Une méthodologie de développement d'applications de traitement d'images

Régis Clouard, Abderrahim Elmoataz, Marinette Revenu

Une méthodologie de développement d'applications de traitement d'images

A methodology for the development of image processing applications

Régis Clouard[1] Abderrahim Elmoataz[1] Marinette Revenu[1]

[1]GREYC - IMAGE

Contact: Régis Clouard
GREYC - IMAGE
6, bd Maréchal Juin
14050 Caen cedex
email: Regis.Clouard@greyc.ismra.fr

Résumé

Nous proposons une méthodologie de développement d'applications de traitement d'image qui se présente comme un guide complet et rigoureux pour la gestion du cycle de vie entier d'une application. Cette méthodologie met en avant des capacités d'aide, de réutilisabilité d'expériences, d'uniformisation des représentations et de communication entre les différents partenaires, par la définition de modèles structurés, de représentations graphiques et de règles de mise en oeuvre de ces modèles à chaque étape du développement. Elle se fonde essentiellement sur le paradigme du pilotage d'une bibliothèque de tâches de traitement, dans lequel la conception d'une solution est vue comme un processus d'agglomération de tâches ponctuelles et indépendantes. Ce travail s'inscrit dans une perspective plus générale d'ingénierie des connaissances en traitement d'images.

Mots Clef

Traitement d'image, méthodologie de développement d'applications, pilotage de programmes, graphe d'opérateurs, programmation visuelle.

Abstract

A methodology of image processing applications development is presented, which is a complete and rigorous guide for the management of all the life cycle of an application. This methodology points out aids, reusing and unifying capabilities for representations and communications between the partners, by defining structured models, graphical representations and implementing rules at each stage of the development proceeding. It is essentially based on the supervision of library of image processing tasks paradigm, in which the conception of a solution is view as a process of agglomeration of punctual and independent tasks. This work is considered as a prerequisite in image processing knowledge management issues.

Keywords

Image processing, methodology of applications development, supervision of a library of programs, visual programming.

1. Introduction

Le développement d'une application de traitement d'images consiste à produire un programme exécutable spécialisé dans la réalisation d'un *objectif de traitement* et dédié à une *classe d'images*. Nous nous limiterons ici aux *objectifs* de transformations d'images en images sans interprétation du contenu. Ainsi, un objectif de traitement d'images n'est pas une fin en soi, mais il s'inscrit comme une étape dans une chaîne plus complète qui va de l'acquisition à l'interprétation. De même, une *classe d'images* se restreint à un catégorie d'images se rapportant à un même sujet, visualisant un même type de scène et acquises avec le même type de capteur dans les mêmes conditions (éclairage, résolution,...).

Ce que l'on exige d'un programme de traitement d'images, c'est qu'il soit capable de traiter toutes les images de la classe étudiée avec les mêmes performances. Le développeur devra donc particulièrement veiller à donner à son application des qualités d'adaptabilité et de robustesse pour lui permettre de faire face à la variabilité inhérente des images numériques.

0.1 Pourquoi est ce si compliqué de développer une application de Traitement d'images?

Malgré la grande masse d'applications développées à ce jour, le développement d'une nouvelle application reste une tâche très difficile qui est à la fois complexe et compliquée (Zamperoni le compare à un art [23]). La tâche est complexe dans la mesure où elle s'inscrit dans un environnement qui comporte un très grand nombre d'éléments à combiner, tant au niveau des opérations, des données, que des techniques à mettre en oeuvre. Elle est compliquée dans la mesure où elle fait appel à l'intégration de compétences de formes (numériques, symboliques), d'origines différentes (traitement du signal, mathématiques, physique, optique, IA, etc.) et de niveaux d'abstraction multiples.

Ceci explique le coût élevé des logiciels spécialisés et fait du traitement des images un ensemble de ressources scientifiques qui reste hors de portée des non-spécialistes [5].

0.2 On pourrait penser s'appuyer sur le génie logiciel, mais il n'est d'aucun secours

Pour tenter de réduire la difficulté de la tâche du développement, les solutions à déployer sont bien évidemment à rechercher dans le génie logiciel. Mais, bien qu'une application puisse être considérée comme un logiciel à part entière, elle s'en distingue par le fait que les méthodologies classiques du génie logiciel (e.g., Merise[16], SADT[11], OMT[19]), de même que les modélisations type UML[19], sont de peu de secours pour développer un système de traitement d'images, à l'instar des systèmes cognitifs [9].

En effet, la difficulté ne porte pas sur le logiciel en tant que tel (i.e., ses interfaces, la modélisation des données, les événements), mais sur le choix, l'adaptation et la programmation des méthodes de calculs numériques. Le développeur traiteur d'image ne peut se baser que sur ses connaissances scientifiques (mathématique, statistiques, traitement du signal, informatiques,etc.), sur son talent heuristique et sur son expérience pour construite de tels programmes [23].

Il manque manifestement une méthodologie complète qui soit:

- un guide rigoureux qui définisse une démarche reproductible,
- un cadre structurant et unificateur qui propose des méthodes pour aborder la complexité et de modèles pour aborder la complication.
- un environnement qui offre une sécurité de programmation.

0.3 Il faut donc concevoir une méthodologie spécifique

Nous proposons ici une méthodologie qui présente de telles caractéristiques. Elle prend sa source d'une part dans le paradigme de la « Programmation Visuelle par Flot de Données » largement popularisés par les environnements tels que Khoros [17], LabView [7] ou AVS/Express [22]), et d'autre part dans celui du « Pilotage de Bibliothèques de Programmes » [15], [20].

Ils reposent tous deux sur l'existence d'une bibliothèque prédéfinie regroupant des opérateurs accessibles sous forme de programmes exécutables. Cette bibliothèque est supposée représenter un bon compromis entre être composée d'un nombre raisonnable d'opérateurs et être suffisamment complète pour envisager une grande variété de traitements. Sous ces conditions, le problème du développement est vu alors comme un problème de sélection et d'assemblage d'opérations de traitement pour former des chaînes de transformations.

Le choix de la « Programmation Visuelle par Flot de Données » comme base s'est imposé par le fait qu'elle permet de faire l'économie d'une programmation réelle. Le choix du modèle de pilotage de programmes s'est imposé par l'intérêt qu'il présente pour le partage et la réutilisation de codes [20].

Ce travail s'inscrit dans une perspective d'ingénierie des connaissances et de capitalisation de savoir-faire en traitement d'images et l'exploitation de bibliothèques de codes, au même titre que les travaux tels que [14] et [6]. Toutefois, nos motivations particulières portent sur les conditions et les moyens d'accroître la production d'applications pour construire un corpus d'expérience significatif et structuré qui ait une réelle plausibilité cognitive.

0.4 Plan de l'article

Les détails de la méthodologie sont présentés dans la section 2. Nous utiliserons le cas d'une application en géographique comme exemple fil rouge. Les images, dont un exemple est donné (Fig. 1), sont des vues aériennes de régions rurales, que les géographes veulent utiliser pour mener une étude diachronique de l'utilisation des sols. Ils recherchent au travers du traitement d'images, une source d'information objective leur permettant de disposer de mesures de comparaisons fiables et stables.

Dans la section section 3, nous discutons de la validité de la méthodologie, puis de ses avantages et ses inconvénients.

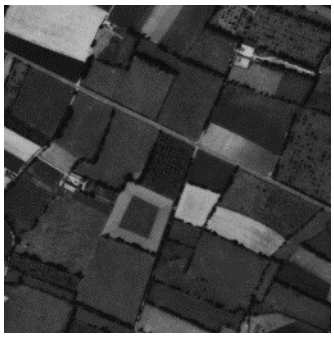

Fig. 1. Les images de l'application sont des vues aériennes de régions du bocage du calvados en France (origine IGN).

1 La méthodologie

Classiquement, notre méthodologie décompose le processus de développement en activités séquentielles pour lesquelles sont définis des modèles plus ou moins formels. Le tout est organisé autour d'un cycle d'abstraction, d'un cycle vie et d'un cycle auteur/lecteurs, ce qui fait de notre méthodologie, une méthodologie complète et rigoureuse.

1.1 Les partenaires

La méthodologie identifie quatre types de partenaires dans le développement d'une solution logicielle complète: le programmeur, le développeur, le client et l'expert de traitement d'images. Chaque type peut être représenté par une ou plusieurs personnes physiques dans un projet. Le rôle du développeur est bien évidemment central, il intervient à toutes les activités. Les trois autres partenaires interviennent à la mesure de leurs compétences lors d'activités bien précises.

1.1.1 Le programmeur

C'est un informaticien dont le rôle est de coder les programmes de la bibliothèque avec un langage de programmation impératif.

1.1.2 Le développeur

C'est un traiteur d'image qui a la connaissance des outils de développement qu'il a sa disposition. Il élabore les solutions et les réalise sous forme de graphe de programmes. Par contre, il est supposé ignorant dans le domaine de l'application qu'il a à développer.

1.1.3 Le client

Il possède le problème à résoudre dans les termes de sa spécialité. Il est capable d'évaluer les résultats finaux et de proposer des moyens d'évaluer ces résultats. Par contre, il est supposé ignorant en traitement d'images.

1.1.4 L'expert de traitement d'images

L'expert de traitement d'images possède les compétences pour juger de la pertinence de solutions face à un problème ponctuel strictement de traitement d'images. Comme le développeur, il est supposé ignorant dans le domaine d'application.

1.2 Les activités

En tant que processus de développement de logiciel, la réalisation d'une application de traitement d'images fait appel à quatre activités principales: Analyse des besoins, Conception, Programmation, Évaluation. Ces activités prennent une forme particulière ici, compte-tenu des paradigmes de base choisis que sont la programmation visuelle par flot de données et le pilotage de programmes.

1.2.1 Analyse des besoins

L'analyse des besoins correspond à la définition du problème à résoudre. Elle relève d'une négociation entre le développeur et le client qui ne partagent pas les mêmes référents scientifiques. Cette négociation a pour but de cerner l'objectif de traitement et de caractériser la classe d'image. Elle correspond à une « traduction » en termes de traitement d'images des concepts métiers véhiculés par les images. Elle s'arrête à l'expression des indices visuels pertinents, et n'a nul besoin d'aboutir à une définition précise et exacte du concept. Par exemple, la définition du concept de parcelle de champs d'herbage dans le bocage normand (Fig. 1) peut s'arrêter à une forme polyédrique uniforme sombre pouvant être entourée d'une haie (qui est un autre concept à décrire).

1.2.2 Conception

La conception a pour but d'élaborer une solution conceptuelle, qui est une description fonctionnelle et abstraite de la future application. Elle décrit une liste d'algorithmes et leur enchaînement pour atteindre l'objectif initial. A ce stade, tous les choix de méthodes et de techniques de traitement doivent être faits, sans qu'il soit encore question de leur mise en oeuvre effective. De par son caractère fonctionnelle, cette activité de la méthodologie de génie logiciel SADT [12] notamment par l'utilisation des diagrammes de type "actigrammes".

L'expert de traitement d'images est sollicité lors cette activité pour critiquer les solutions élaborées ou pour en proposer de nouvelles.

1.2.3 Programmation

La programmation se réduit ici à la sélection et l'enchaînement de codes exécutables dans la bibliothèque, pour implémenter la solution conceptuelle. La construction de ce graphe fait peu appel à des choix subjectifs. Généralement, elle consiste en une traduction des algorithmes en une chaîne d'opérateurs de la bibliothèque. Par contre, une attention toute particulière doit être accordée d'une part au choix des valeurs de paramètres en fonction des contraintes spécifiées dans la tâche, et d'autre part à l'écriture de toute les expressions de contrôle identifiées. Les expressions de contrôle ne peuvent être composées qu'avec des opérateurs de la bibliothèque qui ne posent pas de problèmes de

paramétrisation. Dans le cas contraire, on certainement supposer que les structures de contrôle sont mal choisies et que la solution doit être révisée.

Le résultat est un graphe d'enchaînement des opérateurs de la bibliothèque, intégrant des structures de contrôle classiques, type boucles et alternatives. La seule réelle activité de programmation est la création de nouveaux opérateurs pour la bibliothèque. Elle est réalisé par le programmeur et le développeur.

Le reste de cette activité est entièrement réalisée par le développeur.

1.2.4 Évaluation

L'évaluation vise à éprouver la fiabilité et la robustesse de l'application face à un grand nombre d'images de la classe et à diagnostiquer les causes d'échec. Elle se décline en une validation dont le but est de s'assurer que l'on a défini le bon problème, et une vérification dont le but est de s'assurer que l'on a construit le bon programme.

L'évaluation se fait d'une part avec le client pour valider les images de sorties et avec les experts de traitement d'images pour vérifier les résultats intermédiaires.

1.3 Les modèles

Chaque activité précédente produit et utilise en sortie des modèles de l'application (Error: Reference source not found). Nous définissons 3 types de modèle: le modèle de spécification, le modèle de conception et le modèle de programmation dédiés respectivement à l'expression des besoins, la description d'une solution conceptuelle et la représentation du programme exécutable. Ces modèles s'expriment par des représentations graphiques (Diagramme) ou textuelles (Objectif, Contexte et Justifications) visualisées au travers de formulaires. Nos formulaires empruntent les notations du standard IDEF0 [8] (lié a SADT).

Activité	Modèle	Formulaires
Analyse	Spécification	Objectif Contexte
Conception	Conception	Diagramme Justifications
Implantation	Programme	Diagramme Justifications
Évaluation	Spécification	Objectif contexte

Tableau 1. Chaque activité produit des modèles visualisés à travers des formulaires.

1.3.1 Le modèle de spécification

Le modèle de spécification comporte un partie définition de l'objectif et une partie description du contexte de l'application. Ces deux parties sont conservées indépendantes et seront étudiées séparément (même si la description du contexte est strictement conditionnée par l'objectif et vice versa).

La définition de l'objectif distingue l'objectif propre du traitement d'images et l'objectif global, dont le traitement d'images n'est qu'une composante. Il est important aussi de préciser la façon dont seront utilisés les résultats du traitement d'images (post-traitements). Un objectif s'exprime à l'aide de tâches à accomplir (traduisant une intention sur les images) et de contraintes à respecter (on trouvera dans [3] les justifications de cette modélisation). Les contraintes définissent les critères de qualité attendue sur les résultats, les erreurs acceptables et les restrictions. L'objectif est répertorié dans un formulaire d'objectif. Pour le cas de l'application géographique, un objectif peut être « Extraire uniquement les parcelles de champs. Localiser avec précision les frontières des champs. Préférer couper un même champ en plusieurs que de rassembler plusieurs champs en un. » (Fig. 2).

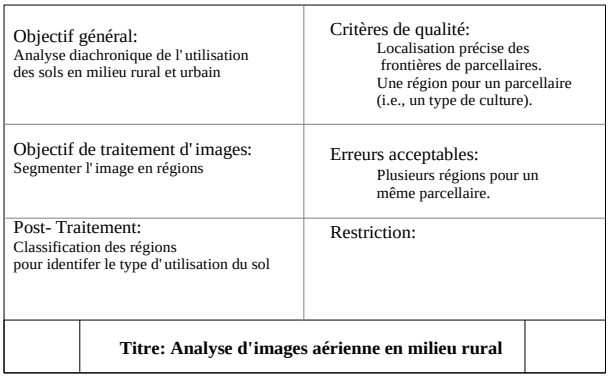

Fig. 2. Le formulaire d'objectif distingue les tâches à accomplir et les contraintes sur ces tâches.

La description du contexte se fait par énumération de caractéristiques mesurables aussi bien en termes d'invariants que de particularités. Par souci de concision et de réutilisabilité, une représentation sous forme d'une paire attribut-valeur est privilégiée à toute autre représentation pour décrire chaque élément d'un contexte. Une bonne description doit comporter des informations sur la formation des images (niveau physique), leur contenu (niveau perceptif) et la scène qu'elles représentent (niveau sémantique) [21, 3].

La description physique renseigne les modes d'acquisition des images (ex : capteur-nature, image-nature), les caractéristiques géométriques et photométriques du numériseurs d'images (ex : signal-type, signal-couleur, image-taille) et les propriétés induites sur les images (ex : image-qualité, bruit-type, bruit-valeur).

La description perceptive détaille les éléments structuraux qui composent les images par description de leurs caractéristiques géométriques, topologiques, spatiales ou de texture (ex : contours-type, régions-taille, histogramme-nature, fond-texture).

La description sémantique définit la notion d'objets réels pour la scène analysée en termes de propriétés

individuelles des objets (ex : objet-taille-min, objet-forme, objet-couleur, objet-texture) et de relations qu'ils entretiennent entre eux (ex : objet-répartition, objet-densité). Lorsqu'il y a plusieurs classes d'objets, on regroupera les attributs-valeurs par classe. Les informations du contexte sont regroupées dans un formulaire de contexte (Fig. 3.).

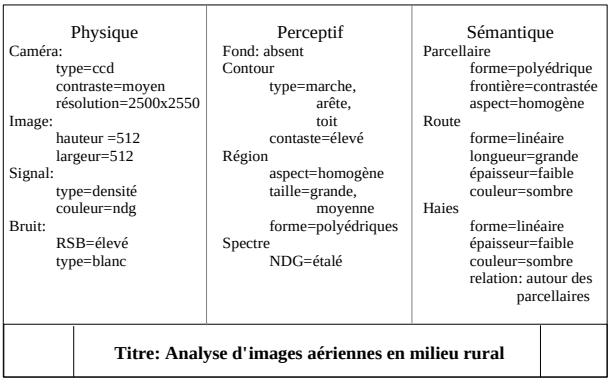

Fig. 3. Le contexte donne un sujet au image en indiquant ce qu'il faut voir dans les images, et ce que cela représente dans le domaine.

1.3.2 Modèle de conception

La conception d'une solution reproduit le schéma d'une méthode d'analyse descendante et consiste à produire des graphes de tâches successifs utilisant des niveaux d'abstraction décroissant. Dans notre méthodologie, une solution conceptuelle d'un problème est envisagée selon les deux axes: horizontal et vertical.

Selon l'axe horizontal, la solution s'exprime par un graphe de tâches. Un graphe décrit la séquence de tâches à accomplir pour construire les images de sortie à partir des images d'entrée. Dans un graphe, toutes les tâches se réfèrent au même niveau d'abstraction, et utilisent le vocabulaire propre du niveau. Une tâche est décrite à l'aide de mots clés par un but à atteindre et des contraintes associées. Les liens de contrôle de ce niveau traduisent donc la relation « est-précédé-de ». La focalisation d'attention porte ici sur les flux de données échangés entre tâches.

Selon l'axe vertical, la solution s'exprime par un arbre de tâches à plusieurs niveaux d'abstraction. Cet arbre traduit un processus de raffinement de tâches d'un niveau en un sous-graphe de tâches plus précises au niveau inférieur. Les liens entre niveau traduisent alors une relation de type « est-réalisé-par ». La focalisation d'attention porte ici sur la décomposition de tâche en sous-tâches.

La représentation d'une solution utilise plusieurs diagrammes (Fig. 4). Un diagramme est consacré à la décomposition d'une tâche en sous-tâches. Une tâche est représentée par une boîte contenant une phrase décrivant l'objet de la tâche et ses contraintes propres. On associe généralement au diagramme, un formulaire de justifications ou les choix faits sont motivés en référence aux éléments du contexte ou des objectifs. Chaque alternative de décomposition fait aussi l'objet d'un diagramme et d'une justification. La structure de graphe est conservée par la numérotation séquentielle des diagrammes, et la structure d'arbre par la numérotation hiérarchique des diagrammes.

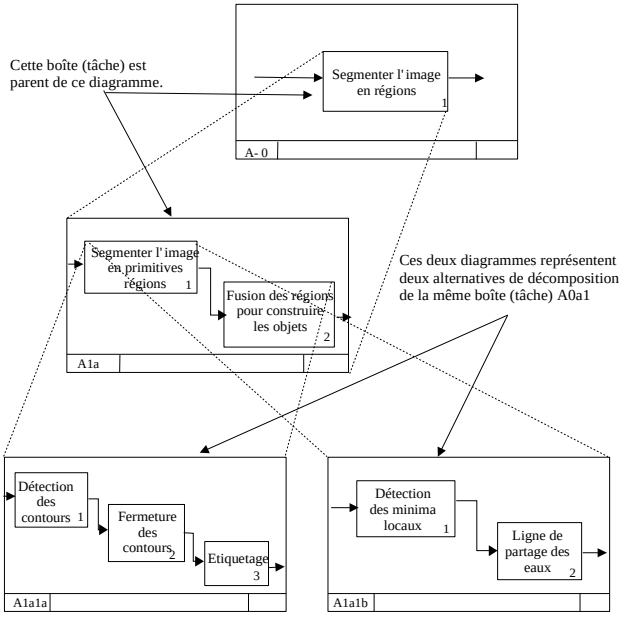

Fig. 4. Une solution conceptuelle est représentée par plusieurs diagrammes organisés séquentiellement sur un même niveau d'abstraction et hiérarchiquement entre niveaux.

1.3.3 Modèle de programmation

Le modèle de programme reprend exactement celui des environnements de programmation visuelle par flots de données. Un programme est un graphe d'opérateurs paramètres. Les liens indiquent le flot de données échangées. Les structures de contrôle sont utilisées pour adapter localement le comportement du graphe à chacune des images présentées.

Sa représentation se fait en utilisant les mêmes formulaires de diagramme et de justifications que précédemment. Chaque opérateur s'écrit comme une tâche; les contraintes sont ici les valeurs de paramètres. Une meilleure représentation serait d'utiliser directement celle proposée par un environnement de programmation visuelle.

1.4 Le cycle d'abstraction

Le cycle d'abstraction est attaché à l'élaboration des solutions conceptuelles. Nous définissons exactement 3 niveaux d'abstraction successifs: intentionnel, fonctionnel, et opérationnel, qui correspondent à 3 niveaux de décision. Une solution s'exprime donc par un arbre de tâche à trois niveaux d'abstraction. A chaque niveau, la solution est complète pour le niveau d'abstraction considéré et est de plus en plus précise au

fur et à mesure que l'on introduit les contraintes d'implémentation.

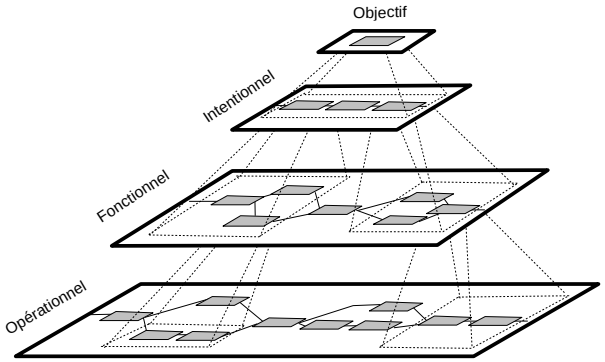

Fig. 5. Le cycle d'abstraction distingue 3 niveaux hiérarchiques. Chaque niveau est l'expression d'une solution complète sous la forme d'un graphe de tâches.

1.4.1 Niveau intentionnel

Le niveau intentionnel est le niveau problème. Il s'intéresse à la détermination d'une stratégie de traitement des images adaptée aux données du problème. Une stratégie définit les étapes de la résolution en terme de sous-problèmes ponctuels à enchaîner. Par exemple, une stratégie classique de segmentation d'images reproduit globalement des séquences du type : 1- prétraitement, 2- segmentation, 3- localisation et 4- resegmentation, avec une orientation descendante, ascendante ou mixte, pouvant être multi-échelle. Une stratégie se décrit par un graphe de tâches intentionnelles, où une tâche est l'expression d'un sous-problème ponctuel avec un vocabulaire lui-même intentionnel. Des tâches typiques de ce niveau sont par exemple : « isoler les objets du fond » ou « former les objets par fusion des régions ».

1.4.2 Niveau fonctionnel

Le niveau fonctionnel est le niveau méthode. Il s'intéresse à la spécification et l'organisation des méthodes de traitement adaptées à la résolution du problème. Une méthode correspond à la mise en oeuvre d'un ensemble de fonctionnalités du traitement d'images, et plusieurs variantes existent généralement pour une même méthode. Par exemple, une méthode de présegmentation en régions peut se réaliser par une classification de pixels seule ou par une détection de contours suivie d'une fermeture de contours suivie d'un étiquetage des composantes connexes.

Une solution de ce niveau se représente par un graphe de tâches fonctionnelles, où une tâche est l'expression d'une fonctionnalité nommée par un substantif, puisqu'une fonctionnalité fait globalement référence à une classe de traitement. Des tâches typiques de ce niveau sont par exemple: « détection de contours », « classification des pixels », « fermeture de contours » ou « fusion de régions ».

1.4.3 Niveau opérationnel

Le niveau opérationnel est le niveau algorithme. Il s'intéresse à la mise en oeuvre technique de la solution. Mais il ne concerne que le choix et l'enchaînement d'algorithmes de traitement d'images sans qu'il soit encore question d'une quelconque bibliothèque d'opérateurs. Ceci fait que ce niveau reste encore dégagé des problèmes liés à la paramétrisation et au contrôle d'exécution des opérateurs. Seuls les directives de la future paramétrisation et le choix des modes d'exécution sont étudiés et constituent les contraintes des tâches de ce niveau.

Une solution de ce niveau se représente par un graphe de tâches opérationnelles, où une tâche est la désignation d'un algorithme. Des tâches typiques de ce niveau sont par exemple: « classification des pixels basée sur une maximisation de la variance inter-classes » ou « fermeture de contours basée sur la poursuite du gradient ».

1.5 Le cycle de vie

Le cycle de vie est attaché aux phases de développement de l'application. Nous définissons un cycle de vie [2] en spirale qui est de plus "colorée" en 4 phases (Fig. 6):
I- Définition, II- Réalisation, III- Validation et IV- Maintenance. Un tour de spire enchaîne les 4 activités du développement et plusieurs tours sont généralement utilisés pour parfaire une phase.

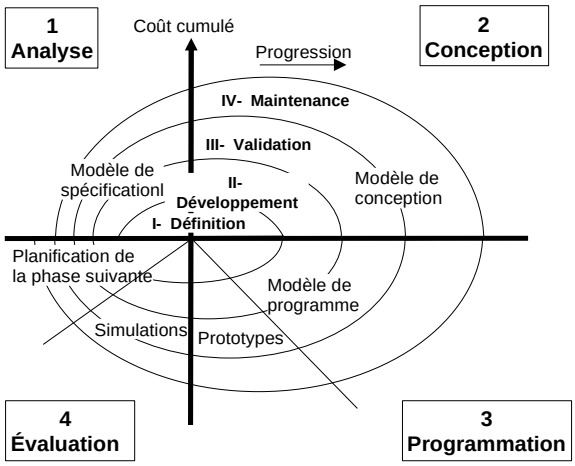

Fig. 6. Le cycle de vie définit 4 groupes spires correspondant à 4 phases de développement. Une phase regroupe plusieurs tours de spires (un groupe de spire définit une couleur). Chaque spire enchaîne les 4 activités de développement.

Le Error: Reference source not found résume la distribution typique des efforts dans chaque activité au cours des phases.

	I Définition	II Développement	III Validation	IV Maintenance
Analyse	*****	***	*	*
Conception		*****	***	**
Program-mation	*	*	*****	**
Évaluation	**	*	****	****

Tableau 2. Chaque phase du cycle de vie (de I à IV) fait appel plus ou moins aux quatre activités du développement. *Le caractère '*' traduit l'importance de l'activité dans la phase.*

Ceci fait du développement un processus itératif et incrémental basé sur l'évolution de prototypes mesurables. Chaque tour de spire produit un ou plusieurs prototypes dont on analyse les résultats pour préparer le tour de spire suivant.

1.5.1 Définition

La première phase a pour but de définir les termes du problème à résoudre, et en aucun cas à construire l'application proprement dite. C'est principalement le contexte de l'application qui est le sujet de l'étude. Le résultat recherché est une description consistante du contexte de l'application. Cette étape ne peut être faite qu'en partenariat avec le client.

La méthode consiste à utiliser autant de cycles enchaînant les quatre activités du développement que nécessaire pour avoir une vision réaliste du problème. Ainsi, les premiers cycles sont concernés par la description physique et perceptive de l'application. Chacun des cycles suivants est concerné par la description d'un objet du contexte: Partant de la description d'un objet donné par le client, le développeur applique plusieurs schémas de segmentation généraux pour vérifier la teneur de sa description, puis affiner ou réviser cette description.

1.5.2 Réalisation

Cette second phase a pour but la programmation de l'application dans sa grandeur nature. Cette fois, c'est la spécification précise de l'objectif qui doit être recherchée. Plusieurs versions concurrentes de l'application doivent être exploitées en parallèle, chacune d'elle étant relative à un point de vue sur le contexte de l'application. Typiquement, un point de vue se réfère à une sélection de caractéristiques.

La méthode consiste à produire différents prototypes d'application qui répondent plus ou moins aux besoins du client. On utilise autant de cycles que nécessaires pour exploiter tous les points de vue valides sur le contexte de l'application; le développement d'un prototype nécessite plusieurs cycles par lui-même. A ce niveau, la réutilisabilité doit être maximale. Les éléments de solution sont d'abord à rechercher dans les archives.

Dans la mesure du possible, on ne gardera pas plus de 3 solutions concurrentes en sortie de cette phase. De même, on limitera le nombre de variantes de décomposition d'une même tâche à l'intérieur d'une même solution.

1.5.3 Validation

Cette phase a pour but de produire la version finale de l'application, dont les capacités de robustesse, de fiabilité et de consistance sont optimisées et toutes les exigences clients sont certifiées.

La méthode consiste à déterminer plusieurs mesures quantitatives qui définissent des critères objectifs pour une sélection appropriée du meilleur programme parmi les candidats issus de la phase de réalisation. On pourra s'appuyer si possible sur la spécification des post-traitements pour déterminer des mesures quantitatives. Dans le pire des cas, ces mesures sont faites par des mesures de différence avec des images de référence dessinées à la main par le client.

1.5.4 Maintenance

Cette dernière phase concerne le déploiement et le suivi de la solution logicielle sur le site jusqu'à ce qu'elle devienne caduque ou trop difficile à maintenir. Le déploiement assure la cohérence avec les systèmes amont et aval du site.

La méthode de déploiement consiste à ajuster les entrées et les sorties du programme pour assurer son intégration sur le système cible. Le suivi concerne la correction des corrections avérées et l'adaptation de l'application à de nouveaux besoins du client. La prise en compte des exigences du clients se fait en utilisant le même processus que les phases de développement et de validation.

1.6 Le cycle Auteur/ Lecteur

Inspiré de la méthodologie SADT [12], ce cycle a pour objectifs de garantir une certaine validité des solutions proposées et de conduire vers une généricité de ces solutions dans l'espoir d'en favoriser la réutilisabilité. Ce cycle est important dans la perspective de maintenance, parce que les lecteurs sont considérés comme une autre source de validation, dans le sens où il réutilisent les modélisations dans un autre contexte que celui pour lequel la modélisation a été écrite. Il repose sur le partage de connaissances expertes et impose une assez forte uniformisation de la représentation de ces connaissances.

1.6.1 Validation

L'auteur produit des programmes et de la documentation associée. On demande alors au lecteur, qui est forcément différent des développeurs du projet, de critiquer les programmes à partir de sa documentation et de l'analyse des images intermédiaires produites par ces programmes.

Étant donnée l'étendue des compétences requises en traitement d'images, les avis d'experts d'origines variées est un atout considérable.

1.6.2 Généralisation

La généralisation des solutions est un préalable à la réutilisabilité. Le lecteur d'applications devient un jour

développeur et profite alors de l'existence de schémas construits par d'autres dans ses propres développements. Ceci doit permettre de dégager tout ce qui relève du contrôle, et tout ce qui relève des méthodes.

1.6.3 Archivage des applications

La validation et la réutilisation suppose un archivage très précis des productions. Ces archives seront considérées comme certifiées et mentionneront l'identification de ses auteurs et de ses lecteurs. Seule, la dernière version des formulaires modélisant la solution finale sera archivée sous la forme d'un Kit de formulaires. Cette version est la seule qui soit réellement validée, à la fois par le client et par les experts. Les autres solutions alternatives n'ont pas fait l'objet d'une validation aussi poussée, elles ne sont pas donc certifiées.

2 Bilan de la méthodologie

Nous voulons démontrer ici la validité de notre méthodologie pour le développement d'applications même complexes, moyennant quelques limites et prouver qu'elle apporte un gain réel pour compenser la difficulté de développement, moyennant quelques inconvénients.

2.1 Quelle validité?

2.1.1 La validité des modèles

La validité de nos modèles est liée à celle du pilotage de bibliothèque d'opérateurs pour le modèle de conception et celle de la programmation visuelle par flots de données pour le modèle de programme.

Le modèle d'arbre de tâches pour le pilotage de programmes a largement prouvé sa validité pour la modélisation de processus cognitifs. On trouvera dans [14] des exemples de développement d'applications complexes en traitement d'images à l'aide du pilotage d'une bibliothèque d'opérateurs. De plus, l'utilisation d'une hiérarchie de tâches fixe en 3 niveaux trouve sa justification d'une part dans la méthodologie Merise [16] pour ce qui est du génie logiciel (les 3 niveaux sont: Conceptuel, Opérationnel, Physique), et d'autre part dans la théorie computationnelle de la vision de Marr [12] pour ce qui est du traitement d'images (les 3 niveaux sont : Computationnel, Algorithmique et Implantation) . Toutefois, un arbre de tâches ne modélise pas le raisonnement mis en oeuvre pour le produire, mais simplement le résultat. Le rôle des formulaires de justifications associés aux diagrammes de tâches est alors de capitaliser une partie de ce raisonnement [18].

Le modèle de programme proposé trouve sa validité dans sa large utilisation au travers des environnements de programmation visuelle par flots de donnés. Il faut pour cela admettre le postulat que l'on peut représenter n'importe quelle application de traitement d'images sous la forme d'un graphe de programmes individualisés. C'est-à-dire qu'un processus de traitement aussi complexe qu'il soit peut toujours se discrétiser en une séquence de traitements ponctuels, dont les liens peuvent se réduire à des fichiers ou à des valeurs numériques. On trouvera dans un exemple de l'utilisation de tels environnements pour le développement d'applications grandeur nature.

2.1.2 La validité des méthodes

Notre méthodologie impose une démarche d'analyse basée sur une vision itérative et incrémentale de la construction de résultats. Elle se justifie par le fait, qu'une construction itérative et incrémentale est préférable à toute autre lorsque la solution à construire est difficile, ce qui est généralement le cas en traitement d'images. La décomposition en activités permet d'introduire le concept de contrôle dans le développement et de mesurer l'avancée du projet.

Nous arguons de plus que notre cycle de vie est parfaitement adapté au domaine du traitement d'images, où la première phase est liée à l'apprentissage des concepts métiers indépendamment du logiciel à produire, et les phases suivantes exploitent les résultats de cette première phase pour se concentrer sur la production puis l'optimisation du logiciel.

2.2 Quelles limites?

Il existe deux limites inhérentes à notre méthodologie: la première porte sur l'unicité de la nature des solutions produites et la seconde sur le coût en temps d'exécution des productions.

L'utilisation d'une bibliothèque d'opérateurs génériques impose de concevoir des solutions qui procèdent par affinage de résultats. Il ne s'agit pas ici d'écrire un programme ad-hoc complètement déterminé par les connaissances a priori sur le modèle de résultat à produire, mais de combiner des opérateurs prédéfinis et généraux, dont le manque d'efficacité individuelle ne peut être compensé que par leur intégration dans un processus d'affinage progressif. Les chaînes de traitement sont donc naturellement plus longues qu'une solution toute entière construite. La première conséquence, c'est que notre méthodologie ne peut pas englober toutes les façons existantes de construire des applications, notamment tout ce qui est basé sur la construction d'application à partir de modèles de la solution à rechercher. La seconde conséquence est que les applications produites ne sont généralement pas optimales, mais plus sûrement acceptables.

Les programmes générés sont très gourmands en temps d'exécution, premièrement parce que les chaînes relèvent d'un processus d'affinage qui réclame l'enchaînement de plusieurs opérateurs et que deuxièmement les opérateurs s'échangent des fichiers entiers d'images. Il est donc parfois nécessaire de réécrire l'application en intégrant le code des opérateurs dans un seul programme, et en optimisant leur enchaînement. Ce faisant, on réintroduit dans l'application de nouvelles erreurs et de nouvelles

imperfections, ce qui nécessite une nouvelle phase de validation. De plus, on perd alors la cohérence entre la modélisation de bas niveau et le code, ce qui est un handicap pour la phase de maintenance.

Ces limitations ne sont pas rédhibitoires à l'utilisation de la méthodologie. En effet, elle peut toujours être utilisée avec profit dans la phase d'étude l'application allant de la phase définition jusqu'à la phase de réalisation. A partir de là, on adoptera un développement classique d'une programme ad-hoc. Le gain de l'utilisation de la méthodologie même pour ces phases initiales est à forte valeur ajoutée.

2.3 Quels bénéfices?

L'intérêt d'adopter une méthodologie est quadruple: sécuriser la programmation, limiter la charge de programmation, aider a priori l'expert et formaliser a posteriori les solutions. Elle répond ainsi aux besoins exprimés dans l'introduction.

2.3.1 Sécuriser la programmation

C'est indéniablement le gain le plus perceptible. Il n'y a plus ici réellement d'activité de programmation, ce qui accroît par conséquent la robustesse des logiciels. Les erreurs de programmation sont repoussées au niveau des opérateurs. Mais là encore, la méthodologie apporte une solution parce qu'elle favorise la réutilisabilité des opérateurs. Ceci permet de les éprouver dans différents contextes. Les erreurs détectés sont alors profitables à tous.

2.3.2 Aider a priori le développeur

Cette méthodologie est une aide au développement d'applications parce qu'elle est un guide formalisateur, complet et rigoureux. Elle répond aux difficultés présentées en introduction, parce que:

- Elle permet le couplage spécification du problème et production de solutions mesurables [11]. La construction de prototypes permet de construire rapidement des résultats visualisables. On évite ainsi la rupture du dialogue entre le développeur et le client.
- Elle structure le développement en une séquence d'étapes parfaitement identifiées définissant ainsi une démarche reproductible pour obtenir des résultats relativement fiables. Pour chacune d'elles, elle précise les informations à renseigner et la façon de les représenter.
- Elle oblige l'expert du traitement d'images à justifier et à formaliser ses choix pour lever les ambiguïtés et préciser les implicites.
- Elle offre un support de communication entre le développeur et les experts du traitement d'images, pour critiquer et réviser les solutions. De ce fait, elle favorise la réutilisation de schémas précédemment définis.

2.3.3 Formaliser a posteriori

L'autre apport de la méthodologie est sa capacité à formaliser les applications sous une forme complète et explicite. Ainsi,

- elle offre un cadre pour capitaliser le savoir-faire ;
- elle fige les représentations et permet de construire des dictionnaires et des ontologies du domaine du traitement d'images. Elle contribue en cela à une formalisation des connaissances du domaine ;
- elle permet enfin d'unifier les représentations des connaissances dans le but de les critiquer pour les affiner ;

2.4 Quels inconvénients

En contre-partie, l'utilisation de la méthodologie impose l'acceptation de deux coûts: un coût initial d'apprentissage de la méthodologie et un coût constant lié au respect d'une démarche rigoureuse et consommatrice de documentation.

Avant d'utiliser la méthodologie, il faut apprendre à repenser sa démarche de développement. Il faut se persuader de l'intérêt à long terme de l'utilisation de cette méthodologie.

Le développement d'une application nécessite de motiver ses choix et oblige à construire des schémas les plus génériques possibles. L'expert du traitement d'images est là pour rappeler cette contraintes parce que les schémas produits doivent être partageables par tous.

3 Conclusion

La méthodologie présentée est le résultat de travaux de recherches portant sur la réalisation d'un système à base de connaissances pour la génération automatique d'applications de traitement d'images [4]. Le système à réaliser doit avoir la capacité de construire seul des applications de traitement d'images en réponse à un problème formulé par un utilisateur. Le goulot d'étranglement inhérent à ces systèmes reste l'acquisition des connaissances, qui ne peut s'envisager qu'au travers de la réalisation d'applications concrètes. Dans ce projet, nous portons un intérêt très fort à l'explicitation des connaissances en plus de son opérationnalisation. En effet, notre but final n'est pas tant le système que la modélisation de la connaissance experte sous une forme explicite. Le système n'est là que pour valider cette explicitation.

Par rapport aux travaux de S. Moisan [14] et P. Dalle [6], notre méthodologie met l'accent sur la modélisation de l'expertise de résolution de problèmes en traitement d'images (en particulier les stratégies et les méthodes), plus que sur la modélisation des connaissances sur les programmes à piloter pour résoudre ces problèmes.

Ainsi, la mise en place de la méthodologie à favoriser la production d'applications et l'échange de schémas de traitement. A l'image du livre des connaissances [10], cette expertise est disponible sous une forme hypertexte, sous une forme plus ou moins explicite selon le degré de respect de la méthodologie. Cette archive constitue pour nous le corpus à partir duquel nous allons éliciter l'expertise de traitement d'images.

Bibliographie

[1] E. Baroth, C. Hartsough, Experience Report: Visual Programming in the Real World, in *Visual Object-Oriented Programming: Concepts and Environments, Burnett, Goldberg & Lewis (eds.)*, Manning, Prentice Hall, p. 21-42, 1994.

[2] B.W. Boehm, A spiral model of software development and enhancement, *IEEE Computer*, Vol. 21, No. 5, pp. 61-72, May 1988.

[3] R. Clouard, A. Elmoataz, M. Revenu, Une modélisation explicite et opérationnelle de la connaissance de traitement d'image, *11e congrès RFIA*, Clermont-Ferrand, Vol. II, pp. 65-74, Jan. 1998.

[4] R. Clouard, A. Elmoataz, C. Porquet, M. Revenu, Borg : A knowledge-based system for automatic generation of image processing programs, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 21, No. 2, pp. 128-144, Feb. 1999.

[5] D. Crevier, R. Lepage, Knowledge-base image understanding systems: a survey, *Computer Vision and Image Understanding*, Vol. 67, n°2, p. 161-18, 1997.

[6] P. Dalle, P. Dejean, Planification en Traitement d'Image : Approche basée sur les données, , *12ème Congrès RFIA*, Vol II, pp. 75-84, Clermont-Ferrand (France)1998.

[7] B. Deen "A graphical user interface for science data processing", *NASA Science information New Letter*, Vol. II, no. 39, pp. 22-27, 1996.

[8] Draft Federal Information Processing Standards Publication 183, 21 décembre 1993, *Announcing the standard for Integration Definition For Function Modeling (IDEF0)*. The national Institute of standards and Technology, 119 pages, 1993.

[9] J.L. Ermine, *Génie logiciel et génie cognitif pour les systèmes à base de connaissances*, Technique et Documentation, Lavoisier, 1993.

[10] J.L. Ermine, *Les systèmes de connaissances*, Hermès, Paris, 1996.

[11] C. Hartsough, E. Baroth, Visual Programming improve communication among the customer, developer and computer, *National Instruments User Symposium*, Austin, Texas, p. 26-28, 1995.

[12] D.A. Marca & C.L. Mc Govan, *SADT: Structured Analysis Design Technique*, Mc Graw Hill, 1987.

[13] D. Marr, *Vision: A computational investigation into the human representation and processing of visual information*, Freeman and co, San Francisco, Calif., 1982.

[14] S. Moisan, J.L. Ermine, Gestion opérationnelle des connaissances sur les codes", *Journées francophones Ingénierie des Connaissances IC'2000*, Toulouse, France, pp. 131-141, Mai 2000.

[15] S. Moisan, D. Ziébelin, Résolution de problèmes en pilotage de programmes, *12ème Congrès RFIA*, Vol III, pp. 387-396, 2000.

[16] Quang & C. Chartier-Kastler, *Merise in practice*, Macmillian, 1991.

[17] J. Rasure & S. Kubica, The Khoros Application Development Environment, *Experimental Environments for Computer Vision and Image Processing*, eds H.I Christensen and J.L Crowley, World Scientific, Singapore, pp. 1-32, 1994.

[18] D.T. Ross, Structured Analysis (SA): A language for communicating ideas, *IEEE transaction on Software Engineering*, Vol. SE-3, No. 1, Jan 1977.

[19] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, Addison-Wesley Object Technology Series, Dec. 1998.

[20] M. Thonnat, S. Moisan. What can Program Supervision do for Program Reuse, *IEE Procedings- Software*, Vol. 147, No. 5, pp. 179-185, Oct. 2000.

[21] J. van den Elst, *Modélisation des connaissances pour le pilotage de programmes de Traitement d'image*, Thèse de l'Université de Nice, 1996, Sophia-Antipolis, Fr.

[22] J. Vroom, *AVS/Express: A new Visual Programming Paradigm*, Rapport Technique, Advanced Visual System, 1997.

[23] P. Zamperoni, Plus ça va, moins ça va, *Pattern Recognition Letters*, n°17, 1996, p. 671-677.