Principes et mise en oeuvre de l'interopérabilité dans le système d'analyse de documents Qgar

Jan Rendek, Philippe Dosch, Gérald Masini, Karl Tombre

Principes et mise en œuvre de l'interopérabilité dans le système d'analyse de documents Qgar

Jan Rendek – Philippe Dosch – Gérald Masini – Karl Tombre

LORIA, UMR 7503
615 rue du jardin botanique, B.P. 101, 54602 Villers-lès-Nancy Cedex, France

Jan.Rendek@loria.fr, Philippe.Dosch@loria.fr, Gerald.Masini@loria.fr, Karl.Tombre@loria.fr

Résumé : *Cet article décrit d'abord brièvement la plate-forme logicielle d'analyse de documents Qgar, son architecture, ses grandes composantes, ainsi que les choix techniques retenus pour les outils et les formats utilisés. Il décrit ensuite plus précisément la solution mise en œuvre pour faciliter l'interopérabilité au sein de la plate-forme, c'est-à-dire l'intégration des applications réalisées avec les briques logicielles fournies par la bibliothèque du système et l'intégration des applications importées. Chaque application est associée à un document XML qui fournit ses caractéristiques : description à l'usage des clients, types et valeurs des paramètres. Bien que simple et pragmatique, cette solution se révèle performante, pour exporter des applications Qgar ou pour importer des applications tierces, comme le montrent les exemples donnés.*

Mots-clés : interopérabilité, plate-forme logicielle, analyse de documents, XML

1 Introduction

La conception de systèmes logiciels pour l'analyse de documents, qui est un domaine pluridisciplinaire, nécessite la maîtrise de nombreuses techniques complexes. Un tel système doit intégrer des modules implantant des méthodes robustes correspondant aux traitements indispensables à la réalisation des applications du domaine : binarisation, séparation texte-graphique, vectorisation, reconnaissance de caractères, etc. Le système doit également être facile à utiliser, et donc être pourvu d'une interface utilisateur ergonomique, bien documentée et facile à installer.

Ces considérations sont bien connues des chercheurs du domaine et ont été la source d'un certain nombre de projets, dont IUE (*Image Understanding Environment*) [KOH 94, LER 98], qui a pour but de fournir un environnement complet de traitement d'images, est certainement l'un des plus célèbres. Le système qui en a résulté a permis de lancer nombre d'idées utiles et intéressantes, mais il reste extrêmement complexe et très difficile à maîtriser, donc d'un intérêt pratique relativement limité. IUE pèche sans conteste par son caractère trop universel. En revanche, beaucoup de systèmes visant des domaines restreints, bien ciblés, ont été mis au point avec succès. En particulier, on a très vite su construire des systèmes OCR à vocations diverses qui soient performants, avec parfois même des post-traitements linguistiques [BAI 92]. Il en est de même pour les systèmes traitant des documents métiers, comme celui de Dengel *et al.* [DEN 02, DEN 03], qui est notamment capable de s'adapter à différents types de documents. Tous les grands domaines d'application ont donné naissance à des systèmes spécialisés : chèques bancaires [GOR 01], tableaux [SHA 97], formulaires [NIY 97], ou encore écriture manuscrite [CRA 99]...

En ce qui concerne les documents à forte teneur graphique, très peu d'outils génériques ont été développés. La plupart des systèmes existants sont très dépendants du contexte et ne sont capables de traiter que des problèmes spécifiques. Citons le système de Pasternak [PAS 96] basé sur une description hiérarchique structurelle couplée à des mécanismes de type « réflexe » pour interpréter le contenu du document. Le système DMOS [COÜ 01, COÜ 03], développé à l'IRISA, est un autre exemple. Le domaine d'application est cette fois décrit grâce à un formalisme syntaxique.

Un critère important — peut-être le plus important — n'a encore pas été évoqué. Nous le nommerons, de manière peut-être impropre, « interopérabilité ». Il est d'ailleurs négligé par la plupart des systèmes évoqués et peut être résumé de la manière suivante : un système d'analyse de documents doit être ouvert et facilement interfaçable avec d'autres systèmes. Il doit en effet pouvoir être continuellement enrichi de nouvelles méthodes, mises au point localement ou bien publiées dans la littérature, les concepteurs du système étant bien souvent ses premiers clients. Toutefois, même avec le plus grand soin donné à tous les aspects des problèmes d'analyse de document, une équipe ne peut pas toujours disposer des meilleures méthodes pour un problème donné. Elle doit donc pouvoir, à la demande, interfacer son système avec une application fournie par un système externe et, inversement, fournir ses propres applications pour qu'elles soient interfacées avec un système externe. Dans tous les cas, cela implique le respect des critères classiques de génie logiciel pour la conception du système (modularité, réutilisabilité, extensibilité, etc.), ainsi que l'adoption de standards pour les formats (de données) et les protocoles d'échange.

Dans le système d'analyse de documents Qgar, développé par l'équipe du même nom au LORIA, nous avons essayé de tenir compte de tous les critères évoqués précédemment, et plus particulièrement de l'interopérabilité. Nous avons volontairement privilégié des solutions simples et pragmatiques, parce que faciles à mettre en œuvre et à utiliser, comme nous allons le montrer dans la suite.

2 Présentation du système

Une équipe de recherche ne peut pas se permettre d'œuvrer dans un même domaine, l'analyse de documents en l'occurrence, en repartant de zéro pour chaque nouvelle application traitée. Il est primordial de pouvoir réutiliser une partie des logiciels réalisés à chaque occasion, ainsi que l'expérience acquise à travers ces réalisations. C'est pourquoi l'équipe Qgar a entrepris depuis plusieurs années un effort considérable pour réaliser une base de travail sous la forme d'un système logiciel permettant de développer des applications d'analyse de documents graphiques [DOS 99].

Le système comprend trois grandes parties. *QgarLib* est une boîte à outils qui fournit les traitements d'images et de graphiques élémentaires. Il s'agit en fait d'une bibliothèque de classes C++, actuellement au nombre d'une centaine, représentant environ 60 000 lignes de code. Elles implantent les outils classiques décrits dans la littérature, adaptés et améliorés, ainsi que les outils mis au point localement dans l'équipe, qui opèrent du niveau du pixel jusqu'au niveau du vecteur, indépendamment d'un modèle de document donné : binarisation, détection de contours, squelettisation, approximation polygonale, etc.

La conception des classes obéit à des considérations pragmatiques, inspirées des solutions adoptées pour la bibliothèque *ImageVision* (IL) de Silicon Graphics [ECK 96]. L'idée sous-jacente est très simple : une classe représentant un objet de base, comme une image, constitue la racine d'une hiérarchie et chaque traitement qui opère sur cet objet est implanté par le constructeur d'une classe dérivée. Par exemple, le calcul du gradient avec l'opérateur de Deriche est implanté par le constructeur de la classe `DericheGradientImage`, qui dérive de la classe `Image`. Cette solution offre de multiples avantages. Différentes méthodes pour effectuer la même opération conceptuelle peuvent être facilement implantées par des classes dérivant d'une même classe abstraite. L'adjonction progressive de nouvelles opérations sur les images ne provoque pas l'extension démesurée de l'interface de la classe `Image`, jusqu'à la rendre inutilisable. Intégrer une nouvelle opération à la bibliothèque, qu'elle soit écrite en C ou en C++, qu'elle ait été mise au point dans l'équipe même, ou dans une équipe tierce, ne nécessite ainsi aucune modification de la hiérarchie des classes. Enfin, le code écrit par les concepteurs aussi bien que par les clients de la bibliothèque obéit aux critères habituellement préconisés en génie logiciel [MEY 97] : compacité, lisibilité, modularité, etc. La figure 1 montre un exemple de code source permettant de réaliser une détection de contours.

La seconde partie du système, *QgarApps*, est une bibliothèque d'applications réalisées à partir des briques de base que constituent les outils de QgarLib. Ces applications sont finalisées et stables, ou encore expérimentales, c'est-à-dire correspondant aux recherches en cours. Enfin, une interface utilisateur, baptisée *QgarGUI*, chapeaute le tout et permet de tester les applications. Elle est également écrite en C++, à partir de la bibliothèque graphique Qt, et représente environ 15 000 lignes de code.

Conception et développement sont menés dans une optique semi-professionnelle, de manière normalisée. Pour chaque (nouvelle) classe sont systématiquement produits une documentation détaillée, en utilisant *Doxygen*, et un module de tests unitaires, avec *CPP Unit*, permettant de valider l'implantation de la plate-forme et de mesurer la fiabilité et la qualité de ses outils. Un soin particulier est consacré au support de tous systèmes Unix/Linux et à la configurabilité (grâce à `autoconf`/`automake`). Le logiciel est déposé à l'Agence pour la Protection des Programmes et il est distribué sous licence LGPL/QPL depuis un site internet dédié[1].

3 Interopérabilité

La deuxième couche du système, *QgarApps*, est donc constituée d'un ensemble d'applications. Certaines sont de simples encapsulations des classes C++ correspondantes de la bibliothèque *QgarLib*. Les autres sont des programmes plus spécifiques que ceux de la bibliothèque, qui sont par essence génériques. Une dizaine d'applications sont pour l'instant finalisées et disponibles : binarisations, morphologie mathématique, séparation texte-graphique, séparation traits forts-traits fins, vectorisations, etc. Ce sont des programmes indépendants, qui communiquent par fichiers. L'utilisation d'outils comme CORBA (pour les échanges) ou d'une architecture répartie, à base d'agents par exemple, a été volontairement écartée au profit d'une solution plus pragmatique, simple et facile à mettre en œuvre, comme nous l'avons mentionné en introduction.

Les applications sont exécutables à partir d'une ligne de commande et fonctionnent comme des boîtes noires, auxquelles le client fournit les paramètres nécessaires à leur exécution : images en entrée, valeurs des paramètres intrinsèques d'exécution, et images en sortie. L'interopérabilité et la capacité d'intégration du système Qgar reposent sur l'utilisation de ces applications, qui peuvent être appelées séquentiellement, pour construire des chaînes de traitements (de graphiques). L'interopérabilité est un critère primordial pour construire de telles chaînes, mais aussi pour faciliter les comparaisons de différentes méthodes exécutant une même tâche, notamment à des fins d'évaluation de performances.

Une application seule ne suffit cependant pas à constituer un module prêt à la (ré)utilisation. Pour être en mesure de l'exploiter, il faut disposer d'un mode d'emploi, comprenant au minimum un descriptif de l'application elle-même, de ses paramètres en entrée et sortie, et de la syntaxe de sa ligne de commande.

Si une documentation exhaustive suffit à un utilisateur humain, une méthode plus fonctionnelle est indispensable pour interagir avec d'autres briques logicielles. Un système intégrant des applications doit pouvoir « comprendre » comment manipuler celles-ci de manière automatisée. Notre première approche a consisté à intégrer directement les informations nécessaires dans l'application. L'exécutable pouvait être interrogé par un appel spécial retournant les informations nécessaires à sa mise en œuvre, le tout étant complété par une documentation utilisateur séparée. Cette approche avait pour intérêt principal de regrouper dans un fichier unique le traitement et les moyens de le lancer, la documentation étant diffusée à part. Elle était cependant limitée car, avant de pouvoir réellement collaborer avec une application, un système devait

[1] `http://www.qgar.org/`

```
PgmFile f("IMG.pgm");   // Fichier contenant l'image originale
GreyLevelImage img(f); // Lire l'image à niveaux de gris
// Opérateur de Deriche, paramètres par défaut
DericheGradientImage gradientImg(img);
// Maxima du gradient
GradientLocalMaxImage maxGradImg(gradientImg);
// Extraction des contours par seuillage hysteresis
HysteresisThresholdBinaryImage edgesImg(maxGradImg, 0, 5);
// Chaînage des contours
LinkedChainList edgesChains(edgesImg, 1, 0);
```

FIG. 1 – Un exemple de code source écrit avec la bibliothèque QgarLib et réalisant une détection de contours.

d'abord l'invoquer. Ceci peut être rédhibitoire dans le cadre d'une architecture répartie, ou encore lors de l'intégration dans un outil de construction de scénarios comme celui du projet DocMining (cf. paragraphe 4), puisque les applications ne sont alors pas nécessairement disponibles pendant la phase de conception de la chaîne de traitements.

Ce modèle n'était pas non plus adapté à l'intégration d'applications tierces, qui devaient être construites sur le même modèle que les applications Qgar pour être prises en compte. Une application tierce devait donc être encapsulée dans un *wrapper* fournissant les services requis. Si le développement de ce module d'interfaçage était simple, une grande partie du code nécessaire étant disponible dans la bibliothèque *QgarLib* sous forme d'une classe abstraite à implanter, cette méthode impliquait forcément une intervention humaine et n'était pas réaliste à grande échelle.

La solution que nous avons retenue consiste à fournir avec l'application un document regroupant les informations nécessaires au client, que ce soit un humain ou une machine, sous la forme d'un document XML. Toute application appelable depuis la ligne de commande et accompagnée d'un tel « mode d'emploi », conforme aux normes prédéfinies, peut ainsi être immédiatement intégrée dans le système Qgar. Ce principe ne nécessite en aucune manière de disposer du code source de l'application à intégrer. Il permet aussi de concevoir des chaînes de traitements sans disposer directement des exécutables, ce qui n'est pas sans avantage en cas d'intégration dans un système non interactif. Enfin, disposer de la description d'une application dans un format normalisé et générique permet de l'interfacer facilement avec tout système utilisant une méthode similaire, même si le format de description est différent.

Le formalisme XML nous a semblé le mieux adapté pour décrire les applications. C'est un format standard de représentation d'information structurée et il existe de nombreux outils du domaine public permettant de l'exploiter, indépendamment de la plate-forme ou du langage de programmation choisis. Visualiser un document XML ou y localiser automatiquement une information précise ne pose aucune difficulté. En outre, le langage de transformation XSLT et les outils qui l'implantent permettent de convertir un document XML dans n'importe quel format de document textuel. Il est ainsi possible de construire des filtres rendant compatible la description d'une application avec celle d'un autre système basé sur le même principe. Un exemple d'une telle adaptation est montré au paragraphe 4.

La figure 2 montre la description de l'application `ThresholdedBinarization`, qui réalise une binarisation à seuil fixe, avec trois paramètres : le nom du fichier de l'image source, le nom du fichier dans lequel stocker l'image résultat et la valeur du seuil à employer. L'application proprement dite est décrite par la balise `descr`, avec le nom de l'application, le nom de ses auteurs, le détail sur les droits d'utilisation (copyright, type de licence), ainsi qu'une documentation à destination des clients, indiquant le rôle de l'application, les services qu'elle propose et les méthodes employées. Une balise `param` donne ensuite la description de chaque paramètre, comprenant le drapeau (`flag`) qui introduit éventuellement le paramètre sur la ligne de commande, le caractère obligatoire ou optionnel du paramètre (`required`), son mode de passage (`passing-mode`), c'est-à-dire « en entrée » ou « en sortie », son type (image binaire, image à niveaux de gris, valeur numérique, etc.), son format (PGM, PBM, chaîne de caractères, etc.), ses bornes inférieure ou supérieure s'il y a lieu, sa valeur par défaut, ainsi qu'une documentation à l'usage du client. Voici une ligne de commande typique pour invoquer l'application :

```
% ThresholdedBinarization  \
    -in  ImageSource.pgm   \
    -out ImageDest.pbm     \
    -thr 126
```

Cette description répond pleinement à nos besoins puisque toutes les informations nécessaires à l'intégration de l'application dans un système de construction de chaînes de traitement sont disponibles : l'agencement des paramètres sur la ligne de commande, le type et le format des paramètres, le fait que ceux-ci sont modifiés ou non à l'exécution... Une documentation complète et structurée sur l'application est également accessible et peut être facilement intégrée dans un système d'aide en ligne.

L'interface QgarGui ne fournit toutefois pas (encore) d'outil permettant de construire automatiquement le fichier de description d'une application écrite avec QgarLib. Elle ne fournit pas non plus d'outil permettant de construire une nouvelle application en enchaînant des applications existantes. C'est l'utilisateur qui doit composer « à la main » le fichier de description correspondant, ainsi que le script regroupant les lignes de commande qui permettent d'invoquer les applications concernées.

```
<application>
  <descr>
    <name>Fixed Binarization</name>
    <author>Qgar Project, LORIA</author>
    <copyright>C 2004, Qgar Project, LORIA</copyright>
  </descr>
  <documentation>... Documentation de l'application ...</documentation>
  <paramlist>
    <param name="Source Image" flag="in" required="true" passing-mode="in"
           type="grayscale" format="PGM">
      <documentation>... Documentation du paramètre ...</documentation>
    </param>
    <param name="Target Image" flag="out" required="true" passing-mode="out"
           type="binary" format="PBM" default=".pbm">
      <documentation>... Documentation du paramètre ...</documentation>
    </param>
    <param name="Threshold" flag="thr" required="true" passing-mode="in"
           type="numeric" format="int" default="127" min="0" max="255">
      <documentation>... Documentation du paramètre ...</documentation>
    </param>
  </paramlist>
</application>
```

FIG. 2 – Description XML de l'application `ThresholdedBinarization` (binarisation à seuil fixe).

4 Exemples d'intégration

Le formalisme décrit précédemment rend l'intégration d'une des applications de *QgarApps* dans un autre système particulièrement aisée. L'analyse du fichier de description permet en effet de construire dynamiquement des boîtes de dialogue présentant à l'utilisateur toutes les informations nécessaires au pilotage de la dite application. Cette faculté peut être exploitée au sein d'interfaces homme-machine (y compris celle du système Qgar elle-même, *QgarGUI*), d'interfaces web ou, plus généralement, de tout autre processus interactif. De façon identique, toute application tierce peut être facilement intégrée dans ces mêmes interfaces, à condition qu'elle soit associée à un fichier de description respectant le formalisme donné. Le critère d'interopérabilité tel que nous l'avons défini — pouvoir exporter ses propres réalisations aussi bien qu'importer les réalisations des autres — est ainsi satisfait, comme le montrent les exemples qui suivent.

QgarGUI, la troisième partie du système Qgar, est, comme nous l'avons dit, une interface homme-machine (IHM) qui fournit aux utilisateurs un environnement convivial permettant de visualiser et d'interagir avec les images manipulées par les processus d'analyse de documents, par exemple pour les corriger manuellement. Elle permet également de contrôler les applications de *QgarApps*, en ajustant les valeurs de leurs paramètres et en construisant interactivement des chaînes de traitements. L'interface *QgarGUI* est complètement indépendante du reste du système, mais le formalisme décrit dans le paragraphe précédent permet évidemment de l'interfacer rapidement et très facilement avec nos applications. Pratiquement, l'approche que nous avons choisie pour cela est de type *plug-in*. Les applications sont stockées avec leurs fichiers de description dans un répertoire particulier, déterminé lors de l'installation de l'interface. Les fichiers de description sont automatiquement analysés au lancement de l'IHM et les applications correspondantes sont dynamiquement ajoutées au menu des traitements de l'IHM. Les boîtes de dialogue permettant d'exécuter les applications sont automatiquement générées à partir des fichiers de description. L'utilisateur peut ainsi consulter interactivement la documentation correspondante et fixer les valeurs des paramètres, comme le montre la figure 3 pour l'application qui effectue la séparation texte-graphique. Les boîtes de dialogue sont réalisées avec des composants (*widgets*) classiques dans toutes les boîtes à outils graphiques, Qt ou autre, ce qui ne remet donc pas en cause notre approche. Comme toute nouvelle application peut être ajoutée (ou retirée) sans avoir à modifier l'IHM, il est très facile d'installer (provisoirement ou non) des applications résultant des recherches en cours dans l'équipe, ou bien encore fournies par une autre équipe de recherche, à des fins de test par exemple.

Toujours selon l'approche préconisée, il est tout aussi facile de mettre une application à disposition depuis un site web. C'est en particulier ce qui a été fait pour les démonstrations dynamiques proposées par le site web dédié au système Qgar. L'analyse du fichier de description est cette fois réalisée par des scripts PHP qui génèrent les formulaires adéquats en HTML. La figure 4 présente le formulaire correspondant à l'application de séparation texte-graphique déjà mentionnée.

Pour finir, les applications Qgar ont pu être intégrées sans effort à une plate-forme tierce conçue pour l'interprétation de documents [CLA 03], en l'occurrence celle du projet RNTL DocMining[2]. Celle-ci intègre un moteur de scénarios d'analyse d'images, qui permet la construction et le pilotage de l'exécution de chaînes de traitements développés séparément par les partenaires. De fait, les applications Qgar ont dû se conformer à un formalisme de description défini au sein de ce projet et basé sur XML. La souplesse de manipula-

[2]Le consortium menant le projet réunissait les universités de Fribourg, La Rochelle, Rouen, le LORIA et France Télécom R&D.

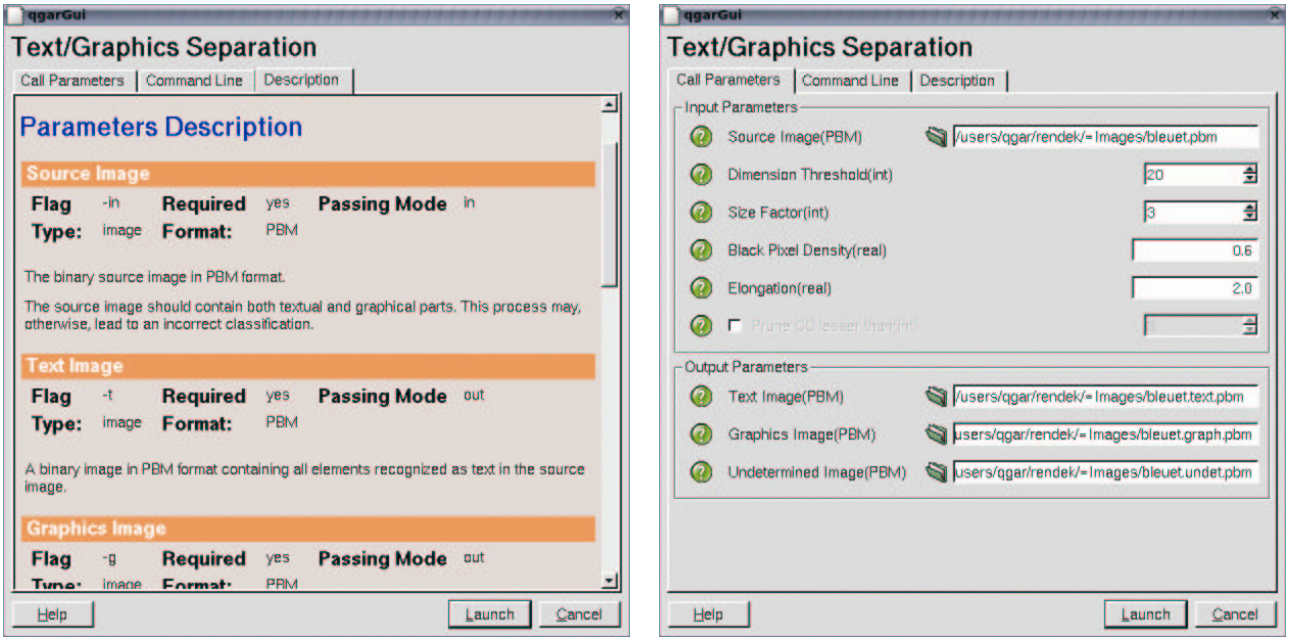

FIG. 3 – Boîtes de dialogues automatiquement construites par *QgarGUI* pour l'invocation de l'application réalisant la segmentation texte-graphique. Les zones de saisie sont automatiquement personnalisées en fonction du type des paramètres et une aide en ligne est disponible pour chacun d'entre eux.

FIG. 4 – Formulaire HTML construit automatiquement pour la segmentation texte-graphique sur le site web du système Qgar.

tion des données XML a permis une conversion automatisée des descriptions au format Qgar dans le format imposé. Mis à part cette conversion, l'intégration n'a requis aucun développement spécifique.

5 Conclusion

Nous pensons avoir montré l'intérêt d'une démarche favorisant la prise en compte de l'interopérabilité des traitements/applications dans le cadre de la réalisation d'une plate-forme d'analyse de documents. Tout en étant simple à mettre en œuvre, le formalisme proposé ne nécessite aucune modification de l'existant et se révèle suffisamment puissant pour permettre l'intégration d'applications dans des environnements différents, comme l'ont montré les exemples donnés. Comme la généricité, thème déjà développé dans nos précédents articles [TOM 98a], l'interopérabilité favorise la réutilisation, qui est un des critères primordiaux dans ce contexte. Ces deux aspects s'avèrent d'ailleurs particulièrement complémentaires et essentiels pour l'évaluation de performances, un des problèmes au cœur des préoccupations actuelles de notre communauté [TOM 98b]. Nous comptons d'ailleurs orienter le développement du système Qgar vers l'intégration d'outils permettant l'étude de cette problématique.

Références

[BAI 92] BAIRD H., Anatomy of a versatile page reader, *Proceedings of the IEEE*, vol. 80, n° 7, 1992, pp. 1059–1065.

[CLA 03] CLAVIER E., MASINI G., DELALANDRE M., RIGAMONTI M., TOMBRE K., GARDES J., DocMining : A cooperative platform for heterogeneous document interpretation according to user-defined scenarios, *Proceedings of the 5th IAPR International Workshop on Graphics Recognition, Barcelona (Spain)*, 2003, pp. 21–32.

[COÜ 01] COÜASNON B., DMOS : A generic document recognition method—Application to an automatic generator of musical scores, mathematical formulae and table structures recognition systems, *Proceedings of the 6th International Conference on Document Analysis and Recognition, Seattle (Washington, USA)*, 2001, pp. 215–220.

[COÜ 03] COÜASNON B., Dealing with noise in DMOS, a generic method for structured document recognition, *Proceedings of the 5th IAPR International Workshop on Graphics Recognition, Barcelona (Spain)*, 2003, pp. 44–54.

[CRA 99] CRACKNELL C., DOWNTON A., A Handwriting Understanding Environment (HUE) for rapid prototyping in handwriting and document analysis research, *Proceedings of the 5th International Conference on Document Analysis and Recognition, Bangalore (India)*, 1999, pp. 362–365.

[DEN 02] DENGEL A., KLEIN B., smartFIX : A requirements-driven system for document analysis and understanding, LOPRESTI D., HU J., KASHI R., Eds., *Proceedings of the 5th IAPR International Workshop on Document Analysis Systems, Princeton (New Jersey, USA)*, Lecture Notes in Computer Science, vol. 2423, pp. 433–444, Springer-Verlag, Berlin, 2002.

[DEN 03] DENGEL A., Making documents work : Challenges for document understanding, *Proceedings of the 7th International Conference on Document Analysis and Recognition, Edinburgh (Scotland)*, 2003, pp. 1026–1035.

[DOS 99] DOSCH P., AH-SOON C., MASINI G., SÁNCHEZ G., TOMBRE K., Design of an integrated environment for the automated analysis of architectural drawings, LEE S., NAKANO Y., Eds., *Document analysis systems : Theory and practice. Selected papers from DAS'98 (Nagano, Japan)*, Lecture Notes in Computer Science, vol. 1655, pp. 295–309, Springer-Verlag, Berlin, 1999.

[ECK 96] ECKEL G., NEIDER J., BASSLE E., *ImageVision Library programming guide*, Silicon Graphics, Inc., 1996.

[GOR 01] GORSKI N., ANISIMOV V., AUGUSTIN E., BARET O., MAXIMOV S., Industrial bank check processing : the A2iA CheckReader, *International Journal on Document Analysis and Recognition*, vol. 3, n° 4, 2001, pp. 196–206.

[KOH 94] KOHL C., MUNDY J., The development of the Image Understanding Environment, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Seattle (Washington, USA)*, 1994, pp. 443–447.

[LER 98] LERNER R., The Image Understanding Environment progress since IUW'97, *Proceedings of the 1998 DARPA Image Understanding Workshop, Monterey (California, USA)*, 1998.

[MEY 97] MEYER B., *Object-oriented software construction, second edition*, The Object-Oriented Series, Prentice-Hall, Englewood Cliffs (New Jersey, USA), 1997.

[NIY 97] NIYOGI D., SRIHARI S., GOVINDARAJU V., Handbook of character recognition and document image analysis, BUNKE H., WANG P., Eds., *Analysis of printed forms*, Chapitre 18, pp. 485–502, World Scientific, 1997.

[PAS 96] PASTERNAK B., Adaptierbares Kernsystem zur Interpretation von Zeichnungen, Dissertation zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.), Universität Hamburg, 1996.

[SHA 97] SHAMILIAN J., BAIRD H., WOOD T., A retargetable table reader, *Proceedings of the 4th International Conference on Document Analysis and Recognition, Ulm (Germany)*, 1997, pp. 158–163.

[TOM 98a] TOMBRE K., AH-SOON C., DOSCH P., HABED A., MASINI G., Stable, robust and off-the-shelf methods for graphics recognition, *Proceedings of the 14th International Conference on Pattern Recognition, Brisbane (Australia)*, 1998, pp. 406–408.

[TOM 98b] TOMBRE K., CHHABRA A. K., Graphics recognition—Algorithms and systems, TOMBRE K., CHHABRA A. K., Eds., *General conclusions*, Lecture Notes in Computer Science, vol. 1389, pp. 411–420, Springer-Verlag, Berlin, 1998.