QUery Engine on Image Database (QUEID)

Rapport Technique
Laboratoire L3i, Université de La Rochelle

Mathieu Delalandre

23 octobre 2006

Table des matières

1 Introduction

Durant l'année 2005-2006 j'ai occupé un poste d'Ingénieur d'Étude au laboratoire *L3i*[1] dans le cadre du projet *MADONNE*[2]. L'objectif de ce projet était de contribuer à la réalisation de systèmes de traitement d'images de document du patrimoine. Il a groupé différents laboratoires français au sein d'un Consortium de 2003 à 2006.

Dans le cadre de ce projet j'ai été amené à réaliser un moteur d'analyse de bases d'images. En effet, les images traitées dans ce projet proviennent de librairies numériques [1] en ligne sur Internet ou locales. Elles ont été numérisées à partir d'ouvrages du patrimoine puis prétraitées à l'aide de différentes plateformes [2]. Ceci introduit une hétérogénéité dans ces images due à différents facteurs [3] :

- Les images numérisées ont été produites par différents prestataires (bibliothèques, sociétés, particuliers ...). utilisant chacun des matériels et modes de numérisation qui leurs sont propres.
- La numérisation est un processus qui s'étale dans le temps : les matériels et modes de numérisation d'un prestataire donné peuvent donc être amenés à évoluer durant cette période.
- La numérisation est un processus conduit par l'homme, des erreurs peuvent donc se produire.
- Les images numérisées sont par la suite pré-traitées par différentes plate-formes pour leur filtrage, redressement, restauration, séparation texte/graphique .... Les images produites sont donc fonction des plateformes et traitements mis en oeuvre.
- ...

Cette hétérogénéité pose deux types de problème pour un système de traitement d'images souhaitant exploiter ces images :

Technique : Les images peuvent être sauvegardées de différentes façons : jpg ou tiff, couleur ou niveaux de gris, compressées ou non compressées, ... Or un système a souvent des contraintes sur le type des images qu'il traite, celles-ci doivent donc être correctement formatées en entrée.

Sémantique : Les images peuvent être numérisées à des résolutions différentes et compressées différemment (avec ou sans perte). Elles sont donc sémantiquement différentes, or tous les systèmes ne sont pas capables de supporter ces différences. Les images doivent donc pouvoir être distinguées en entrée.

[1] Laboratoire informatique image interaction
http ://www.univ-lr.fr/labo/l3i/site_statique

[2] MAsse de DONnées issues de la Numérisation du patrimoiNE
http ://l3iexp.univ-lr.fr/madonne/

De façon à résoudre ces problèmes nous avons développé un moteur d'analyse de bases d'images baptisé *QUEID* (*QUery Engine on Image Database*). *QUEID* permet d'extraire des bases les attributs des images comme les modèles "binaire, niveaux de gris, couleur", les formats "jpg, tiff, bmp, ...", les résolutions, ... À partir de cette extraction *QUEID* peut être utilisé en deux modes différents. Ceux-ci sont présentés sur la Figure 1 suivante et détaillés sur la liste ci-dessous.

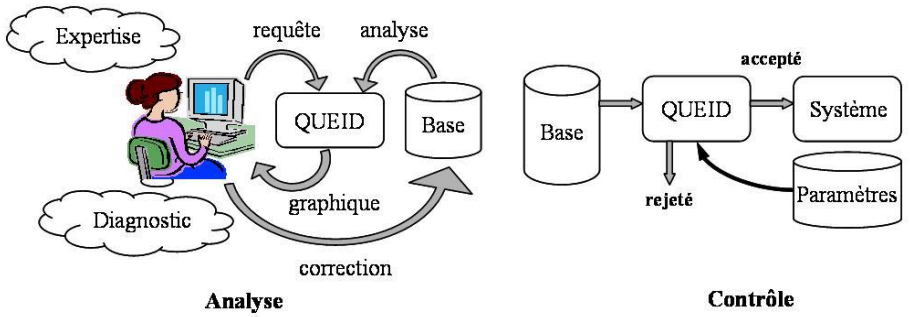

FIG. 1 – Modes d'utilisation de *QUEID*

Analyse : *QUEID* peut tout d'abord être utilisé en mode "analyse" par un utilisateur. Il permet d'effectuer une analyse statistique des attributs extraits d'une base d'images présentée sous la forme de graphiques à l'utilisateur. Ce dernier peut ensuite utiliser *QUEID* en mode requête pour la recherche d'images données. Le but est de naviguer au sein de la base afin d'identifier les images mal formatées. L'utilisateur peut alors corriger les images détectées à l'aide des moyens à sa disposition sur son système (recherche de sauvegarde, re-numérisation, édition, ...).

Contrôle : *QUEID* peut également être utilisé en mode "contrôle" en entrée d'un système. *QUEID* analyse pour cela les attributs de chacune des images d'entrée. En fonction de ces attributs *QUEID* procède alors au rejet ou à l'acceptation de l'image. Cette décision est alors fonction d'un ensemble de paramètres propres au système et externalisés de façon à être exploitables par *QUEID*.

Dans la suite de ce rapport nous présentons notre moteur *QUEID*. *QUEID* est une application *Java* reposant sur l'utilisation de la librairie *JAI*. Nous présentons donc tout d'abord cette librairie dans la section (2). Dans les sections (3) et (4) nous présentons *QUEID* puis et un cas d'usage d'utilisation. Finalement dans la section (5) nous concluons sur ce travail.

2 La Java Advanced Imaging (JAI)

2.1 Introduction

La *JAI* [4] est une librairie de traitement d'images *Java* [5] distribuée par *Sun* et libre de droit d'utilisation. Cette librairie est en fait une sur-couche à la librairie standard *java.awt.image* de la plate-forme *J2SDK*[3] [6]. Elle regroupe un nombre conséquent de traitements d'images et propose des possibilités d'extension. Son implémentation *Java* en assure une portabilité aisée. De plus, nombre des traitements sont implémentés nativement. L'utilisation de ces implémentations natives est alors gérée automatiquement par la *JAI* de façon transparente pour l'utilisateur. Ceci rend donc la *JAI* aussi performante (en terme de temps de calcul) que des librairies écrites en *C/C++*. Enfin, la *JAI* propose un mécanisme original de gestion de graphes d'opérateurs. Toutes ces caractéristiques font de la *JAI* une librairie attractive de plus en plus utilisée pour l'implémentation de systèmes de traitement d'images. La *JAI* est disponible à l'adresse[4]. Différentes ressources sont proposées, nous présentons ci-dessous les principales.

1. La *JAI* en tant que telle se compose de trois librairies (.jar) : *jai_codec.jar*, *jai_core.jar* et *mlibwrapper_jai.jar*.
2. L'API[5] documentation *JAI* correspond à la documentation *Javadoc* des classes de la *JAI*.
3. Le Guide *JAI* fournit une documentation d'utilisation de la *JAI*. Un tutorial d'introduction de la *JAI* peut également être trouvé dans [7].
4. Le Tutorial *JAI* est un exécutable *Java* illustrant les possibilités de la *JAI* tout en fournissant des exemples de codes.
5. Aux ressources précédentes on se doit d'ajouter L'API documentation de la plate-forme *J2SDK* concernant le package *java.awt.image*. La maîtrise de ce package est en effet indispensable à une bonne compréhension de la *JAI* [4].

2.2 Représentation des images

Les algorithmes de traitement d'images requièrent l'utilisation d'objets images. Dans cette section, nous présentons les différentes classes utilisées par la *JAI* pour l'instanciation de ces objets. Les images de la *JAI* peuvent être multi-dimensionnelles (c-à-d avec plusieurs valeurs associées à un seul pixel) et codées de différentes façons (booléen, entier, flottant, ...). Différents modèles de couleur peuvent également être utilisés. De façon à représenter l'ensemble des codages possibles la *JAI* repose sur l'utilisation d'une variété de classes. Nous présentons dans les paragraphes suivants une description des principales.

[3] Java 2 Software Development Kit
[4] http ://java.sun.com/products/java-media/jai/
[5] Application Programming Interface

La classe *PlanarImage* constitue la base pour la manipulation des images dans la *JAI*. Elle utilise une composition d'objets pour représentation des images présentée dans la Figure 2 ci-dessous. Les données pixels sont stockées dans un objet *DataBuffer* composant un objet *Raster*. Ces deux objets sont associés selon un ensemble de règles spécifiées dans un objet *SampleModel*. Ce dernier décrit la façon dont les données pixels sont échantillonnées au sein des différentes bandes de couleur composant l'image, il compose également l'objet *Raster*. Un objet *PlanarImage* se compose aussi d'un objet *ColorModel*. Ce dernier sert à la représentation graphique des images pour leur affichage. Il se compose d'un objet *ColorSpace* définissant les règles de codage des couleurs (*RGB*, *YCbCr*, *GRAY*). Un objet *PlanarImage* est accessible en lecture uniquement : les pixels peuvent être lus de différentes manières mais il est impossible de les modifier.

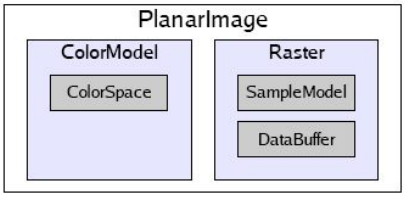

FIG. 2 – Composition d'un objet *PlanarImage*

La classe *TiledImage* (ou image tuilée) est sous-classe de la classe *PlanarImage*. Un objet *TiledImage* correspond à un ensemble de sous-images. Chaque sous-image est alors qualifiée de tuile. La Figure suivante donne un exemple d'image tuilée. Toutes les tuiles sont de même dimension. Chacune d'entre elles peut être traitée indépendamment par un algorithme. De cette manière, les images de larges dimensions sont traitées avec de raisonnables performances par la *JAI*. En effet, il n'est pas nécessaire de charger l'image d'un seul coup en mémoire pour son traitement. Un objet *TiledImage* peut être lu mai aussi modifié en écriture

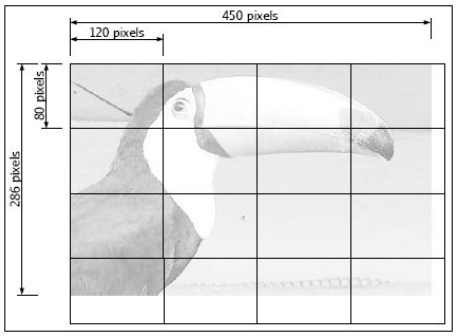

FIG. 3 – Exemple de *TiledImage*

2.3 Opérateurs *JAI*

Chaque mise en oeuvre d'un traitement d'image donné dans la *JAI* se fait par appel d'un opérateur spécifique. Il existe différents opérateurs dans la *JAI* regroupés en 8 familles principales. Elles sont décrites dans la section 3.6 du guide de la *JAI*, le Tableau 1 suivant les présente brièvement.

Familles	**Description**	**Exemples**
Point	Les opérateurs points transforment une image d'entrée en une image de sortie de façon à ce que chacun des pixels de l'image de sortie dépendent uniquement du pixel d'entrée correspondant.	valeur absolue, logarithme, ...
Aire	Les opérateurs aires effectuent des transformations mathématiques afin de translater les pixels de l'image d'entrée vers de nouvelles coordonnées dans l'image résultat.	bord, découpe, convolution, ...
Géométrique	Les opérateurs géométriques permettent de modifier la taille et l'orientation des formes de l'image.	rotation, échelle, translation, ...
Tramage	Les opérateurs tramages, aussi connu sous le terme de quantification de couleur, simulent la représentation des couleurs d'une image à partir d'un groupe de couleurs primaires.	quantification, diffusion, ...
Fichier	Les opérateurs fichiers sont utilisés pour lire et écrire les images.	lecture, écriture, conversion, ...
Fréquence	Les opérateurs fréquences ont pour but de décomposer le domaine spatial de l'image vers un domaine fréquence.	cosinus discret, fourrier, ...
Statistique	Les opérateurs statistiques permettent une analyse du contenu de l'image.	histogramme, moyennage, ...

TAB. 1 – Familles d'opérateurs de la *JAI*

3 QUEID

3.1 Introduction

Dans cette section nous présentons notre moteur *QUEID* d'analyse de base d'images. Celui-ci se compose de trois packages principaux : *queid.img*, *queid* et *queid.ihm*. Nous présentons chacun d'entre eux dans les sous-sections suivantes (3.2), (3.3) et (3.4).

3.2 Le package *queid.img*

Le package *queid.img* est central au moteur *QUEID*, il permet la gestion des fichiers images. Pour cela il utilise différentes classes de mise en oeuvre et plus particulièrement les classes *ImgNameMng* et *ImgFileMng*. La Figure 4 ci-dessous présente le le diagramme de classes de ce package.

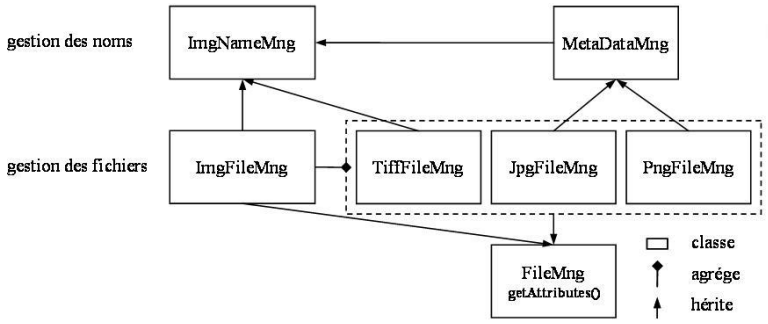

FIG. 4 – Diagramme de classes du package *queid.img*

Les classes *ImgNameMng* et *ImgFileMng* sont les interfaces principales du package *queid.img*. Elles permettent respectivement la gestion des noms et formats de fichier. La classe *ImgNameMng* propose des fonctionnalités classiques d'analyse de chaînes de caractères appliquées aux noms de fichier pour l'extraction des chemins, des extensions, ... La classe *ImgFileMng* met en oeuvre le reste du package pour la gestion des formats de fichier. Cette dernière emploie différentes classes *FileMng* spécifiques à chaque format de fichier traité. *QUEID* prend en charge dans sa version actuel trois types de format [8] : *Tiff*, *Jpg* et *Png*. Cependant il peut être étendu aux autres types de par les possibilités de la *JAI* [4]. Les classes *FileMng* ont alors pour objectif l'extraction des attributs des fichiers. Pour cela elles implémentent une fonction *getAttributes()* retournant un objet *Attributes* à partir du fichier analysé. Le Tableau 2 suivant détaille les attributs extraits.

Attributs	**Valeurs**
Nom	une chaîne
Format	jpg, tiff, png
Modèle	couleur, niveaux de gris, binaire
Compression	sans, packbits, jpg, ...
Résolution	un entier (en ppi)
Hauteur	un entier (en pixel)
Largeur	un entier (en pixel)

TAB. 2 – Attributs d'un fichier image

3.3 Le package *queid*

Le package *queid* exploite les attributs extraits par le package *queid.img* afin d'effectuer l'analyse statistique de la base d'images. Pour ce faire ce package utilise deux algorithmes spécifiques : *ajout* et *tri*. La Figure 5 en donne les pseudo-codes[6].

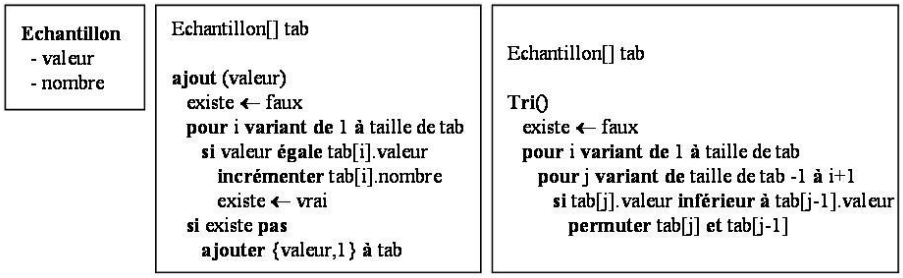

FIG. 5 – Algorithmes d'analyse *ajout* et *tri*

Ces algorithmes manipulent des objets de type *échantillon* composés de deux membres : la *valeur* de l'échantillon son *nombre* d'exemplaire. L'algorithme *ajout* a alors pour but de créer les objets *échantillon* ou d'incrémenter le *nombre* d'exemplaire d'un objet existant. Une fois tous les attributs traités l'algorithme *tri* les ordonne. Il utilise pour cela une méthode dite de tri à bulles [9].

3.4 Le package *queid.ihm*

Le package *queid.ihm* permet la gestion des interfaces console et graphique de *QUEID*. Il implémente deux fonctionnalités principales : l'IHM[7] de *QUEID* et le parser de fichier de paramètre.

L'IHM de *QUEID* se compose de trois composants comme le montre la Figure 6 ci-dessous : une barre d'outil, la fenêtre console et le panneau de graphiques. La barre d'outil permet la commande de *QUEID*. Elle repose sur les librairies *Java Swing/AWT*[8] [5]. Elle permet l'affichage d'une fenêtre de dialogue via l'action du bouton *Open* pour la sélection du répertoire d'entrée de la base d'images. Une fois ce répertoire validé son nom est affiché dans le champ texte central de *QUEID*.

[6] Algorithme où les actions à entreprendre sont exprimées en langage naturel.
[7] Interface Homme Machine
[8] Abstract Window Toolkit

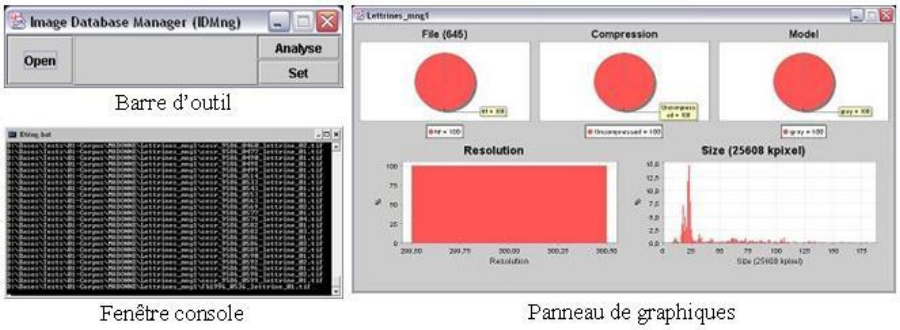

FIG. 6 – IHM de *QUEID*

A partir du répertoire d'entrée sélectionné, le bouton *Analysis* permet de lancer l'analyse de la base d'images en mettant en oeuvre le package *queid*. Pour les besoins de cette analyse *QUEID* effectue une lecture récursive des répertoires à partir du répertoire d'entrée. Toutes les images incluses dans les sous-répertoires sont donc analysées également. Durant l'analyse la fenêtre console affiche différentes informations comme le répertoire en cours d'analyse, les images analysées, ...

Une fois l'analyse terminée *QUEID* affiche les résultats sur un panneau de graphiques. Ce panneau utilise de la librairie *JFreeChart*[9] [10]. Cette dernière permet la conception de nombreux graphiques, *QUEID* l'utilise plus particulièrement pour ceux de types histogramme et camembert. Le Tableau 3 ci-dessous détaille chacun des graphiques généré par *QUEID*.

Titre	**Type**	**Description**
Format	Camembert	Ce graphique donne la distribution des formats de fichier. Il spécifie le également le nombre total de fichier de la base dans la barre de titre.
Compression	Camembert	Ce graphique donne la distribution des modes de compression utilisés dans les fichiers.
Modèle	Camembert	Ce graphique donne la distribution des modèles des images (couleur, niveaux de gris et binaire).
Résolution	Histogramme	Ce graphique donne la distribution des résolutions des images. Les résolutions inconnues sont étiquetées à -1.
Taille	Histogramme	Ce graphique donne la distribution des tailles des images en kilo pixel $\frac{hauteur \times largeur}{1000}$. La taille totale de la base est également indiquée dans la barre de titre.

TAB. 3 – Graphiques de *QUEID*

[9] http ://www.jfree.org/jfreechart/

Le package *queid.ihm* permet également la définition de requêtes. Ces dernières reposent sur l'utilisation d'un objet de type *Setting* instancié par l'utilisateur à l'aide du panneau de configuration de paramètre présenté sur la Figure 7. Ce panneau s'ouvre par action du bouton *Setting* de la barre d'outil. Chacun de ses champs instancie alors un membre de l'objet *Setting*. Le Tableau 4 suivant détaille ces différents champs. L'objet de type *Setting* dispose d'une méthode *accept* prenant en argument un objet de type *Attributes*. Cette méthode est alors mise en oeuvre dans le package *queid* afin de filtrer en entrée les fichiers de la base d'images.

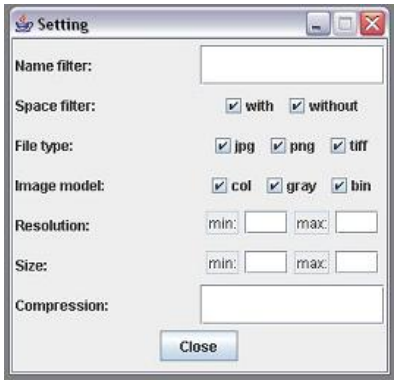

FIG. 7 – Panneau de configuration de paramètre

L'objet *Setting* peut être également instancié à partir de la lecture de fichiers de paramètre. Ces derniers permettent alors d'externaliser des paramètres propres à un système donné, ils sont exploités dans le mode contrôle de *QUEID*. Ces fichiers sont au format *XML* [11], le Tableau 5 en détaille la structure et le code ci-dessous en donne un exemple. Leur lecture repose sur deux classes principales dans le package *queid.ihm* : *SettingHandler* et *SettingParser*. Elles servent à l'implémentation d'un parser de type *SAX*[10]. Elle retourne alors un objet de type *Setting* correspondant au contenu d'un fichier.

```
<?xml version='1.0'?>
<setting>
    <name filter="bgmw"/>
    <space without="true" with="false"/>
    <file types="jpg;tif"/>
    <compression types="lzw;jpg"/>
    <model types="gray;color"/>
    <resolution min="250" max="350"/>
    <size min="25" max="50"/>
 </setting>
```

[10]Simple API for XML

Nom	Description
Filtre de nom	Ce filtre est appliqué sur les noms de fichier. Si un nom de fichier contient la chaîne de caractères spécifiée alors il sera sélectionné. Ce filtre ne tient pas en compte les cas majuscule/minuscule, par exemple "RTV" et "rtv" seront considérés comme des filtres identiques.
Filtre d'espace	Ce filtre est appliqué sur les noms de fichier. D'un coté un fichier sera sélectionné si il contient des espaces blancs dans son nom (*with*). De l'autre coté un fichier sera sélectionné si il ne contient aucun espace blanc dans son nom (*without*). Les sélections de (*with*) et (*without*) retourneront tous les fichiers.
Filtre de format	Ce filtre est appliqué sur les formats de fichier. Il spécifie les formats sur lesquels l'analyse doit exclusivement se porter (*Jpeg* et/ou *Png* et/ou *Tiff*).
Filtre de modèle	Ce filtre est appliqué sur les modèles des images. Il spécifie les modèles sur lesquels l'analyse doit exclusivement se porter (couleur et/ou niveaux de gris et/ou binaire).
Filtre de résolution	Ce filtre est appliqué sur les résolutions des images. Il permet de définir une bande de valeurs de résolution ($min \leq r \leq max$) sur lesquelles l'analyse doit exclusivement se porter. Les résolutions inconnues sont étiquetées à -1.
Filtre de taille	Ce filtre est appliqué sur les tailles des images. Il permet de définir une bande de valeurs de taille ($min \leq t \leq max$) sur lesquelles l'analyse doit exclusivement se porter. Les tailles sont exprimées en kilo pixel $\frac{hauteur \times largeur}{1000}$.
Filtre de compression	Ce filtre est appliqué sur les modes de compression des images. Il permet de spécifier différents modes de compression sur lesquels l'analyse doit exclusivement se porter. La séparation des modes se fait par utilisation du caractère ";" (exemple : jpg ;uncompressed).

TAB. 4 – Champs du panneau de configuration de paramètre

Balise(s) XML	Attribut(s)
name	L'attribut *filter* spécifie la chaîne de caractères à utiliser pour le filtre de nom.
space	Les attributs *with* et *without* valident (*true*) ou invalident (*false*) le filtre d'espace.
file, compression, model	L'attribut *type* spécifie les types sur lesquels l'analyse doit se porter. La séparation des types se fait par utilisation du caractère ";" (exemple : jpg ;uncompressed).
resolution, size	Les attributs *min* et *max* déterminent la bande de valeurs du filtre.

TAB. 5 – Balises et attributs des fichiers de paramètre *XML*

4 Cas d'usage

Nous présentons ici un cas d'usage de *QUEID*. Celui-ci a été utilisé pour l'analyse[11] de la base d'images *OLDB*[12] produite par les *BVH*[13] du *CESR*[14] de Tours. Cette base regroupe des images de lettrines de livres anciens imprimés du XV° et XVI° siècles. Le résultat de l'analyse est présenté sur la Figure 8 suivante, le Tableau 6 en donne une synthèse.

Taille de la base	3312 Fichiers, 433.8 Mp
Formats	jpg, tiff
Modes de compression	jpg, packbits, sans
Modèles	niveaux de gris, couleur
Résolutions (ppi)	?, 72, $\simeq 225$, $\simeq 300$, $\simeq 400$
Tailles des images (kp)	$\simeq 750$, $\simeq 1000$, $\simeq 2500$

TAB. 6 – Synthèse de l'analyse de *OLDB*

À la vue de l'analyse de *OLDB* produite par *QUEID* différentes hétérogénéités apparaissent au sein de cette base :

1. Les images sont à 98% au format *Jpg* et à 2% au format *Tiff*.
2. Dans le cas des images *Tiff* celles-ci sont à 97% non compressées et à 3% compressées en mode *packbits*.
3. Les images sont à 85% en niveaux de gris et à 15% en couleur.
4. Les images sont de résolutions très hétérogènes. Elles sont pour environ 65% de résolutions de 275 à 325 ppi. Environ 10% de la base concerne des images de plus faibles résolutions (de 200 à 250 ppi) et de plus fortes résolutions (de 375 à 425 ppi). Enfin, environ 25% des images est non exploitable car de résolution inconnue à -1 et de faible résolution à 72 ppi.
5. Les images sont en quasi totalité de tailles inférieures à 750 kp. Environ 1% d'entre elles sont de tailles entre 750 et 1100 kp. Enfin, une image apparaît comme étant de taille nettement supérieure aux alentours de 2500 kp.

[11] Analyse effectuée le 09/10/2006
[12] Ornamental Letters DataBase
http ://www.bvh.univ-tours.fr/madonne.asp
[13] Bibliothèques Virtuelles Humanistes
http ://www.bvh.univ-tours.fr/
[14] Centre d'Études Supérieures de la Renaissance
http ://www.cesr.univ-tours.fr/

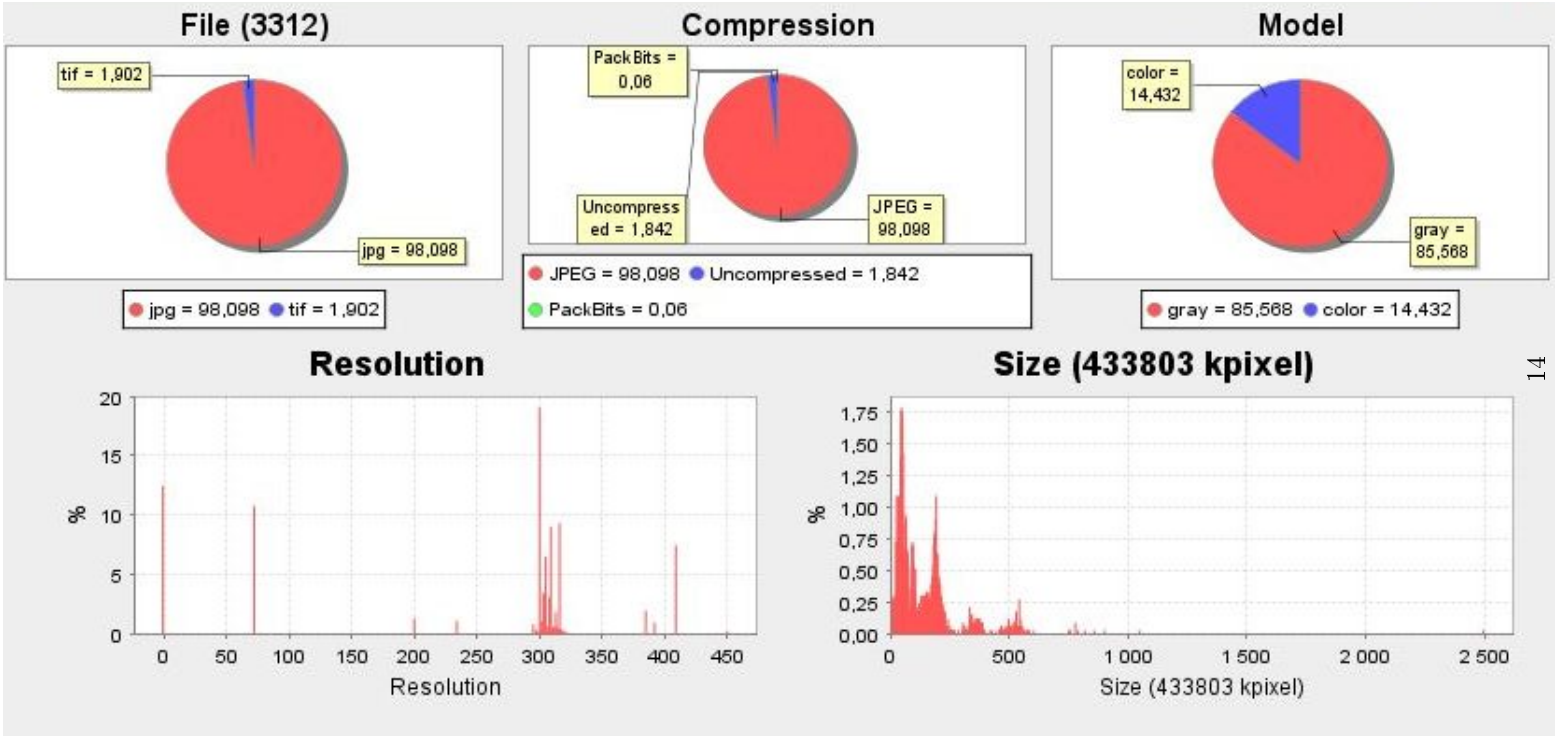

FIG. 8 – Analyse de *OLDB*

L'expertise des membres des *BVH* et l'utilisation de *QUEID* en mode requête a permis d'identifier les différentes sources d'hétérogénéité au sein de *OLDB*. La liste qui suit résume les principales :

1. La politique de numérisation a changé en cours de projet, le format de stockage est passé du *Tiff* non compressé au *Jpg* faiblement compressé.
2. Les variations des modes de compression du format *Tiff* sont des erreurs de numérisation.
3. La présence d'images couleur est liée à l'utilisation d'une plate-forme de segmentation. Celle-ci convertit par défaut les images en niveaux de gris en images couleur.
4. La présence d'images de résolution à 72 ppi est liée à une mauvaise utilisation d'un éditeur d'images. Des sauvegardes malencontreuses ont écrasé les résolutions d'origines par une résolution par défaut à 72 ppi.
5. Les images de résolutions inconnues sont de deux origines. Tout d'abord, elles sont liées à une mauvaise prise en compte du format *Jpg* par la *JAI* et sont donc mal traitées par *QUEID*. Ensuite, certaines images ont été re-dimensionnées pour des besoins de visualisation par le *CESR*, leurs résolutions d'origines ont donc été écrasées.
6. La présence d'images de plus faibles résolutions de 200 à 250 ppi est liée à l'utilisation d'un scanner à plat en début de projet.
7. La présence d'images de plus fortes résolutions de 375 à 425 ppi est liée à l'incorporation d'images issues d'autres bibliothèques numériques.

5 Conclusions

Dans ce rapport nous avons présenté notre moteur *QUEID* d'analyse de base d'images. Ce moteur permet de détecter les hétérogénéités au sein des bases par extraction des attributs des images. Basé sur cette extraction *QUEID* peut être utilisé en deux modes différents : analyse et contrôle. Le mode analyse permet à un utilisateur de détecter des images mal formatées au sein d'une base. Le mode contrôle permet lui à un système de filtrer en entrée les images à traiter en fonction de paramètres qui lui sont propres.

Nous avons ensuite présenté les packages principaux de *QUEID* : *queid.img* et *queid.ihm*. Le premier est central au moteur *QUEID*, il permet l'analyse des images et repose sur l'utilisation de la librairie *JAI*. Le second permet principalement la définition de paramètres via l'utilisation de l'IHM de *QUEID* et la lecture de fichiers *XML*. Ces paramètres servent alors aussi bien à la définition de requêtes en mode analyse qu'à celle des filtres à appliquer en mode contrôle. Finalement nous avons présenté un cas d'usage de *QUEID* sur une base d'images numérisées. Nous avons illustré les hétérogénéités existantes au sein de cette base et montré comment *QUEID* permettait l'identification de leurs origines.

6 Remerciements

Je tiens à remercier le *CESR* d'avoir mis à disposition les images utilisées pour le développement et test de *QUEID*. Plus particulièrement, je tiens à remercier *Sébastien Busson* du projet *BVH* pour s'être intéressé à ce moteur et pour avoir accepté de mettre au "banc d'essai" les bases d'images produites par le *CESR*.

Références

[1] G. Cleveland, Digital libraries : Definitions, issues, and challenges, International Federation of Library Associations and Institutions (IFLA), Universal Dataflow and Telecommunications Core Program (UDT), Occasional Paper 8 (1998).

[2] F. Lebourgeois, al, Documents images analysis solutions for digital libraries, in : Workshop on Document Image Analysis for Libraries (DIAL), 2004, pp. 2–24.

[3] Minerva, Recommandations techniques pour les programmes de création de contenus culturels numériques, Tech. rep., Université de Bath, Bath, UK (2004).

[4] L. Rodrigues, Building Imaging Applications with Java Technology, Addison Wesley Publishing, 2001.

[5] B. Eckel, Thinking in Java, 2nd Edition, MindView Inc, 2000.

[6] Sun, Programmer's guide to the java 2d api, Tech. rep., Sun Microsystems, Inc (2001).

[7] R. Santos, Java Advanced Imaging API : A Tutorial, Divisão de Sensoriamento Remoto, Instituto de Estudos Avançados, Centro Técnico Aeroespacial, São Paulo, Brasil (2004).

[8] J. Murray, W. V. Ryper, Encyclopedia of Graphic File Formats, 2nd Edition, Editions O'Reilly, 1996.

[9] J. Velu, Méthodes Mathèmatiques pour l'Informatique, Editions Dunod, 1999.

[10] D. Gilbert, The jfreechart class library, developer guide, Tech. rep., Object Refinery Limited (2005).

[11] A. Michard, XML Langage et Application, 2nd Edition, Editions Eyrolles, 2000.

Table des figures

Liste des tableaux