Dix-septième colloque GRETSI, Vannes, 13-17 septembre 1999

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

Régis CLOUARD[1], Abderrahim ELMOATAZ[1, 2], Marinette REVENU[1]

[1]GREYC - Image, UPRESA CNRS 6072
6 boulevard du Maréchal Juin, 14050 Caen Cedex, France

[2]LUSAC, EA 2607
Site universitaire de Cherbourg, BP 78, 50130 Octeville, France

{Regis.Clouard, Abder.Elmoataz , Marinette.Revenu}@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 des applications. Cette méthodologie met en avant des capacités d'aide, de réutilisabilité d'expériences, d'uniformisation des représentations et de communication entre les différents personnes impliquées, par la définition de modèles structurés, de représentations graphiques et de règles de mise en œuvre de ces modèles à chaque étape du développement. Elle se fonde essentiellement sur le paradigme du pilotage d'une bibliothèque de tâches, avec lequel la conception d'une solution est vue comme un processus d'agglomération de tâches ponctuelles et indépendantes. Le formalisme retenu distingue le cycle d'abstraction, qui considère trois niveaux pour la modélisation d'une solution conceptuelle, plus un niveau pour le programme proprement dit, et le cycle de vie qui préconise quatre phases successives pour la gestion complète de l'application.

Abstract – A methodology for the development of image processing applications that is a complete and rigorous guide for the management of the whole life cycle of applications is proposed. This methodology points out assistance, reusing and unifying capabilities for representations and communications between the different intervening parties, by defining structured models, graphical representations and implementing rules at each stage of the development process. It is essentially based on the paradigm of supervision of library of task, along which the design of a solution is viewed as an agglomeration process of punctual and independent tasks. The formalism distinguishes the abstraction cycle that considers three levels of abstraction for conceptual solution modeling, plus one for the program itself, and the life cycle that distinguishes four successive phases for the application management.

1. Introduction

Le développement d'une application de traitement d'image en tant qu'activité de production de logiciels, présente un certain nombre de particularités qui rendent peu adaptées les méthodologies traditionnelles du génie logiciel type Merise [1], SADT [2] ou UML [3]. La notion d'application se définit en traitement d'image comme un programme spécialisé dans la réalisation d'un objectif donné de transformation d'images sans interprétation du contenu, et qui peut s'exécuter sur toutes images d'une classe donnée. Parmi les particularités rencontrées dans le développement de telles applications, on peut citer :

- L'expression de l'objectif à atteindre relève d'une négociation entre un expert du domaine d'application qui possède le problème à résoudre de façon non explicite et l'expert du traitement d'image qui doit le formuler de façon explicite en choisissant les bonnes tâches à accomplir et en caractérisant les bonnes informations sur les images et le contexte. En pratique, la définition complète et précise d'un problème est indissociable de la résolution qui en est faite ;
- Les images, qui sont les données premières du problème, sont à la fois trop complexes, incomplètes, entachées d'erreurs et sous-contraintes pour être le support de la modélisation des connaissances et du raisonnement ;
- Le savoir-faire accumulé, pourtant considérable, reste difficilement réutilisable parce qu'il est non structuré en raison de l'absence de cadre formalisateur unique et de la multiplicité des domaines de référence théorique dont il relève (TS, optique, IA...) ;
- L'écriture du programme proprement dit se heurte à la mise en œuvre d'un grand nombre de calculs sur un grand nombre de données ;
- L'évaluation ne peut s'appuyer ni sur l'examen visuel, qui est insuffisant, ni sur la vérification numérique des valeurs de résultats, qui est fastidieuse.

Dans ce papier, nous nous intéressons à la définition d'une méthodologie de développement d'applications de traitement d'image qui tente d'apporter une réponse aussi satisfaisante que possible à la prise en compte de chacune des particularités précédentes, en veillant à ce que ces réponses s'organisent entre elles en un schéma méthodologique unique et cohérent. Le but est de réduire la complexité de la tâche de développement en proposant de rationaliser le processus de développement et de favoriser la réutilisabilité de solutions préalablement testées.

2. La méthodologie proposée

2.1.Le paradigme de base

La méthodologie que nous proposons prend sa source dans le paradigme du **pilotage d'une bibliothèque**, mis en lumière par les environnements de programmation visuelle type Khoros [4] ou AVS/Express [5]. Ces environnements proposent de coder une application par un graphe d'opérateurs paramétrés. Ceci suppose l'existence d'une bibliothèque prédéfinie dans laquelle les opérateurs sont des commandes exécutables du système d'exploitation. L'exécution du graphe résulte de l'exécution de chacun des opérateurs dans l'ordre spécifié, avec les valeurs de paramètres spécifiées. L'introduction explicite de structures de contrôle classiques (itératives, répétitives ou alternatives) donnent aux graphes la possibilité d'ajuster dynamiquement leur comportement sur chacune des images présentées en entrée.

La création d'un graphe selon ce paradigme revient à :

1) Sélectionner des opérateurs dans la bibliothèque ;
2) Paramétrer ces opérateurs avec des valeurs fixes ou des valeurs calculées par d'autres opérateurs ;
3) Enchaîner ces opérateurs selon le flux de données et les structures de contrôle.

Nous proposons de généraliser ce paradigme pour définir une démarche de développement descendante, modulaire, hiérarchique et structurée. La généralisation du paradigme conduit à considérer la conception d'une solution comme la construction d'un graphe de tâches et plusieurs niveaux d'abstraction sont envisagés. Une tâche est la description précise d'un but à atteindre, d'une méthode à employer ou d'une technique à utiliser selon son niveau d'abstraction. Le dernier graphe de tâches produit devient alors le support de la construction du graphe d'opérateurs final.

La démarche de développement proposée et qui utilise le paradigme du pilotage organise les activités techniques du développement de logiciels (ici essentiellement l'analyse des besoins, la conception, la programmation et l'évaluation) autour d'un cycle d'abstraction en trois niveaux (intentionnel, fonctionnel et opérationnel) et d'un cycle de vie en quatre phases (initialisation, réalisation, validation et maintenance).

2.2.Les activités techniques

Les quatre activités identifiables dans un processus de développement profitent du paradigme du pilotage d'une bibliothèque de tâches ou de programmes:

1. L'**analyse des besoins** est vue ici comme la définition de la tâche initiale. Une tâche distingue les objectifs et le contexte [5]. Les objectifs sont choisis parmi ceux du traitement d'image (segmenter, localiser, détecter...) et leurs contraintes parmi quatre catégories: critères à optimiser, niveaux de détail, erreurs acceptables et restrictions. Le contexte se décrit en renseignant des descripteurs d'image à trois niveaux: physique (capteur, signal, bruit...), perceptif (gradient, contour, région...) et sémantique (objets, relation, groupe d'objets...). Nous développons actuellement un outil utilisant les réseaux de communication qui favorise le dialogue entre les deux types d'experts ;
2. La **conception** d'une solution conceptuelle reproduit le schéma d'une analyse descendante et consiste à produire des graphes de tâches successifs utilisant des niveaux d'abstraction décroissants. Ce sont à chaque fois des solutions complètes de la tâche initiale, mais avec une formulation plus précise. Chaque solution est modélisée par un graphe de tâches utilisant les mêmes conventions que les graphes d'opérateurs. Les tâches exprimées en terme de [objectifs + contraintes] sont décrites avec un vocabulaire propre au niveau d'abstraction d'appartenance et sont reliées entre elles par le flux de données échangées et les structures de contrôle. Le modèle de conception se ramène à celui du pilotage d'une bibliothèque (en fait un dictionnaire de tâches constitué au fur et à mesure des réalisations) et consiste à:
    1) Sélectionner des tâches dans le dictionnaire ;
    2) Déterminer leurs contraintes ;
    3) Enchaîner ces tâches entre elles selon le flux de données et les structures de contrôle.
3. La **programmation** se réduit à la composition du graphe d'opérateurs en s'appuyant sur le dernier graphe de tâches produit. On se retrouve ici au niveau des environnements classiques de programmation visuelle. Toutefois, nous proposons une description plus fine de la structure des graphes d'opérateurs et de la notion d'opérateur. Ainsi, les graphes intègrent les structures de contrôle classiques (alternatives, répétitives et itérative), mais aussi des structures plus originales du type optimisation ou satisfaction. Nous distinguons de façon explicite cinq types de flux de données échangeables entre opérateurs : 1) les données, 2) les variables de contrôle, 3) les images de référence utilisées pour le contrôle 4) les valeurs de paramètres et 5) les masques de données. Les opérateurs considérés ici sont des opérateurs polymorphes, masquables et atomiques. Nous développons actuellement un outil de programmation visuelle propre qui intègre ces principes.
4. L'**évaluation** profite des modélisations précédentes. L'utilisation de graphes discrets de tâches ou d'opérateurs permet une bonne localisation des sources d'erreurs. L'utilisation de graphes à plusieurs niveaux d'abstraction permet une vérification hiérarchique des résultats intermédiaires. De plus, les sources d'erreurs restent circonscrites à la construction des graphes pour peu que la bibliothèque soit suffisamment éprouvée. Les corrections se réduisent à la suppression, l'ajout ou l'échange de tâches ou d'opérateurs.

Pour chaque activité, nous avons défini des formulaires types de représentation graphique : pour la définition de la tâche initiale, pour la représentation des différents graphes de tâches et pour la représentation des graphes d'opérateurs. Ces formulaires formalisent a posteriori le problème a résoudre et les solutions correspondantes, mais aident aussi a priori à la formulation du problème et à la conception d'une solution parce qu'ils listent en détail les rubriques à renseigner.

2.3.Le cycle d'abstraction

Le cycle d'abstraction est attaché à la modélisation des solutions conceptuelles. La méthodologie utilise trois formalismes hiérarchiques successifs pour modéliser une application (FIG 1). Ces trois formalismes s'identifient fortement aux trois niveaux d'abstraction définis par Marr [6] pour sa théorie computationelle de la vision :

1. Le niveau **intentionnel** répond aux questions « quoi faire? » et « pourquoi? » par la mise en place d'une stratégie de traitement des images adaptée. Une stratégie définit les étapes de la résolution en terme de sous-problèmes ponctuels à enchaîner. Par exemple une segmentation peut s'écrire comme une instanciation des étapes 1) prétraitement, 2) pré-segmentation, 3) localisation et 4) resegmentation;
2. Le niveau **fonctionnel** répond à la question « comment? » par la recherche de solutions pour chacun des sous-problèmes posés, en termes de méthodes de traitement des images. Une méthode de traitement s'écrit comme une chaîne de fonctionnalités (où une fonctionnalité fait globalement référence à une classe d'opérateurs). Par exemple, une pré-segmentation peut se composer de 1) classification de pixels puis 2) marquage des régions obtenues ;
3. Le niveau **opérationnel** répond à la question « avec quoi? » en construisant une solution technique en terme d'une chaîne d'algorithmes. Par exemple, une classification de pixels en deux classes peut s'écrire 1) lissage moyenneur puis 2) binarisation selon la maximisation de la variance interclasse.

Compte-tenu de ces trois niveaux, l'activité de conception d'une solution consiste alors à produire successivement les trois graphes de tâches correspondants. L'utilisation d'une hiérarchisation fixe pour la modélisation des solutions en exactement trois niveaux permet une uniformisation des solutions et facilite alors la comparaison et la critique. Elle est surtout une garantie pour la réutilisabillité.

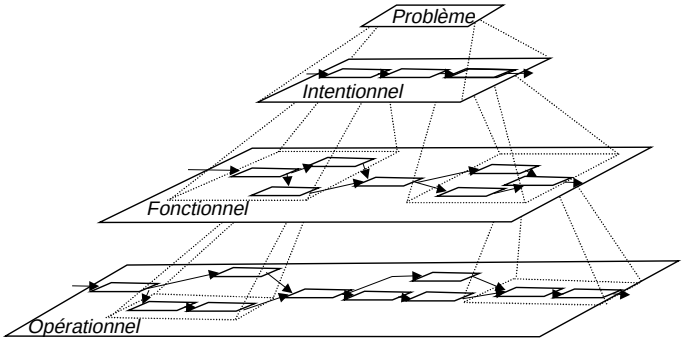

FIG 1. Une solution conceptuelle est représentée verticalement par un arbre de tâches à 3 niveaux d'abstraction et horizontalement par 3 graphes de tâches.

2.4.Le cycle de vie

Le cycle de vie est attaché aux phases de développement de l'application et organise les quatre activités précédentes dans un modèle en spirale [8] que nous définissons en plus de « colorée » (FIG 2). Un cycle en spirale définit un processus de développement itératif et incrémental. Une spire implique plus ou moins l'utilisation des quatre activités et se termine par la production d'une version de la formulation du problème et d'une version pour tous les niveaux de la solution conceptuelle et pour le programme lui-même. Ces résultats sont tout de suite réinvestis dans la spire suivante après leur évaluation. Notre spirale est colorée parce que nous distinguons quatre zones de spires (les phases) séquentielles distinctes sans que l'on sache à l'avance le nombre de spires que contient chacune des phases. Les quatre phases que nous avons identifiées sont :

1. L'**initialisation** consiste à écrire des prototypes pour segmenter les différents objets du domaine sans pour cela chercher à produire l'application finale. Le but est de s'assurer que les informations définies dans le contexte caractérisent parfaitement les objets du domaine ;
2. La **réalisation** s'attache cette fois au développement réel de l'application pour répondre aux objectifs spécifiques du problème. Elle utilise alors les résultats de la phase d'initialisation pour définir précisément le problème à résoudre et générer le bon programme ;
3. La **validation** teste l'application en grandeur nature en confrontant le programme généré à toute une série d'images caractéristiques de la classe. L'application est ainsi peu à peu fiabilisée ;
4. La **maintenance** assure le suivi de l'application pour toutes les évolutions liées à la détection de défauts, à l'incomplétude ou aux ajouts demandés.

L'évaluation des différents programmes écrits est réalisée avec le concours de l'expert du domaine d'application qui devient un partenaire à part entière du développement.

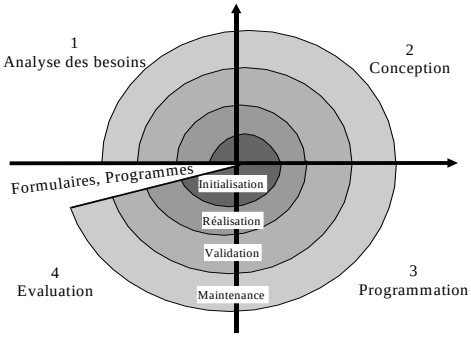

FIG 2. Le cycle de vie organise les 4 activités techniques du développement en une spirale de 4 "couleurs".

3. Exemple d'une application

Cette section ne présente que le résultat d'une modélisation des connaissances pour le cas d'une application d'imagerie aérienne (plus de détails sur l'application et ses résultats peuvent être trouvés dans [9]). Il n'est pas du tout rendu compte ici du cycle de vie (i.e., les prototypes produits).

Le problème de l'application peut se formuler comme FIG 3. La solution peut s'écrire comme FIG 4, et le programme résultant comme FIG 5.

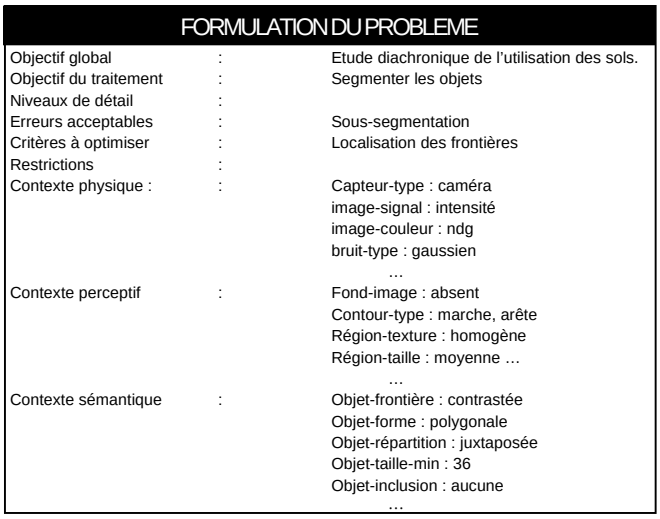

FIG 3. Les termes du problème.

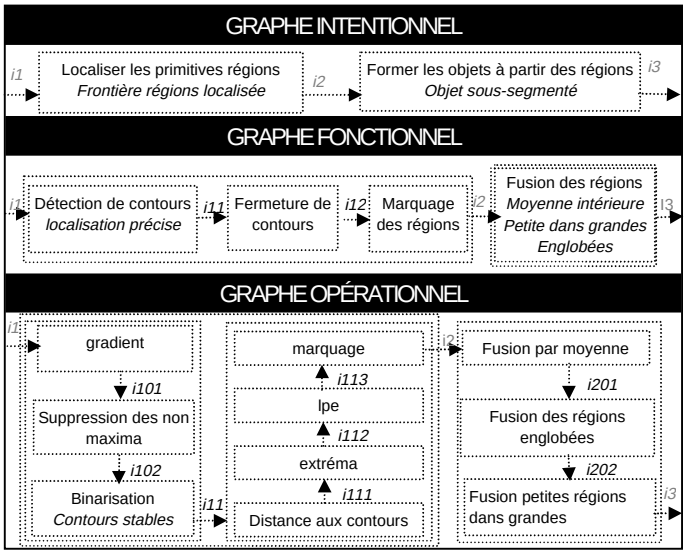

FIG 4. Les trois niveaux hiérarchiques de la solution.

4. Conclusion

La méthodologie présentée prend en compte les particularités présentées en introduction. Ainsi, elle structure le développement en une séquence d'étapes parfaitement identifiées constituant une démarche reproductible pour obtenir des résultats fiables. Elle définit un formalisme pour capitaliser le savoir-faire sous forme de graphes de tâches en exactement trois niveaux d'abstraction et des modèles pour la formulation des problèmes. Elle rend possible la production de nombreux prototypes mesurables, et elle profite des qualités des environnements de programmation visuelle pour réduire la charge cognitive des développeurs.

En contre-partie, elle engendre des coûts pour le développeur qui sont liés au respect d'une démarche rigoureuse, à l'obligation de motiver tous ses choix et à la production de programmes relativement consommateurs de ressources et de temps (les chaînes relèvent de processus d'affinage progressifs et les opérateurs s'échangent des fichiers entiers) même si des optimisations a posteriori sont possibles.

Références

[1] P.T. Quang et C. Chartier-Kastler, *Merise in practice*, Macmillian, 1991.

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

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

[4] J. Rasure et 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.

[5] R. Clouard, A. Elmoataz, C. Porquet et M. Revenu, "Borg: A knowledge-based system for automatic generation of image processing programs", *IEEE Trans. PAMI*, Vol. 21, No. 2, pp. 128-144, Feb. 1999.

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

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

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

[9] S. Coudé, *Analyse d'images aériennes, Mise en œuvre de l'atelier logiciel d'intégration de connaissances en traitement et en interprétation d'images*, Rapport de DEA, GREYC, Caen, 1996.

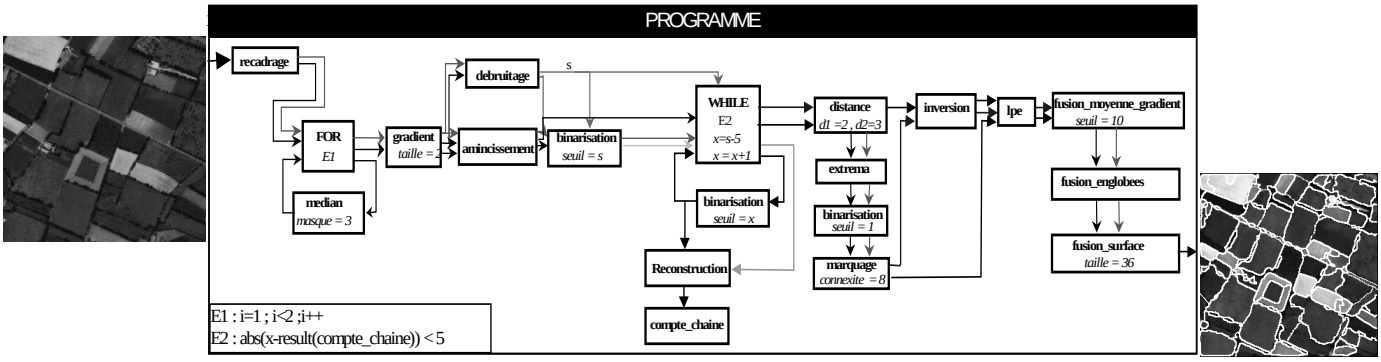

FIG 5.Le graphe d'opérateurs Pandore résultant.