Si un groupe d'enregistrements dans R forme une classe d'équivalence ℓ-diverse qui ne chevauche aucune autre des classes d'équivalence existantes, alors ajouter ce groupe d'enregistrements à $\hat{T}$ en tant que nouvelle classe d'équivalence.

b. **(Insertion)** Les enregistrements qui ne peuvent être ajoutés comme des classes d'équivalence à part entière doivent être insérés dans des classes d'équivalence existantes. De manière à minimiser la distorsion des données dans $\hat{T}'$, chaque enregistrement r_i doit être inséré dans une classe d'équivalence e_j dans $\hat{T}$ qui minimise la différence $IL(e_j \cup \{r_i\}) - IL(e_j)$.

c. **(Division)** Après avoir ajouter ou insérer tous les enregistrements de R dans $\hat{T}$, il est possible que le nombre des valeurs distinctes dans quelques classes d'équivalence dépasse 2ℓ. Si une telle classe d'équivalence existe, alors on peut la diviser en deux classes d'équivalence pour une meilleure qualité des données. Il se peut qu'il ne soit pas possible de diviser une classe d'équivalence dépendemment de la distribution des enregistrements dans la classe d'équivalence.

À aucun moment dans cet algorithme il n'a été question des canaux d'inférence qui peuvent être introduits par l'insertion de nouveaux enregistrements dans un ensemble de données déjà anonymisé ni de la manière de prévenir ces canaux d'inférence. Il faut pourtant s'assurer que l'algorithme précédent produise des ensembles qui sont ℓ-diverse pour les trois opérations d'ajout, d'insertion et de division des classes d'équivalence.

Disons d'abord que l'opération d'ajout n'introduit aucun canaux d'inférence étant donné qu'elle ne fait qu'ajouter de nouvelles classe d'équivalence qui ne présentent aucune compatibi-

lité avec les classes existantes. L'opération d'insertion, quant à elle, peut introduire des canaux d'inférence lorsque de nouveaux enregistrements sont ajoutés et que les classes d'équivalence qui en résultent contiennent moins que ℓ instance de valeur d'attributs sensibles ($e_j[S] \setminus e_i[S]$). Tel que mentionné précédemment, les classes d'équivalence deviennent vulnérables aux attaques à travers $\delta(\neg\hat{T}_i, \hat{T}_j)$. Cette sorte d'insertion ne peut être permise. Pour pallier ce problème on doit modifier l'opération d'insertion comme suit : Durant la phase d'insertion, plutôt que d'insérer les enregistrements dans les classes d'équivalence, on insère les enregistrements dans une liste d'attente de classes d'équivalence. Au fur et à mesure que les enregistrements sont ajoutés à la liste d'attente, ils finissent par former des classes d'équivalence ℓ-diverse qui peuvent être insérées dans les ensembles de données anonymisés. Comme de nouveaux enregistrements sont continuellement ajoutés à la liste d'attente, le temps d'attente est relativement court pour la plupart des enregistrements.

Deux canaux d'inférence peuvent être introduits quand une classe d'équivalence e_i est divisée en deux classes e_{j1} et e_{j2}. La première possibilité est $((e_{j1}[S] \cup e_{j2}[S]) \setminus e_i[S]) \cap e_{jk}[S], k = 1, 2$. Si l'ensemble résultant n'est pas ℓ-diverse, les enregistrements sont vulnérables aux attaques à travers $\delta(\neg\hat{T}_i, \hat{T}_j)$. La condition doit être vérifiée avant de diviser e_i. La deuxième possibilité est $e_i[S] \cap e_{jk}[S], k = 1, 2$. Cela implique que s'il n'y a pas suffisamment de valeurs d'attribut sensibles se chevauchant entre la classe d'équivalence et les classes d'équivalence divisées, alors les classes d'équivalence divisées deviennent vulnérables aux attaques par inférence à travers $\delta(\hat{T}_i, \hat{T}_j)$. Donc, à moins que cette condition ne soit remplie, e_i ne doit pas être divisée. La question épineuse dans ce cas est que les canaux d'inférence peuvent exister avec toutes les classes d'équivalence publiée antérieurement. Par exemple, s'il existe une classe d'équivalence $e_i{}'$ qui a été publiée avant e_i, alors la condition de division doit être satisfaite avec $e_i{}'$ également. Cela signifie que le système doit maintenir l'information sur toutes les versions précédentes. De manière à faciliter ceci, on stocke toute l'information au sujet des états précédents pour chaque classe d'équivalence. Cela ne représente pas une quantité si importante d'espace de stockage puisque l'on ne conserve que l'information au sujet des attributs sensibles (pas tous

les enregistrements). On purge également cette information lorsque les classes d'équivalence deviennent incompatibles à la classe d'équivalence courante.

Comme on peut le constater, les mécanismes de prévention des canaux d'inférence peuvent contribuer à atténuer la qualité des données anonymisées, mais il s'agit du prix à payer pour bénéficier d'une meilleure confidentialité.

1.3 La pseudonymisation

Le dossier médical électronique du patient (l'enregistrement patient), quand il est utilisé dans un contexte clinique où des soins sont promulgués au patient, devrait contenir toute l'information nécessaire à l'identification de ce dernier. Cependant, l'enregistrement patient possède aussi les usages alternatifs suivants : il peut servir dans les projet de recherche sur des maladies spécifiques, en recherche épidémiologique, en recherche sur les soins de santé, lors de l'évaluation de la qualité des traitements et en recherche en économie de la santé. Ces utilisations secondaires comportent les aspects typiques suivants : 1) l'enregistrement patient quitte le contexte intitutionnel où il bénéficie généralement de la protection conférée par le secret professionnel et 2) l'identité du patient n'a plus vraiment sa raison d'être dans l'enregistrement. Dans ce cas spécifique, toute l'information nominative pouvant mener directement à l'identification de l'individu doit être retirée de l'enregistrement patient. Certaines dispositions spéciales doivent être prises pour empêcher la réidentification du patient soit par déduction ou par quelques manières que ce soit.

Malheureusement, ce n'est pas toujours aussi simple. Dans plusieurs situations où l'on fait un usage secondaire de l'enregistrement patient, il peut être souhaité, voire même nécessaire, de retracer l'identité du répondant. Cela peut être le cas lorsque les données proviennent de différentes sources ou lorsque l'information devient disponible à des moments précis dans le temps ou qu'elle doit être mise à jour à des intervalles plus ou moins rapprochés comme ce que nous cherchons à réaliser dans ce mémoire. On peut même imaginer des situations où il serait nécessaire de conserver un lien à l'identité du patient pour l'informer du résultat de la recherche effectuée à son sujet (par exemple, s'il présente des dispositions génétiques spéciales). On peut

également vouloir constituer un lot d'individus prédisposés à une étude épidémiologique ou clinique spécifique. L'utilisation de pseudonymes est la solution à ce type de problème.

La pseudonymisation sert à remplaçer un vrai identificateur tel que le nom ou le numéro d'identification du patient par un pseudonyme qui est unique à l'individu, mais qui n'a aucun rapport avec cet individu. Un pseudonyme ne peut mener à l'identification du patient puisque le lien qui l'unit au patient est conservé de façon sécuritaire et séparément de l'information à traiter.

Dans leur ouvrage sur la manière d'assurer la confidentialité des données de recherche, Boruch et Cecil [2] expliquent qu'il existe des études transversales (« cross-sectional studies ») et des recherches longitudinales (« longitudinal researches »). Les études transversales permettent d'obtenir un échantillon d'information à un ou plusieurs moments dans le temps, mais aucune tentative n'est faite pour lier ces échantillons entre eux. On prend par exemple le bureau de recensement qui investigue à tous les 10 ans sur le nombre de personnes que comporte un foyer, mais qui ne fait jamais de liens sur le nombre d'enfants qu'il y a à l'intérieur de chaque maison d'une fois à l'autre. Les études longitudinales, quant à elles, permettent d'obtenir de l'information sur les mêmes répondants à intervalle régulier. On essaie d'obtenir un cheminement logique en faisant des liens avec l'information obtenue précédemment. On donne l'exemple de l'étude correlationnelle dans le développement émotionnel des enfants désavantagés en établissant la relation entre le comportement précédent de l'enfant et le comportement présent. Ce mémoire tient compte du fait qu'on veuille suivre un patient sur une période couvrant toute la durée de son traitement ; il s'agit donc d'une étude longitudinale et cela nécessite l'utilisation d'un pseudonyme.

La pseudonymisation est essentiellement de deux types : réversible ou à sens unique. La pseudonymisation réversible permet la réidentification des individus alors que la pseudonymisation à sens unique la rend impossible. Dans les deux cas, les pseudonymes permettent la liaison des enregistrements entre le secteur cliniques et celui de la recherche. L'utilisation de pseudonymes réversibles exige que la réidentification dépende d'une clé secrète et que l'opération de pseudonymisation soit effectuée par un tier de confiance (« trusted third party »). De plus, la

réidentification, si elle est possible et si elle est nécessaire, ne peut se faire qu'avec le consentement explicite du patient.

La figure 1.5 illustre un exemple de pseudonymisation utilisant une fonction de hachage à sens unique présenté par K. Pommerening [28]. Dans cette figure, MDAT représente les données médicales du patient, IDAT est l'information d'identification du patient, PID est le numéro du patient et PSN représente son pseudonyme.

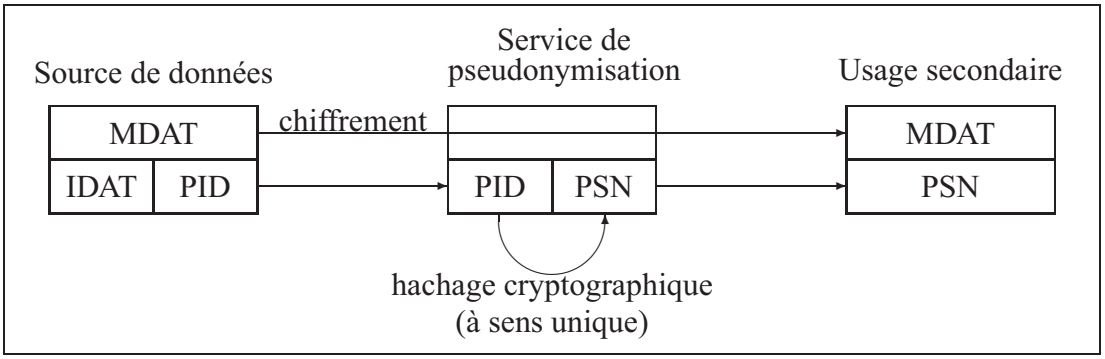


Figure 1.5 Pseudonymisation à sens unique.
Tiré de Pommerening (2004, p. 3)

Ici, on peut imaginer que les données proviennent de diverses sources ou bien qu'elles sont disponibles à des moments différents dans le temps. Une condition préalable essentielle est qu'il existe une façon unique d'identifier le patient qui est partagée entre toutes les sources de données. Celle-ci est le « patient ID » ou PID. Le pseudonyme est généré à partir d'une fonction à sens unique (ce pourrait être du hachage) qui ne permet aucun retour en arrière sur l'information originale. Ce service devrait être fourni par un tier de confiance.

Une caractéristique typique de ce service est l'utilisation d'un algorithme de chiffrement asymmétrique : la source des données chiffre l'information médicale avec la clé publique de l'utilisateur secondaire et envoie le PID (pas l'information d'identification) avec les données médicales chiffrées au service de pseudonymisation qui chiffre le PID pour générer le pseudonyme avant d'envoyer les données médicales accompagnée du pseudonyme. Noter que le service de pseudonymisation ne peut déchiffrer les données médicales, seul l'utilisateur secondaire peut

les déchiffrer grâce à sa clé privée. Cependant, l'utilisateur secondaire ne peut déchiffrer le pseudonyme et ainsi dévoiler le PID.

1.4 Conclusion

Dans ce chapitre nous avons étudié en détails les opérations impliquées dans le processus d'anonymisation qui sont la désidentification, l'anonymisation et la pseudonymisation. La désidentification est l'opération qui consiste à retirer toute l'information nominative pouvant mener à l'identification du patient. Les opérations qu'il est possible de réaliser sur un attribut pour le désidentifier sont de l'effacer ou de modifier sa valeur. L'effacement d'un attribut signifie son retrait définitif de l'instance anonymisée. Les modifications qu'on effectue sur un attribut pour le désidentifier sont de le remplir de caractères blancs ou de changer sa valeur par une constante. Les attributs de type SQ (« Sequence of Items ») utilisent la récursivité pour se désidentifier. Les attribut de type UID doivent faire l'objet d'un processus de désidentification particulier tenant compte du fait que leur valeur revêt une importance toute particulière en ce qui concerne l'intégrité référentielle qui existe entre les différents objets DICOM.

Une deuxième technique, complémentaire à la première, est l'anonymisation. L'anonymisation protège une image DICOM par la généralisation des données qu'elle contient. La généralisation des données est une forme de modification que peut subir un attribut d'une instance DICOM. Deux modèles sont communément employés et ont fait leur preuve sur des ensembles de données statiques : il s'agit des modèle k-anonymat et ℓ-diversité. Le premier modèle stipule qu'un élément ne devrait pas se distinguer d'au moins $k-1$ autres éléments dans l'ensemble de données par rapport à son quasi-identificateur de manière à garder la probabilité de réidentification de l'information inférieure à $1/k$. Un quasi-identificateur étant un ensemble d'attributs considérés inoffensifs lorqu'ils sont pris individuellement, mais qui constituent une menace à la protection de la confidentialité du patient lorsqu'ils sont combinés entre eux. Le modèle ℓ-diversité vient compléter le premier modèle en stipulant qu'aucun élément de donnée ne devrait se distinguer de moins de $\ell-1$ autres éléments par rapport à ses attributs sensibles de manière à garder la probabilité de réidentification inférieure à $1/\ell$. Comme nous venons de le mentionner,

ces modèles sont prévus s'appliquer parfaitement aux ensembles de données statiques, mais il en est tout autrement pour les ensembles de données qui sont mis à jour régulièrement (incrémentaux) comme nous tentons de faire dans cadre de ce mémoire. Dans ce cas spécifique, il faut tenir compte des différentes situations où des canaux d'inférence risquent d'être introduits entre les différentes versions d'ensembles de données générés. Un canal d'inférence étant une manière de déterminer avec précision à qui appartient l'information par simple comparaison des classes d'équivalence. Une classe d'équivalence étant un ensemble d'éléments de données ayant la même valeur de quasi-identificateur. On présente un algorithme d'anonymisation basé sur les travaux de Byun et al. [3] traitant de l'anonymisation des ensembles de données incrémentaux.

La troisième et dernière technique mise à contribution dans l'atteinte de l'objectif de mettre à jour la banque de données secondaire de façon incrémentale est la pseudonymisation. Cette technique consiste à remplacer la valeur d'un identificateur unique de l'image par un autre identificateur (le pseudonyme) n'ayant rien à voir avec le sujet dans le monde réel. Nous avons vu qu'il existe des pseudonymes réversibles et à sens unique. Les premiers permettent de retracer l'identité du patient alors que les seconds rendent impossible une telle réidentification. Les pseudonymes réversibles peuvent être implémentés de deux manières : la première consiste à maintenir une table de correspondance entre l'identificateur original et le pseudonyme, la seconde consiste à chiffrer l'identificateur à l'aide d'un algorithme de chiffrement à clé secrète. Les pseudonymes irréversibles peuvent être implémentés à l'aide d'une fonction de hachage à sens unique. En bénéficiant d'un identificateur unique qui perdurera dans le temps, on peut envisager faire un suivi à long terme du patient et c'est l'objectif visé dans ce mémoire.

Les connaissances acquises dans ce chapitre constituent notre assise pour aborder les chapitres suivants. Dans le prochain chapitre, nous effectuons l'analyse du problème et commençons à concevoir le système capable de répondre aux exigences auxquels nous avons à faire face.

CHAPITRE 2

ANALYSE ET CONCEPTION

2.1 Introduction

Notre objectif consiste à mettre en place une banque d'images médicales provenant du secteur clinique pour la rendre disponible à la recherche. Cela implique que l'image quitte les murs de l'établissement de santé et que la confidentialité du patient auquel l'image appartient n'est plus garantie par le secret professionnel de l'institution. Cela signifie également que l'identité du patient rattachée à l'image n'a plus du tout sa raison d'être. De plus, cette banque d'images doit être mise à jour périodiquement par l'ajout de nouvelles images. On veut mettre au point une solution logicielle implémentant les mécanismes capables de satisfaire à ces deux exigences. Considérant que l'obtention du consentement explicite du patient soit essentielle avant que l'information devienne accessible dans la banque d'images destinée à la recherche, il est néanmoins souhaitable qu'un niveau minimal de confidentialité soit assuré aux enregistrements patients avant qu'ils ne quittent l'institution.

Ce chapitre présente l'analyse et la conception rattachée à notre solution d'anonymisation. Nous débutons par la description de notre problème. Nous mettons au point un système nous permetant de classifier les attributs DICOM en fonction des points contraignants de ce standard. Nous élaborons un algorithme général d'anonymisation des attributs. Nous faisons la classification des attributs que le supplément 55 nous suggère de protéger et nous présentons notre conception des opérations de désidentification, d'anonymisation et de pseudonymisation.

2.2 Description du problème

Supposons que l'on veuille anonymiser une instance d'image DICOM. Dorénavant, nous utiliserons les termes « anonymiser » et « anonymisation » pour exprimer, dans un contexte plus large, l'application des opérations de désidentification, d'anonymisation et de pseudonymisation appliquées aux valeurs d'attributs de l'image dans le but de dissocier le mieux possible

cette image du patient auquel elle appartient. Nous préciserons « opération d'anonymisation » lorsque nous voudrons parler de l'opération d'anonymisation en tant que telle. Comme notre solution peut s'appliquer à une vaste gamme de domaines, nous généraliserons la partie clinique et la partie recherche par les secteurs dits « primaire » et « secondaire » respectivement. Supposons qu'une image est acquise à un moment donné. Elle emprunte deux itinéraires : le premier consiste à stocker l'image dans une banque destinée à l'usage exclusif de l'institution. Nous appelons cette banque d'images la banque primaire et l'information qu'elle contient ne quitte jamais les murs de l'institution. Le second trajet consiste à faire passer l'image par un processus d'anonymisation qui, une fois complété, rend possible la sauvegarde d'une copie anonymisée de l'instance dans une banque d'images, que nous appelons banque secondaire destinée à quitter les murs de l'établissement pour atteindre le secteur secondaire entièrment dévoué à la recherche. De plus, dans ce processus d'exportation de l'information, si la nouvelle image exportée appartient à un patient possédant déjà de l'information dans la banque secondaire, il faut prévoir les mécanismes permettant d'ajouter cette images à la série et à l'étude correspondantes au patient dans la banque secondaire. Ceci est pour donner la possibilité au secteur secondaire de suivre l'évolution de la maladie et du diagnostic sur une longue période de temps. C'est pour cette raison que l'intégrité des relations hiérarchiques entre les images doit être rigoureusement maintenue lors du passage du secteur primaire vers le secteur secondaire.

Considérons la tâche à effectuer : anonymiser une instance DICOM. Une telle instance se compose d'un ensemble d'attributs disposant tous d'une valeur. L'anonymisation consiste à prendre chacun de ces attributs et lui faire subir un changement plus ou moins important selon un ensemble de directives précises déterminées par l'utilisateur. On appelle cet ensemble de directives la politique d'anonymisation. Les attributs ainsi modifiés sont ensuite copiés dans une nouvelle image afin de produire l'instance anonymisée. Notre travail consiste à fournir les algorithmes nécessaires pour permettre cette anonymisation. Nous abordons le problème avec une approche orientée objet pour élaborer notre solution, donc notre analyse et notre conception sont également orientées objet.

Chaque attribut du standard DICOM est unique. Leur rôle est de fournir une information spécifique appartenant à une image DICOM. Les attributs se caractérisent par une étiquette (« tag »), une représentation de valeur (VR) et par le fait que leur valeur est requise ou non dans telle ou telle circonstance, etc. On constate que tous les attributs, de par leur nature, peuvent représenter un traitement spécifique dépendant de l'opération de désidentification, d'anonymisation et de pseudonymisation qu'on veuille y appliquer. Le nombre d'attributs DICOM multiplié par les opérations d'anonymisation qu'il est possible d'appliquer sur chacun d'eux représente un nombre considérable d'algorithmes. C'est une solution envisageable, mais difficilement réalisable. Nous disons envisageable parce que cette solution n'est pas impossible à réaliser et qu'elle pourrait être nécessaire dans une situation qu'on pourrait qualifier de « pire des cas d'anonymisation ». Cependant, dans le cadre de notre analyse, nous chercherons à aborder le problème de façon plus efficace en tentant de catégoriser les attributs, de discriminer les circonstances et de factoriser les traitements afin d'éviter la redondance.

Selon notre analyse, les attributs à anonymiser se divisent en trois groupes spécifiques : les attributs à désidentifier, les attributs dont il faut généraliser la valeur et ceux dont on doit remplacer la valeur par un pseudonyme. La présentation des attributs prescrits à l'anonymisation est faite à la section suivante. Les attributs de la première catégorie voient leur présence supprimée ou modifiée dans l'instance secondaire alors que pour les deux autres catégories, il s'agit d'une application spécifique de la modification de valeurs. Les attributs dont il n'est pas explicitement question dans la politique d'anonymisation sont passés directement dans l'instance secondaire sans aucune autre forme d'anonymisation. La catégorie dans laquelle un attribut se situe constitue la première et la plus importante contrainte de détermination de l'algorithme d'anonymisation à appliquer, les autres contraintes suivent par ordre de priorité.

La deuxième contrainte consiste dans la représentation de valeur (« Value Representation ») qui dicte la taille, le format et l'ensemble des caractères acceptés par l'attribut en question. Pour chaque type d'anonymisation, il faut prévoir un algorithme d'anonymisation différent par représentation de valeur pour tenir compte des spécificités relatives à chacune d'elles.

La troisième contrainte se rapporte au type d'attribut qui spécifie si un attribut est optionnel ou obligatoire et, dans ce dernier cas, si sa valeur peut être ou non laissée vide.

Les deux dernières contraintes identifiées par notre analyse se rapportent au type de définition d'objet d'information (en anglais « Information Object Definition » ou IOD) et aux besoins spécifiques dictés par la recherche. Ces contraintes n'ont pas de rôle déterminant dans l'élaboration des algorithmes d'anonymisation en tant que tel soit parce que nous ne nous sommes pas suffisamment penchés sur la question parce que trop compliqués à gérer, soit qu'il sont pris en charge simplement par la politique d'anonymisation élaborée par l'utilisateur. Dans tous les cas, notre solution s'applique à la vaste majorité des cas. Pour un besoin plus spécifique, il est toujours possible d'ajouter à notre solution de nouveaux algorithmes.

Maintenant que nous pensons avoir colligé tous les aspects contraignants du problème, nous pouvons mentionner les points en faveur de notre conception. Le premier point positif est que l'ensemble d'attributs que l'on doit anonymiser est fini et connu : ils ont tous été répertoriés dans le supplément 55 de DICOM. Cela aura pour effet bénéfique de limiter le nombre de classes aux attributs qu'il nous est demandé d'anonymiser. D'ailleurs il est judicieux de mentionner ici qu'il n'est pas du ressort de ce mémoire de discuter les raisons pour lesquelles un attribut est présent ou non dans la liste des attributs à protéger prescrite par le supplément 55 de DICOM. Pour cela, nous nous en remettons entièrement à la compétence des spécialistes DICOM qui ont participé à la rédaction de ce supplément. Un deuxième point positif est que les classes d'anonymisation doivent travailler avec des objets provenant d'une classe ayant une interface unique. Ce qui améliore grandement notre conception. Troisièmement, la programmation orientée objet nous permet de gérer certaines contraintes à l'aide de la classe. C'est l'approche privilégiée pour tenir compte des spécificités propres à un traitement. Pour finir, le quatrième et dernier point favorisant notre conception est que certaines spécificités sont prises en charge par l'instance.

Ayant considéré les contraintes et des points favorables à notre conception, nous pouvons résumer le processus d'anonymisation que nous tentons de mettre au point dans le cadre de notre solution :

- L'opération par défaut consiste à copier l'attribut DICOM tel quel dans l'instance secondaire ;
- l'opération de suppression ignore tout simplement l'attribut ;
- la mise à vide consiste à remplir l'attribut de blancs ;
- le changement de valeur affecte une valeur arbitraire à l'attribut ;
- les séquences d'items (SQ) sont des ensembles d'éléments DICOM qui peuvent contenir d'autres séquences d'items, elles sont anonymisées récursivement ;
- l'anonymisation des identificateurs uniques (UI) est un traitement qui s'apparente à la pseudonymisation, il constitue un cas particulier qui nécessite des algorithmes spécifiques ;
- la généralisation permet de modifier le contenu d'un attribut afin de conserver une certaine qualité à l'information ;
- la pseudonymisation permet les mises à jour incrémentales.

2.3 Description de la solution

Concrètement, nous cherchons à satisfaire aux contraintes de conception en séparant les algorithmes relatifs à chaque ensemble d'attributs dans une classe distincte. Cela nous donne un ensemble de classes auquel il nous est possible d'ajouter de nouvelles classes. Nous voulons que les classes qui utilisent nos classes d'anonymisation ignorent tout de la spécificité relative à chaque classe. Cela nous mène à la conception présentée à la figure 2.1.

Il s'agit du mariage entre un patron de conception *Strategy* et le patron de conception *Hashed Adapter Objects* (voir les annexes pour comprendre ces patrons de conception). Voici comment les classes fonctionnent entre elles. Un objet `Deidentifier` consulte

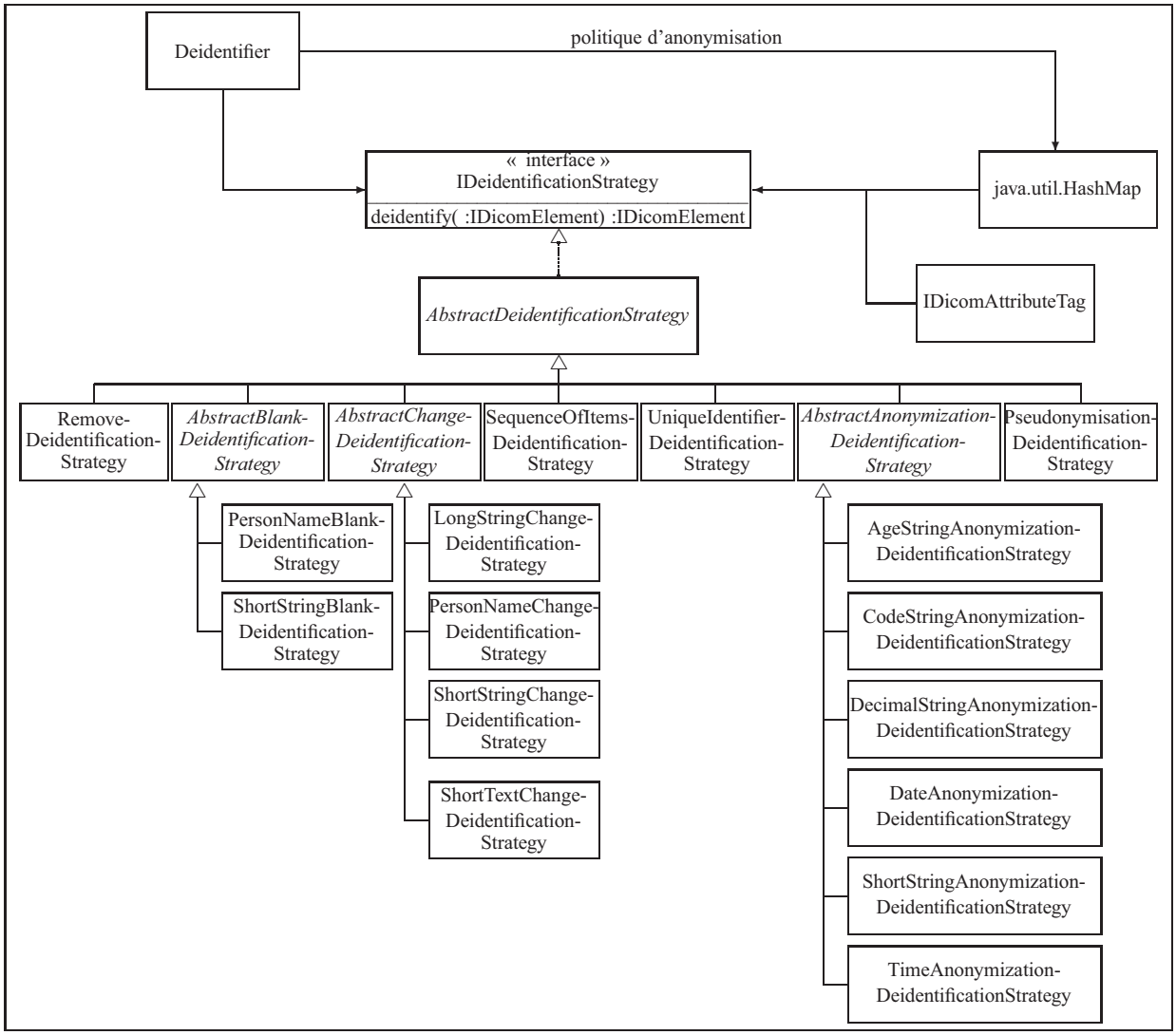


Figure 2.1 Classes d'anonymisation.

un objet `java.util.HashMap` qui contient une table de correspondance entre un objet `IDicomAttributeTag` et un objet de type interface `IDeidentificationStrategy` chargé de l'anonymisation de cet attribut (`IDicomElement`). L'utilisation d'un `HashMap` découle de l'application directe du patron *Hashed Adapter Objects* qui permet d'éviter l'utilisation d'une longue chaîne de « *if* » imbriqués. Cette organisation permet à un objet `Deidentifier` d'anonymiser n'importe quelle attribut DICOM seulement en appelant la méthode `deidentify` de l'objet `IDeidentificationStrategy`.

L'utilisation du patron de conception *Strategy* est particulièrement appropriée pour les raisons suivantes :

- Le patron Strategy permet d'offrir plusieurs algorithmes d'anonymisation ;
- il permet de varier le comportement d'une instance d'une classe ;
- il permet de faire varier le comportement d'un objet à l'éxécution ;
- la délégation d'un comportement envers une interface permet aux classes utilisant ce comportement d'ignorer complètement les détails d'implémentation des classes implémentant cette interface et ce comportement.

Nous allons maintenant revoir, pour chacune des catégories vues précédemment, les attributs, leur représentation, leurs spécificités et les classes du modèle qui s'y appliquent.

2.4 Première catégorie : les attribut à désidentifier

Cette section présente les attributs DICOM à désidentifier tel que stipulé par le supplément 55 du standard DICOM. Rappelons que la désidentification consiste en le retrait de toute information du fichier DICOM pouvant mener à l'identité du patient. Parmi ces informations, il y a les données nominatives faisant partie intégrante de l'image DICOM. Les attributs à désidentifier sont présentés au tableau 2.1. Parmi les opérations de désidentification, il y a : a) la suppression, b) la mise à vide, c) la modification, d) les séquences d'items, e) les identificateurs uniques (UID) et f) les attributs qu'il ne faut pas modifier.

Les attributs à supprimer

Le tableau 2.1 (a) présente les attributs à supprimer de l'instance DICOM. L'opération de suppression des attributs est l'opération de désidentification la plus simple à appliquer à une instance DICOM. On supprime ces attributs en passant outre leur enregistrement dans la version désidentifiée de l'instance. La présence d'un attribut dans l'instance est prescrite par le standard.

Tableau 2.1 Les attributs à désidentifier

Nom de l'attribut	Tag	Commentaire
a) Les attributs à supprimer		
Referring Physician's Address (ST)	(0008,0092)	Leur valeur n'est généralement pas utile à la recherche.
Referring Physician's Telephone Numbers (SH)	(0008,0094)	
Institutional Department Name (LO)	(0008,1040)	
Physician(s) of Record (PN)	(0008,1048)	
Performing Physicians' Name (PN)	(0008,1050)	
Name of Physician(s) Reading Study (PN)	(0008,1060)	
Admitting Diagnoses Description (LO)	(0008,1080)	
Derivation Description (ST)	(0008,2111)	
Additional Patient's History (LT)	(0010,21B0)	Champs à texte libre pouvant dévoiler l'identité du patient.
Patient Comments (LT)	(0010,4000)	
Image Comments (LT)	(0020,4000)	
Other Patient Names (PN)	(0010,1001)	Ces attributs peuvent servir à la ré-identification du patient et ne sont pas utiles la recherche.
Medical Record Locator (LO)	(0010,1090)	
Occupation (SH)	(0010,2180)	
Accession Number (SH)	(0008,0050)	
b) Les attributs à vider		
Referring Physician's Name (PN)	(0008,0090)	Sa valeur n'est généralement pas utile à la recherche, mais sa présence est requise pour des raisons de conformité.
Study ID (SH)	(0020,0010)	
Patient's Name (PN)	(0010,0010)	
c) Les attributs à modifier		
Station Name (SH)	(0008,1010)	Les valeurs de ces attributs ne sont pertinentes que pour l'équipement utilisé.
Device Serial Number (LO)	(0018,1000)	
Institution Name (LO)	(0008,0080)	Les valeurs de ces attributs ne sont généralement pas pertinentes pour la recherche en traitement d'image ou d'aide au diagnostic.
Institution Addresse (ST)	(0008,0081)	
Operators' Name (PN)	(0008,1070)	
d) Les séquences d'items		
Content Sequence (SQ)	(0040,A730)	Ces séquences d'items sont traitées récursivement.
Request Attributes (SQ) Sequence	(0040,0275)	
e) Les identificateurs uniques		
Instance Creator UID (UI)	(0008,0014)	Les UIDs générés devraient préserver les relations entre les objets et la hiérarchie qui pourrait exister entre eux.
SOP Instance UID (UI)	(0008,0018)	
Referenced SOP Instance UID (UI)	(0008,1155)	
Study Instance UID (UI)	(0020,000D)	
Series Instance UID (UI)	(0020,000E)	
Frame of Reference UID (UI)	(0020,0052)	
Synchronization Frame of Reference UID (UI)	(0020,0200)	
UID (UI)	(0040,A124)	
Storage Media File-set UID (UI)	(0088,0140)	
Referenced Frame of Reference UID (UI)	(3006,0024)	
Related Frame of Reference UID (UI)	(3006,00C2)	
f) Les attributs à ne pas modifier		
Study Description (LO)	(0008,1030)	Leur valeur est importante pour les algorithmes de traitement d'image.
Series Description (LO)	(0008,103E)	
Protocol Name (LO)	(0018,1030)	

Les attributs menant à l'identité du médecin soignant tels que le « Referring Physician's Address » (0008,0092) et le « Referring Physician's Telephone Numbers » (0008,0094) sont des attributs « Short Text » (ST) et « Short String » (SH) respectivement.

Parmi les attributs « Long String » (LO) qu'on peut éliminer il y a les attributs « Admitting Diagnoses Description » (0008,1080) et « Institutional Department Name » (0008,1040) qui contiennent respectivement la description du (des) diagnostic(s) d'admission et le nom du département. L'attribut « Medical Record Locator » (0010,1090) est un autre attribut LO qui contient une référence au dossier (physique celui-là) du patient.

L'attribut « Occupation » (0010,2180) est un attribut « Short String » (SH) et un champs optionnel décrivant l'occupation du patient. Sa présence n'est pas pertinente à la recherche. L'attribut « Accession Number » (0008,0050) est un attribut SH et est un identificateur de la demande de service d'imagerie pour cette procédure. Cet attribut n'est pas requis par le standard et on peut le supprimer.

L'attribut « Additional Patient's History » (0010,21B0) contient de l'information additionnelle concernant l'historique médicale du patient. Le « Patient Comments » (0010,4000) contient un commentaire à propos du patient. L'attribut « Image Comments » (0020,4000) contient un commentaire à propos de l'image. Ils sont tous trois des attributs « Long Text » (LT) dont la présence n'est pas requise et peuvent donc être supprimés de l'instance DICOM sans souci de compromettre la conformité de l'instance.

L'attribut « Derivation Description » (0008,2111), un « Short Text » (ST) contient une description de la manière dont l'image a été dérivée. Cet attribut devrait être retiré de l'instance puisque sa présence n'est pas obligatoire et qu'elle pourrait compromettre la confidentialité du patient.

Parmi les attributs « Person Name » considérés, on retrouve le « Physician(s) of Record » (0008,1048), le « Performing Physicians' Name » (0008,1050) et le « Name of Physician(s) Reading Study » (0008,1060) qui représentent tous un nom de médecin ayant joué un rôle pen-

dant l'étude. L'attribut « Other Patient Names » (0010,1001) contient les autres noms utilisés pour identifier le patient.

Pour toutes les opérations de désidentification destinées à supprimer l'attribut du fichier DICOM, l'opération est relativement simple et consiste à ne pas tenir compte de l'attribut lors de la copie dans l'instance secondaire. Ce comportement s'apparente à un patron de conception *Null Object*. La même instance de la classe `RemoveDeidentificationStrategy` est affectée à tous les `IDicomAttributTag` qu'on doit supprimer.

Les attributs à vider

Le tableau 2.1 (b) présente les attributs dont la présence est requise par le standard, mais dont la valeur peut être laissée vide à l'intérieur de l'instance DICOM. L'attribut « Referring Physician's Name » (0008,0090), un attribut « Person Name » relatif au nom du médecin traitant, est requis par le standard même s'il ne contient pas de valeur (réf. PS 3.3-2008 section C.7.2.1). L'attribut « Study ID » (0020,0010) est un attribut « Short String » (SH) généré par l'équipement (ou l'utilisateur) pour représenter l'étude. Cet identificateur doit être vidé pour éviter qu'il ne mène à l'identité du patient. L'attribut « Patient's Name » (0010,0010), un attribut « Person Name » (PN), contient le nom du patient et doit être vidé de l'instance DICOM pour des raisons évidentes de confidentialité.

La mise à vide des attributs de cette catégorie est prise en charge par les classes dérivant de la classe `AbstractBlankDeidentificationStrategy`. Dans ce cas-ci, deux classes ont été prévues pour assurer le comportement approprié aux deux types d'attribut qu'il y a à traiter. La classe `PersonNameBlankDeidentificationStrategy` qui sert aux attributs « Referring Physician's Name » (0008,0090) et « Patient's Name » (0010,0010) et la classe et `ShortStringBlankDeidentificationStrategy`.

Les attributs à modifier

Le tableau 2.1 (c) présente les attributs dont il faut changer le contenu par une valeur factice parce que le standard exige que ces attributs possèdent une valeur. Ici, ces attributs se distinguent des attributs qu'on doit vider par le fait qu'ils ne peuvent être laissés sans valeur. L'attribut « Station Name » (0008,1010), un attribut « Short String » (SH), et le « Device Serial Number » (0018,1000), un attribut « Long String » (LO), représentent respectivement le nom et le numéro de série de l'équipement ayant servi à produire l'instance. Ils représentent tous deux de l'information en rapport avec l'équipement. Ces attributs sont requis d'avoir une valeur à certaines conditions et, pour cette raison, on ne peut les retirer de l'instance inconditionnellement. Les attributs porteurs d'informations relatives à l'identification de l'institution tels que le « Intitution Name » (0008,0080) ou le « Institution Address » (0008,0081), deux attributs « Long String » (LO) et « Short Text » (ST) respectivement, devraient être remplaçés par des valeurs factices ne menant pas à l'identification de l'institution. Ils sont conservés dans l'instance pour des raisons de conformité au standard. L'attribut « Operators' Name » (0008,1070), un attribut « Person Name » (PN), représente le nom des opérateurs soutenant l'étude. Cet attribut doit avoir une valeur s'il est présent dans l'instance et doit être consistant dans les instances SOP constituant l'étude.

Les attributs dont il faut changer la valeur font partie de quatre représentations de valeur différentes bien qu'il s'agisse de chaînes de caractères. Les classes dérivant de la classe abstraite `AbstractChangeDeidentificationStrategy` servent à définir de façon plus spécifique le traitement relatif au changement de valeur des attribut de type « Long String, » « Person Name, » « Short String » et « Short Text. » Nous vous épargnons le nom de ces classes concrètes pour des raisons de lisibilité. Ce qu'il est important de comprendre est que si un attribut d'un autre type s'ajoute ou si un attribut avec un comportement spécifique s'ajoute, la seule modification à apporter au modèle sera d'ajouter une nouvelle classe. Le traitement consiste principalement à changer la valeur de l'attribut par une chaîne de caractères constante afin de dissimuler l'ancienne valeur et ainsi préserver l'identité du patient.

Les séquences d'items

Le tableau 2.1 (d) présente les séquences d'items qu'on doit soumettre à la désidentification. Les attributs « Content Sequence » (0040,A730) et « Request Attributes Sequence » (0040,0275) consituent des cas particuliers car ils sont tous deux des attributs « Sequence of Items » (SQ) qui contiennent des sous-ensembles d'éléments de données DICOM. Ils sont désidentifiés de façon récursive selon la politique d'anonymisation établie. Cette opération est assurée par la classe concrète `SequenceOfItemsDeidentificationStrategy` dérivant directement de la classe abstraite `AbstractDeidentificationStrategy`.

Les identificateurs uniques

Le tableau 2.1 (e) présente les attributs « Unique Identifier » (UI) qu'on doit désidentifier. Ils représentent un cas particulier d'attributs à modifier car les UIDs sont utilisés en DICOM pour identifier une vaste gamme d'items de façon unique : une voie toute privilégiée à l'identification des objets. Ces attributs sont le « Instance Creator UID » (0008,0014) qui identifie le dispositif qui a créé l'instance, le « SOP Instance UID » (0008,0018) et le « Referenced SOP Instance UID » (0008,1155) qui identifient une instance SOP spécifique. L'attribut « Study Instance UID » (0020,000D) est l'identificateur pour l'étude. Le « Series Instance UID » (0020,000E) identifie la série. Le « Frame of Reference UID » (0020,0052) identifie le cadre (« frame ») de référence de la série. L'attribut « Synchronization Frame of Reference UID » (0020,0200) est l'UID de l'environnement de synchronisation. Le « UID » (0040,A124) est utilisé dans les items de type « nom-valeur » dont le « Value Type » (0040,A040) est égal à « UIDREF ». L'attribut « Storage Media File-set UID » (0088,0140) permet d'identifier un support de stockage sur lequel l'instance SOP réside. Le « Referenced Frame of Reference UID » (3006,0024) identifie un cadre de référence dans lequel une ROI « region of interest » est définie. L'attribut « Related Frame of Reference UID » (3006,00C2) contient un identificateur vers le cadre de référence dont le système de coordonnées doit être tranformé pour celui du cadre actuel.

Chaque type d'UID est traité par une instance différente de la classe `UniqueIdentifierDeidentificationStrategy`. Si le problème l'exige, il est aussi

possible de dériver une classe pour chaque type d'UID que l'on doit traiter. Dans le cas des UIDs, la seule garantie que nous possédons est qu'il s'agit d'un champs de 64 caractères dont le contenu est composé des chiffres décimaux (« 0 » à « 9 ») et du point décimal (« . »). Selon le standard DICOM, chaque UID est représenté par une partie « org root » fournie par une instance émetrice autorisée (ISO, ANSI) et une partie « suffixe » qui est laissée à l'entière discrétion de l'organisation qui en est tributaire. En outre, c'est le rôle de l'organisation d'assurer l'unicité de ses UIDs. D'ailleurs, il est clairement spécifié dan le standard qu'en raison de la souplesse permise par DICOM dans la création des UIDs privés, toute implémentation ne devrait présumer qu'un UID puisse fournir une information sémantique par l'analyse de certaines de ses composantes.

Le fonctionnement de notre algorithme est le suivant : nous générons un nouveau « SOP Instance UID » pour chaque nouvelle image reçue. Pour ce faire, le contenu des attributs « Study Instance UID » (0020,000D) et « Series Instance UID » (0020,000E) est comparé au contenu de tables de correspondance. Si l'une des deux valeurs ne se retrouve pas dans l'une des tables, une nouvelle entrée est créée et un numéro séquentiel unique à la table est assigné à cette entrée. Ce numéro séquentiel fera partie du nouveau « SOP Instance UID » (0008,0018) généré. Le nouvel UID généré pour identifier l'instance unique prendra la forme suivante :

« 1.2.840.xxxxx.3.152.235.2.12.187636473 »

La racine se compose des éléments suivants :

1	Signifie ISO
2	Identifie un membre ANSI
840	Indique le code de pays (840 = U.S.)
xxxxx	Identifie l'implémentation DICOM utilisé

Les deux premiers composants du suffixe se rapportent à l'identification de l'équipement :

3	Manufacturier
152	Numéro de série

Les quatre derniers composants se rapportent à l'identification de l'image :

235 Numéro d'étude (« Study number »)

2 Numéro de la série (« Series number »)

12 Numéro de l'image (« Image number »)

187636473 estampille temporelle (« Timestamp ») de l'acquisition de l'image

Cette manière de faire permet d'assurer l'intégrité hiérarchique des instances DICOM et facilite le suivi des images sur une longue période de temps tout en conservant la confidentialité des instances DICOM.

Les attributs à ne pas modifier

Le tableau 2.1 (f) présente les attributs qui ne peuvent être modifiés puisque leur valeur peut être importante pour les algorithmes de traitement d'image. L'attribut « Study Description » (0008,1030), un « Long String » (LO), est une description ou classement de l'étude généré par l'institution. L'attribut « Series Description » (0008,103E), un « Long String » (LO), est une description de la série fournie par l'utilisateur. L'attribut « Protocol Name » (0018,1030), un « Long String » (LO), est la description définie par l'utilisateur des conditions dans lesquelles la série a été réalisée.

Ces attributs ainsi que tous les autres attributs dont la stratégie d'anonymisation n'est pas explicitement spécifiée dans la politique d'anonymisation sont copiés directement dans l'instance secondaire.

2.5 Deuxième catégorie : les attributs à anonymiser

Cette section présente les mécanismes élaborés pour effectuer la généralisation des attributs DICOM afin de préserver la confidentialité des patients. L'opération d'anonymisation consiste en la généralisation des valeurs d'attribut afin que même seuls ou regroupés, la possibilité d'en déduire la provenance par inférence soit gardée en dessous d'une certaine probabilité. Les attributs sujets à l'anonymisation sont présentés au tableau 2.2.

Tableau 2.2 Les attributs à anonymiser

Nom de l'attribut	Tag	Commentaire
Patient's Birth Date (DA)	(0010,0030)	Ces attributs peuvent permettre d'identifier le patient s'il ne sont convenablement pas anonymisés.
Patient's Birth Time (TM)	(0010,0032)	
Patient's Age (AS)	(0010,1010)	
Patient's Sex (CS)	(0010,0040)	
Patient's Size (DS)	(0010,1020)	
Patient's Weight (DS)	(0010,1030)	
Ethnic Group (SH)	(0010,2160)	

Les attributs « Patient's Birth Date » (0010,0030) et « Patient's Birth Time » (0010,0032) sont des attributs « Date » (DA) et « Time » (TM) respectivement et correspondent à la date et l'heure de la naissance du patient. L'attribut « Patient's Age » (0010,1010), de valeur de représentation « Age String » (AS), contient l'âge du patient. L'attribut « Patient's Sex » (0010,0040) est un attribut « Code String » (CS) qui représente le sexe du patient. Les attributs « Patient's Size » (0010,1020) et « Patient's Weight » (0010,1030), deux « Decimal String » (DS), contiennent respectivement la taille et le poids du patient. L'attribut « Ethnic Group » (0010,2160) est un « Short String » (SH) qui représente le groupe ethnique ou la race du patient.

Les algorithmes d'anonymisation sont très limités par les règles strictes dictées par les représentation de valeur. Néanmoins, il est tout de même possible de généraliser un peu l'information des attributs. Par exemple, il est possible de changer la valeur du jour d'un attribut « Date » (DA) pour lui affecter la valeur « 01 ». Ainsi, la date de naissance du 21 juin 1971, représentée par la chaîne « 19710621 », serait généralisée par « 19710601 » ainsi que toutes les autres dates du mois de juin 1971. La date peut être généralisée jusqu'au mois en ne laissant que l'année à sa valeur initial (« 19710101 »). Il va sans dire qu'un attribut de représentation de valeur DA ne peut être laissé dans un état incohérent (p. ex., « 19710000 »). Il existe également en DICOM la notion de « range matching » qui permet l'introduction d'un caractère « - » dans la valeur de l'attribut et qui permet de représenter un intervalle (p. ex., « 19710601-19710615 » pour représenter une date située entre le 1er et le 15 juin 1971). C'est une avenue qui peut s'avérer forte intéressante, mais qui demande encore une investigation puisque ce format est censé n'être utilisé que dans le cadre d'une requête (« Query »).

Un attribut « Time » (TM) peut être anonymisé d'une manière un peu similaire à un attribut « Date » (DA), c'est à dire en affectant la chaîne « 00 » à la partie de l'heure que l'on veut anonymiser. Ici, encore on peut utiliser le « - » pour exprimer un intervalle, mais il nous reste à confirmer que cette valeur est permise en dehors d'une « Query. »

L'attribut « Patient's Sex » (0010,0040) est une attribut du type « Code String » pouvant contenir les caractères « M=male, » « F=female, » ou « O=other. »

Les attributs du type « Decimal String » (DS) peuvent contenir les caractères « 0-9, » « +, » « -, » « . » et le caractère d'espacement du répertoire par défaut. Éventuellement, un tel attribut pourrait contenir une valeur de type « a-b » tel que « 100-150 ».

Un attribut du type « Short String » peut contenir n'importe quel chaîne de caractères provenant du répertoire par défaut [spécifié par l'attribut « Specific Character Set » (0008,0005)] dans la mesure où sa taille n'excède pas 16 caractères.

En ce qui concerne l'algorithme d'anonymisation utilisé, nous tentons d'implémenter l'algorithme décrit par Byun et al. [3] dans son article sur les ensembles de données incrémentaux. Bien que cette implémentation soit considérablement compromise par le fait que les données que nous manipulons ne sont pas stockées sour forme de table alors que cet algorithme a été conçu pour ce genre d'organisation des données. Pour pouvoir appliquer cet algorithme, il faut maintenir une liste des valeurs d'attributs que nous cherchons à anonymiser en parallèle avec les instances de la banque secondaire DICOM. De plus l'algorithme d'insertion doit remplir les trois exigences suivantes :

a. À chaque ajout d'information à la banque secondaire, celle-ci doit demeurer ℓ-diverse ;

b. la qualité de l'information contenue dans la banque secondaire doit être maintenue aussi élevée que possible ;

c. la banque secondaire doit être dépourvue de canaux d'inférence.

L'idée maîtresse derrière cet algorithme consiste à insérer les nouveaux enregistrements dans la classe d'équivalence la plus rapprochée possible de telle sorte que les changements causés par la généralisation sont gardés à leur niveau minimal.

Il existe trois types d'opérations lors d'une mise à jour d'un ensemble de données anonymisé : l'ajout, l'insertion et la séparation.

a. (**Ajout**) Si un groupe d'enregistrements forme une classe d'équivalence qui ne chevauche aucune autre classe d'équivalence, alors nous devons tout simplement ajouter cette nouvelle classe d'équivalence à la banque secondaire sans risquer de compromettre la sécurité des versions précédentes.

b. (**Insertion**) Si les enregistrements ne peuvent être ajoutés comme une nouvelle classe d'équivalence, ils doivent donc être insérés dans des classes d'équivalence existantes. Ceci doit se faire en minimisant la distortion de l'information qui pourrait survenir dans la banque secondaire.

c. (**Séparation**) Après avoir ajouté ou inséré tous les nouveaux enregistrements dans la banque secondaire, il est possible que le nombre de valeurs distinctes dans quelques classes d'équivalence excède 2ℓ. Dans ce cas il est possible de séparer cette classe d'équivalence de manière à améliorer la qualité de l'information.

Pour prévenir les canaux d'inférence, il faut traiter les cas d'insertion et de séparation avec un soin particulier. En effet, ces deux opérations peuvent introduire, si on n'y prête pas attention, des canaux d'inférence entre les mises à jour successives des tables. Pour pallier cette situation, la solution consiste à instaurer une file d'attente où les enregistrements doivent séjourner le temps de s'assurer qu'ils n'introduisent pas de nouveaux canaux d'inférence. La séparation des classes d'équivalence implique de maintenir de l'information au sujet de toutes les versions précédentes.

2.6 Troisième catégorie : les attributs à pseudonymiser

Cette section présente le fonctionnement général de l'algorithme pour permettre la mise à jour incrémentale des instances DICOM de la banque secondaire. La pseudonymisation consiste à remplacer un numéro d'identification du patient par une autre valeur unique ne menant pas à l'identification du patient. Les attributs destinés à la pseudonymisation sont présentés au tableau 2.3.

Tableau 2.3 Les attributs à pseudonymiser

Nom de l'attribut	Tag	Commentaire
Patient ID (LO) Other Patient Ids (LO)	(0010,0020) (0010,1000)	Doivent être remplaçés par un pseudonyme.

Le tableau 2.3 présente les attributs qu'on doit remplacer par un pseudonyme. Les attributs « Patient ID » (0010,0020) et « Other Patient Ids » (0010,1000) sont des attributs « Long String » (LO). Le « Patient ID » (0010,0020) est le numéro d'identification primaire du patient. L'attribut « Other Patient Ids » (0010,1000) contient les autres numéros (ou codes) pour identifier le patient.

La solution que nous avons adopté est de maintenir une table de correspondance entre le numéro de patient (« Patient ID ») et le pseudonyme. C'est la manière la plus simple d'assurer le suivi à long terme du patient.

2.7 Conclusion

Dans ce chapitre, nous avons vu que les attributs sont classifiés selon le type d'anonymisation qui leur est assignée. Il doit exister un algorithme différent par représentation de valeur des attributs présents dans la catégorie d'anonymisation. Cette multiplication d'algorithmes apparentés est destinée à prendre en charge toutes les subtilités relatives à la taille, au format et aux caractères acceptés par les différents attributs. Ceci afin de favoriser le plus possible la réutilisation du code. À la limite, cette granularité peut descendre jusqu'à l'attribut lui-même où, dans ce cas, chaque attribut possède un algorithme de traitement qui lui est propre.

La conception et le développement orientés objet permettent d'accéder à ce niveau de granularité dans la détermination de l'algorithme à appliquer pour anonymiser un attribut. En développant une classe par algorithme, on met en application le patron de conception *Strategy* qui permet de développer une stratégie différente pour chaque représentation de valeur dans une catégorie d'anonymisation donnée. En utilisant une interface commune à toutes les classes d'anonymisation, on rend le programme d'anonymisation indépendant de l'organisation hiérarchique des classes qui fournissent la fonctionnalité. La classe de type interface `IDeidentificationStrategy` fournit une méthode `deidentify` qui prend en paramètre l'attribut original (`IDicomElement`) et retourne sa version anonymisée. L'objet `IDeidentificationStrategy` encapsule toute l'information nécessaire à l'accomplissement de sa tâche et l'objet de type interface `IDicomElement` passé en paramètre encapsule toute l'information utile à sa propre anonymisation.

La politique d'anonymisation est représentée par une structure faisant la correspondance entre une étiquette d'attribut (`IDicomAttributeTag`) et une stratégie d'anonymisation (`IDeidentificationStrategy`). À chaque étiquette correspond une et une seule stratégie d'anonymisation ce qui permet l'application directe du patron *Hashed Adapter Objects* qui est un patron d'optimisation de code. Les attributs qui n'ont pas leur étiquette listée dans la politique d'anonymisation ne sont pas anonymisés et sont tout simplement copiés dans l'instance secondaire. C'est ce qui arrive pour la plupart des attributs d'une instance DICOM.

CHAPITRE 3

IMPLÉMENTATION

Introduction

Le besoin consiste à fournir le secteur de la recherche en données provenant de patients pour tester et valider les algorithmes et méthodes mis au point pour améliorer la qualité des images ou détecter des caractéristiques spécifiques dans le but d'aider au diagnostic.

La solution à ce problème consiste à mettre au point une banque d'images médicales provenant du secteur clinique. Cela ne doit cependant pas se faire au détriment de la protection de la confidentialité de l'information, du droit à la vie privée du patient ou même du consentement du patient. De plus, afin de pouvoir suivre un patient sur une longue période de temps, il faudra être en mesure d'effectuer des mises à jour incrémentales par l'ajout successif de nouvelles informations.

Ce chapitre présente les détails techniques relatifs à l'implémentation du système Anonym 1.0 qui a été développé dans le cadre de ce mémoire. Les différentes sections de ce chapitre traitent des contraintes dont il a fallu tenir compte pour atteindre les objectifs visés. Une section traite de l'API (« Application Programming Interface ») utilisé. On fait un bref retour sur les contraintes dictées par le standard DICOM lui-même et on présente les différents patrons de conception utilisés dans le cadre de la mise au point de la politique d'anonymisation qui constitue le cœur de notre application. Il est également question des autres patrons de conception qui jouent des rôles secondaires mais néanmoins importants dans l'élaboration de notre application. Ce chapitre présente finalement les points qui n'ont malheureusement pas été abordés dans le cadre de notre projet.

3.1 Retour sur les contraintes

Les premières contraintes qui se présentent à nous concernent le type, le format et le média sur lequel les données à anonymiser sont disponibles. Le type de donnée consiste en des images médicales. Ces images doivent être disponibles électroniquement pour permettre de tester les algorithmes de traitement d'images développés en recherche. Le format d'image utilisé est le format DICOM. Ce format s'impose par son usage très répandu dans le monde médical. Le type de média utilisé est le fichier DICOM.

L'utilisation du format DICOM est souhaitable car il s'agit d'un standard largement utilisé dans le domaine médical ce qui facilite l'intégration de notre application à l'intérieur des systèmes existants. C'est pour cette raison que notre application est en mesure de communiquer avec des modalités DICOM. De plus, comme il s'agit d'un standard ouvert qui est abondamment documenté, il existe de nombreux outils pour nous aider dans le développement de notre application.

Concernant le média utilisé, mentionnons que les images sont conservées sous forme de fichiers DICOM sur le disque. En conservant le format de stockage le plus générique possible, nous facilitons la manipulation des images anonymisées par des applications tierces qui sont indépendantes du fonctionnement de notre système. De plus, étant donné la complexité du standard DICOM, nous ne voulons pas devoir maintenir une structure d'information similaire à l'intérieur de notre système. Nous nous contentons plutôt de conserver dans notre système que certaines informations utiles aux traitements que nous souhaitons appliquer à l'image en plus d'une référence au fichier contenant l'image en question.

3.2 La modalité DICOM

Notre application est capable d'accepter des images médicales en utilisant le standard DICOM. Le rôle de notre application consiste à attendre l'établissement d'une nouvelle association, un fois l'association établie, elle reçoit une requête de stockage (C_STORE) avec toute l'information nécessaire pour anonymiser l'image DICOM. Une fois l'image originale reçue, le système

itère parmi les attributs et anonymise chacun des items en tenant compte des règles énoncées dans la politique d'anonymisation. Les attributs de l'image sont ensuite récupérés puis assemblés en une nouvelle image avant son enregistrement définitif dans la base de données secondaire. Ce fonctionnement de l'application est illustré à la figure 3.1.

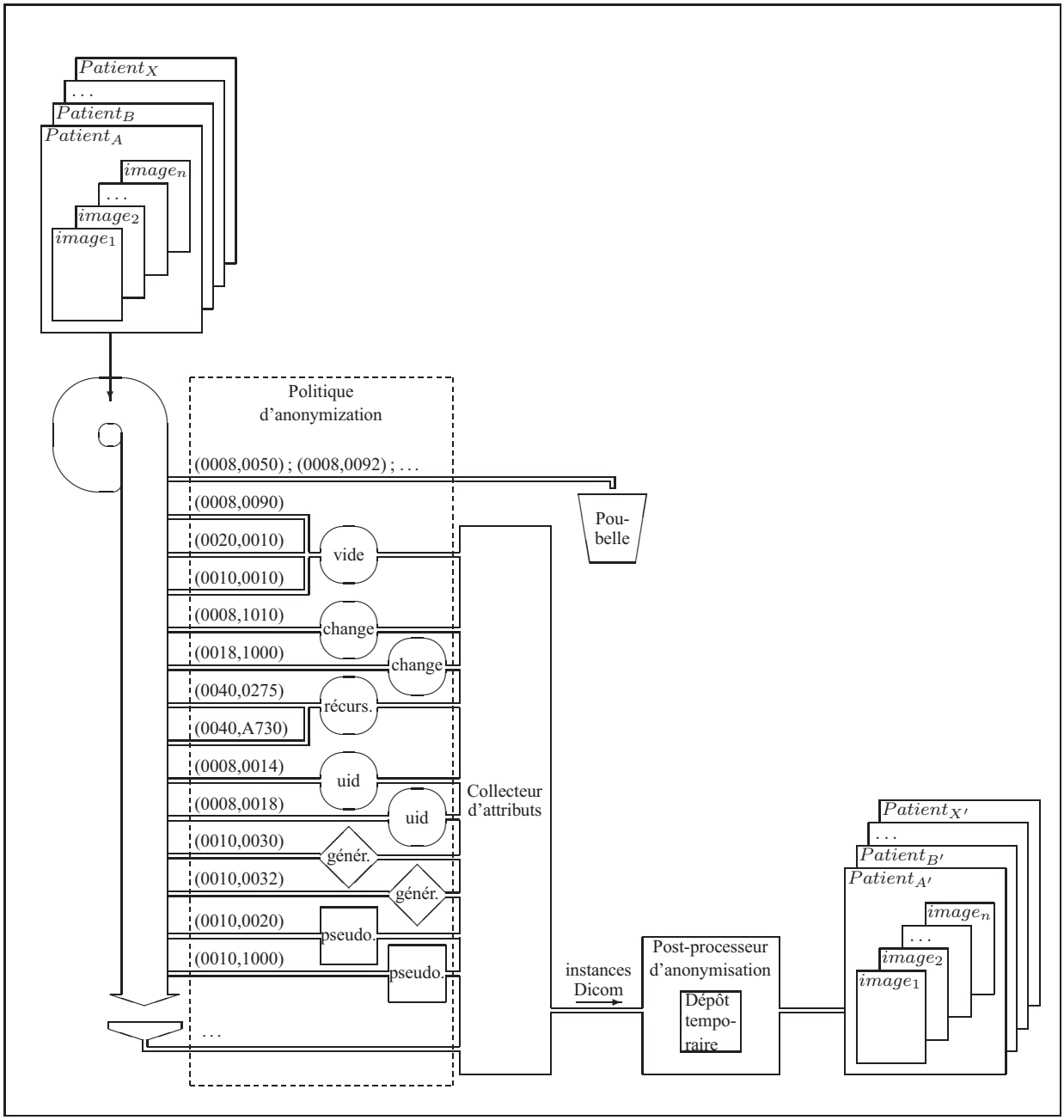


Figure 3.1 Système d'anonymisation.

3.3 L'utilisation d'un API

Tous les détails relatifs au format DICOM, de l'établissement de l'association à l'écriture du fichier dans la banque d'images secondaire, sont pris en charge par un « Application Programming Interface » ou API. Pour manipuler les images DICOM, nous optons pour l'utilisation d'un API programmé en Java. Le choix du langage de programmation pour la réalisation de notre système est évident : il existe bon nombre d'API Java tels que PixelMed, JDCM, dcm4che ou dicom4j. La multitude des API disponibles a justifié l'utilisation des patrons *Interface & Abstract Class* dans le cadre du patron *Adapter* afin de conserver l'indépendance du système avec les classes de l'API employé. Il va sans dire que l'analyse et la conception orientées objets nous a grandement aidé dans la réalisation de ce projet, l'utilisation abondante des patrons de conception orientés objets n'est pas étrangère non plus au choix que nous avons fait de l'approche d'analyse et de conception ainsi que du langage de programmation.

3.4 Les contraintes dictées par le standard DICOM

Les fichiers DICOM possèdent des attributs contenant de l'information nominative permettant d'identifier le patient. C'est pour cette raison qu'il faut y appliquer des algortihmes d'anonymisation de manière à modifier le contenu des attributs dans le but d'assurer la confidentialité du patient. Cependant, nous savons que le standard DICOM possède certaines contraintes comme par exemple l'intégrité référentielle existant entre les UID ou les VR dictant rigoureusement la manière de spécifier la valeur des attributs. Malheureusement, ces contraintes ne nous offrent pas toute la liberté escomptée pour atteindre le niveau d'anonymisation souhaité. Comme nous l'avons spécifié dans notre analyse, pour faire face à ces contraintes, nous devons sélectionner et faire subir à nos attributs des algorithmes d'anonymisation, un par attribut, dont le contenu sera dicté par certains critères comme le type d'anonymisation (désidentification, anonymisation et pseudonymisation), la représentation de valeur qui spécifie la taille, le format et le jeu de caractères acceptés par les différents attributs, le type d'élément de donnée qui stipule si un élément doit être présent ou non dans l'instance et si oui, quelle importance prend sa valeur et qui sont intimement liés aux définitions d'objets d'information (IOD) ainsi que les besoins

spécifiques à la recherche qui peuvent être pris en charges par les différents niveaux formulés à l'intérieur de la politique d'anonymisation.

3.5 La politique d'anonymisation

Il existe cependant des facteurs favorisant notre implémentation et nous pouvons catégoriser ces facteurs comme étant les « niveaux » de la politique d'anonymisation. Le premier point en faveur est la liste connue des attributs à anonymiser présentée dans le supplément 55 du standard DICOM qui nous permet de restreindre le nombre d'algorithmes d'anonymisation à développer au nombre d'attributs présents dans la liste. À moins qu'on ne puisse partager une classe d'anonymisation entre deux attributs à protéger, il doit y avoir au moins une classe d'anonymisation pour chaque attribut de manière à prendre en charge toutes les particularités spécifiques à l'anonymisation de l'attribut auquel il s'applique. Le deuxième point en faveur est l'interface unique aux attributs d'une image DICOM ce qui signifie qu'il sera possible d'établir notre propre interface d'anonymisation et qu'elle pourra également être unique. Troisièmement, les mécanisme de sélection ainsi que la prise en charge des nombreuses variations des algorithmes d'anonymisation est un contexte idéal pour l'application des patrons de conception orientés-objets *Hashed Adapter Objects* et *Strategy*. Certains attributs, parce qu'ils présentent des similitudes ou parce que leur procédure d'anonymisation est vraiment très simple peuvent partager la même classe de stratégie d'anonymisation. L'important est que notre implémentation offre toute la souplesse nécessaire pour tenir compte de toutes les stratégies possibles d'anonymisation de la plus simple à la plus complexe.

3.6 Les patrons de conception utilisés

Concernant l'utilisation des patrons de conception, le système exploite pleinement le patron *Strategy* pour tenir compte des différences qui existent entre les différentes techniques d'anonymisation appliquées aux attributs d'un fichier DICOM et permet l'encapsulation de toute l'information nécessaire à la mise en œuvre de l'algorithme sélectionné.

La politique d'anonymisation est en fait un assemblage entre les stratégies d'anonymisation présentes dans le système et les étiquettes des attributs DICOM faisant partie d'un fichier à anonymiser. Pour tous les attributs du fichier, le système recherhe l'étiquette correspondante dans la table pour trouver la stratégie à appliquer à l'attribut en question. C'est l'utilisation concrète du patron *Hashed Adapter Objects* qui permet non seulement d'améliorer considérablement les performances d'un alogrithme effectuant une sélection parmi un grand nombre d'alternatives, mais qui permet également de réarranger les associations entre les étiquettes et les stratégies afin de formuler de nouvelles politiques d'anonymisation.

La structure hiérarchique des classes découlant de l'interface `IDeidentificationStrategy` est laissée entièrement à la discrétion du développeur. Les classes peuvent être séparées par type d'anonymisation puis, à un deuxième niveau, par valeur de représentation (VR). Des classes abstraites peuvent être insérées à tout emplacement dans la structure pour favoriser la réutilisation de code commun à deux sous-classes (factorisation). Cela peut être utilisé, par exemple, pour les stratégies de désidentification de changement des chaînes longues et courtes (`LongStringChangeDeidentificationStrategy` et `ShortStringChangeDeidentificationStrategy`) qui partagent assurément des similitudes. La classe `Deidentifier` exige seulement que chaque classe concrète fournisse une implémentation de la méthode `deidentify`. L'objet `IDicomElement` encapsule toute l'information nécessaire à la réalisation de son anonymisation. Les classes de type interface `IDeidentificationStrategy` profitent aussi de l'encapsulation parce que grâce à elle, les utilisateurs de la classe concrète n'ont aucune idée des détails ou du degré de complexité de la mise en œuvre de la classe utilitaire.

Pour ce qui est de la pseudonymisation, comme pour toutes les autres stratégies d'anonymisation, il s'agit d'un cas spécial de la modification des attributs. L'objectif est de garantir que toute l'information relative à un sujet dans le secteur primaire devrait appartenir au même sujet anonymisé dans le secteur secondaire. La stratégie de pseudonymisation se concrétise simplement à garantir que le numéro d'identification du patient corresponde toujours du secteur primaire au secteur secondaire toute la durée du traitement du patient. Cela peut être facile-

ment réalisé en conservant une table de correspondance entre les numéro de patient du secteur primaire et les pseudonymes.

Les patrons de conception facilitent la conception et la maintenance des programmes orientés objets. Notre système regorge d'endroits où les problèmes rencontrés sont sujets à l'utilisation de tels patrons. En plus des patrons *Strategy* et *Hashed Adapter Objects*, voici quelques exemples de patrons qui nous ont été forts utiles :

Le patron *Factory Method* sert à encapsuler le processus de sélection et de création des objets de manière à garder les classes utilisatrices (classes clientes) le plus indépendantes possibles des classes des instances créées.

Le patron *Abstract Factory* assure la cohérence entre les classes d'objets instanciées en offrant une interface unique à la création des différents type d'objets qui sont utilisés par le système.

Le patron *Mediator* permet de réduire le couplage (patron GRASP *Low coupling*) entre les différentes classes en rassemblant toutes les dépendances à l'intérieur d'une classe unique responsable des interactions entre les objets (patron *High cohesion*). Ce patron est utilisé principalement par les classes décrivant le fonctionnement des boîtes de dialogue de l'interface utilisateur, mais il pourrait aussi servir à l'intérieur des stratégies d'anonymisation concrètes qui doivent effectuer des opérations complexes d'anonymisation sur les attributs impliquant de nombreux objets.

Le patron *State* permet de séparer toute la logique de transition d'état d'un objet en classe disctintes : une classe par état. De cette manière, il est possible de séparer de façon cohérente les états, les traitements qui s'y rapportent, la gestion des événements et les opérations relatives à la transition vers un autre état. Toute la gestion de l'association avec la modalité est prise en charge de façon cohérente par le patron *State*.

Le patron *Template Method* offre une infrastructure commune à l'ensemble des classes de l'application qui facilite l'écriture de nouvelles classes utilitaires en favorisant la réutilisation de code. Par exemple, le patron de conception *Template Method* sert à factoriser les parties de

code redondants qui se retrouvent couramment dans les classes de boîtes de dialogue ou les classes médiatrices.

Nombreux sont les patrons de conception orientés-objets qui ont été mis à contribution dans ce système et qui ne se retrouvent pas explicitement décrits dans cette section. Ce n'était pas le but non plus d'en faire une liste exhaustive, mais de mentionner les principaux patrons qui ont été utilisés et les plus intéressants. Les patrons de conception plus complexes utilisent abondamment les patrons de conception fondamentaux et ne sont bien souvent qu'un agencement différents des patrons de conception plus simples. Pour le bénéfice du lecteur, les patrons de conception utilisés dans notre système sont présentés plus en détails à l'annexe II à la fin de ce mémoire.

3.7 Technicalités

Il y a trois points dont nous n'avons pas tenu compte dans le cadre de l'élaboration de notre système soit parce qu'ils dépassent les objectifs visés, qu'ils sont trop compliqués à circonscrir ou qu'ils s'avèrent tout simplement insolubles. Il s'agit des points suivants :

- La gestion du consentement du patient ;
- l'information nominative brûlée dans l'image ;
- la reconstitution des images volumétriques en 3D.

La gestion du consentement du patient

L'utilisation de l'information du patient doit être rigoureusement encadrée. Des procédures strictes mises en oeuvre sous le contrôle du patient sont essentielles. Un patient peut autoriser une pseudonymisation de son information clinique à des fins de recherche pour qu'une ré-identification soit possible. Il peut l'autoriser, la révoquer, la limiter à certaines conditions ou non, quelle que soit les résultats de la recherche. Même si nous n'en avons pas tenu compte dans la mise au point de notre système, nous sommes conscients qu'un système prenant en charge le consentement du patient est non seulement important mais essentiel. Le consentement du

patient est en dehors et prédomine toute règle d'anonymisation et ne peut être pris en charge par une politique d'anonymisation telle que nous l'entendons. Il est responsable de la gestion du consentement du patient et permet la ré-identification à l'aide du pseudonyme. Il fait l'objet d'un système très complexe en soi et qui doit être considéré à part.

Les informations nominatives du patient brûlée dans l'image

Pour les modalités telles que les machines à ultrasons qui « brûlent » littéralement l'information nominatives du patient au sein même des pixels formant l'image à l'aide de lettres de plomb ou de tout autre procédé, une attention spéciale doit être portée. Bien que nous n'ayons pas abordé ce problème, une ébauche de solution serait que si l'emplacement et la taille de la région dans laquelle l'information nominative se trouve est connue et constante, elle peut être effacée facilement en changeant la couleur des pixels de cette région pour la couleur du fond de l'image. Si l'emplacement ou la taille de la région varie dans l'image, on peut employer une technique de reconnaissance optique de caractères (OCR) ou l'entrée manuelle de points par l'utilisateur.

Les images volumétriques

Finalement, il existe un défi de taille auquel l'anonymisation des images médicales fait face. Les images volumétriques peuvent être segmentées et affichées en trois dimensions de manière à reconstituer les traits du patient. Même si les données ont été complètement anonymisées, la reconstruction du visage du patient à partir des seuls pixels et ce même partiellement, conduit inévitablement à l'identification du patient.

Conclusion

Dans ce chapitre, nous avons détaillé tous les aspects relatifs à l'implémentation du système Anonym 1.0. Nous avons révisé les contraintes relatives aux objectifs visé et au standard DICOM lui-même pour leur apporter une solution qui intègre efficacité et élégance. La réalisation des différents algorithmes d'anonymisation est prise en charge par l'application directe du pa-

tron *Strategy*. Le patron de conception *Hashed Adapter Objects* offre un mécanisme permettant l'association entre l'attribut à anonymiser et la stratégie à adopter pour chacun d'eux en plus d'un mécanisme de sélection qui n'est pas abordé par le premier patron. Les autres patrons de conception sont présentés sommairement et, pour le bénéfice du lecteur, une présentation plus substantielle se trouve en annexe. Le chapitre se termine par la présentation des points qui n'ont pu être abordés dans le cadre de ce mémoire, mais qui constituent des problèmes intéressants pour la poursuite du projet.

CONCLUSION

Les algorithmes d'analyse et de traitement d'image mis au point par la recherche doivent être testés et validés. Pour cela, les concepteurs de ces algorithmes ont besoin de données provenant du monde réel. Ce besoin sans cesse grandissant de nouvelles données se heurte à la protection de la confidentialité, au droit à la vie privée et au respect du consentement du patient. Une approche offrant une protection raisonable de la confidentialité des informations cliniques du patient tout en donnant accès à une information médicale vaste et de qualité est essentielle. De plus, le suivi d'un patient pouvant s'échelonner sur une période plus ou moins longue, il serait bon de faire bénéficier la recherche de l'information découlant de la progression de la maladie et de l'évolution du diagnostic. Pour cela, il faut permettre la mise à jour incrémentale de l'information par l'ajout successif de nouvelles données.

Nous avons d'abord commencé par étudier les différentes opérations d'anonymisation qui sont à notre disposition : il y a la désidentification, l'anonymisation et la pseudonymisation. La désidentification est vue comme le moyen d'assurer la confidentialité de l'information en enlevant toute l'information nominative pouvant mener de manière directe ou indirecte à l'identification du patient. La désidentification ne pouvant suffire à anonymiser pleinement l'information, l'anonymisation (ou la généralisation) agit comme une technique complémentaire à la première. Cette technique consiste principalement à rendre l'information d'un patient indistinguable des données d'un autre patient. Les notions de quasi-identificateurs et d'attibuts sensibles sont introduites. Un quasi-identificateur étant un assemblage d'attributs pouvant mener à l'identification d'un individu dans un ensemble de données. Un attribut sensible étant un attribut ne faisant pas partie d'un quasi-identificateur mais qui peut mener à l'identité du patient à l'intérieur d'une classe d'équivalence. Une classe d'équivalence est un groupe d'enregistrements qui ont tous la même valeur de quasi-identificateur. Nous avons par la suite appris qu'un ensemble de données est considéré comme anonymisé s'il respecte les modèles de k-anonymat et de ℓ-diversité. Ces modèles stipulent que chaque élément doit être indistinguable de $k-1$ autres éléments dans l'ensemble par rapport à la valeur de son quasi-identificateur et de $\ell-1$ autres éléments par rapport à son (ses) attribut(s) sensible(s) à l'intérieur de la même classe

d'équivalence. Ceci afin de conserver la probabilité d'identifier un patient inférieure à $1/k$ pour son quasi-identificateur et $1/\ell$ pour ses attributs sensibles. Les travaux de Byun et al. [3] nous ont rapidement fait comprendre que s'ils s'appliquent parfaitement aux ensembles de données statiques, ces modèles ne peuvent s'appliquer aux ensembles de données mis à jour de façon incrémentale sans tenir compte de la possibilité de voir apparaître des canaux d'inférence à chaque nouvelle mise à jour. Les canaux d'inférence étant une façon dont dispose l'adversaire pour identifier précisément à qui appartient l'information en comparant les différentes versions d'ensemble de données. La troisième et dernière technique d'anonymisation est la pseudonymisation. La pseudonymisation consiste à changer un identificateur unique du patient par un pseudonyme mais sans qu'il y ait aucun lien avec la personne réelle. Cette technique est utilisée lorsque l'information provient de sources multiples ou lorsqu'elle devient disponible à des moments précis dans le temps. Les pseudonymes sont réversibles ou à sens unique. Les pseudonymes réversibles permettent, si le besoin s'en fait sentir et avec son consentement, de retracer l'identité du patient. Ils sont implémentés à l'aide d'une table de correspondance (« mapping ») ou d'un algorithme de chiffrement à clé secrète. Les pseudonymes à sens-unique, quant à eux, empêchent toute possibilité de ré-identification du patient. Leur implémentation se fait à l'aide d'une fonction de hachage.

Suite à notre analyse, il s'avère qu'un attribut d'une image DICOM peut subir l'un des trois traitements suivants : il peut être copié tel quel dans l'instance secondaire, être ignoré (ce qui résulte en son absence ou sa suppression de l'instance secondaire) et il peut subir une modification de sa valeur avant d'être copié dans l'instance secondaire. Toutes les opérations du processus d'anonymisation appliqué à un attribut d'une instance DICOM constituent une altération de la valeur de l'attribut en question. Parmi ces modifications il y a 1) de le mettre à vide (le remplir de caractères blancs), 2) lui affecter une valeur constante arbitraire, 3) de l'anonymiser de façon récursive (dans le cas des séquences d'items), 4) de lui affecter un UID cohérent, 5) de généraliser sa valeur (opération d'anonymisation) et 6) de lui affecter un pseudonyme (afin de permettre le suivi du patient à long terme). Chaque attribut devant subir une opération du processus d'anonymisation est pris en charge par un algorithme différent. Si possible,

nous chercherons à faire en sorte qu'un algorithme puisse servir à plus d'un attribut. Le pire des cas consiste à devoir écrire un algorithme différent pour chaque attribut. La classification des attributs permet de déterminer l'alogrithme d'anonymisation qu'un attribut doit subir. On sépare premièrement les attributs en fonction du type d'anonymisation qu'il doit subir : est-ce que l'attribut doit être désidentifié, anonymisé (sa valeur doit être généralisée) ou pseudonymisé ? Le type d'anonymisation est le premier critère de classification. Le deuxième critère consiste en la représentation de valeur (« Value Representation ») ou VR qui définit la taille, le format et l'ensemble des caractères qui sont acceptés par l'attribut. Le troisième critère correspond au type d'élément de données (« data element type ») qui spécifie si la présence d'un attribut est requise et, si elle l'est, quelle importance prend sa valeur dans l'instance. Les deux derniers critères se rapportent à la définition d'objet d'information (« Information Object Definition ») ou IOD et aux besoins spécifiques dictés par le domaine de recherche. Les facteurs favorisant notre conception sont que les attributs à protéger sont connus (parce que mentionnés dans le supplément 55 de DICOM), ils possèdent une interface unique dont nous saurons tirer profit et, finalement, l'analyse, la conception et la programmation orientée objet (AAOD) nous permettent de tenir compte des spécificités du problème par les classes et, si cela s'avère nécessaire, par les instances.

Notre application est capable d'accepter de images médicales à l'aide du standard DICOM. Nous avons choisi cette approche pour faciliter l'intégration avec les systèmes médicaux existants. Nous faisons usage d'une interface de programmation applicative (« Application programming interface ») ou API pour tout ce qui concerne le standard DICOM. Toutes les classes utilitaires sont néanmoins dotées d'adaptateurs afin d'assurer l'indépendance de notre application vis-à-vis l'API choisie. La politique d'anonymisation est implémentée à l'aide des patrons de conception *Strategy* et *Hashed Adapter Objects*, ce qui assure toute la souplesse nécessaire pour déterminer les opérations du processus d'anonymisation à appliquer aux différents attributs pouvant être présents dans l'instance. Les concepts orientés-objets tels que l'encapsultation et le polymorphisme sont pleinement exploités pour, dans un premier temps, fournir toute l'information nécessaire à la réalisation de l'opération tout en dissimulant la complexité

d'implémentation qu'il y a à l'arrière-scène et, dans un second temps, sélectionner un algorithme spécifique pour effectuer l'anonymisation d'un attribut en particulier. D'autres patrons de conception sont utilisés pour permettre la réalisation de ce projet. Finalement, nous mentionons certains défis qu'il nous reste à surmonter.

Ceci complète la rédaction de ce mémoire. Il décrit une solution fonctionnelle aux problèmes liés à l'anonymisation des données médicales dans le but d'en faire usage dans le domaine de la recherche. Il est aussi question de le mise à jour incrémentale de l'information. La section suivante fait état des recommandations pour la suite du projet.

RECOMMANDATIONS

Mes recommandations sont les suivantes :

- Étudier les concepts de ℓ-diversity et de k-anonymity car il s'agit d'une avenue prometteuse pour le projet. De nouveaux articles sont publiés à chaque semaine sur le sujet. Voir comment il peuvent être appliqués dans le cadre des images Dicom.
- Compléter le développement de l'application Anonym 1.0. Le code source du programme est disponisble sur le site SourceForge.net à l'URL http ://acronym.sourceforge.net
- Envisager l'utilisation d'un modèle relationnel plus complet pour représenter les images Dicom et ainsi manipuler l'information comme de gros tableaux. Voir ce que le standard Dicom propose à ce sujet.
- Tester l'application en milieu médical pour vérifier son bon fonctionnement.
- Intégrer dcm4che au logiciel Anonym 1.0. Il s'agit d'une implémentation Dicom développée en Java et disponible en logiciel libre (http ://www.dcm4che.org/). Il ne s'agit en fait que d'une nouvelle implémentation pour les interfaces mises au point dans le système.

ANNEXE I

TABLEAUX

Tableau I.1 Attributs à protéger pour respecter le profil de confidentialité au niveau de l'application
Adapté du supplément 55 (2002, p. 13)

Attribute Name	Étiquette	Type	Attribute Description
Instance Creator UID	(0008,0014)	3	Identifie le dispositif qui a créé l'instance.
SOP Instance UID	(0008,0018)	1	Identifie une instance SOP spécifique.
Accession Number	(0008,0050)	3	Identifie la demande de service d'imagerie pour cette procédure.
Institution Name	(0008,0080)	1C	Institution ou organisation à laquelle la personne identifiée est sous la responsabilité.
Institution Address	(0008,0081)	3	Adresse postale de l'institution ou de l'organisation à laquelle la personne identifiée est sous la responsabilité.
Referring Physician's Name	(0008,0090)	2	Nom du médecin traitant.
Referring Physician's Address	(0008,0092)	3	Adresse du médecin traitant.
Referring Physician's Telephone Numbers	(0008,0094)		Numéro de téléphone du médecin traitant.
Station Name	(0008,1010)	1C	Nom de l'équipement ayant servi à produire l'instance.
Study Description	(0008,1030)	3	Description de l'étude générée par l'institution.
Series Description	(0008,103E)	3	Description de la série fournie par l'institution.
Institutional Department Name	(0008,1040)	3	Nom du département où l'équipement est situé.
Physician(s) of Record	(0008,1048)	3	Noms des médecins responsables de l'ensemble des soins à donner au patient au moment de l'étude.
Performing Physicians' Name	(0008,1050)	3	Noms des médecins administrateurs de la série.
Name of Physician(s) Reading Study	(0008,1060)	3	Noms des médecins lors de la lecture de l'étude.
Operators' Name	(0008,1070)	1C	Noms des opérateurs pour la série.

Tableau I.1 Attributs à protéger pour respecter le profil de confidentialité au niveau de l'application (suite)

Attribute Name	Étiquette	Type	Attribute Description
Admitting Diagnoses Description	(0008,1080)	3	Description du diagnostic d'admission.
Referenced SOP Instance UID	(0008,1155)	1	Permet d'identifier l'instance SOP.
Derivation Description	(0008,2111)	3	Description de la manière dont l'image a été dérivée.
Patient's Name	(0010,0010)	2	Nom complet du patient.
Patient ID	(0010,0020)	2	Numéro du patient.
Patient's Birth Date	(0010,0030)	2	Date de naissance du patient.
Patient's Birth Time	(0010,0032)	3	Heure de naissance du patient.
Patient's Sex	(0010,0040)	2	Sexe du patient. (valeurs possibles : M = mâle ; F = femelle ; O = autre [other])
Other Patient Ids	(0010,1000)	3	Autres numéros servant à identifier le patient.
Other Patient Names	(0010,1001)	3	Autres noms servant à identifier le patient.
Patient's Age	(0010,1010)	3	Âge du patient.
Patient's Size	(0010,1020)	3	Taille du patient en mètres.
Patient's Weight	(0010,1030)	3	Poids du patient en kilogrammes.
Medical Record Locator	(0010,1090)	3	Référence au dossier physique du patient.
Ethnic Group	(0010,2160)	3	Groupe ethnique du patient.
Occupation	(0010,2180)	3	Occupation du patient.
Additional Patient's History	(0010,21B0)	3	Information additionnelle concernant l'historique médical du patient.
Patient Comments	(0010,4000)	3	Commentaire additionnel au sujet du patient.
Device Serial Number	(0018,1000)	1C	Numéro de série de l'équipement ayant produit les sources.
Protocol Name	(0018,1030)	1C	La description des conditions dans lesquelles la série a été réalisée.
Study Instance UID	(0020,000D)	1	Identifie l'étude.
Series Instance UID	(0020,000E)	1	Identifie les séries.
Study ID	(0020,0010)	2	Identifie l'étude générée par l'équipement.
Frame of Reference UID	(0020,0052)	1	Identifie le cadre de référence de la série.

Tableau I.1 Attributs à protéger pour respecter le profil de confidentialité au niveau de l'application (suite)

Attribute Name	**Étiquette**	**Type**	**Attribute Description**
Synchronization Frame of Reference UID	(0020,0200)	1	Identifie l'environnement de synchronisation.
Image Comments	(0020,4000)	3	Commentaire sur l'image.
Request Attributes Sequence	(0040,0275)	3	Séquence qui contient les attributs de la demande de service d'imagerie.
UID	(0040,A124)	1C	C'est la valeur de l'élément de contenu.
Content Sequence	(0040,A730)	1C	Séquence de contenu.
Storage Media File-set UID	(0088,0140)	3	Permet d'identifier un support de stockage sur lequel les instances SOP résident.
Referenced Frame of Reference UID	(3006,0024)	1	Identifie de façon unique le cadre de référence dans lequel la région d'intérêt est définie.
Related Frame of Reference UID	(3006,00C2)	1	Cadre de référence du système de coordonnées.

ANNEXE II

PATRONS DE CONCEPTION ORIENTÉS-OBJETS

Les patrons de conception présentés dans cet annexe sont inspirés des références suivantes : [10, 12, 14, 18]

1 Le patron de conception *Factory Method*

Le patron *Factory Method* sert à une classe utilisatrice d'objets à créer les objets qu'elle utilise. Ce patron de conception garantit l'indépendance entre les classes Client et les classes Produit en fournissant un type d'objet responsable de la sélection et l'instanciation des classes Produit.

Pour illustrer ce concept, supposons une classe `Client` faisant usage d'objets de la classe `Product` telle que présenté à la figure II.1.

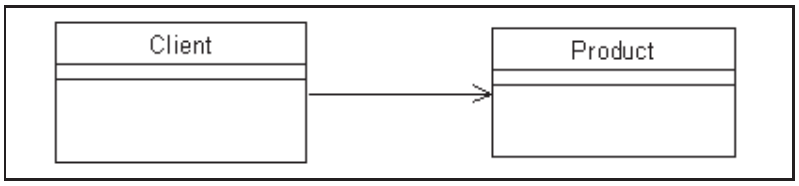


Figure II.1 Association Client-Product.

Si on veut assurer l'indépendance entre ces deux classes, la première chose à faire est d'ajouter à notre système une interface `IProduct` et peut-être une classe abstraite séparant l'interface des classes concrètes. Cette classe abstraite, appelée `AbstractProduct`, pourra contenir tous les services communs aux classes qui, autrement, devraient être répliqués dans toutes les classes `Product`. De cette manière, la classe `Client` n'a aucune idée de quoi sont faites les classes `Product` ne connaissant d'elles que l'interface qu'elles implémentent. Cette organisation des classes est présentée à la figure II.2.

L'application du patron de conception *Interface And Abstract Class* tel qu'énoncé précédemment vient assurer l'indépendance entre les classes Client et les classes Produit. Il n'a cepen-

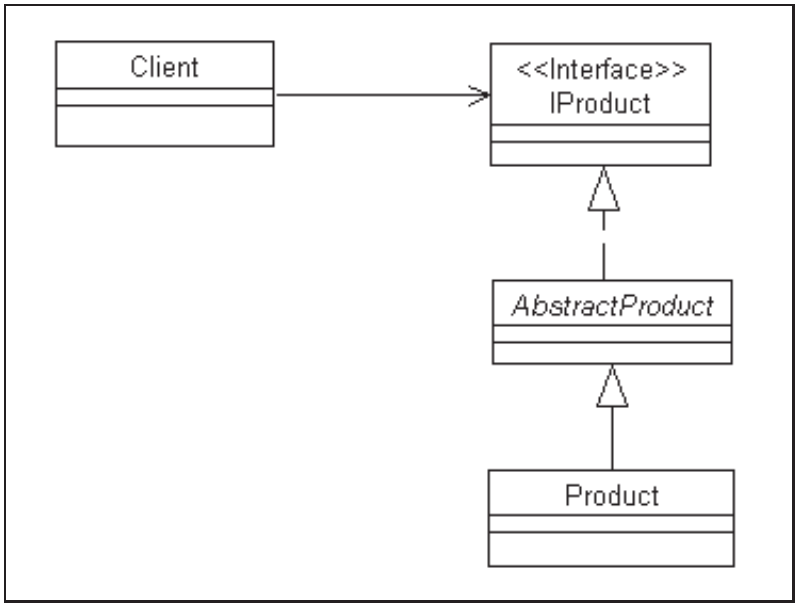


Figure II.2 Client-Product framework.

dant jamais été question de l'opération de construction des objets Produit. Comment demeurer indépendant d'une classe à tel point qu'on ne devrait même pas la connaître lors de son instanciation ? Rien de la figure II.2 ni de la discussion précédente n'expliquent comment l'objet `Client` crée les instances de Produit sans être lui-même rattaché à classe qu'il s'apprête à instancier.

La manière d'assurer cette indépendance consiste à fournir une classe encapsulant la logique relative à la sélection et à l'instanciation des classes concrètes. En utilisant l'organisation présentée à la figure II.3, un objet `Client` appelle la méthode `createProduct` de l'interface `IProductFactory` qui retourne un objet de type interface `IProduct`.

Les caractéristiques du patron *Factory Method* est qu'il doit permettre la construction d'objets `Produit` sans créer aucun lien de dépendance avec les classes qui composent ces produits. Il doit, de plus, faciliter l'ajout de nouvelles classes à mesure qu'elles apparaîssent.

Voici la description des différentes classes et interfaces faisant parties du modèle général du patron *Factory Method* :

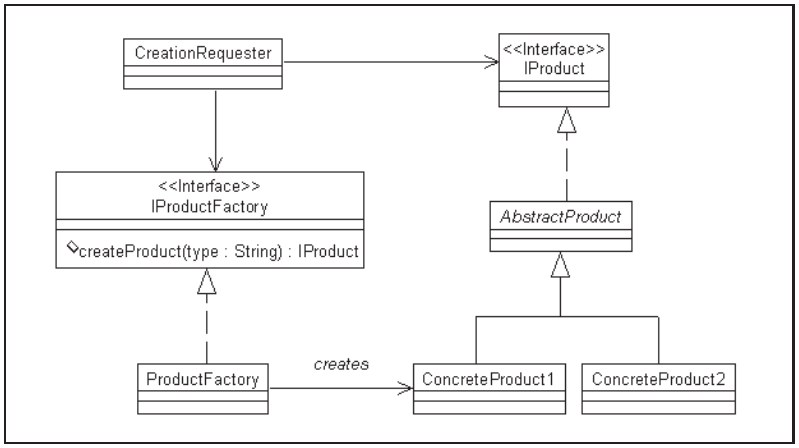


Figure II.3 Organisation générale du patron de conception *Factory Method*.

IProduct. Les objets créés en utilisant ce patron de conception doivent implémenter cette interface.

ConcreteProduct1, ConcreteProduct2, . . . Les classes jouant ce rôle sont instanciées par les objets `Factory`. Elles implémentent l'interface `IProduct`.

CreationRequester. Une classe jouant ce rôle est une classe indépendante de l'application qui veut créer une classe spécifique à l'application. Elle fait cela à partir d'un objet provenant d'une classe implémentant l'interface `IProductFactory`.

IProductFactory. Les objets qui créent les objets `IProduct` au nom des objets `CreationRequester` doivent implémenter cette interface. Les interfaces remplissant ce rôle ont bien souvent le mot « Factory » dans leur nom.

ProductFactory. C'est une classe spécifique à l'application qui implémente l'interface `IProductFactory` appropriée et qui a une méthode pour créer des objets `ConcreteProduct`. Les classes jouant ce rôle auront typiquement un nom contenant le mot « Factory », tel que `DocumentFactory` ou `ImageFactory`

Il existe principalement 2 variations du patron *Factory Method*. Il y a le cas général où la classe de l'objet est déterminée au moment de la création de l'objet. Il y a aussi le cas moins fréquent où la classe de l'objet qui sera créés est toujours la même et est déterminée avant que la création de l'objet ne soit initiée.

Un programme peut utiliser un objet *factory* pour créer une instance de la même classe s'il est configuré pour le faire. Dans ce cas, la méthode `createProduct` n'a besoin d'aucun paramètre puisqu'elle renvoie toujours le même type d'objet.

Dans le cas de la détermination de la classe par les données, le choix de la classe à instancier est guidé par l'information que l'objet doit encapsuler. On doit donc passer un paramètre à la méthode `createProduct` représentant la classe à instancier. Ce genre de méthode `createProduct` prend souvent la forme d'un gros *switch-case* ou d'un chaîne de *if-then-else*. Cette séquence de *if* fonctionne bien dans la mesure où le nombre de classes qu'il est possible d'instancier est connu à l'avance . Si le nombre de classes à instancier est très grand ou s'il n'est pas connu à l'avance, on peut utiliser le patron *Hashed Adapter Objects* pour palier à ce problème.

Noter qu'il est approprié de retrouver ce genre de structure *switch-case* à l'intérieur de la méthode `createProduct` d'une `ProductFactory`. Normalement, en conception orientée objet, ce genre de structure représente un défaut de conception et devrait plutôt être implémentée à l'aide du polymorphisme, mais cela implique qu'un objet a déjà été créé. Le patron *Factory Method* ne peut être implémenté en utilisant le polymorphisme car aucun objet n'a encore été créé.

Enfin, la détermination de la classe par les données peut se faire à plusieurs niveaux. Dans ce cas, il y a une `Factory` responsable de créer une autre classe *factory* qui, elle, sert à créer les produits finaux.

Un point négatif par rapport au patron *Factory Method* est qu'il engendre un (ou des) niveau(x) d'indirection entre l'objet qui initie la construction d'un objet et la classe qui instancie la classe dans les faits ce qui peut rendre le programme plus difficile à comprendre.

L'exemple suivant montre un endroit dans le code de l'application Anonym 1.0 où il est fait usage du patron de conception *Factory Method*.

Dans cet exemple, il est question de la manière dont est prise en charge toute la politique de désidentification d'un fichier DICOM. Dans cette partie du système, on associe à une politique de désidentification un ensemble de règles qui, appliquées à un fichier DICOM, le dissocient du patient à qui l'information appartient. L'organisation des classes impliquées dans l'opération de désidentification est présentée à la figure II.4.

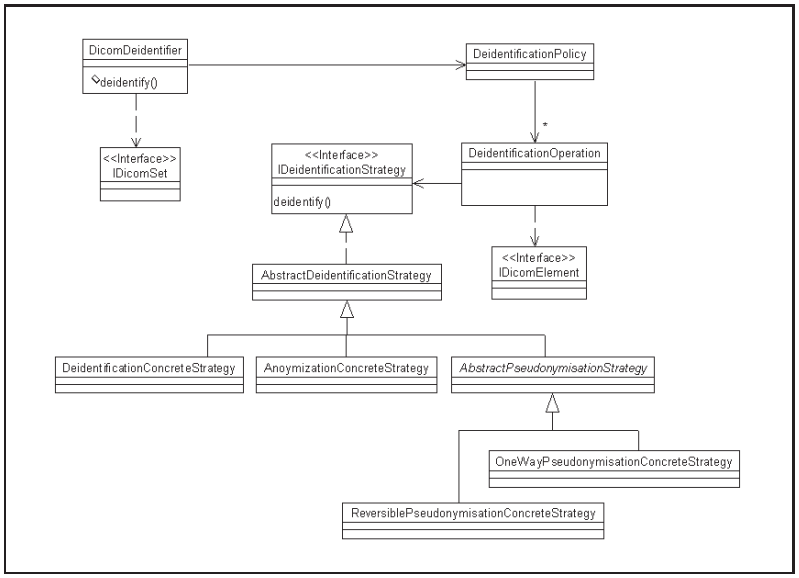


Figure II.4 Diagramme de classe du système de désidentification.

La classe `DicomDeidentifier` prend un objet de type interface `IDicomSet` provenant d'un fichier DICOM et lui applique une politique de désidentification représentée par la classe `DeidentificationPolicy`. La classe `DeidentificationPolicy` est une aggrégation d'objets de classe `DeidentificationOperation` qui associent une stratégie de désidentification à chaque élément DICOM (représenté ici par l'interface `IDicomElement`).

La partie qui nous intéresse concerne les stratégies de désidentification appliquées aux éléments DICOM. Les stratégies de désidentification sont basées sur le patron *Strategy* que nous expliquerons plus tard. Pour l'instant, ce qu'il faut comprendre c'est que chaque stratégie de désidentification est représentée par une classe différente responsable d'une stratégie donnée.

Les stratégies de désidentification actuellement implémentées sont :

Remplacement par des blancs. Consiste à conserver l'élément dans le fichier DICOM mais avec une valeur vide.

Changement de la valeur. Cette stratégie modifie la valeur de l'élément DICOM par une autre valeur.

Pseudonymisation réversible à l'aide de DES. Stratégie qui consiste à générer un pseudonyme réversible à l'aide de l'algorithme de chiffrement DES.

Effacement de l'élément DICOM. Cette stratégie retire l'élément du fichier DICOM.

Génération d'un nouvel UID. Cette stratégie est appliquée aux UID du fichier DICOM et consiste à générer un nouvel UID pour remplacer l'élément en question.

Les opérations de désidentification sont différentes dépendant de chaque politique de désidentification. Les classes qui déterminent quelle stratégie de désidentification appliquer à chaque élément DICOM ne sont pas connues par la classe responsable d'appliquer la politique de désidentification. Tout ce que cette classe connaît est que chaque stratégie implémente une interface `IDeidentificationStrategy` fournissant une méthode `deidentify` prenant en paramètre un objet de type interface `IDicomElement` et un de `IDicomSet` représentant l'ensemble des éléments DICOM auquel l'élément appartient.

À priori, la procédure de désidentification pour le fichier DICOM est indépendante des classes implémentant les stratégies de désidentification et cette indépendance est poussée jusqu'à la construction des objets stratégies. En effet, afin d'assurer une indépendance totale entre la classe qui applique la procédure de désidentification au fichier DICOM et les stratégies elles-mêmes, on délègue une classe de type interface `IDeidentificationStrategy` la responsabilité de création des instances de classes stratégies. Une implémentation de l'interface est représentée par la classe `DeidentificationStrategyFactory`. Ainsi, on peut ajouter de nouvelles classes stratégies, les modifier, sans influencer la classe responsable de la désidentification. On peut également changer la classe *factory* dynamiquement pour donner accès à un nouvel ensemble de classes.

Cette organisation des classes est présentée à la figure II.5.

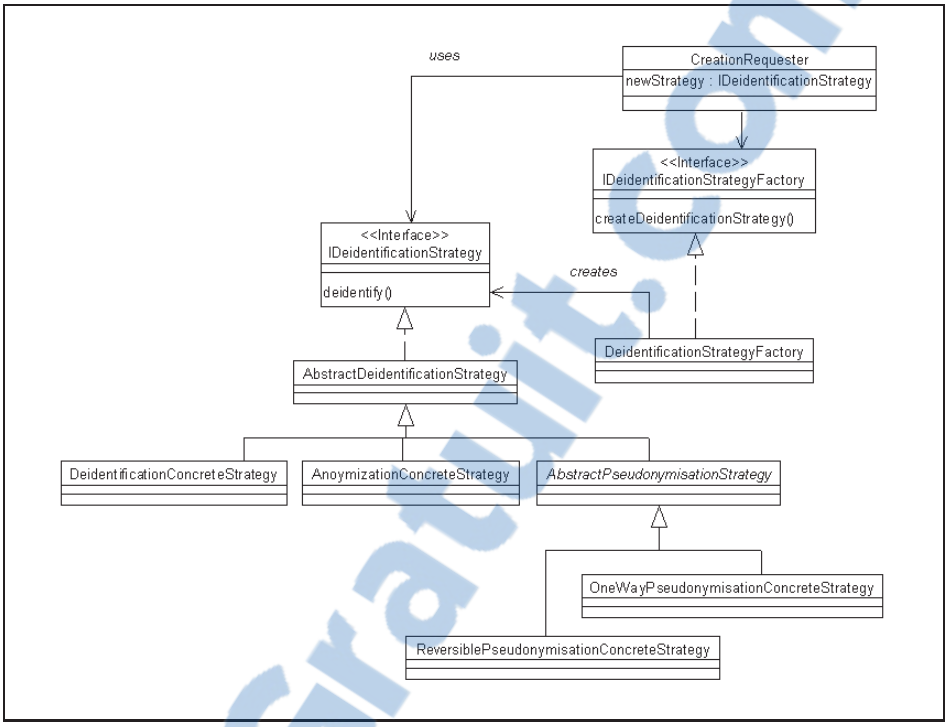


Figure II.5 Patron *Factory Method* des stratégies de désidentification.

En utilisant cette organisation de classes, l'objet de type `CreationRequester` délègue à un objet de type interface `IDeidentificationStrategyFactory` la charge de sélectionner et d'instancier les classes de type `IDeidentificationStrategy`. De cette façon, on vient couper le lien de dépendance qui pourrait exister entre l'objet `CreationRequester` et les classes de type interface `IDeidentificationStrategy`.

2 Le patron de conception *Abstract Factory*

Lors du développement d'un logiciel, il arrive souvent qu'on ait besoin d'utiliser une librairie d'une tierce partie. Ces librairies sont conçues de façon très génériques et offrent une panoplie de classes utilitaires paramétrisables capables de s'adapter aux tâches à réaliser dans le cadre spécifique d'une application. L'utilisation de ces librairies est à ce point importante qu'il serait improbable qu'un projet voit le jour sans leur présence vu la quantité impressionnante de travail qu'elle permettent d'économiser. C'est notamment le cas pour les applications interagissant avec des bases de données, les interfaces utilisateur graphiques (GUI) évoluées ou dans des domaines très spécifiques comme l'utilisation du standard DICOM. Il arrive également qu'on doive offrir la possibilité d'utiliser des librairies similaires provenant de fabricants différents. C'est le cas lorsqu'on doit supporter différents SGBD (Oracle, DB2, PostgresQL, MySQL, etc), différents GUI (Windows, Motif, MacOS X, etc) ou différentes implémentations de librairies Java DICOM (JDCM, PixelMed ou dcm4che). Ces librairies offrant sensiblement le même sous-ensemble de services, on peut facilement mettre en facteur des interfaces communes permettant de les utiliser toutes de manière transparente à l'intérieur d'une même application.

C'est exactement ce qu'on a tenté de faire lors du développement de l'application Anonym 1.0 dans le cadre de ce mémoire de maîtrise. En effet, dans ce projet, on a tenu compte du fait que le logiciel pourrait devoir travailler avec différentes implémentations de librairies servant à interagir avec les structures de données DICOM. Cette possibilité fut offerte pour permettre de comparer les différents produits disponibles sur le marché. Pour commencer, il a été convenu de ne permettre l'utilisation des produits PixelMed et JDCM, étant donné que la mise en place des mécanismes pour supporter ces deux fabricants ouvrirait la porte à toutes les autres implémentations. Chaque implémentation possèdent ses avantages et ses inconvénients comme par exemple JDCM est payante alors que PixelMed est gratuite, JDCM offre une granularité plus fine alors que PixelMed implémente des fonction à des niveaux plus élevés, etc. Il n'en demeure néanmoins que les deux implémentations offrent sensiblement la même gamme de services et

de structures qu'il est possible d'utiliser les deux implémentations de façon cohérente à partir d'interfaces communes. Ces interfaces sont présentées à la figure II.6.

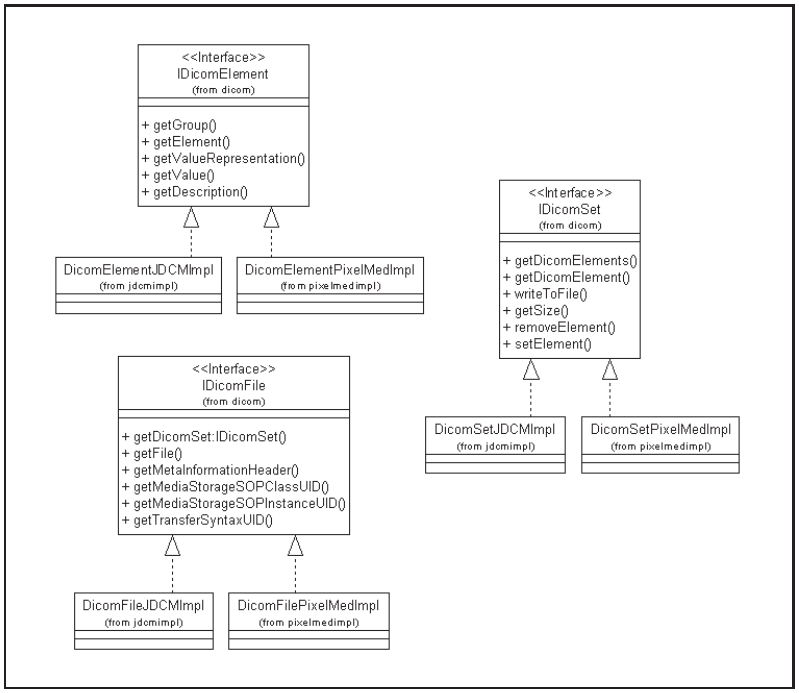


Figure II.6 Les classes DICOM.

Évidemment, cette flexibilité implique certaines contraintes de conception qui ne sont pas triviales à prime abord :

- Il faudra premièrement prévoir à mettre en place l'ensemble des interfaces, appelons-les produits, présentées auparavant et déterminer la gamme de services communs à toutes les implémentations qui seront couverts par ces interfaces ;
- il faudra développer de telle sorte qu'un système travaillant avec plusieurs familles de produits soit en mesure de fonctionner de façon transparente par rapport aux produits spécifiques qu'il utilise ;
- un système est configuré pour ne travailler qu'avec une famille de produits spécifiques à la fois ;
- les produits d'une même famille sont prévus pour être utilisés ensemble et l'on doit s'assurer que cette contrainte est scrupuleusement respectée ;

Le patron de conception *Abstract Factory* s'applique parfaitement à ce genre de situation. Comme le montre la figure II.7, le patron *Abstract Factory*, définit un ensemble d'interfaces qui sont directement manipulées par la classe client et qui assurent à ce dernier une indépendance totale avec la spécificité des classes qu'il utilise. L'interface `IAbstractFactory` offre une gamme complète de méthodes `create` qui, implémentées par les différentes classes `ConcreteFactory`, donne au client la possibilité de créer les différents produits d'une même famille de produits et ce, de manière robuste et cohérente. En effet, tous les produits générés par les méthodes `create` d'une même `ConcreteFactory` sont garantis faire partie de la même famille. L'indépendance des produits face au client est assurée par les interfaces qui sont implémentées par les différents produits.

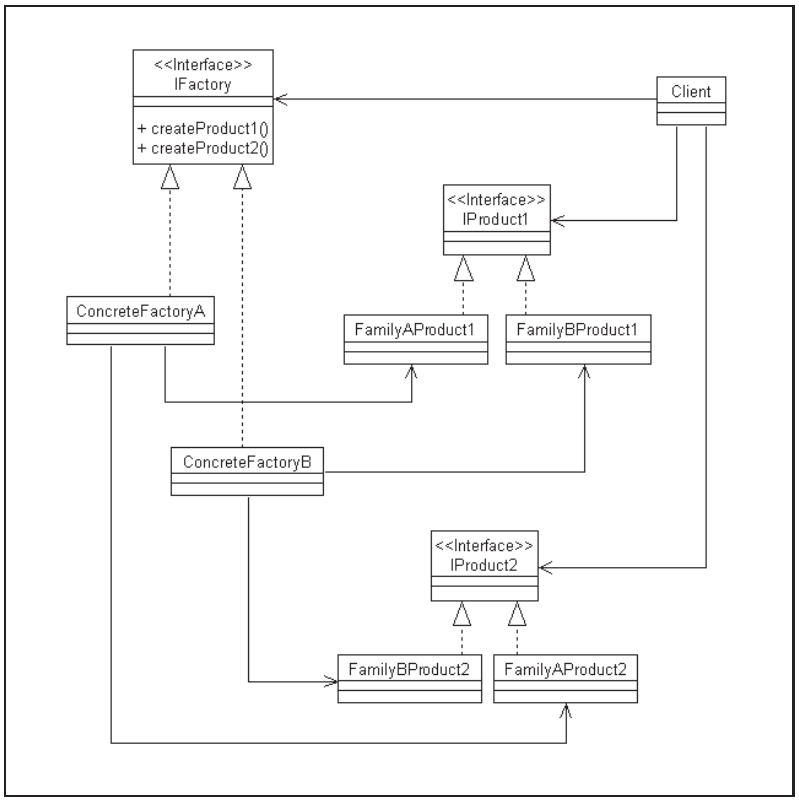


Figure II.7 Le patron *Abstract Factory*.

La figure II.8 montre l'implémentation du parton *Abstract Factory* dans le cadre du projet Anonym 1.0. Le rôle des produits est joué par les classes implémentant les interfaces `IDicomFile`, `IDicomSet`, `IDicomElement` et `IDicomUIDGenerator`. Les deux familles de produits cor-

respondent aux implémentations de librairies DICOM qui sont actuellement supportées, soit JDCM et PixelMed. Les classes implémentant l'interface `IDicomAbstractFactory` jouent le rôle de fabriques concrètes et permettent de créer les produits spécifiques à chacune des librairies disponibles.

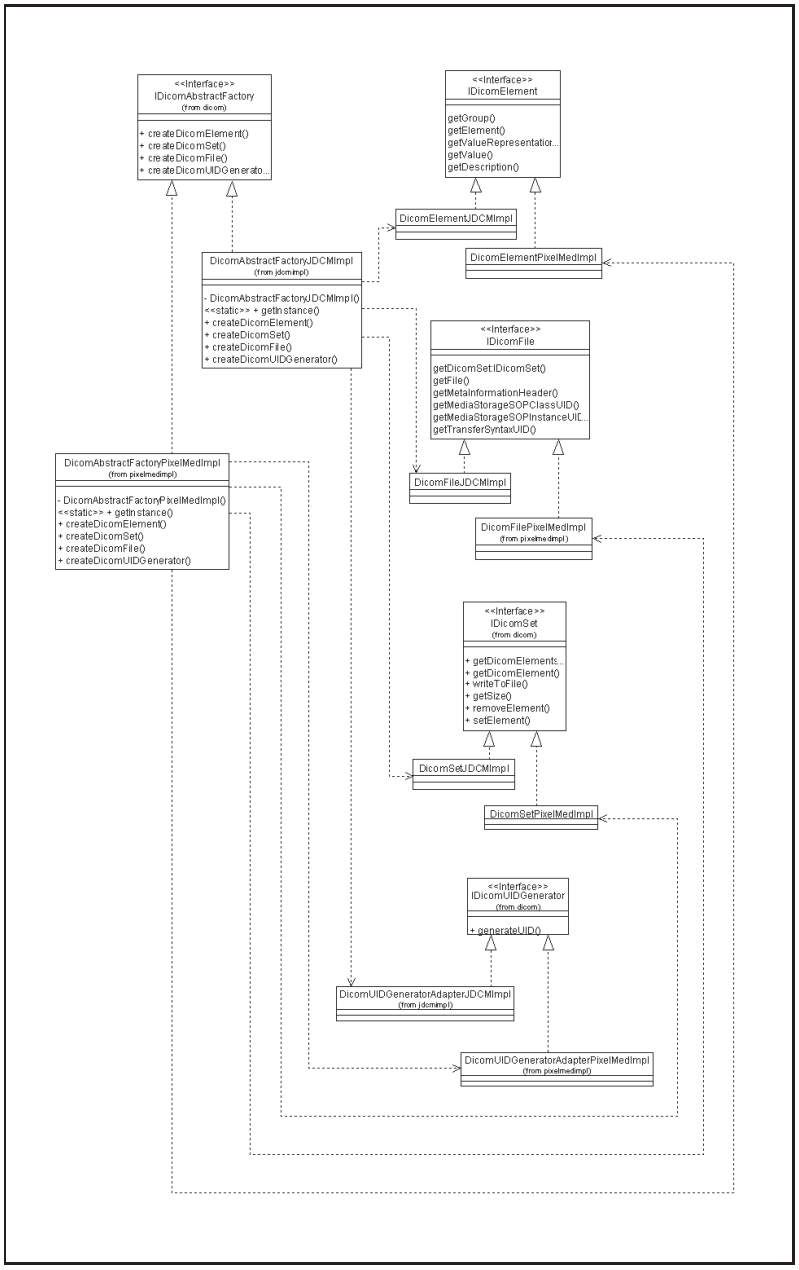


Figure II.8 Diagramme de classe du patron *Abstract Factory* dans Anonym 1.0.

Une telle solution isole les classes concrètes et organise les dépendances entre les classes de manière à rendre la maintenance plus facile. Il est également plus facile de changer de famille de produits : dans notre cas, changer le paramètre general.dicom.scp.implementation.provider.id à la valeur « JDCM » ou « PixelMed » change la librairie qui est utilisée au démarrage de l'application. Le patron *Abstract Factory* permet de s'assurer que les objets qui sont créés font tous partie de la même famille de produits. Un des aspects négatifs de ce patron est qu'il rend plus complexe l'ajout de nouveaux produits étant donné qu'il faille fournir une méthode différente par produit dans chacune des fabriques concrètes. Ajouter une nouvelle famille de produits n'est pas une mince affaire non plus dans la mesure où il faille fournir une implémentation différente pour chacune des interfaces de produits disponibles en plus de développer la nouvelle fabrique concrète.

3 Le patron de conception *Mediator*

Le patron de conception *Mediator* sert à gérer les changements d'état entre des objets de différentes natures. Il sert à régler les problèmes qui surviennent souvent lors de la mise au point des boîtes de dialogue. Par exemple, la boîte de dialogue présentée à la figure II.9.

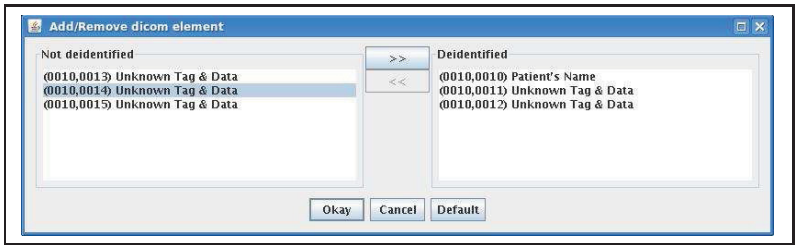


Figure II.9 La boîte de dialogue d'ajout/retrait des élément DICOM.

Cette boîte de dialogue sert à ajouter ou à retirer des éléments DICOM de la politique de désidentification.

- Quand la boîte de dialogue s'ouvre en premier, aucun élément des deux listes « Not deidentified » et « Deidentified » ne sont sélectionnés. Les deux boutons « » » et « « » sont désactivés.
- Lorsqu'on sélectionne un élément d'une des listes, tout ce qui avait éventuellement été sélectionné dans l'autre liste retombe dans un mode non-sélectionné.
- Le bouton « » » ne s'active que si un des éléments de la liste *Not deidentified* est sélectionné.
- Le bouton « « » ne s'active que si un des éléments de la liste *Deidentified* est sélectionné.

Si tous les objets prennent la responsabilité des dépendances qui existent avec les autres objets, il en résulte une organisation d'objets fortement couplés avec une cohésion faible. La figure II.10 montre les relations qui unissent les objets entre eux.

Lorsqu'on considère la figure II.10, à cause de ces nombreux liens de dépenance, on s'aperçoit que la boîte de dialogue est difficile à maintenir. Un programmeur chargé de la maintenance

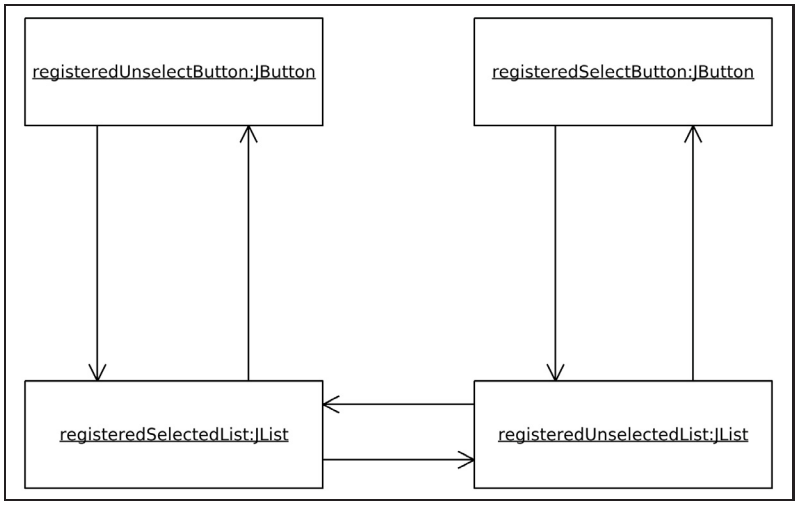


Figure II.10 Gestion décentralisée des dépendances.

de cette boîte de dialogue ne voit qu'une partie seulement de la prise en charge de ces dépendances. Comme la prise en charge des dépendances est divisée entre plusieurs classes, il est difficile de bien la comprendre. À cause du manque de cohésion, la maintenance demande plus de temps et est souvent de piètre qualité.

Une bonne manière de faire consiste à réorganisation de ces objets de manière à réduire au minimum le nombre de connexions. Rassembler la prise en charge des dépendances en un objet cohérent est une approche souhaitable. C'est ce à quoi sert le patron *Mediator*. La figure II.11 montre les objets impliqués dans la boîte de dialogue avec un objet additionnel de prise en charge centralisée des dépendances.

En plus d'être plus facile à maintenir et à programmer, la conception présentée à la figure II.11 est également plus facile à comprendre.

La figure II.12 montre comment les classes et les interfaces participent au patron *Mediator* en général.

Colleague1, Colleague2, . . . Les instances de classes jouant ce rôle ont des dépendances entre elles. Ces dépendances prennent 2 formes :

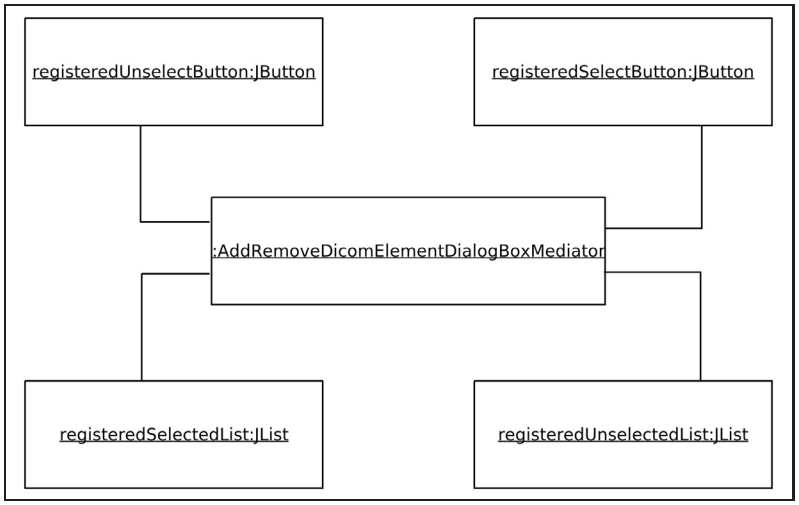


Figure II.11 Gestion centralisée des dépendances.

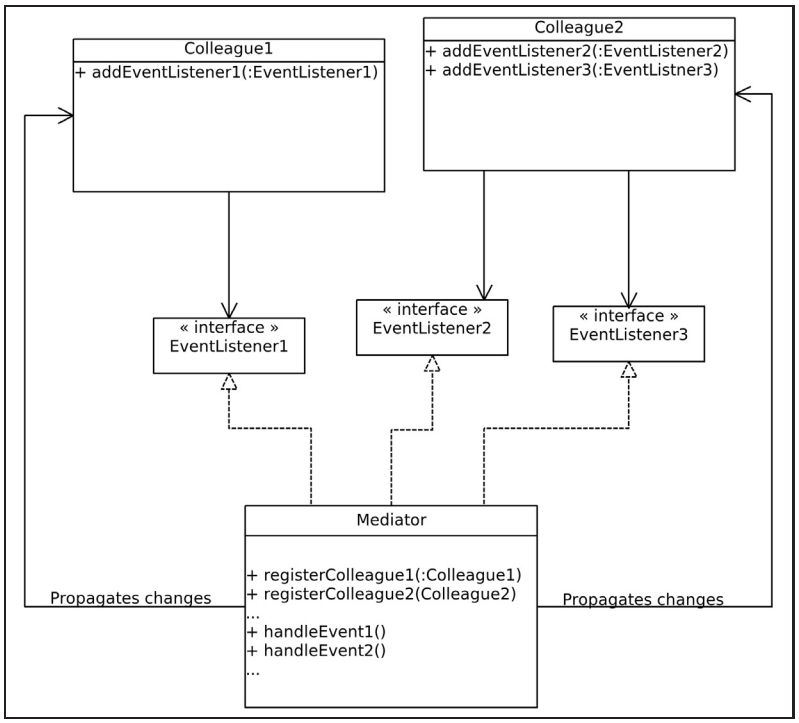


Figure II.12 les classes du patron *Mediator*.

– Un premier type de dépendance consiste à obtenir l'approbation des autres objets avant d'effectuer un type spécifique de changement d'état.

– le deuxième type de dépendance consiste à notifier d'autres objets lorsqu'un changement d'état spécifique survient.

Les 2 types de dépendances sont pris en charge de façon similaire. Les instances de `Colleague1, Colleague2, ...` sont associées avec l'objet `Mediator`. Quand ils doivent notifier les autres objets d'une changement relatif à leur état, ils appellent une méthode qu'ils connaissent de l'objet `Mediator`. L'objet `Mediator` s'occupe du reste.

EventListener1, EventListener2, ... Les interfaces jouant ce rôle permettent aux classes `Colleague1`, `Colleague2` et autres d'avoir d'un haut niveau de réutilisabilité. Cela est fait en rendant ces classes indépendantes du fait qu'elles utilisent un objet de classe `Mediator`. Chacune de ces interfaces offre une ou plusieurs méthodes pour répondre à un événement particulier. Les objets collègues appellent la méthode de l'interface appropriée sans être informée de la classe de l'objet médiateur qui implémente la méthode.

Mediator. Les instances de classes jouant ce rôle possède la logique nécessaire pour s'occuper des changements d'état des objets `Colleague1`, `Colleague2` et autres. La classe `Mediator` implémente les interfaces `EventListener`. Les objet `Colleague1`, `Colleague2` et autres appellent les méthodes déclarées par les interfaces et défninies par la classe `Mediator` pour l'informer de leurs changements d'état. L'objet `Mediator` encapsule la logique relative à ces changements d'état. L'objet `Mediator` possède des méthodes qui peuvent être appelées pour associer les objets `Colleague1`, `Colleague2` et autres avec le `Mediator`. Ces méthodes sont indiquées sur le diagrammes en tant que `registerColleague1`, `registerColleague1` et autres. Elles reçoive un objet `Colleague` en paramètre et se charge d'appeler une ou plusieurs de ses méthodes `add...Listener` afin d'indiquer l'objet `Colleague` qu'il doit informer l'objet `Mediator` d'un changement d'état.

Le mécanisme utilisé par `Colleague1`, `Colleague2`, ... de permettre aux autres objets d'exprimer leur intérêt pour un changement d'état et le mécanisme pour fournir une notification de ces changements d'état passe souvent par le *Java's delegation event model*. Souvent un seul objet est responsable de créer les objets `Colleague` ainsi que l'objet `Mediator`. Cet objet

est appelé à devenir un conteneur (« container ») pour ces objets qu'il crée. Dans ces conditions, pour ajouter à la robustesse du code, l'objet `Mediator` vient d'une classe interne privée (« inner class ») à la classe conteneur.

Les conséquences positives d'une telle organisation sont :

- La majeure partie de la complexité de prise en charge des dépendances est ramenée vers l'objet `Mediator`. Cela facilite l'implémentation et la maintenance de ces objets.
- Concentrer les dépendances à l'intérieur d'une classe unique facilite la maintenance de ces dépendances.
- L'utilisation du patron *Mediator* réduit le nombre de trajets dans le code ce qui facilite par le fait même les tests en boîte blanche (« White Box Tests »).
- Utiliser le patron *Mediator* signifie qu'il n'est pas nécessaire de dériver les classes `Colleague`.
- Les classes `Colleague` sont plus réutilisables car elles ne sont pas entichées du code relatif à la prise en charge des dépendances.

Les conséquences négatives sont :

- Ramener la prise en charge des dépendances à l'intérieur d'une classe unique peut faciliter la compréhension jusqu'à un certain point. Dépassé ce stade la classe `Mediator` peut devenir trop volumineuse. Si la classe `Mediator` devient énorme, penser à la séparer en plus petits morceaux pour la rendre plus compréhensible.
- Les classes `Mediator` sont difficilement réutilisable parce que trop liées à l'application.

L'application développée dans le cadre de ce projet de maîtrise fait un usage intensif du patron *Mediator* pour la mise au point des boîtes de dialogue utilisées dans l'interface utilisateur graphique. C'est un exemple complexe. Cela montre la nature du patron *Mediator* qui permet de rassembler toute la complexitée de la gestion des événements au-dessous d'une classe unique.

La classe `AddRemoveDicomElementDialogBoxMediator` est une classe interne privée à la classe conteneur, dans ce cas-ci, la classe boîte de dialogue `AddRemoveDicomElementDialogBox`:

La classe possède des variables d'instance qui réfère aux objets qui sont enregistrés avec elle.

```
private class AddRemoveDicomElementDialogBoxMediator extends
   AbstractMediatorTemplate {
  ...
  /** The registered 'select' button. */
  JButton registeredSelectButton;

  /** The registered 'unselect' button. */
  JButton registeredUnselectButton;

  /** The registered 'select' list of elements. */
  JList registeredSelectedList;

  /** The registered 'unselect' list of elements. */
  JList registeredUnselectedList;
```

La classe `AddRemoveDicomElementDialogBoxMediator` dérive de la classe `AbstractMediatorTemplate` qui offre des balises pour faciliter le développement d'une classe `Mediator`. Nous verrons plus loin à quoi cela peut bien servir. La classe médiatrice n'implémente aucune interface `EventListener`. Au lieu de cela, elle utilise des objets adaptateurs définis comme des classes internes privées ou des classes anonymes. On utilise cette manière de faire pour deux excellentes raisons :

- Cela garantit que ce ne sont que les objets qui ont le droit de s'enregistrer avec l'objet `Mediator` qui pourront transmettre leurs événements. Les classes internes ou les classes

anonymes ne pouvant être instanciée que par l'objet `Mediator`, ce n'est pas n'importe quel objet qui pourra s'enregistrer avec ces objets adaptateurs.

– Cela décharge la classe médiatrice d'avoir à déterminer de quel collègue un événement survient.

Les différents types d'événements pouvant se produire à l'intérieur de la boîte de dialogue sont pris en charge par des classes adaptatrices anonymes puisqu'elles doivent réagir différement en fonction de la source de l'événement. Les classes adaptatrice privée nommées servent à traiter les événements demandant un traitement similaire peu importe la source de l'événement. Par exemple, elles peuvent servir pour traiter les événements provenant de boutons radio ou d'autre objets simples. La classe `AbstractMediatorTemplate` déclare une classe interne privée appelée `ItemAdapter` dont l'unique instance est conservée dans la variable membre `itemAdapter`. Cette instance est accessible à toutes les classes dérivées de la classe `AbstractMediatorTemplate` et est un bon exemple de réutilisation de code.

```
/**
 * This is the private named adapter use to process the
   event which
 * are not source dependant.
 */
private ItemAdapter itemAdapter = new ItemAdapter();
```

La classe `AbstractMediatorTemplate` définit 2 constructeurs prenant en paramètre un objet de classe `Frame` dans un cas et un de type `Dialog` dans l'autre. C'est que la classe à laquelle peut appartenir le `Mediator` peut être soit un `JFrame` ou un `JDialog`. L'un des 2 constructeurs doit obligatoirement être appelée par la classe dérivée. Ces constructeurs appellent chacun la méthode `mediatorInit` qui déclare et instancie une classe adaptatrice anonyme prenant en charge les événements que la fenêtre lance lorsqu'elle s'ouvre. Cet adaptateur appelle la méthode de la classe `Mediator` responsable de mettre les composants de l'interface utilisateur graphique à leur état initial.

```
private void mediatorInit(Window owner) {
  WindowAdapter windowAdapter = new WindowAdapter() {
    public void windowOpened(WindowEvent windowEvent) {
      initialState();
    } // windowOpened(WinodwEvent)
  };
  owner.addWindowListener(windowAdapter);
}
```

La méthode `initialState` appelée à l'intérieur de la méthode `windowOpened` de l'adaptateur `WindowAdapter` est une méthode protégée abstraite dont l'implémentation est spécifique à la classe `Mediator` dérivée. C'est l'application concrète du patron de conception *Template Method*.

Pour enregistrer le bouton *Select* dans la boîte de dialogue est très simple car il n'est pas de la responsabilité de la classe `Mediator` de prendre en charge des événements issus de ce bouton, mais seulement de déterminer si le bouton doit être activé ou désactivé.

```
public void registerSelectButton(JButton
   registeredSelectButton) {
  this.registeredSelectButton = registeredSelectButton;
}
```

Les méthodes d'enregistrement des autres composants sont cependant beaucoup plus complexe parce qu'elles sont concernées par la gestion des événements qui peuvent interférer avec d'autres composants enregistrés avec le médiateur. La méthode d'enregistrement typique ressemble à la suivante :

```
public void registerSelectedList(final JList
   registeredSelectedList) {
  this.registeredSelectedList = registeredSelectedList;
```

```
    MouseAdapter adapter = new MouseAdapter() {

      private final Logger logger = Logger.getLogger(this.
         getClass());

      public void parseMouseEvent(MouseEvent mouseEvent) {
        if (MouseEvent.MOUSE_CLICKED == mouseEvent.getID()) {
          registeredUnselectedList.clearSelection();
        }
      }
    };
    registeredSelectedList.addMouseListener(adapter);
    registeredSelectedList.addListSelectionListener(new
       ListSelectionAdapter() {
      private final Logger logger = Logger.getLogger(this.
         getClass());
      public void parseSelection() {
        if (registeredSelectedList.getModel().getSize() == 0)
           {
          registeredSelectedList.clearSelection();
        }
      }
    });
  }
```

Cette méthode d'enregistrement fournit un objet adaptateur anonyme qui va un peu plus loin que d'appeler la méthode `enforceInvariants`. La classe `AbstractMediatorTemplate` met à la disposition du programmeur de la classe dérivée une gamme de classes adaptatrices internes qui sont protégées et qui peuvent être instanciée sous la forme de classes

anonymes pouvant répondre aux événements plus spécifiques d'un composant donné. Dans le programme précédent, on utilise une instance d'une classe dérivée de `MouseAdapter` et `ListSelectionAdapter` qu'on ajoute à l'instance de `JList` dont la référence est conservée à l'intérieur de la variable d'instance `registeredSelectedList`.

La méthode `enforceInvariant` peut modifier l'état de certains composants de l'interface utilisateur de manière à respecter les relations invariantes. Pour éviter que la méthode `enforceInvariant` ne soit appelée récursivement, la classe médiatrice utilise un indicateur booléen afin de reconnaître les appels récursifs à la méthode `enforceInvariants`.

```
private void enforceInvariants() {
  if (! busy) {
    busy = true;
    protectedEnforceInvariants();
    busy = false;
  }
}
```

La méthode `enforceInvariants` ne fait pas directement le travail de s'occuper des invariants, au lieu de cela, elle renvoie immédiatement le contrôle à la méthode appelante dans le cas d'un appel récursif sinon elle appelle la méthode `protectedEnforceInvariants`. Comme Java et son modèle de gestion des événements assure que tous les événements livrés de façon synchrone, la méthode `enforceInvariants` ne tient pas compte qu'elle pourrait être appelée pour prendre en charge un événement alors quelle en traite un autre.

Les relations invariantes dont la méthode `protectedEnforceInvariants` tient compte sont de désactiver le bouton < < lorsque la liste des éléments sélectionnés est vide et désactiver le bouton » lorsque la liste des éléments non-sélectionnés est vide. Cette méthode est déclarée

protected et *abstract* au niveau de la classe `AbstractMediatorTemplate` puisqu'elle doit être spécifique à la fenêtre pour laquelle de `Mediator` doit servir.

```
protected void protectedEnforceInvariants() {
  registeredUnselectButton.setEnabled(!
     registeredSelectedList.isSelectionEmpty());
  registeredSelectButton.setEnabled(!
     registeredUnselectedList.isSelectionEmpty());
}
```

4 Le patron de conception *State*

Le patron de conception *State* sert à représenter les états d'un objet par d'autres objets, chacun héritant d'une classe commune. Un objet peut prendre un nombre fini d'états qui sont connus à l'avance. Le comportement d'un objet est déterminé par son état et les événements appelés à survenir.

Un exemple de l'application du patron *State* se trouve dans l'implémentation de la boîte de dialogue de gestion des actions à effectuer sur un fichier DICOM. Cette boîte de dialogue ne s'applique que pour l'implémentation JDCM de l'application client, la version PixelMed offrant un autre niveau d'abstraction fonctionne d'une manière tout à fait différente.

Figure II.13 Boîte de dialogue des actions DICOM.

Dans ce modèle orienté objet, il y a la superclasse `AssociationState` de laquelle toutes les classes d'état héritent. Cette superclasse définit un ensemble de constantes symboliques représentant chacune un événement pouvant survenir sur la boîte de dialogue. La boîte de dialogue représente le contexte sur lequel le patron *State* s'applique.

La classe `AssociationState` déclare également 5 variables statiques qui contiennent chacune la référence à une classe d'état concrète. Ces variables sont `notConnected`, `notAssociated`, `associated`, `released` et `closed`. En effet, à moins d'avoir besoin de variables d'instance dans les classes, il n'est nullement besoin de définir plus d'une instance

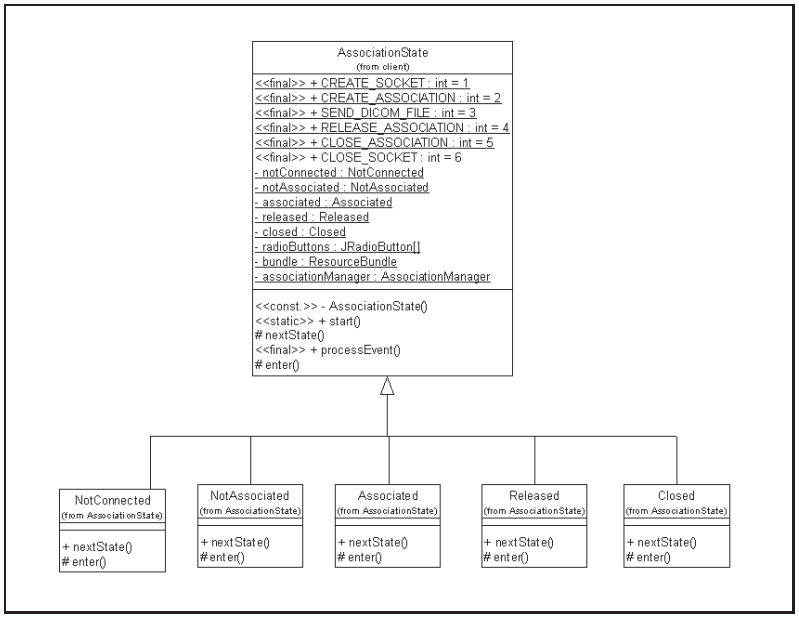


Figure II.14 Diagramme de classes.

des classes concrètes d'états. les classes concrètes d'état peuvent être instanciées une fois pour toute au début de l'éxécution du programme. C'est ce qui est fait à l'intérieur d'un bloc d'initialisation static au début de la superclasse.

La méthode statique `start` effectue toutes les opérations d'initialisation et retourne l'état initial de la machine d'état. Cette méthode n'est appelée qu'une fois pour initialiser et démarrer la machine d'état. Dans notre cas, nous l'appelons au moment d'ouvrir la boîte de dialogue des actions DICOM, c'est à ce moment que les boutons de la boîte de dialogue sont initialisés à leur état d'activation par défaut.

La méthode `nextState` est une méthode protégée. Cette méthode prend en paramètre un nombre entier représentant l'événement qui vient de survenir et retourne l'état suivant. Cette méthode n'est pas définie au niveau de la superclasse, mais elle l'est au niveau des classes concrètes. Son rôle consiste à déterminer l'état résultant. C'est cette méthode qui factorise les comportements à adopter de l'objet contextuel en fonction des différents états.

La méthode `processEvent` est une méthode publique qui prend en paramètre un événement et appelle la méthode nextState avec cet événement. Si l'état retourné par la méthode nextState

est différent de l'état courant (c'est à dire s'il y a changement d'état), la méthode `enter` est invoquée sur le nouvel état et le nouvel état est retourné par la méthode.

La méthode `enter` est invoquée quand l'objet état devient l'état courant. La classe `AssociationState` fournit une implémentation vide à cette éthode. Les sous-classes de la classe `AssociationState` redéfinissent le contenu de cette méthode pour effectuer un traitement dans l'objet état devient l'état courant pour le contexte.

Chaque état est représenté par une sous-classe différente de la classe mère `AssociationState`. Il y a les classes `NotConnected`, `NotAssociated`, `Associated`, `Released` et `Closed` qui redéfinissent chacune les méthode `nextState` et `enter` pour faire réagir la boîte de dialogue en fonction de l'état sélectionné. Aucune autre classe n'a besoin de connaître l'existence des classes d'état, c'est pourquoi il est approprié de déclarer ces classes comme des classes internes privées à la superclasse d'état. Le code relatif à chacun des états réside dans une classe distincte, il est donc plus facile de limiter les conséquences de l'ajout de nouveaux états en plus d'avoir la possibilité de pouvoir gérer un très grand nombre d'états. Pour les clients des objets états, la transition entre les états est vue comme une opération atomique où le nouvel état est retourné par un appel à la méthode `processEvent` sur l'état courant. Ce patron de conception permet également d'éviter les énormes structures *switch-case* ou les *if-then-else* imbriquées qui sont incompréhensibles et s'avèrent difficiles à maintenir. Il réduit le nombre de chemins possibles dans l'algorithme et facilite ainsi les tests en boîtes blanches (*White Box Tests*). À moins que les classes d'état possèdent des variables d'instance, il n'y a pas raison d'avoir plus d'une instance d'une classe d'état. Dans un tel cas, les classes d'état sont instanciées au démarrage de la machine d'état et les références sont conservées dans des variables statiques.

5 Le patron de conception *Strategy*

Le patron *Strategy* sert à rendre un comportement indépendant de la classe qui l'utilise. Cela permet non seulement de sélectionner un comportement spécifique pour un objet, mais offre la possibilité de changer ce comportement dans le temps.

Comme cela fut mentionné précédemment, les stratégies de désidentification du système Anonym 1.0 sont basées sur le patron *Strategy*. En effet, une des exigences du programme stipule que le logiciel puisse désidentifier un fichier DICOM. De plus, l'utilisateur doit pouvoir spécifier la politique de désidentification qu'il désire et celle-ci doit être modifiable facilement. Ces méthodes de désidentification sont appliquées à chacun des éléments DICOM et peuvent consister en les opérations suivantes :

- Remplacer la valeur par des blancs ;
- changer la valeur ;
- générer un pseudonyme ;
- retirer l'élément DICOM de l'ensemble (« DICOM set ») ;
- générer un nouvel UID.

On remplit la première exigence en séparant la logique de chacune des opérations de désidentification dans une classe différente. Cela nous donne un petit nombre de classes représentant les différentes méthodes de désidentification auxquelles il est facile d'ajouter d'autres classes. Nous voulons également que les classes qui utilisent nos classes de désidentification ne soient aucunement au courant des détails d'implémentation de ces dernières. Cela nous mène à la conception présentée à la figure II.15

Voici comment fonctionnent les classes de la figure II.15 entre elles. Si un objet `DeidentificationOperation` possède un objet `IDeidentificationStrategy` avec lequel travailler, il lui délègue l'opération de désidentification qu'il doit appliquer sur l'élément DICOM. Cet objet est d'une des classes suivantes :

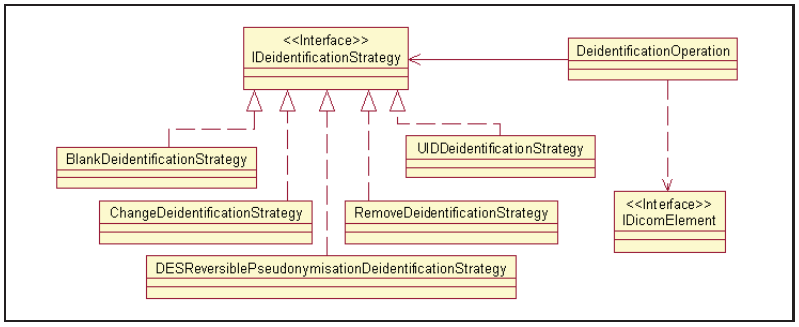


Figure II.15 Organisation des classes de désidentification.

- `BlankDeidentificationStrategy`;
- `ChangeDeidentificationStrategy`;
- `DESReversiblePseudonymisationDeidentificationStrategy`;
- `RemoveDeidentificationStrategy`;
- `UIDDeidentificationStrategy`.

Chacune des classes mentionnées implémente une opération de désidentification différente. Comme on peut le constater, cette organisation des classes facilite grandement l'ajout de nouvelles opérations et la modification de celles-ci.

On utilise le patron *Strategy* dans les cas où l'on doit fournir à une classe plusieurs algorithmes ou comportement. On doit faire varier le comportement d'une classe pour chaque instance de cette classe. Là où il est nécessaire de faire varier le comportement de l'objet à l'éxécution. De plus, faire appel à une classe implémentant une interface permet à la classe qui utilise la classe d'ignorer la manière dont la classe a été implémentée. Cependant, si le comportement d'une classe ne varie pas d'une instance à l'autre ni dans le temps, il est plus simple pour la classe de contenir directement le comportement ou d'avoir une référence statique à ce comportement. Cet arrangement des classes fait en sorte qu'un objet de type `DeidentificationOperation` puisse effectuer une opération de désidentification simplement en appelant la méthode `deidentify` de l'interface `IDeidentificationStrategy`.

Le patron de conception *Strategy* sert à fournir de multiples variations d'un alogrithme. Il est utilisé pour faire varier le comportement d'une instance d'une classe ou changer le comportement d'un objet lors de l'éxécution. De plus, déléguer le comportement d'une classe à une autre classe implémentant une interface fait en sorte que les classes qui utilisent ce comportement ne sont en aucune façon mises au fait des détails d'implémentation des classes qui implémentent ces interfaces. Toutefois si le comportement d'une instance d'une classe ne varie pas dans le temps, alors il est plus simple de penser inclure le comportement dans la classe ou de maintenir un référence statique au comportement.

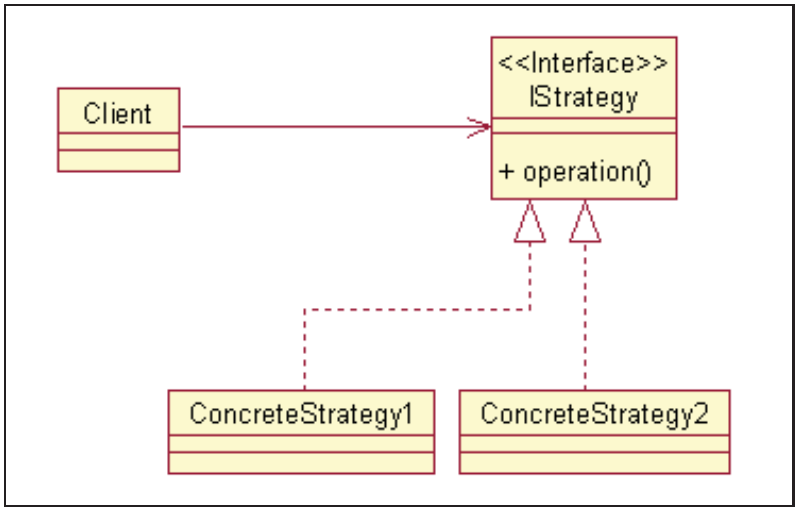


Figure II.16 Organisation générale des classes du patron Strategy.

Voici la description des rôles que les interfaces et les classes jouent dans la figure II.16 :

Client. Une classe jouant le rôle de `Client` délègue une opération à une interface. Il le fait sans savoir à quelle classe d'objet il délègue ni comment la classe implémente l'opération.

StrategyIF. une interface ayant ce rôle fournit une façon commune d'accéder aux opérations encapsulées par ses sous-classes.

ConcreteStrategy1, ConcreteStrategy2, et les autres. Les classes ayant ce rôle représentent différentes implémentations de l'opération que la classe `Client` délègue.

Le patron *Strategy* se présente bien souvent avec un mécanisme pour déterminer quel `ConcreteStrategy` l'objet `Client` utilise. Le processus de sélection est souvent guidé par une donnée de configuration ou un événement. Dans notre cas, la construction des objets `ConcreteStrategy` est confiée à un objet fabrique (« factory ») et cette information est gardée dans un fichier de configuration. Les classes `ConcreteStrategy`, si elles ont des comportements qui leurs sont communs, devraient implémenter ces comportement à l'intérieur d'une superclasse abstraite.

Il existe des situations où aucun des comportements encapsulés dans les classes `ConcreteStrategy` n'est approprié. Une façon facile de tenir compte de cette situation consiste à conserver une référence *null* au lieu d'une référence valide à un objet `Stategy`. Cette pratique nous oblige cependant à vérifier que la référence est différente de *null* avant chaque appel à la méthode de l'objet `Strategy`. Si la structure du `Client` ne s'y prête pas, il vaut mieux considérer utiliser le patron de conception *Null Object*.

Le patron *Strategy* permet de changer le comportement des objets `Client` de façon dynamique. Il permet également de simplifier les classes `Client` en les relevant de la responsabilité de sélectionner un comportement ou d'implémenter des comportements différents. Il simplifie le programme des classes `Client` en éliminant les structures complexes de sélection telles que les *if* et les *switch*. Il peut même parfois améliorer la performance des objets `Client` en leur enlevant la charge d'avoir à sélectionner le comportement approprié.

6 Le patron de conception Template Method

Un autre patron de conception qui fut mis à contribution dans le développement de l'application est le patron *Template Method*. Ce patron favorise la réutilisation du code en offrant une structure de haut niveau s'appliquant au plus grand nombre de cas possibles et en laissant au programmeur utilisateur le soin d'ajouter les parties plus spécifiques à l'application qu'il est en train de développer.

Dans le cas plus spécifique de l'application développée dans le cadre de ce mémoire, une utilisation qui est faite du patron *Template Method* sert à rassembler la logique commune à toutes les boîtes de dialogue du système. En effet, une large part du code servant à effectuer la gestion de la boîte de dialogue est relativement complexe et cela pourrait consituer un problème à long terme d'avoir à répéter ce code pour toutes les boîtes de dialogue du système. En fait, la seule partie qui change réellement entre deux boîtes de dialogue est ce qu'elle contient. Chaque classe définit donc ses méthodes primitives `getMessage` et `getComponentAdapter` qui sont appelées par la méthode *template* `dialogInit`. La figure II.17 présente quelques boîtes de dialogue qui appartiennent au système d'anonymisation et dont le fonctionnement est tout basé sur le patron *Template Method*.

Le diagramme de classes de la figure II.18 montre l'application du patron *Template Method* pour ces boîtes de dialogue. On peut voir que toutes les classes dérivent de la superclasse abstraite `AbstractDialogBoxTemplate`. Cette superclasse définit la méthode *template* `dialogInit` qui contient l'algorithme de base qui doit être le même pour toutes les sous-classes. À l'intérieur de cette méthode sont appelées les méthodes `getMessage` et `getComponentAdapter` qui sont deux méthodes auxquelles les classes concrètes doivent absolument donner une implémentation minimale. La superclasse définit également les méthodes (« hook ») `doOkOption`, `doDefaultOption`, `doCancelOption` et `doClosedDialog` qui traitent les actions pour les différents boutons présents dans la boîte de dialogue. Ces méthodes dîtes (« hook ») sont des méthodes offrant une implémentation minimaliste qu'il est possible d'écraser pour leur donner une implémentation plus élaborée.

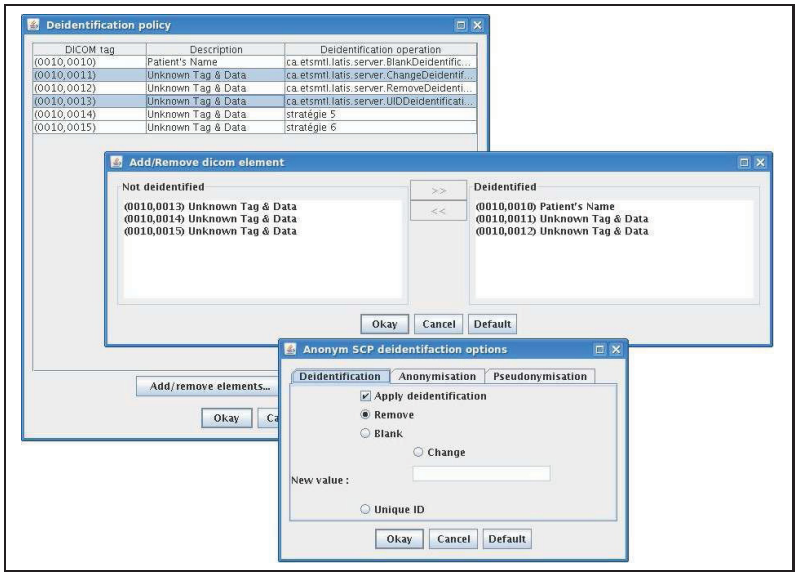


Figure II.17 Quelques boîtes de dialogue de l'application.

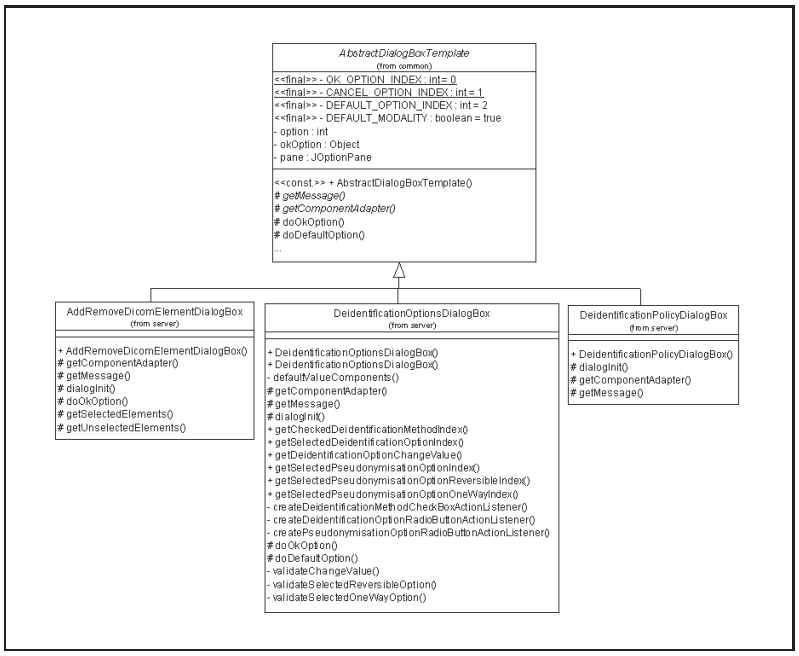


Figure II.18 Les classes des boîtes de dialogue de l'application.

Cette organisation des classes se concrétise par une superclasse déclarant une ou plusieurs méthodes abstraites appelées primitives dont l'implémentation sera fournie par ses classes dérivées. La superclasse abstraite définit une méthode appelée méthode (« template ») que tous les programmes qui l'utilisent. Cette méthode appelle les primitives dans un ordre pré-déterminé

et c'est aux classes dérivées de définir le contenu des primitives en fonction des opérations qu'elles souhaitent réaliser. La figure II.19 est un diagramme de classes qui montre l'organisation entre les classes lorsqu'on utilise le patron en question.

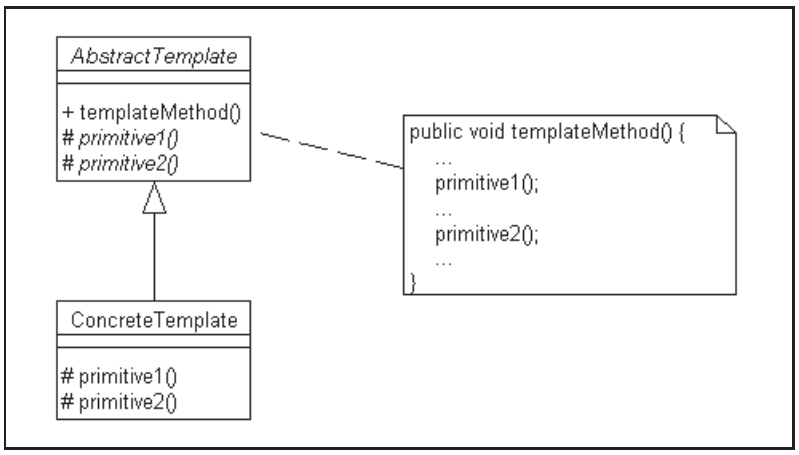


Figure II.19 Le patron *Template Method*.

`AbstractTemplate`. Cette classe abstraite contient une méthode *template* appelée `templateMethod` dont la responsabilité est de définir la logique élémentaire qui sera identique pour toutes les classes dérivées de `AbstractTemplate`. Cette méthode concrète appelle les méthodes de bas niveau appelées ici `primitive1` et `primitive2` dont l'implémentation est fournie par la classe dérivée. C'est ce qui permet d'adapter, en partie, le fonctionnement de la méthode `templateMethod` pour une utilisation plus spécifique tout en offrant un cadre rigide de conception.

`ConcreteTemplate`. Cette classe est une classe concrète dérivée de la classe `AbstractTemplate`. C'est cette classe qui fournit une implémentation aux méthodes de bas niveau `primitive1` et `primitive2` de manière à compléter la logique de la méthode `templateMethod`.

Ce patron de conception convient parfaitement lorsqu'on veut réutiliser du code, lorsque la responsabilité d'une classe est la même partout où on l'utilise et ne diffère qu'en quelques points spécifiques qui sont clairement définis. Cette organisation des classes permet aussi au programmeur d'avoir un guide, un formulaire en blanc dans lequel il ne suffit que de remplir

les espaces vides, ce qui élimine complètement toute ambiguïté quant à la manière d'utiliser la classe. Cela est d'autant plus vrai en Java où si on laisse une méthode abstraite sans implémentation, le compilateur refuse de compiler la classe tout simpelement. Toutefois, une telle conception des classes demande un effort supplémentaire qui est perdu s'il n'est pas réutilisé à plusieurs endroits dans le programme.

7 Le patron de conception *Hashed Adapter Objects*

Introduction

Ce patron de conception permet d'appeler une méthode d'un objet adaptateur qui est associé à un objet arbitraire. L'objet arbitraire sert à localiser l'objet adaptateur dans une structure de données.

Mise en contexte

Supposons qu'une méthode doivent accomplir différentes actions en fonction de la référence à un objet donné. Une technique couramment utilisée pour implémenter ce genre de structure de décision consiste à utiliser une chaîne de if imbriqués.

Si le nombre de comparaisons effectuées est très élevé, le temps nécessaire pour effectuer les comparaisons peut représenter un problème de performance. Quand une référence est comparée à un nombre élevé d'autres références, il existe des techniques plus rapides qu'une série d'alternatives pour obtenir le résultat escompté.

Le problème se résume à effectuer une recherche dans une structure de données. L'idée est de sélectionner une structure de données dans laquelle on peut effectuer des recherches le plus rapidement possible. La structure de données doit également permettre à n'importe quel objet adaptateur d'être associé à n'importe quel objet référence. Le but de cet objet est d'encapsuler le comportement à adopter lorsqu'une référence à un objet au sein de la structure de données correspond à une référence donnée.

Les autres objets sont des adaptateurs, c'est à dire qu'ils proviennent tous de classes implémentant une interface commune. L'interface définit une méthode qui est appelée lorsque l'objet adaptateur est récupéré de la structure de données. Bien souvent, les objets adaptateurs n'implémentent pas le comportement eux-même. Leur rôle se limite plutôt à appeler une mé-

thode définie à l'intérieur d'un autre objet dont le rôle consiste à éxécuter l'opération de façon concrète.

La structure de données qui répond le mieux à ces exigences est une table de hachage. En moyenne, seulement une comparaison est nécessaire pour localiser un objet dans une table de hachage. L'inconvénient d'utiliser une table de hachage est que l'ordre séquentiel dans lequel les objets sont listés est incertain. Puisque cela ne représente pas un problème dans le cas qui nous intéresse, une table de hachage s'avère être un choix judicieux. Le temps nécessaire pour localiser un objet dans la plupart des autres structures de données varie en fonction du nombre d'objets qu'elle contient, ce n'est pas le cas pour les tables de hachage où le temps de recherche reste relativement constant.

Réalisation

L'application à développer doit, à la fois, lire et traiter les données contenues dans un ensemble d'éléments DICOM (`IDicomSet`). Chaque élément de l'ensemble représente un attribut DICOM (un objet de type interface `IDicomElement`). La conception du programme implique qu'on doive itérer parmi les attributs de l'ensemble pour leur appliquer un algorithme d'anonymisation spécifique. La figure II.20 est un diagramme de classes qui montre une partie de cette conception.

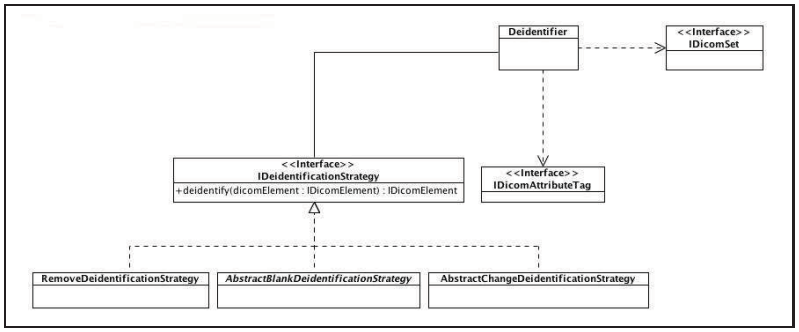


Figure II.20 Classes d'anonymisation de base.

Un objet `Deidentifier` lit une suite d'attributs DICOM provenant de l'objet de type interface `IDicomSet`. Chaque attribut possède une étiquette servant à le distinguer des autres attributs de

l'ensemble. Chaque attribut contient une valeur et c'est cette valeur qui doit subir le processus d'anonymisation.

L'objet `IDicomSet` donne accès aux attributs DICOM (IDicomElement). L'objet `Deidentifier` permet de passer ces attributs à la méthode `deidentify` de l'objet `IDeidentificationStrategy`. La méthode `deidentify` renvoie une version anonymisé de l'attribut DICOM dans une instance du type interface `IDicomElement`.

Lorsque le moment est venu d'optimiser le programme, il s'avère que l'application passe trop de temps à éxécuter la méthode `deidentify` de la classe `Deidentifier`. C'était prévisible puisqu'il existe plus d'un millier d'attributs différents à traiter. Le test pour déterminer le type d'un document se résume à une chaîne de if imbriqués comme celle-ci :

```
if (attributeTag.equals(DICOM_TAG_PATIENT_NAME))
  return new PersonNameChangeDeidentificationStrategy().
    deidentify(attribute);
else if (attributeTag.
    equals(DICOM_TAG_REFERRING_PHYSICIANS_NAME))
  return new SS1Record(record);
...
```

La conclusion à tirer est qu'une recherche dans une table de hachage serait beaucoup plus rapide que l'utilisation d'une cascade de comparaisons d'objet `IDicomTag`. Le coût de la recherche dans une table de hachage est relativement moins élevé que l'appel répétitif à la méthode `equals` d'un objet de type interface `IDicomTag`. Pour mettre en œuvre cette solution, il faut ajouter des classes au projet, comme le montre le diagramme de classes dans la figure II.21.

Lors de son initialisation, l'objet `Deidentifier` crée une instance de chaque classe implémentant l'interface `IDeidentificationStrategy`. Il associe à chacun de ces objets une instance de type interface `IDicomAttributeTag` permettant d'identifier quelle straté-

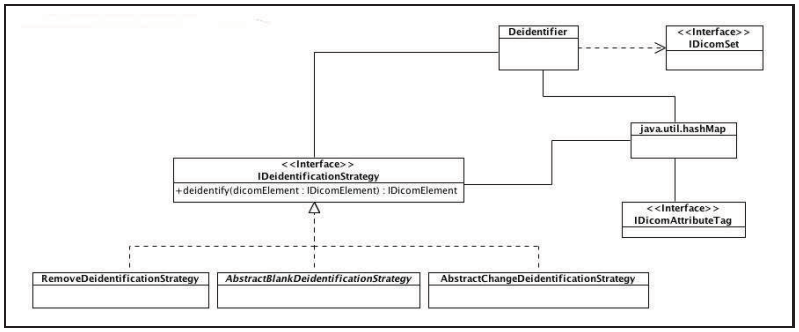


Figure II.21 Hashed adapters pour anonymiser des attributs.

gie d'anonymisation associer à l'attribut en question. Il délègue le maintien de ces associations à un objet `java.util.HashMap`.

L'objet `Deidentifier` passe chaque paire d'objets composée d'un `IDicomAttributeTag` et d'un objet de type interface `IDeidentificationStrategy` à la méthode `put` d'un objet `HashMap` de manière à ce que chaque étiquette soit la clé vers un objet de type interface `IDeidentificationStrategy` qui est sa valeur associée.

Alors que le programme traite le fichier DICOM, il passe l'ensemble des attributs à la méthode `deidentify` de l'objet `Deidentifier`. Pour chaque attribut DICOM contenu dans l'ensemble, la méthode passe l'étiquette qui identifie l'attribut à la méthode `get` de l'objet `HashMap`. La méthode `get` retourne l'objet `IDeidentificationStrategy` associé à la chaîne. La méthode `deidentify` fait ensuite un appel polymorphique à la méthode `deidentify` de l'objet `IDeidentificationStrategy` en lui passant l'attribut à anonymiser. Cette méthode `deidentify` retourne une instance de la classe de `IDicomElement` anonymisée. La figure II.22 illustre, en partie, le fonctionnement de cet algorithme.

Forces

– Il existe une longue chaîne de if imbriqués qui teste l'égalité entre un objet et de nombreux autres ; la chaîne de if imbriqués peut représenter un temps considérable au moment de l'éxécution.

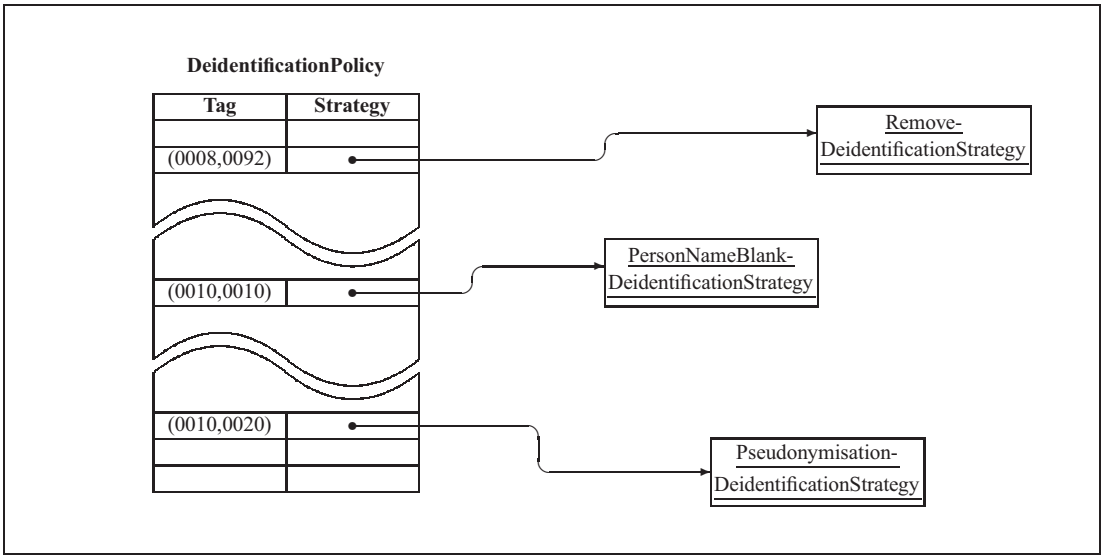


Figure II.22 La classe `HashMap` au service de l'anonymisation.

– L'action exécutée est sélectionnée par une suite de comparaisons d'égalité entre un objet et un ensemble variable d'autres objets.

– L'ensemble des asscociations objet-action peut varier dans le temps ainsi que les associations qui existent entre les objets clés et les objets adaptateurs.

– Une table de hachage propose une structure de données où la valeur associée à un objet clé peut être trouvée dans un délai qui est relativement indépendant du nombre d'objets qu'il y a dans la table, pour autant que certaines conditions soient respectées. Les tables de hachage sont présentées plus en détails à la section « Implémentation » de cet annexe.

– S'il n'y a que quelques tests à effectuer, l'utilisation d'une table de hachage pour la recherche peut s'avérer moins efficace qu'une chaîne de if imbriqués. Comme pour toutes les optimisations, il est préférable de s'assurer par un chronométrage approprié que l'utilisation d'une table de hachage constitue une amélioration appréciable au niveau des performance par rapport à l'utilisation d'une chaîne de if imbriqués. Même si la chaîne de if imbriqués est longue, lorsque quelques objets seulement sont sélectionnés nettement plus fréquemment que d'autres, il est possible d'obtenir de meilleurs résultats en plaçant ces objets au début de la chaîne. La règle à adopter est de ne pas compliquer

inutilement le fonctionnement d'un algorithme à moins que ces améliorations ne soient clairement justifiées.

- Bien que l'utilisation d'une table de hachage puisse améliorer la vitesse d'exécution dans certaines circonstances, cette solution consomme plus de mémoire qu'une chaîne de if imbriqués.
- La rédaction de classes adaptatrices représente parfois une tâche colossale. Toutefois, il s'avère que les classes adaptatrices ont de fortes ressemblances structurelles d'une classe à l'autre. Parfois, les différences se limitent à aussi peu que le nom de la classe et le nom de la méthode qu'ils appellent sur l'objet encapsulé par l'adaptateur.

Solution

L'hypothèse relative au patron Hashed Adapter Objects est un ensemble d'actions associées à un ensemble d'objets. Quand une action est sélectionnée, puis exécutée, un programme fait des comparaisons entre un objet de référence et les objets contenus dans un ensemble en utilisant leur méthode `equals`. Si l'un des objets de la série est égal à l'objet donné, l'action correspondante est exécutée. Le diagramme de classes de la figure II.23 montre l'organisation des classes du patron *Hashed Adapter Objects*.

Voici la description des rôles que jouent les objets, les classes et les interfaces dans le patron *Hashed Adapter Objects* :

ActionIF Une interface jouant ce rôle, définit une méthode, illustrée dans le diagramme de la figure II.23 par `doIt`, que la classe cliente peut appeler pour obtenir l'action désirée.

Action1, Action2 Les classes jouant ce rôle implémentent l'interface `ActionIF` et permettent d'encapsuler un comportement utilisé par un objet client. Bon nombre de ces classes n'implémentent pas le comportement en question, elles ne sont qu'un adaptateur pour appeler les méthodes d'un objet d'une autre classe n'implémentant pas l'interface `ActionIF`.

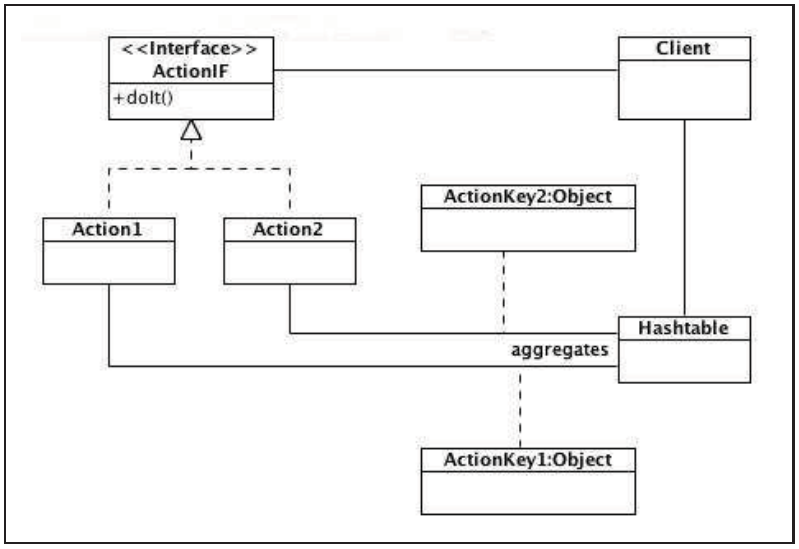


Figure II.23 Organisation des classes du patron Hashed Adapters Objects.

ActionKey1, ActionKey2 Les objets jouant ce rôle ont une instance de `Action1`, `Action2` qui leur sont associée.

Hashtable Les instances de classe jouant ce rôle, sont responsables de maintenir l'association entre les instances des classes `Action1`, `Action2`, ... avec les objets jouant le rôle `ActionKey1`, `ActionKey2`, ... Quand un client présente une instance d'une classe `ActionKey1`, `ActionKey2`, ... à un objet `Hashtable`, il cherche un objet `ActionKey1`, `ActionKey2`, ... correspondant parmi les objets déjà stockés dans la table de hachage. Pour être un objet correspondant, l'appel à la méthode `equals` de l'objet doit retourner vrai quand l'objet `Hashtable` lui passe une des instances de `ActionKey1`, `ActionKey2`, ... stockées dans la table de hachage. Après qu'il ait trouvé l'objet correspondant, il retourne l'instance de l'une des classes `Action1`, `Action2`, ... qui lui est associée.

Client Quand une instance d'une classe jouant ce rôle doit effectuer une action associée à un objet, il présente cet objet à un objet `Hashtable`. L'objet `Hashtable`, renvoie un objet qui implémente une interface `ActionIF`. Le client appelle ensuite la méthode `doIt` que la classe de l'objet implémente pour effectuer l'action requise.

Conséquences

- Si la classe Hashtable est bien programmée, le coût pour récupérer un objet action à partir d'un objet Hashtable se résumera à un appel à la méthode `hashCode` et en moyenne un appel à la méthode `equals` de l'objet `ActionKey`.
- L'ensemble des objets `ActionKey` dans un objet `Hashtable` peut être modifié en cours d'exécution, ainsi que les objets actions qui leur sont associés. Cela signifie que cette façon de faire permet de modifier le comportement d'un objet client simplement en ajoutant ou en enlevant des éléments à l'ensemble des associations objet-action. Cela signifie également qu'il est possible d'affecter une action différente à n'importe objet clé et bénéficer ainsi d'un degré de liberté supplémentaire dans la détermination du comportement général de l'objet client. Finalement, il est important de souligner que ces modifications peuvent être apportées à un objet en cours d'éxécution.

Implémentation

Une table de hachage maintient le lien entre des objets clés et des objets valeurs. Cela permet, pour tout objet clé se trouvant dans la table, de retrouver rapidement l'objet valeur qui lui est associé.

La façon dont une table de hachage fonctionne quand on lui demande de trouver ou de stocker un objet clé est d'appeler la méthode `hashCode` de l'objet. Il utilise la valeur de hachage retournée par la méthode `hashCode` pour déterminer où l'objet sera recherché ou conservé dans la structure de données. La méthode `hashCode` retourne une valeur entière (int), ce qui signifie qu'il y aura plus de valeurs de code de hachage qu'il y aura de place dans la structure des données. Les algorithmes de table de hachage tentent de résoudre cette difficulté en associant chaque place dans la structure de données à plusieurs valeurs de code de hachage. Cela signifie que plusieurs objets voudront potentiellement être stockés au même endroit dans une table de hachage. Lorsque cela arrive, on dit qu'il y a collision.

Quand il n'y a pas de collision, obtenir la valeur de l'objet associé à un objet clé se résume aux étapes suivantes :

a. Appelez la méthode `hashCode` de l'objet clé.

b. Rechercher dans la structure de données, à l'endroit indiqué par le code de hachage de l'objet clé, où l'objet devrait être situé, s'il est présent dans la structure de données.

c. Déterminer si l'objet clé se trouve bien dans la structure de données.

d. Si l'objet clé existe dans la structure des données, retourner l'objet valeur qui lui est associé.

Les algorithmes des tables de hachage tiennent compte des collisions de différentes façons. Cependant, toutes les techniques de prise en charge des collisions ont besoin d'effectuer des recherches supplémentaires pour trouver l'emplaçement d'un objet dans une structure de données dans laquelle il peut y avoir collision. Cependant, s'il n'y a aucun risque de collision, le temps de recherche à l'intérieur de la structure de données est indépendant du nombre d'éléments dans la table.

Il n'est généralement pas possible de prévenir les collisions dans une table de hachage. Cependant, on peut faire en sorte de diminuer le risque de collision. Habituellement, la chose la plus importante à contrôler est la quantité d'objets que la table peut accepter. Si une table contient plus d'objets qu'il y a d'emplacements pour les stocker, il est certain qu'il y aura collision. Quand une table de hachage passe d'un état de 100 % remplie à un état vide, la probabilité que les élément subissent des collisions à l'intérieur de la table passe de 100 % à 0 %. Les classes qui implémentent une table de hachage fournissent généralement un moyen de spécifier une limite supérieure du nombre d'éléments que la table de hachage peut accepter. Si le nombre d'éléments dans la table excède cette limite, la taille de la table est augmentée pour que le nombre d'éléments se situe à l'intérieur de la limite.

La performance des tables de hachage est également affectée par la qualité des méthodes `hashCode` implémentées par les objets stockés dans la table de hachage. Si la méthode hashCode de deux objets différents retourne la même valeur, les stocker dans une table de hachage produira toujours une collision. Par conséquent, il est important que les méthodes `hashCode`

soient implémentées de manière à ce qu'il soit peu probable que deux objets différents produisent la même valeur de hachage.

L'API Java fournit deux implémentations de table de hachage : ce sont les classes `java.util.Hashtable` et `java.util.HashMap`. On utilise normalement l'une de ces deux classes lorsqu'on implémente le patron *Hashed Adapter Objects*. D'un point de vue optimisation, la principale différence entre les deux classes est que les méthodes d'accès aux données dans la classe `java.util.Hashtable` sont synchronisées alors que les méthodes dans le `java.util.hashMap` ne le sont pas.

Pour les applications du patron *Hashed Adapter Objects* dans lequel le contenu de la table de hachage est initialisé et n'est jamais changé, la classe `java.util.HashMap` s'avère un choix judicieux car, pour de nombreuses implémentations de Java, les appels aux méthodes non-synchronisées de la classe `java.util.HashMap` prendront moins de temps à s'exécuter que leur équivalent synchronisés de la classe `java.util.Hashtable`.

Pour les applications du patron *Hashed Adapter Objects* dans lequel le contenu de la table de hachage est accessible par plusieurs « threads », la classe `java.util.Hashtable` est souvent plus appropriée. Toutes les mesures doivent être prises pour s'assurer qu'un seul « thread » à la fois accède à la table de hachage. Habituellement, la façon la plus simple de faire est de profiter du fait que les méthodes de la classe `java.util.Hashtable` sont synchronisées.

Patrons apparentés

Adapter Le patron *Hashed Adapter Objects* utilise le patron *Adapter*.

Lookup Table Les deux patrons *Hashed Adapter Objects* et *Lookup Table* impliquent l'utilisation d'une agrégation. Toutefois, les deux utilisent l'agrégation avec un objectif différent. Le patron *Lookup table* utilise une agrégation pour stocker les résultats précalculés dans le but d'économiser du temps de calcul pour plus tard. Pour le patron *Hashed Adapter Objects*, c'est la structure de données qui implémente l'agrégation qui est la source d'économie de temps.

Polymorphism Quand il est possible de choisir un comportement en fonction du type d'un objet, le patron *Polymorphism* offre une solution plus simple que le patron *Hashed Adapter Objects*.

Single Threaded Execution Le patron *Single Threaded Execution* est utilisé pour coordonner l'accès par de multiples « threads » à la table de hachage utilisée par le patron *Hashed Adapter Objects*.

Strategy Le patron *Strategy* ne propose aucun mécanisme particulier de sélection de la stratégie à appliquer. Le patron *Hashed Adapter Objects* offre un mécanisme pour sélectionner les objets stratégies dans le patron *Strategy*.

ANNEXE III

GUIDE D'EXPLOITATION POUR ANONYM V1.0

Ce document contient l'information nécessaire pour être en mesure d'installer et d'exploiter le logiciel.

1 Liste des pré-requis pour exécuter l'application

L'application Anonym 1.0 exige d'avoir les éléments suivants pour s'éxécuter :

- Une JVM version 1.6 ;
- une connexion réseau configurée en TCP/IP ;
- une instance fonctionnelle du système de gestion de base de donnée PostgreSQL avec les relations de Anonym 1.0 montées dessus ;
- la liste des librairies décrites dans la section suivante :

2 Liste des fichiers

Le fichier d'archive peut être décompressé à n'importe quel emplacement du disque. Il contient tous les fichiers et répertoires nécessaires au déploiement et à l'exécution de l'application. Les seules dépendances qui ne font pas partie de l'archive sont la base de données.

2.1 Le répertoire racine du projet (. /)

Ce répertoire contient tous les sous-répertoires utiles à la mise en œuvre du projet. Il contient également le fichier `build.xml` mais, bien que ce fichier puisse être éxécuté à l'aide de la commande `ant` directement, il nécessite l'utilisation d'un nombre considérable d'options de compilation et de déclaration (`-D<property>=<value>`) qui peuvent rendre son utilisation un peu compliquée et nébuleuse. Il est conseillé d'utiliser la commande `./scripts/runAnt.sh`

depuis le répertoire racine du projet qui se charge d'appeler la commande `ant` avec les paramètres appropriés.

2.2 Le répertoire `./build`

Ce répertoire contient tous les fichiers générés par la procédure de compilation (`./scripts/runAnt.sh`). Ce répertoire peut ne pas être présent dans le fichier d'archive pour des raisons d'économie d'espace disque, il sera créé automatiquement par la procédure de compilation ainsi que les sous-répertoires qu'il contient (classes, javadoc, jar, etc.) au moment d'éxécuter la commande `./scripts/runAnt.sh`.

Le répertoire `./build` contient les sous-répertoires suivants :

`./classes`: Ce répertoire contient un sous-répertoire au nom du projet (ici `pixelmed_test_01`) contenant la structure hiérarchique des fichiers classes composant le projet. C'est cette même structure hiérarchique qui se retrouve dans le fichier `.jar` à la différence qu'elle est déployée sur le disque. Ce sous-répertoire au nom du projet sert également de répertoire de sortie par défault (*Default output folder*) à l'environnement de développement intégré [*Integrated Development Environment (IDE)*] Eclipse. Ce répertoire est généré lorsque la commande `./scripts/runAnt.sh` est invoquée avec la cible `compile` de la façon suivante :

```
[alemay@localhost pixelmed_test_01]$ ./scripts/runAnt.sh
   compile
Buildfile: build.xml

init:
    [mkdir] Created dir: /home/alemay/Documents/Projets/
      pixelmed_test_01/build/classes/pixelmed_test_01
```

```
     [copy] Copying 1 file to /home/alemay/Documents/
        Projets/pixelmed_test_01/build/classes/
        pixelmed_test_01

compile:
    [javac] Compiling 138 source files to /home/alemay/
      Documents/Projets/pixelmed_test_01/build/classes/
      pixelmed_test_01
    [javac] Note: /home/alemay/Documents/Projets/
      pixelmed_test_01/src/ca/etsmtl/latis/dicom/
      DicomAbstractFactoryFactory.java uses unchecked or
      unsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for
      details.

BUILD SUCCESSFUL
Total time: 4 seconds
[alemay@localhost pixelmed_test_01]$ ll build
total 8
drwxrwxr-x 3 alemay alemay 4096 Oct 27 16:18 classes
```

./jar : Ce répertoire contient le résultat de l'éxécution de la commande `scripts/runAnt.sh jar` qui permet de générer l'archive Java de l'application.

./javadoc : Ce répertoire contient le résultat de l'éxécution de la commande `scripts/runAnt.sh javadoc` qui permet de générer le javadoc du programme.

2.3 Le répertoire `./database`

Ce répertoire contient tous les fichiers relatifs à la création et à l'effacement de la base de données relationnelle qui est utilisée par le logiciel. En effet, le système a besoin pour fonctionner

d'un système de gestion de base de données (SGBD) afin de conserver l'information du patient et de ses images médicales. Pour l'instant, nous utilisons une base de données PostgreSQL, mais cela pourrait très bien changer dans l'avenir puisque nous utilisons l'interface JDBC pour passer nos requêtes à la base de données ce qui nous assure un niveau d'indépendance avec la base de données utilisée.

Le répertoire `./database` contient les fichiers et sous-répertoires suivants :

`create_database.psql` Ce script crée toutes les relations et séquences de la base de données en PostgreSQL.

`create_psql_database.sh:` Ce script shell éxécute le script `create_database.psql` avec les paramètres appropriés.

`create_psql_user.sh:` Ce script shell crée l'utilisateur PostgreSQL qui sera utilisé pour accéder à la base de données du système. L'utilisateur créé par ce script doit exister avant de créer la base de données car c'est par le biais de cet utilisateur que le système pourra accéder à la base de donnée par la suite. Pour éxécuter ce script, on doit d'abord s'approprier l'identité de l'utilisateur système postgres à l'aide de la commande `su`.

`ddl:` Ce répertoire contient les scripts de définition des relations (« Data Definition Language »). Le répertoire `ddl` contient un nombre important de fichiers dont le nom est suffisant pour expliquer la fonction.

`drop_database.psql:` Ce script détruit toutes les relations et séquences de la base de données. Il n'efface toutefois pas la base de données comme l'indique son nom.

`mdl:` Ce répertoire contient tous les scripts généraux de manipulation des données (« Manipulation Definition Language »).

2.4 Le répertoire `./dcm_samples`

Ce répertoire contient un ensemble de fichiers Dicom servant de tests pour l'application.

2.5 Le répertoire `./doc`

Ce répertoire contient la documentation relative au projet. Il contient notamment le fichier `pixelmed_test_01.mdl` qui est la version la plus à jour du fichier modèle pour Rational Rose Enterprise Edition.

2.6 Le répertoire `./etc`

Ce répertoire contient pour le moment deux fichiers de configuration :

`anonym.properties:` Ce fichier contient les paramètres de configuration du logiciel Anonym 1.0.

`logforj.properties:` Ce fichier contient les paramètres de configuration pour le système de journalisation Log4j.

Le répertoire `./etc` fait partie de la variable `CLASSPATH` de telle sorte que l'application Java puisse accéder aux fichiers de configuration sans qu'il soit nécessaire de préciser l'emplacement de ces derniers.

2.7 Le répertoire `./images`

Ce répertoire contient la liste des images rattachées à chacun des patients de la base de données. Il existe un répertoire spécifique pour chacun des patients de la base de données. Normalement, ce répertoire ne devrait pas se retrouver dans le répertoire du projet, mais cela est tolérable tant que le logiciel est en cours de développement.

2.8 Le répertoire `./jars`

Ce répertoire contient toutes les archives java des librairies qui sont utilisées par l'application. Ces fichiers `.jar` sont ensuite pris en grappe par le script `build.xlm` pour construire l'application.

2.9 Le répertoire `./keystores`

Ce répertoire contient les clés (publiques et privées) qui sont utilisées par les algorithmes de chiffrement (DES pour l'instant).

2.10 Le répertoire `./log`

Ce répertoire contient la liste des fichiers journaux (fichiers `.log`) générés par l'application.

2.11 Le répertoire `./scripts`

Ce répertoire contient tous les scripts utilitaires dont un pour exécuter l'application.

2.12 Le répertoire `./src`

Ce répertoire contient l'arborescence des fichiers de programme source java de l'application.

2.13 Le répertoire `./tmp`

Ce répertoire contient des fichiers temporaires.

3 La base de données

L'application nécessite l'utilisation d'un système de gestion de base de donnée (SGBD) pour fonctionner. En effet, quand l'application DICOM Anonym 1.0 reçoit des images, elle doit conserver de l'information concernant l'image qu'elle vient d'anonymiser. Pour l'instant, cette information qu'elle conserve consiste principalement en l'emplacement sur le disque du fichier Dicom. Cela permet de recréer l'historique des examens reçus pour un même patient. Une fois l'information du fichier Dicom stockée dans la base de données, il devient plus simple de lancer des requêtes concernant les fichiers sans avoir à accéder physiquement aux fichiers sur le disque.

Avant de lancer l'application, il faut créer les relations qui contiendront l'information des patients. Pour cela, il existe une gamme de scripts qui se trouvent dans le sous-répertoire `./database` du répertoire du projet.

a. S'assurer que le service postgresql soit démarré. Pour ce faire, utiliser la commande `service` de la manière suivante :

```
# service postgresql start
```

Le système devrait confirmer que le service est bel et bien démarré :

```
Starting postgresql service:                               [
    OK  ]
#
```

b. Depuis le sous-répertoire `./database` du répertoire du projet, lancer la commande pour générer l'utilisateur principal de la base de données :

```
# ./create_psql_user.sh
```

Il est important d'éxécuter le script `create_psql_user.sh` en premier car l'utilisateur principal de la BD doit avoir été créé avant de lancer le script de création des relations de la base de données. S'il est impossible de se connecter à la base de données, c'est que la permission n'a pas été accordée. Le SGBD PostgreSQLvient toujours avec des privilèges de connexion assez restreints, pour modifier la sécurité de la base de données, éditer le fichier `pg_hba.conf` :

```
# vi /var/lib/pgsql/data/pg_hba.conf
```

Pour plus de détails concernant le format du fichier `pg_hba.conf`, on recommande de lire la documentation en ligne de votre version de PostgreSQL.

c. Toujours depuis le sous-répertoire `./database`, lancer la création de la base de donneés grâce au script `create_psql_database.sh` :

```
# ./create_psql_database.sh
```

Une fois le processus de création des relations de la base de données complété avec succès, on peut lancer l'éxécution de l'application Anonym 1.0.

4 Le fichier de configuration `./etc/anonym.properties`

Ce fichier de configuration contient tous les paramètres qui servent à contrôler le fonctionnement du système Anonym 1.0. Le fichier de configuration est basé sur les fichiers de propriétés (*properties files*) de Java. Voici la description de chacune de ces variables :

network.hostname Nom d'hôte sur lequel s'éxécute le service.

network.port.number Numéro du port sur lequel le service attend les requêtes.

general.image.repository Emplacement des images de la base de données.

general.image.extension Extension par défaut des fichiers Dicom.

general.message.bundle Nom du fichier de propriétés contenant la liste des messages à afficher.

general.dicom.implementation.provider.id Le type d'implémentation pour le SCU. Les deux implémentations qui sont actuellement disponibles sont JDCM et PixelMed.

general.dicom.scp.implementation.provider.id le type d'implémentation pour le SCP. Les deux valeurs possibles pour ce paramètres sont « JDCM » et « PixelMed » car ce ne sont que les deux seules implémentations disponibles pour le moment.

general.dicom.scp.pixelmed.implementation.network.port.number Numéro du port sur lequel l'implémentation PixelMed du SCP attend les connexions.

general.dicom.scp.pixelmed.implementation.our.maximum.length.received Longeur maximale du PDU que l'implémentation PixelMed du SCP peut permettre de recevoir.

general.dicom.scp.pixelmed.implementation.socket.receive.buffer.size Taille du tampon de réception de la socket TCP de l'implémentation PixelMed du SCP.

general.dicom.scp.pixelmed.implementation.socket.send.buffer.size Taille du tampon d'envoi de la socket TCP du SCP dans l'implémentation PixelMed (une taille égale à zéro signifie d'utiliser la taille par défaut).

general.dicom.scp.pixelmed.implementation.called.application.entity.title Titre de l'AE qui est appelée pour le SCP PixelMed.

general.dicom.scp.pixelmed.implementation.secure.transport Vrai si on utilise un protocole de transport sécure avec le SCP PixelMed.

general.dicom.scp.pixelmed.implementation.debug.level Niveau des messages de déboguage pour le SCP PixelMed (0 pour aucun message).

general.dicom.pixelmed.implementation.hostname Nom d'hôte vers lequel l'implémentation PixelMed du SCU envoie sa requête de stockage.

general.dicom.pixelmed.implementation.port Numéro du port vers lequel l'implémentation PixelMed du SCU tente d'établir une connexion.

general.dicom.pixelmed.implementation.called.application.entity.title Nom de l'AE appelée par l'implémentation PixelMed du SCU.

general.dicom.pixelmed.implementation.calling.application.entity.title Nom de l'AE appelante de l'implémentation PixelMed du SCU.

general.dicom.pixelmed.implementation.compression.level Niveau de compression utilisé par l'implémentation PixelMed du SCU (0=*none*, 1=*propose deflate*, 2=*propose deflate and bzip2*).

general.dicom.pixelmed.implementation.move.originator.application.entity.title L'AET du C_MOVE d'où origine la requête C_STORE (null si absent) pour l'implémentation PixelMed du SCU.

general.dicom.pixelmed.implementation.move.originator.message.id L'identificateur du message du C_MOVE d'où originie la requête C_STORE (-1 s'il n'y en a aucun) pour l'implémentation PixelMed du SCU.

general.dicom.pixelmed.implementation.debug.level Niveau des messages de déboguage pour l'implémentation PixelMed du SCU.

general.dicom.temporary.directory Emplacement sur le disque pour les fichiers temporaires.

locale.language Code à deux lettres minuscules représentant la langue en ISO-639.

locale.country Code à deux lettres majuscules représentant le pays en ISO-3166.

BIBLIOGRAPHIE

[1] Wayne D. Blizard. Multiset theory. *Notre Dame Journal of Formal Logic*, 30(1), 1989.

[2] Robert F. Boruch and Joe S. Cecil. *Assuring the confidentiality of social research data*. University of Pennsylvania Press, Philadelphia, 1979.

[3] Ji-Won Byun, Yonglak Sohn, Elisa Bertino, and Ninghui Li. Secure anonymization for incremental datasets. volume 4165 LNCS, pages 48–63, Seoul, South Korea, 2006. Springer Verlag, Heidelberg, Germany.

[4] Mary Campione, Kathy Walrath, and Alison Huml. *The Java tutorial : a short course on the basics*. Java series. Addison-Wesley, Boston ; Montreal, 3rd edition, 2001.

[5] Yu-Cheng Chiang, Tsan-Sheng Hsu, Sun Kuo, Churn-Jung Liau, and Da-Wei Wang. Preserving confidentiality when sharing medical database with the cellsecu system. *International Journal of Medical Informatics*, 71(1) :17–23, 2003.

[6] Josep Domingo-Ferrer, Agusti Solanas, and Antoni Martinez-Balleste. Privacy in statistical databases : K-anonymity through microaggregation. 2006 IEEE International Conference on Granular Computing, pages 774–777, Atlanta, GA, United States, 2006. Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ , United States.

[7] Bruce Eckel. *Thinking in Java*. Prentice-Hall, Upper Saddle River, N.J., 2nd edition, 2000.

[8] Robert Eckstein and Marc Loy. *Java swing*. O'Reilly, Sebastopol, CA, 2nd edition, 2003.

[9] D.C. Feldmeier and P.R. Karn. Unix password security–ten years later. Advances in cryptology–CRYPTO '89 Proceeding, pages 44 – 63. Springer-Verlag, 1990.

[10] Erich Gamma. *Design patterns : elements of reusable Object-Oriented software*. Addison-Wesley professional computing series. Addison-Wesley, Reading, Mass. ; Don Mills, Ont., 1995.

[11] Li Gong, Gary Ellison, and Mary Dageforde. *Inside Java 2 platform security : architecture, API design, and implementation*. Java series. Addison-Wesley, Boston, 2nd edition, 2003.

[12] Mark Grand. *Patterns in Java, volume 1 a catalog of reusable design patterns illustrated with UML*. Wiley Pub., Indianapolis, Ind., 2nd edition, 2002.

[13] Ceki Gülcü. *The complete Log4j manual*. QOS.ch, Lausanne, Suisse, 1st edition, 2003.

[14] Cay S. Horstmann. *Object-oriented design & patterns*. Wiley, Hoboken, NJ, 2nd edition, 2006.

[15] D.V. Klein. "'foiling the cracker' : A survey of, and implications to, password security,". Proceedings of the USENIX UNIX Security Workshop, pages 5 – 14, Aug 1990.

[16] Henry F. Korth and Abraham Silberschatz. *Systèmes de gestion des bases de données*. McGraw-Hill, Toronto, 1988.

[17] Leslie Lamport. *LATEX : a document preparation system : user's guide and reference manual*. Addison-Wesley, Reading, Mass. ; Toronto, 2nd edition, 1994.

[18] Craig Larman. *Applying UML and patterns : an introduction to object-oriented analysis and design and the unified process*. Prentice Hall PTR, Upper Saddle River, NJ, 2nd edition, 2002.

[19] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. l-script -diversity : Privacy beyond k-anonymity. volume 2006 of *Proceedings - International Conference on Data Engineering*, page 24, Atlanta, GA, United States, 2006. Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ, United States.

[20] Steven C. McConnell. *Code complete : a practical handbook of software construction*. Microsoft Press, Redmond, Wash., 1993.

[21] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, Chris Rowley, Christine Detig, and Joachim Schrod. *The LaTeX companion*. Addison-Wesley series on tools and techniques for computer typesetting. Addison-Wesley, Boston ; Toronto, 2nd edition, 2004.

[22] National Electrical Manufacturers Association. *Digital Imaging and Communications in Medicine (DICOM), Supplement 55 : Attribute Level Confidentiality (including De-identification)*, 2002.

[23] National Electrical Manufacturers Association. *The Digital Imaging and Communications in Medicine (DICOM) Standard*, 2008.

[24] Rita Noumeir, Alain Lemay, and Jean-Marc Lina. Pseudonymisation of radiology data for research purposes. volume 5748 of *Progress in Biomedical Optics and Imaging - Proceedings of SPIE*, pages 298–305, San Diego, CA, United States, 2005. International Society for Optical Engineering, Bellingham, WA, United States.

[25] Lucila Ohno-Machado, Paulo Sergio Panse Silveira, and Staal Vinterbo. Protecting patient privacy by quantifiable control of disclosures in disseminated databases. *International Journal of Medical Informatics*, 73(7-8) :599 – 606, 2004.

[26] Aleksander Ohrn and Lucila Ohno-Machado. Using boolean reasoning to anonymize databases. *Artificial Intelligence in Medicine*, 15(3) :235 – 254, 1999.

[27] M. Peyraviana, A. Roginskya, and A. Kshemkalyanib. On probabilities of hash value matches. *Computers & Security*, 17(2) :171 – 174, 1998.

[28] K. Pommerening and M. Reng. Secondary use of the electronic health record via pseudonymisation. *L. Bos, S. Laxminarayan, and A. Marsh (ed.) Medical Care Compunetics I, Amsterdam*, pages 441 – 446, 2004.

[29] Bruce Schneier. *Applied cryptography : protocols, algorithms, and source code in C*. Wiley, New York, 2nd edition, 1996.

[30] Latanya Sweeney. k-anonymity : A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems*, 10(5) :557–570, 2002.

[31] Sharon Zakhour and Mary Campione. *The Java tutorial : a short course on the basics*. The Java series. Addison-Wesley, Upper Saddle River, NJ, 4th edition, 2006.