Agents négociateurs appliqués à la télédétection

par

Martin Labrie

mémoire présenté à la Faculté des sciences en vue de l'obtention du grade de maître ès sciences (M. Sc.)

FACULTE DES SCIENCES
UNIVERSITE DE SHERBROOKE

Sherbrooke, Québec, Canada, avril 2003

Your file Votre référence
ISBN: 0-612-86661-0
Our file Notre référence
ISBN: 0-612-86661-0

Le 12 septembre 2003,
Date

le jury a accepté le mémoire de M. Martin Labrie dans sa version finale.

Membres du jury

M. Froduald Kabanza
Directeur
Département de mathématiques et d'informatique

M. Goze B. Bénié
Codirecteur
Département de géographie et télédétection

M. Richard St-Denis
Membre
Département de mathématiques et d'informatique

Mme Hélène Pigot
Président-rapporteur
Département de mathématiques et d'informatique

SOMMAIRE

Agents négociateurs appliqués à la télédétection

par Martin Labrie

Directeur de thèse : Professeur Froduald Kabanza
Département de mathématiques et d'informatique

La négociation a toujours existé dans le monde du commerce. L'explosion du commerce électronique depuis quelques années a entraîné un besoin d'automatiser les mécanismes qui régissent cette négociation. Face à la surabondance d'informations sur Internet, les acheteurs utilisent maintenant des *agents logiciels* pour les aider à trouver les meilleures offres. Face à la quantité grandissante d'acheteurs, les vendeurs font aussi appel à des agents pour les représenter. Ce sont donc ces agents qui entreprennent les négociations en lieu et place des commerçants. Les différents algorithmes possibles utilisés dans une négociation commerciale permettent aux agents d'arriver à une entente sur le prix du produit/service ou sur un échec.

Le mécanisme de négociation ne se limite pas au commerce. Il peut survenir partout où existe un échange de service, par exemple lorsqu'une application informatique fait appel à une autre composante pour effectuer une tâche bien précise. En télédétection, par exemple, la surabondance actuelle d'algorithmes de traitement d'images devient telle qu'il est difficile pour un utilisateur de choisir l'algorithme qui est le plus approprié. C'est dans ce cadre que nous avons développé un logiciel de traitement d'images pour la télédétection, appelé I3Soft. Ce logiciel offre à ses utilisateurs non seulement plusieurs algorithmes de traitement d'images satellitaires, mais aussi un outil d'aide à la

sélection d'algorithmes de traitement d'images. Pour une tâche de traitement d'images bien précise (par exemple, le filtrage, la segmentation ou la classification), en général il peut y avoir plusieurs algorithmes plus ou moins appropriés selon différents critères tels que les propriétés de l'image, la précision des résultats désirée et le temps de traitement disponible. Le problème est que ces critères sont généralement contradictoires. Par exemple, un algorithme peut donner des résultats plus précis qu'un autre pour un certain type d'image, mais exiger un temps de calcul plus long.

Le problème de sélection de l'algorithme approprié devient donc un problème de négociation : l'interface est vue comme un *agent acheteur* (du service de traitement d'images) et chaque algorithme de traitement d'images est vu comme un *agent fournisseur* qui entre en compétition avec les autres pour offrir le meilleur service de traitement d'images. L'agent acheteur négocie donc avec les différents agents fournisseurs qui sont en compétition et choisit celui qui offre le meilleur service selon les critères définis par l'interface. Ainsi, du point de vue de la conception et de la programmation de cette interface, nous avons un système *multi-agents*. La méthodologie de conception et de programmation à base d'agents est assez populaire dans le domaine de l'intelligence artificielle et nous nous sommes basés sur cette méthodologie pour concevoir notre logiciel I3Soft.

I3Soft est une application complexe, conçue et programmée par toute une équipe, et comportant plusieurs composantes, incluant les algorithmes de traitement d'images proprement dit, un système expert d'interprétation d'images et l'interface graphique dont nous venons de parler. Ce mémoire concerne uniquement l'interface graphique, c'est-à-dire notre contribution au projet I3Soft. Le mémoire présente donc l'interface graphique, explique le

système multi-agents utilisé, décrit la technique de négociation codée dans le système et discute des forces et faiblesses d'une telle approche pour la conception et la programmation des interfaces graphiques en général et d'une interface graphique pour le traitement d'images en particulier.

REMERCIEMENTS

Je voudrais avant tout remercier mon directeur, Froduald Kabanza, pour ses judicieux conseils, son expertise et ses commentaires constructifs. Merci également à mon co-directeur, Goze Bertin Bénié, pour son support constant tout au long de ce projet.

J'aimerais en outre remercier Yves Voirin, d'une part, pour son expertise en télédétection et, d'autre part, pour m'avoir présenté sa sœur. Merci aussi au reste de l'équipe Estritel pour les échanges enrichissants lors des réunions. Merci à toute l'équipe du Cartel pour tous les bons moments passés ensemble.

Enfin, je voudrais exprimer ma gratitude à ma famille, mes amis et ma Virginie pour leur soutien et leurs encouragements constants.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

Introduction

L'avènement et le développement de l'informatique au cours des dernières décennies ont bouleversé la plupart des domaines scientifiques et des méthodes de recherche. La vulgarisation de la technologie facilite maintenant l'accès aux outils autrefois réservés aux experts. Les ordinateurs étant maintenant accessibles à tous, l'acquisition d'informations ne requiert que le bon logiciel. Malheureusement les utilisateurs de ces outils se retrouvent ensevelis sous une avalanche d'informations et comme la plupart d'entre eux n'ont pas suivi de formation en sciences de l'information, ils ne savent comment discerner l'information pertinente du reste.

Un des domaines des plus touchés par le développement technologique est la télédétection. Cette discipline se définit comme étant l'acquisition d'informations sur un objet sans être en contact avec lui. Plus précisément, la télédétection est la détection de la surface de la terre à partir de l'espace en utilisant les propriétés des ondes électromagnétiques émises, réfléchies ou diffractées par les objets détectés, dans le but d'améliorer la gestion des ressources naturelles, l'utilisation du territoire et la protection de l'environnement.

En télédétection, l'informatique a permis de développer de nouveaux algorithmes de traitement d'images satellitaires. Plusieurs logiciels de traitement d'images ont été mis sur pied et commercialisés. Par ailleurs, ce développement de l'informatique et des nouvelles technologies est si fulgurant qu'il en devient impossible, même pour un expert, d'en suivre la trace avec précision. En télédétection de nouveaux algorithmes de traitement sont mis sur

pied tous les jours, et pour un utilisateur qui veut extraire de l'information d'une image, le choix du meilleur algorithme à utiliser est de plus en plus difficile.

Prenons par exemple un forestier qui veut effectuer la mise à jour d'une carte forestière d'une région du Labrador. Il possède l'ancienne carte forestière et de nouvelles images satellitaires à partir desquelles il doit extraire les nouvelles informations, c'est-à-dire pour les zones de coupe à blanc, ou bien encore les zones dévastées par un feu de forêt. Il doit pour cela, en premier lieu, passer certains filtres pour éliminer le bruit (dû aux interférences des ondes utilisées pour détecter le sol) et apporter les corrections géométriques et atmosphériques (dues à l'ombrage, aux nuages, à l'inclinaison du sol) sur ces nouvelles images. Le forestier doit ensuite choisir une méthode de segmentation pour découper les différentes zones et il aura par la suite à identifier, classifier ces zones et interpréter les résultats obtenus. Bien que ce forestier sache bien ce qu'il veut obtenir comme résultat (l'identification des zones de coupe à blanc et de feux de forêt), il ne sait pas quelles opérations et quels algorithmes utiliser pour extraire l'information la plus précise possible.

Les logiciels de traitement d'images classiques permettent d'effectuer la plupart des opérations nécessaires à l'extraction d'informations sur des images satellitaires, mais ils n'offrent pas d'assistance aux utilisateurs non-experts, comme le forestier, qui ne connaissent pas ces opérations. Ces logiciels sont dotés d'une interface utilisateur complexe et ils s'adressent avant tout à une clientèle de scientifiques environnementalistes et gestionnaires de ressources naturelles, qui sont des experts en traitement et en interprétation d'images satellitaires. Pour les non-experts comme le forestier, une automatisation de la sélection des algorithmes de traitement d'images est donc nécessaire.

Cette problématique est à la base des travaux menés par notre groupe de recherche. Dans ce mémoire, nous présentons le logiciel que nous avons développé dans le cadre du projet SITI (Système Intelligent de Traitement d'Images) et qui effectue une telle automatisation. Ce logiciel s'appelle I3Soft (Intelligent Image Interpretation Software). Notre groupe de recherche est composé de professeurs chercheurs et d'étudiants en informatique et en géomatique. Comme chaque membre de l'équipe a des objectifs académiques personnels à atteindre au niveau du projet, I3Soft est le résultat de la conjonction des travaux de chacun des membres de cette équipe. C'est pourquoi, comme vous le verrez au cours de ce mémoire, I3Soft est un système modulaire, système qui contient les modules suivants :

- les algorithmes et les méthodes de traitement d'images ;
- un système expert d'interprétation d'images ;
- une interface utilisateur.

Les différents traitements d'images sont des algorithmes développés au CARTEL (Centre d'Applications et de Recherches en TELédétection de l'Université de Sherbrooke) . Ce sont entre autres des algorithmes de filtrage, de correction géométrique et géographique et de segmentation d'images. Le système expert d'interprétation d'images est réalisé conjointement par les étudiants de géomatique et d'informatique. Le but est d'arriver à réaliser un système capable de classifier automatiquement et correctement les zones d'une image segmentée.

Notre contribution dans ce projet par rapport aux travaux de toute l'équipe se situe au niveau de l'interface utilisateur du système. Effectivement, nous

verrons dans ce mémoire les méthodes que nous avons utilisées pour remplir nos objectifs personnels, qui sont de :

- développer un système de traitement d'images satellitaires comportant une interface utilisateur conviviale et adaptée au niveau de connaissance en télédétection de l'utilisateur ;

- bâtir ce système sur une architecture modulaire pour simplifier l'ajout de fonctionnalités et d'opérations de traitement d'images développées par le reste de notre équipe ;

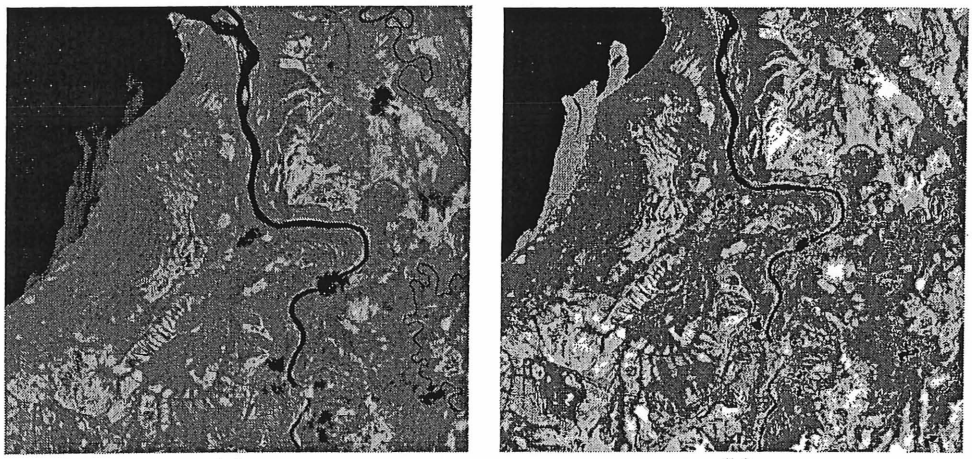
- doter ce système d'une forme d'intelligence pour assister l'utilisateur.

Le chapitre 1 de ce mémoire fait le point sur les différentes méthodes de segmentation dont nous nous servirons pour élaborer notre module d'aide à la décision. Le chapitre 2 traite des principaux systèmes d'information géographiques disponibles sur le marché. Nous y faisons également une comparaison de ces systèmes avec I3Soft. Le chapitre 3 est une introduction au concept d'agent et de systèmes multi-agents. Nous y faisons aussi un survol de MADKit, la librairie dont nous nous sommes servie dans l'architecture d'agents d'I3Soft. Les chapitres 4 et 5 traitent I3Soft de façon complémentaire ; le chapitre 4 aborde le logiciel plutôt du point de vue de l'utilisateur, avec ses interfaces et les fonctionnalités qu'il offre, tandis que le chapitre 5 fait la lumière du point de vue du développeur, en élaborant sur l'architecture interne du logiciel. Le chapitre 6 présente le module d'aide à la décision que nous avons développé. Nous y expliquons notamment les mécanismes de négociation du système d'aide à la décision. Le chapitre 7 fait une présentation des résultats obtenus, et le tout est suivi d'une conclusion sur la validité de ces résultats et les recommandations pour de futurs développeurs.

Chapitre 1

LE TRAITEMENT D'IMAGES

En imagerie satellitaire, lorsqu'on veut extraire de l'information d'une image, on doit dans la plupart des cas effectuer une segmentation sur cette image. Ce procédé consiste à regrouper les pixels homogènes voisins dans l'image. Ces regroupements sont appelés zones, ou segments, et sont délimités à leur périphérie par les segments avoisinants ou par les limites de l'image. La figure 1 illustre bien ce procédé. On y voit le résultat d'une segmentation sur une image Landsat d'une région du Labrador. L'image originale (a) est segmentée puis séparée de façon, dans ce cas-ci, à donner quatre types de zones (ou classes) différentes indiquées par des couleurs différentes dans l'image (b).



(a) (b)

Figure 1. Résultat d'une segmentation sur une image Landsat.

Ces segments peuvent être déterminés par différents algorithmes avec des résultats variables, mais dans tous les cas le but est d'identifier et d'étiqueter

ensuite chacun de ces segments. On distingue principalement deux grandes familles de segmentation: les segmentations supervisées et les segmentations non supervisées. Dans la première catégorie, c'est l'utilisateur qui détermine le nombre de classes différentes. Dans la deuxième, il est déterminé automatiquement.

Dans ce chapitre nous voyons les quatre méthodes de segmentation que nous utilisons dans le module d'aide à la décision que nous avons développé. Ces méthodes sont des algorithmes courants dans le domaine de la télédétection, mais ceux que l'on utilise ont été codés par Yves Voirin, un membre du groupe ESTRITEL. Notre approche est généralisable à un plus grand nombre d'algorithmes et à d'autres types de traitements que la segmentation, mais nous nous sommes limités à seulement quatre méthodes, vu les limites de temps dont nous disposions pour créer un tel système.

Quand l'utilisateur envoie une requête de traitement d'une image au module d'aide à la décision, celui-ci devra choisir laquelle des quatre méthodes correspond le plus aux besoins de cet l'utilisateur. Pour ce, il se base sur les propriétés des images traitées d'une part, et sur les caractéristiques des méthodes d'autre part. L'analyse que nous faisons dans ce chapitre a pour but de bien cerner ces caractéristiques, qui sont le temps d'exécution et la qualité de l'image résultant, et qui sont directement reliés aux propriétés des images.

1.1 Méthodes utilisées

1.1.1 Segmentation *Maximum Likelihood*

Le *Maximum Likelihood* (ML) est une méthode d'approximation de point, qui utilise comme un estimé d'un paramètre d'une population inobservable[1] le membre de l'espace du paramètre qui maximise la fonction de vraisemblance. L'algorithme ML se divise en deux étapes principales : estimation des paramètres et segmentation.

Pour effectuer l'estimation des paramètres nous devons tout d'abord poser le problème dans le contexte du traitement d'images satellitaires. Dans un tel cas, la population est composée des pixels constituant l'image. La segmentation a pour but d'assigner chaque pixel à une zone. Pour ce faire, nous devons tout d'abord identifier les différentes classes de l'image. Chaque classe est caractérisée par trois paramètres : la moyenne U, la variance V et la corrélation spectrale du bruit W. L'estimation se fait à partir d'un échantillon de la population des pixels parce que la taille des images est bien souvent trop importante pour calculer les paramètres à partir de toute la population en un temps raisonnable.

La moyenne et la variance permettent de trouver la probabilité P pour un pixel p_{xy} d'appartenir à une classe k selon la formule

$$P_k(p_{xy}) = (1/(2\pi)^{1/2} V_k)exp(-(p_{xy}-U_k)^2/2V_k^2)$$

Cette formule est celle d'une distribution de probabilité gaussienne. Comme tous les points contribuent de façon différente à l'incertitude sur les moyennes U_k, nous devons donc, pour chaque pixel de l'échantillon, additionner les probabilités d'appartenir à chaque classe k en pondérant avec la valeur de

[1] En statistique, on appelle population l'ensemble des données duquel on retire des informations comme la moyenne.

poids de chaque classe W_k, c'est-à-dire en mettant plus de poids aux pixels qui ajoutent le moins d'incertitude à la somme G.

$$G = \sum_k P_k(p_{xy}) * W_k$$

Il ne reste plus par la suite qu'à extraire le logarithme de cette fonction ($LogG$) et de normaliser les probabilités. Par la suite l'algorithme prévoit de successivement raffiner les paramètres U, V et W, et de recalculer $P_k(p_{xy})$, G et $LogG$ autant de fois que spécifié par l'utilisateur.

Une fois que le nombre d'itérations a été suffisant pour bien ajuster les valeurs de U, V et W, nous passons à la phase de segmentation en attribuant à chaque pixel la valeur de la classe k pour laquelle la probabilité $P_k(p_{xy})$ est la plus élevée.

Pour lancer cet algorithme, le système a besoin de certaines valeurs que l'utilisateur pourra fournir en entrée :

- k, le nombre de classes désirées ;
- n, le nombre d'itérations de raffinement des paramètres U, V et W ;
- s, step, la distance en pixels entre deux pixels d'échantillon lors de l'estimation ;
- $kMax$, le nombre maximal de classes.

Après une analyse détaillée de l'algorithme ML, nous avons pu établir une formule déterminant sa durée approximative d'exécution. On obtient cette formule en parcourant le code de la méthode et en portant une attention

particulière aux boucles d'itération qui s'y trouvent. Par exemple, si à un endroit de l'algorithme on doit exécuter une fonction à tous les pixels, on appellera cette fonction dans une double boucle du genre :

```
...
for (int i=0;i<x;i++) {
for (int j=0;j<y;j++) {
fonction(i,j);
...
}
}
...
```

Ce code se traduit par un terme dans l'équation Cxy, où C est une constante déterminée par la durée d'exécution des lignes de code à l'intérieur des boucles, x et y sont respectivement le nombre de colonnes et le nombre de lignes de l'image traitée. En additionnant toutes les boucles de l'algorithme, on obtient la formule suivante:

$$\Theta(ML) = C_1 + C_2k + xy/s(C_3 + C_4k + C_5k^2 + nk(C_6 + C_7s + C_8ks + C_9xy/s)),$$

où C_1 à C_9 sont des constantes quelconques, x est la largeur de l'image et y la hauteur. Si on ne retient que les éléments les plus significatifs, c'est-à-dire les sections de code qui exigent un temps d'exécution significativement plus long, et qu'on considère les autres éléments comme négligeables, on obtient une durée d'exécution de l'ordre de

$$\Theta(ML) = C_1 + xy/s(C_2nk + C_3nsk^2 + C_4nkxy/s) \text{ (éq. 1.1)}$$

1.1.2 Segmentation *Stochastic Expectation Maximization*

Le *Stochastic Expectation Maximization* (ou SEM) est un estimateur de mélange de densités, qui est une variante stochastique de la méthode *Expectation Maximisation* (EM). La méthode EM est basée sur les mêmes principes que ML. Comme ML, EM et SEM sont des segmentations non supervisées, elles font appel aux mêmes paramètres. L'estimation de ces paramètres peut être faite par EM, *Gradient Stochastique* ou *Estimation Conditionnelle Itérative* (ICE). Toutefois, ces méthodes peuvent devenir très instables avec des images non homogènes. Elles ne sont pas bien adaptées aux images prises avec le satellite SPOT qui contiennent des zones urbaines, et elles possèdent aussi, excepté SEM, des limitations :

- le nombre de classes est présumé connu ;
- la solution dépend fortement de l'initialisation.

L'algorithme EM ne converge pas nécessairement au ML et le résultat dépend des valeurs initiales. Alors il est souvent efficace d'exécuter plusieurs fois EM avec des valeurs initiales différentes et de seulement garder la meilleure valeur.

Le SEM est une amélioration du EM obtenue par l'addition d'une composante stochastique. Cette addition ajoute au SEM, par rapport au EM, les points suivants :

- il suffit de connaître une limite supérieure du nombre de classes ;
- la solution est essentiellement indépendante des valeurs d'initialisation ;
- la vitesse de convergence est améliorée.

L'algorithme SEM est principalement constitué de trois étapes :

- étape-E (Estimation), où pour chaque échantillon, on définit la prochaine distribution sur l'ensemble des classes ;

- étape-S (classification stochastique), où pour chaque échantillon on sélectionne, à partir de l'ensemble de classes, un élément selon la distribution des probabilités de classes ;

- étape-M (maximisation), où on calcule les paramètres et on ajuste les distributions de classes [P.Masson, W. Pieczynski 1993].

L'étude de l'algorithme pour cette méthode (qui a été faite selon les mêmes principes que pour l'algorithme ML) nous donne, après réduction des éléments négligeables, un temps d'exécution de l'ordre de

$$\Theta(SEM) = C_1 + C_2k + C_3xyk + C_4xyk^2 \text{ (éq. 1.2)}$$

La durée d'exécution sera alors proportionnelle au carré du nombre de classes multiplié par la taille de l'image en pixels.

1.1.3. Segmentation *Split & Merge*

La méthode *Split & Merge* (ou S&M) consiste en premier lieu à fragmenter l'image en quatre cadrans, et de fragmenter chacun de ces cadrans à leur tour de façon récursive jusqu'à ce que chacun de ces cadrans constitue une zone homogène. Ensuite, il faut fusionner les régions adjacentes semblables pour éliminer les frontières artificielles causées par ces fragmentations récursives. Les procédés de fragmentation et de fusion utilisent la variance de l'intensité dans une région comme mesure de son homogénéité. Cette variance, déterminée par l'utilisateur, contrôle indirectement le nombre et la taille des

régions produites par l'algorithme. L'image résultante contient la valeur moyenne de l'intensité de chaque région identifiée.

Il existe toutefois une lacune dans l'adaptabilité à la sémantique de l'image à cause de sa structure arborescente carrée. Une technique de seuil est employée dans la phase de segmentation pour refléter directement la sémantique de l'image dans le résultat de la segmentation de l'image. Alors, les régions qui contiennent des sous-régions distinctes peuvent être extraites en une seule étape, ce qui allège considérablement la charge de calcul et la mémoire nécessaire.

En examinant le code, nous avons cerné la portion critique pour la durée de l'exécution de l'algorithme :

```
void SplitScalar(int Xlow, int Xhigh, int Ylow, int Yhigh, float[][] Data1, int[][]
Data2)
{...
 if (Variance <= VarThresh)  /* Condition d'arrêt */
 {...
 }
 else /* Récursivement séparer et fusionner les régions */
 {
        Xmid = (Xhigh + Xlow) / 2;
        Ymid = (Yhigh + Ylow) / 2;
        SplitScalar(Xlow, Xmid, Ylow, Ymid, Data1, Data2);
        SplitScalar(Xlow, Xmid, Ymid, Yhigh, Data1, Data2);
        SplitScalar(Xmid, Xhigh, Ylow, Ymid, Data1, Data2);
        SplitScalar(Xmid, Xhigh, Ymid, Yhigh, Data1, Data2);
        for (y = Ylow; y < Yhigh; y++)
              MergeScalar(Data2[y][Xmid], Data2[y][Xmid - 1]);
              if (Ymid > 0)
        for (x = Xlow; x < Xhigh; x++)
              MergeScalar(Data2[Ymid][x], Data2[Ymid - 1][x]);
 }
...
}
```

La fonction *SplitScalar()*, qui est appelée initialement pour toute l'image et qui la sépare en quatre cadrans, s'appelle récursivement sur chacun de ces quatre cadrans jusqu'à ce qu'une des conditions d'arrêt soit atteinte. La séparation récursive se terminera soit lorsque les pixels constituant la région seront suffisamment homogènes ou, dans le pire des cas, lorsque la région atteindra une seule dimension, c'est-à-dire lorsqu'elle aura une hauteur ou une largeur d'un seul pixel. On peut voir sur la figure 2 la séparation causée par les trois premiers appels de *SplitScalar(n)* dans le premier cadran.

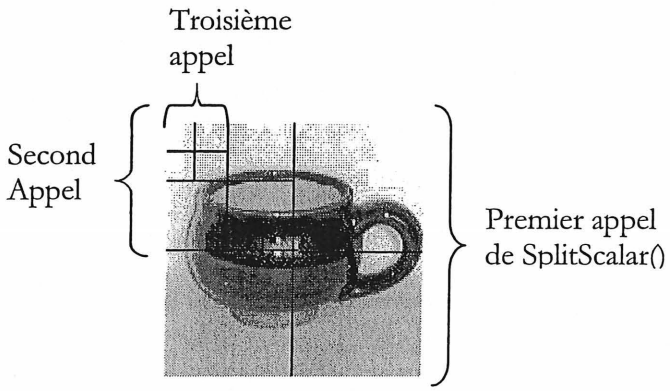


Figure 2. Phases de séparation

Donc au premier appel de *SplitScalar(n=1)*, on aura une durée d'exécution de l'ordre de xy, plus les quatre appels récursifs de *SplitScalar(n=2)*, qui s'exécuteront sur une zone de x/2 colonnes par y/2 lignes de pixels, et ainsi de suite, ce qui donne comme équation

$$\Theta(S\&M) = xy + 4(x/2)(y/2) + 4(4((x/2)/2)((y/2)/2)) + \ldots$$

En généralisant, on obtient la suite:

$$\Theta(S\&M) = xy + 4xy/4 + 16xy/16 + \ldots = \sum 2^{2n} xy/2^{2n}$$

Nous devons maintenant calculer le nombre d'appels récursifs n nécessaires avant de rencontrer la condition d'arrêt. Comme nous l'avons déjà mentionné, nous atteignons cette condition dans deux cas. Tout d'abord en divisant les régions jusqu'à ce qu'elles soient homogènes, ce qui est impossible à déterminer d'avance. L'autre condition d'arrêt, survenant dans le pire cas, est quand les régions se réduisent à une dimension (x ou y = 1). Donc, les divisions cesseront quand

$$min(x,y)/2^{2n} = 1$$

Donc, à la n ième itération, où

$$n = (log_2 min(x,y)/2)$$

L'analyse de durée du S&M nous donne donc un temps d'exécution de l'ordre de

$$\Theta(S\&M) = C_1 + C_2 \sum_{n=0,\, n=(log2(min(x,\, y))/2)} xy \quad \text{(éq. 1.3)}$$

1.1.4. Segmentation *Watermerge*

L'algorithme *Watermerge* (ou WM) repose sur les principes du *Watershed* (WS). La segmentation WS tient son nom de la façon dont l'algorithme segmente les régions de l'image en forme de bassins versants. Ces bassins sont des points bas dans l'intensité de l'image segmentée. Les bassins y sont référés comme segments ou régions. Ces régions partagent des frontières entre elles. Ces frontières peuvent être vues comme étant des points hauts dans l'intensité

de l'image, tout comme des crêtes des montagnes qui séparent des vallées. Pour mieux comprendre, on peut imaginer la pluie tombant dans ces bassins. Comme le niveau de l'eau monte, le bassin se remplit et l'eau qui atteint le niveau de la crête se déverse dans un autre bassin. De cette façon nous pouvons voir la segmentation de l'image en bassins à différents niveaux d'échelle, une mesure continue spécifiée par le niveau de l'eau.

Le résultat d'un WS est une hiérarchie de bassins qui peuvent être vus à différents niveaux d'échelle. Ce qu'il reste à faire pour l'utilisateur est de déterminer quels bassins à quels niveaux d'échelle représentent des structures d'intérêt.

Les entrées de l'algorithme du WS sont critiques pour la qualité de sa sortie. L'algorithme attend une image de hauteur en entrée. Une image de hauteur est définie dans ce contexte comme une image où les intensités d'images les plus hautes correspondent aux frontières logiques entre les régions d'intérêt.

Le WS ne cible pas une région particulière de l'image, il la traite en entier. Pour pouvoir étiqueter une structure particulière dans une image, le résultat du WS doit être manipulé à la main. C'est pourquoi on qualifiera cet algorithme de semi-automatique.

Voici les avantages du WS :

- n'utilise pas de valeur de seuil ;
- les régions résultant sont fermées, alors il est facile de procéder à la fusion après la segmentation.

Voici les désavantages du WS :

- dépend beaucoup de la méthode utilisée pour produire l'image du gradient ;

- produit généralement une image trop texturée (trop segmentée).

Pour pallier ce désavantage de texture excessive, il convient d'utiliser l'algorithme WM, qui calcule le gradient des régions tracées par le WS, et ensuite exécuter une procédure itérative de fusion pour combiner certaines de ces régions selon leurs similarités. Le résultat donne une image de segmentation avec une très bonne localisation des contours.

L'analyse de durée du WM nous donne un temps d'exécution de l'ordre de :

$$\Theta(WM) = C_1 + C_2 r + C_3 xy + m(C_4 i + C_5 xy) \quad \text{(éq. 1.4)}$$

où r est le nombre de régions répertoriées, m est le nombre d'itérations de fusion (établi par l'utilisateur) et i est la quantité d'informations de l'image ($i = 5r + 1000$)

1.2. Différentiation entre les méthodes

Ces quatre méthodes constituent actuellement les seuls choix disponibles pour le module d'aide à la décision. Nous devons donc bien cerner les caractéristiques de chacune de ces méthodes parce ce sont elles qui influenceront le choix automatique de traitement. Nous étudierons principalement deux caractéristiques : le temps d'exécution de l'algorithme et la qualité du résultat en fonction de l'image en entrée.

1.2.1. Temps d'exécution

Dans le domaine de la télédétection, on travaille habituellement avec des images de grande taille qui contiennent beaucoup d'informations. Lorsqu'on effectue un traitement sur une de ces images, il arrive que le résultat mette du temps à s'afficher. Plusieurs facteurs entrent en cause ici : la taille de l'image, la texture et le nombre de classes.

Voici le tableau récapitulatif des formules présentées dans la section précédente.

Tableau 1. Temps d'exécution des algorithmes de segmentation

Méthode	Ordre de grandeur du temps d'exécution Θ
ML	$C_1+xy/s(C_2nk + C_3nsk^2+C_4nkxy/s)$
SEM	$C_1+C_2k+ C_3xyk+C_4xyk^2$
S&M	$C_1+ C_2\sum_{n=0,\ n=log2(min(x,\ y))}(2^{2n}/2^n)xy$
WM	$C_1+C_2r+ C_3xy+m(C_4i+C_5xy)$

Il est difficile de tirer des conclusions immuables de ce tableau étant donné le grand nombre de paramètres influençant les durées. On peut toutefois émettre certaines observations. Par exemple, en regardant du côté de ML, on voit que c'est le seul des quatre algorithmes dont le temps de traitement augmente avec le carré de la taille de l'échantillon d'image $(xy/s)^2$. Il est alors probable de penser que ML sera pénalisé pour le traitement de grandes images. On peut quand même toutefois pallier un peu à ce problème si l'utilisateur choisit une

plus grande distance s entre les échantillons de pixels. Pour ce qui est du SEM, la durée dépend aussi de la taille de l'image, mais il semble tout de même plus reliée au nombre de classes k que le ML. Cet algorithme sera alors plus rapide pour les images ne comportant pas trop de classes différentes, mais il reste que le SEM est une méthode très lente comparativement aux trois autres. En ce qui concerne le S&M, il est difficile de le comparer aux autres segmentations de par sa formule, mais on voit que sa durée est fortement dépendante de la taille de l'image et aussi de la variance minimale établie par l'usager au-delà de laquelle on sépare encore deux régions. Cette variance est une condition d'arrêt de l'algorithme. Soulignons tout de même que le S&M est la méthode la plus simple et la plus rapide des quatre. Du côté du WM, l'algorithme sera généralement lent, plus particulièrement pour les images très texturées. En effet, plus il y aura de régions r et d'informations i à traiter, plus le temps de traitement sera long. De plus, si une image est trop texturée, il y aura plus de traitements de fusion m à effectuer. La figure 3 permet d'avoir un aperçu concret des durées relatives d'exécution (en millisecondes) des quatre algorithmes en fonction de la taille de l'image (en pixels). On y voit que le S&M est effectivement beaucoup plus rapide que les autres algorithmes, peu importe la taille de l'image. On constate aussi que la courbe du ML tend à suivre une ligne droite, ce qui signifie effectivement que le temps est proportionnel au carré de la taille de l'image (ou le nombre de pixels). Le temps d'exécution de WM n'est pas directement relié à la taille de l'image, mais on constate tout de même qu'il peut être significativement plus lent pour certaines images. Finalement, SEM est visiblement plus lent que les trois autres algorithmes.

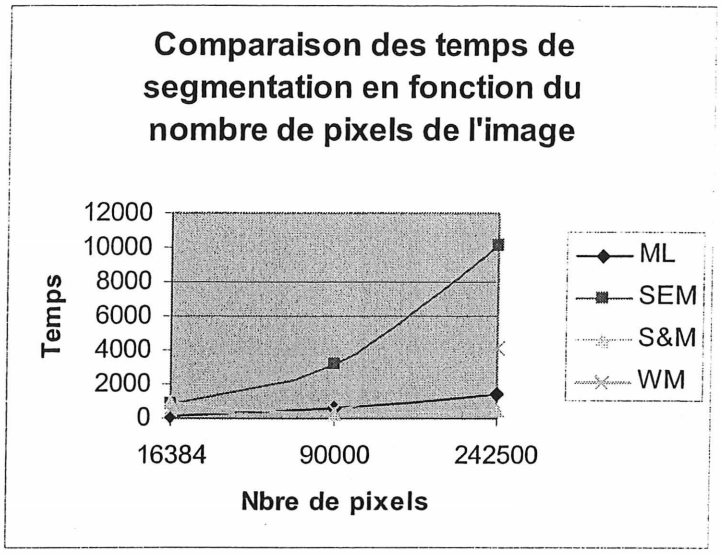


Figure 3. Temps d'exécution des algorithmes de segmentation

1.2.2. Qualité du résultat

La qualité du résultat dépendra principalement de la combinaison d'images d'entrée avec le choix du traitement, car certains traitements sont plus adaptés à certains types d'images. Par ailleurs, le choix des paramètres d'entrée et leur valeur joue aussi un rôle capital.

Nous connaissons déjà quelques contraintes imposées par certains algorithmes pour des types d'images données. La figure 4 démontre les différents résultats obtenus par chacun des algorithmes. On y retrouve d'abord l'image à segmenter en (a), qui est une image du Labrador de type Landsat. Les images (b), (c), (d) et (e) représentent respectivement les résultats des segmentations ML, SEM, S&M et WM. On constate que le ML (b) donne un résultat

satisfaisant. Il en résulte une délimitation précise et fidèle des différentes zones.

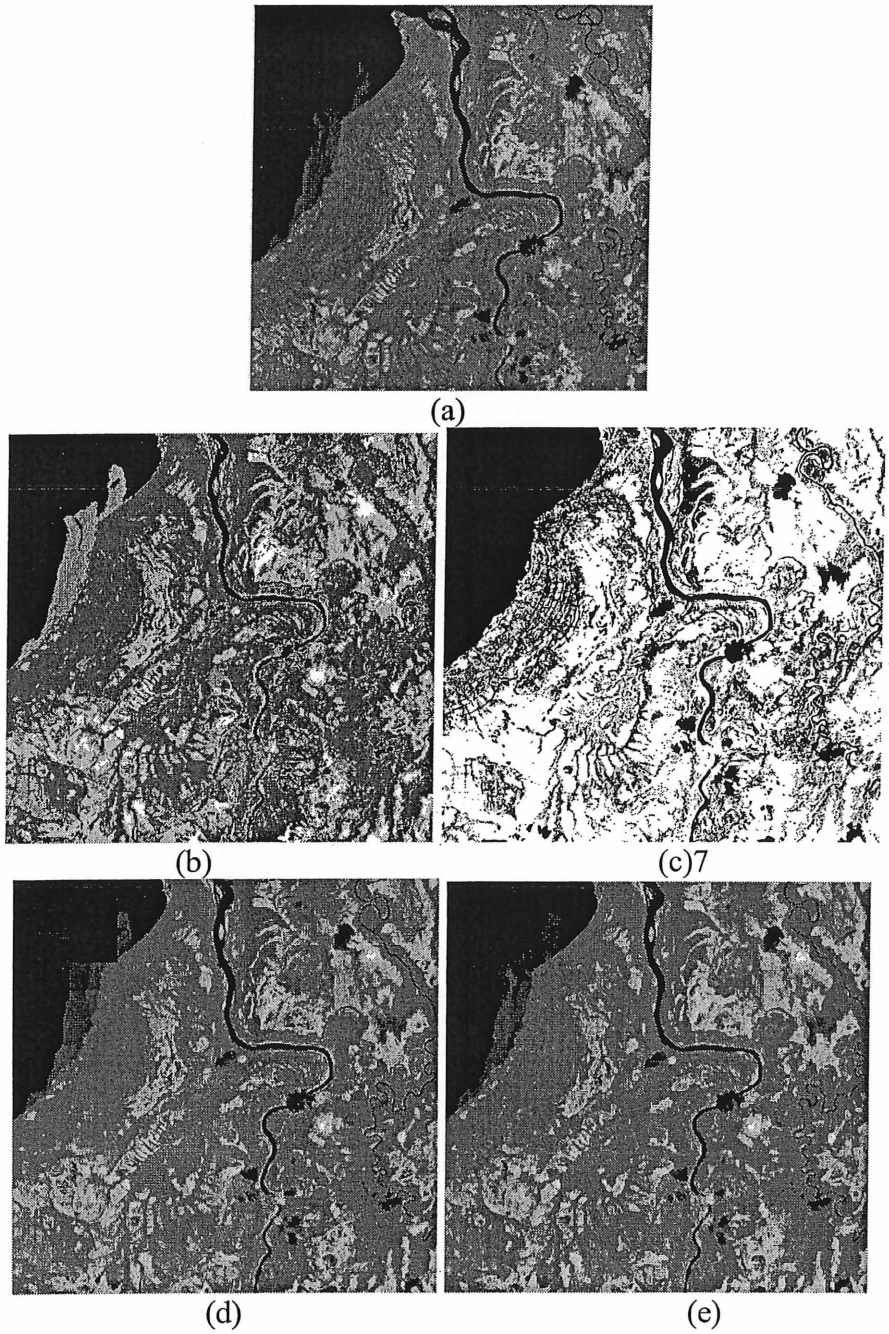


Figure 4. Comparaison des résultats des quatre types de segmentation

Par exemple, les eaux peu profondes au large de la côte au nord-ouest de l'image ont relativement bien été reconnues et délimitées. Pour ce qui est du SEM, le résultat (c) est précis, mais trop texturé et présentant quelques défaillances, comme la disparition de la région d'eau peu profonde au nord-ouest. Par ailleurs, le S&M, qui effectue des segmentations d'architecture plutôt rectangulaires, est déconseillé si l'on accorde de l'importance aux contours des segments et que l'on veut une image de précision, et ceci est particulièrement vrai pour des images d'un milieu naturel comme la forêt. Par exemple, on remarque pour la région d'eau peu profonde que le résultat de l'algorithme S&M est faussé (d). Les contours des zones sont, de par la nature de l'algorithme, trop carrés. Le S&M serait plus recommandé pour des images du milieu urbain, où les zones sont plutôt rectangulaires. Le résultat du WM (e) dans ce cas-ci est plutôt satisfaisant. On constate que les régions y sont plutôt fidèlement reconnues, même si elles sont de forme plutôt arrondies, comme des bassins creux. Ce sont toutes ces caractéristiques qui nous servent de ligne directrice pour l'élaboration de notre module d'aide à la sélection de modules de traitement d'images

Chapitre 2

TRAVAUX CONNEXES

Dans ce chapitre, nous présentons les principaux logiciels de traitement d'images satellites existant sur le marché. Nous y dresserons également un portrait comparatif des caractéristiques et des fonctionnalités offertes par ces logiciels, et celles offertes par I3Soft.

2.1. Les systèmes d'information géographique

Un système d'information géographique est conçu pour capturer, emmagasiner, restaurer, analyser, manipuler et afficher des données référencées spatialement ou géographiquement. On entend par données référencées, des images satellites ou aéroportées de la Terre.

Il existe déjà sur le marché une panoplie de logiciels de traitement d'images qui effectuent sensiblement les mêmes tâches qu'I3Soft. Nous ne survolerons que les trois principaux logiciels commerciaux: Arc/Info, Envi et Geomatica.

2.1.1 Arc/Info

Ce logiciel créé par la société américaine ESRI (Environmental Systems Research Inc) [ESRI] possède des outils pour la modification, la gestion, l'analyse et l'affichage d'informations géographiques. La partie Arc fournit les outils de base pour la collecte de données, l'analyse et les sorties. La partie Info est simplement le système de base de données dont se sert Arc.

Les fonctionnalités de Arc incluent, entre autres, Arcplot, un système cartographique compréhensible pour générer des cartes de qualité, et Arcedit, une librairie graphique pour gérer et éditer des données géographiques.

Arc fournit une approche à base de commandes qui offre la possibilité de développer des macros. Ces dernières donnent la possibilité de définir des menus et d'automatiser des opérations. Arc offre également la possibilité, via Arc-Objet, d'éditer et d'ajouter ses propres algorithmes de traitement. Il existe plusieurs extensions pour ajouter des fonctionnalités au système. Arc adhère aux standards modernes d'informatique et est portable sur plusieurs plates-formes comme UNIX et Windows NT.

Bien que Arc/Info soit plutôt complexe à apprendre et à utiliser, il existe un produit séparé, appelé ArcView, qui est conçu pour donner aux non-experts un accès facile aux fonctionnalités. Il est orienté *Windows,* axé sur le menu et est facile à utiliser.

Le coût d'un tel logiciel s'évaluait en 1995 entre 10 000 et 20 000 $US.

2.1.2. ENVI (ENvironment for Visualizing Images)

ENVI est l'un des logiciels le plus complet de télédétection. Développé par Research Systems [Research Systems Inc], ENVI permet, entre autres, d'effectuer une myriade de traitements d'images satellitaires avec des outils faciles d'utilisation. Il procure également des outils de correction géométrique, d'analyse de terrain et radar, de support pour une grande variété de source d'images, et il permet même d'éditer et d'incorporer ses propres routines via le langage IDL

(Interactive Data Language). ENVI est compatible avec les principaux types de plates-formes, comme Windows, Mac, Unix et Linux.

Le coût de ENVI se situe aux alentours de 7 000 $US.

2.1.3. Geomatica

Ce logiciel développé par PCI Geomatics [PCI Geomatics] est un *package* de programmes conçu pour offrir des solutions complètes pour le marché de la géomatique. Il réunit des technologies qui permettent d'effectuer des traitements d'images satellitaires, des analyses spatiales ou géographiques et de la cartographie en un seul environnement. Avec tous ces outils maintenant disponibles sur un même environnement de visualisation, Geomatica est plus facile d'utilisation. Il peut être installé sur Windows (98, NT et XP), Solaris, IRIX, Linux, AIX et Unix. Il peut être installé sur un système distribué grâce à WebServer.

2.1.4 Comparaison des caractéristiques

Tableau 2. Comparaison entre logiciels de traitement d'images

Nom du logiciel	Conviviali té	Multi plate-formes	Ajout de fonctionnalités	Choix automatique de traitement	Coût estimé ($US)
Arc/Info	oui	oui		non	10 K – 20 K
ENVI		oui	oui	non	7 K
Geomatica	amélioré	oui		non	
I3Soft	oui	oui	à venir	oui	non spécifié

Il est difficile de comparer I3Soft avec ces logiciels puisque son budget et son temps de développement est significativement inférieur aux trois autres logiciels, ce qui se traduit par un éventail plus restreint de ses fonctionnalités. Toutefois, comme illustré dans le tableau 2, I 3Soft bénéficie d'une convivialité comparable à Arc/Info, ENVI et Géomatica. Il est également multi plate-formes et sa modularité lui permettra éventuellement d'ajouter des fonctionnalités.

Ce qui distingue principalement I3Soft des autres logiciels, c'est qu'I3Soft s'adresse à un éventail plus large d'utilisateurs. Alors qu'Arc/Info, ENVI et Géomatica s'adressent aux géomaticiens et autres experts en télédétection, I3Soft s'adresse plutôt à des utilisateurs non experts de la télédétection qui doivent extraire des informations à partir d'images satellitaires mais qui ne savent pas nécessairement de quelle façon. C'est grâce à une interface conviviale mais surtout grâce à son module de choix automatique de traitement qu'I3Soft se démarque et est accessible à ce nouveau marché d'utilisateurs.

Chapitre 3

AGENTS ET MADKIT

Ce chapitre constitue une introduction au concept d'agents. Nous y verrons ce qui caractérise les agents et les mécanismes qui gèrent ces entités. Nous parlerons en second lieu de MADKit, la librairie que nous avons utilisée pour implanter nos agents dans le logiciel I3Soft.

3.1 Agents

Lorsqu'on discute d'agents, ce qu'il faut faire avant toute chose est d'en donner une définition claire. Il est important que l'on cerne ce qui distingue un agent d'un simple programme informatique et ce qui lui confère un caractère intelligent. Il existe plusieurs définitions d'agents, allant de la plus simple à la plus complexe. Une des définitions des plus populaires est celle des agents AIMA (Artificial Intelligence: a Modern Approach) qui est le titre d'un ouvrage très prisé en intelligence artificielle, et où on y retrouve la définition suivante: *«An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.»*[2] [S. Russell, P. Norvig 1995] (figure 5). Clairement, cette définition dépend de ce que l'on entend par *environment* et sur la signification de *sensing* et *acting*. Si l'on définit *environment* comme étant tout ce qui fournit les entrées à l'agent et en reçoit les sorties, que l'on dit de *sensing* que c'est percevoir les entrées, et que l'on dit de *acting* que c'est produire des sorties,

[2]Un agent est tout ce qui peut être vu comme étant une entité qui perçoit son environnement par l'entremise de capteurs et qui agit sur cet environnement par l'entremise de ses actionneurs.

alors tous les programmes informatiques sont des agents. Si l'on veut marquer un contraste entre agent et programme, nous devons restreindre certaines notions d'environnement, de perception et d'action.

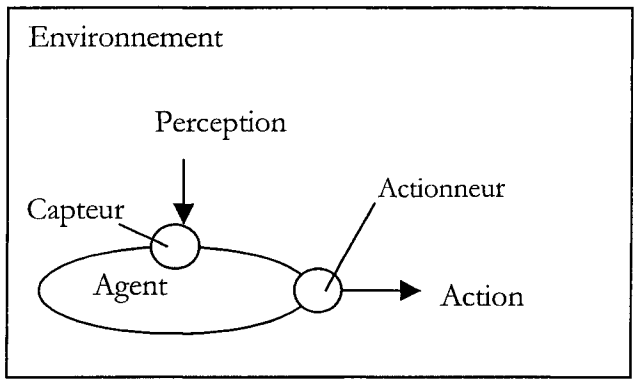


Figure 5. Agent dans son environnement

Prenons par exemple le cas d'un logiciel de traitement de textes qui comporte un agent de correction d'orthographe. Le but de cet agent est donc d'obtenir un texte sans faute. Le texte rédigé et les commandes entrées par l'utilisateur constituent l'environnement. L'usager peut donc envoyer, pour corriger son texte, une commande d'exécution qui est perçue par l'agent. L'agent passe alors à l'action en vérifiant tous les mots et la grammaire du texte et en suggérant des corrections lorsqu'il détecte des erreurs ou même en corrigeant lui-même les erreurs, modifiant ainsi son environnement. Cet exemple simpliste nous laisse un peu perplexe quant à la nécessité d'utiliser un agent, car on pourrait obtenir le même résultat avec un simple logiciel. Nous avons donc besoin d'une définition plus précise.

Pattie Maes, du MIT, est l'une des pionnières de la recherche sur les agents. Dans sa définition : *«Autonomous agents are computational systems that*

inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed.» [3] [P. Maes 1995], elle ajoute un élément crucial dans la définition d'agent : les agents doivent agir de façon autonome pour réaliser un ensemble de buts. Aussi, les environnements sont restreints à être complexes et dynamiques.

Reprenons l'exemple de l'agent correcteur d'orthographe. Cette fois-ci, l'agent ne se contente plus de s'exécuter seulement lorsque l'usager le lui ordonne, mais également au fur et à mesure qu'il tapera de nouvelles fautes. L'environnement devient dynamique puisque le texte change continuellement, et l'agent devient autonome puisqu'il n'attend plus les ordres de l'usager pour s'exécuter.

La définition de Wooldridge et Jennings [M. Wooldridge, N. Jennings 1995], en plus d'énumérer l'autonomie, la perception et l'action, est permissive à un éventail large mais fini d'environnements. Cette définition ajoute également une contrainte de communication. Voici cette définition : *«... a hardware or (more usually) software-based computer system that enjoys the following properties:*

- *autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;*

[3] Les agents autonomes sont des systèmes informatique qui baignent dans un environnement complexe et dynamique, qui perçoivent et agissent de façon autonome dans cet environnement et, ce faisant, réalisent un ensemble de buts ou de tâches pour lesquels ils sont conçus.

- *social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language;*

- *reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;*

- *pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative.»* [4]

Cette définition diffère de la précédente principalement sur le point de la pro-activité. Si l'on voulait que notre correcteur d'orthographe devienne pro-actif, on pourrait le programmer de sorte que si l'usager refuse toujours les corrections de l'agent avant que son texte ne soit terminé, l'agent pourrait alors prendre l'initiative de demander à l'usager s'il veut travailler sans son aide. Dans ce cas, nous commençons à bien discerner le caractère intelligent de l'agent.

---

[4] Traduction : "... un système informatique basé sur le matériel ou (plus souvent) sur le logiciel qui respecte les propriétés suivantes :

- autonomie : les agents opèrent sans l'intervention directe des humains ou des autres, et ont un certain contrôle sur leurs actions et sur leur état interne;
- capacité sociale : les agents interagissent avec d'autres agents (et possiblement des humains) via une sorte de langage de communication entre agents;
- réactivité : les agents perçoivent leur environnement, (qui peut être le monde physique, un utilisateur via une interface graphique, un ensemble d'autres agents, l'Internet, ou peut-être toutes ces choses combinées), et répond aux changements qui s'y produisent;
- pro-activité : les agents n'agissent pas simplement en réponse à leur environnement, ils peuvent exhiber un comportement dirigé pour atteindre un but en prenant des initiatives."

Quand un système devient plus complexe et qu'il nécessite plusieurs agents, on parle alors d'un système multi-agents. Un système multi-agents est constitué d'agents (sous forme de processus parallèles) qui ont chacun leur tâche simple à effectuer et qui, lorsqu'ils travaillent de pair, peuvent réaliser des tâches plus complexes comme les prises de décision. Nous verrons au chapitre 5 une application de ce type d'architecture. De tels systèmes permettent d'exploiter le caractère social propre à l'agent. On entend par ce caractère social la capacité de l'agent à communiquer avec ses semblables via un langage qui leur est propre.

Le chapitre 5 sera consacré aux agents que l'on a implémenté dans le système et à l'intelligence qui émerge de leurs actions combinées.

3.2 MADKit (Multi-agent Development Kit)

Pour le développement de notre système multi-agents, deux choix nous étaient offerts. Nous pouvions choisir en premier lieu de développer nos propres outils de développement d'agents. Dans ce cas, nous perdons quelques semaines, voire quelques mois (tout dépendant du degré de performance que l'on veut attribuer à cet outil) à écrire le code. Par ailleurs de cette façon, nous sommes sûrs d'obtenir un produit sur mesure. Nous avions aussi comme choix d'utiliser des librairies qui ont déjà été développées par d'autres programmeurs. Dans ce cas il importe de bien cerner nos besoins et de bien comparer tous les produits disponibles pour faire le meilleur choix possible, car il existe sur le marché une panoplie d'outils commerciaux et non commerciaux pour faciliter l'implantation d'une architecture multi-agents. Dans notre cas, nous avons décidé d'utiliser une librairie déjà disponible.

3.2.1. Caractéristiques

Parmi tous les choix qui nous étaient offerts, nous avons choisi MADKit (Multi-Agent Development Kit), qui est une plate-forme multi-agent basée en Java et bâtie sur un modèle organisationnel (agent/groupe/rôle). MADKit procure des fonctionnalités générales de gestion d'agents (cycle de vie, messagerie, distribution). Nous avons choisi MADKit pour plusieurs raisons :

- il est multi-plateforme. Cette caractéristique facilite son intégration dans notre système, qui possède aussi cette caractéristique, les deux étant développés en JAVA ;

- il est extensible. Parce qu'il a été développé avec un langage orienté objet, il est facile de personnaliser certaines classes et fonctionnalités pour les besoins du projet ;

- il supporte plusieurs modèles de communications. En effet, MADKit supporte le standard ACL, établi par la FIPA 97 (Foundation For Intelligent Physical Agent), KQML (Knowledge Query Manipulation Langage) ;

- il permet le transfert entre agents de documents XML, de simples Strings, ainsi que d'Objects JAVA ;

- il est compatible avec JESS, un langage qui est déjà utilisé pour l'implémentation de nos systèmes experts.

3.2.2 Agents de MADKit

La logique des agents qui sont développés à partir de MADKit se retrouve sur la classe *Agent* et l'interface *AbstractAgent*. La classe *Agent* permet la gestion du comportement de ses descendants lorsque ces derniers sont activés, mis en

pause, lorsqu'ils attendent ou reçoivent des messages. L'interface *AbstractAgent*, quant à elle, offre toutes les fonctionnalités nécessaires à une gestion complexe de la messagerie entre agents.

Tous les agents créés avec MADKit sont sous la supervision de l'agent *Kernel.* Cet agent constitue le véritable noyau de tous les agents. Il effectue les tâches suivantes :

- *contrôle des groupes et rôles locaux :* Le Kernel est responsable de maintenir les informations correctes quant aux membres des groupes et des rôles gérés à même ces groupes. Il vérifie également si les requêtes faites aux structures des groupes et des rôles sont correctes;

- *gestion du cycle de vie d'un agent* : Le Kernel lance et tue les agents, et maintient les tables et les références des instances de ces agents. Il gère aussi les informations inhérentes aux agents et leur assigne, à la création, un identifiant global et unique, l'AgentAddress ;

- *passage de messages locaux* : Le Kernel gère le routage et la distribution de messages entre les agents.

La structure des agents utilisés est basée sur trois concepts, Agent/Groupe/Rôle, qui définit les relations entre les agents.

- **Agent** : Du point de vue de cette architecture, un agent est perçu comme une entité active communicante, qui joue un rôle au sein d'un groupe.

- **Groupe** : Un groupe est défini comme étant comme une agrégation d'agents. Chaque agent fait partie d'un ou plusieurs groupes. Dans sa

forme la plus primaire, le groupe permet de libeller un ensemble d'agents ayant un point ou un but commun, qu'ils soient en coopération ou en compétition. Le groupe est utile pour la diffusion de messages.

- **Rôle** : Le rôle est une représentation abstraite de la fonction d'un agent, d'un service ou d'une identification dans le groupe. Chaque agent peut posséder plusieurs rôles, et chaque rôle manipulé par un agent est local au groupe.

L'utilisation d'une librairie externe comme MADKit trouve sa justification dans le fait qu'elle permet d'épargner des heures considérables de programmation et qu'elle permet d'obtenir un prototype fonctionnel et performant du système. Si le produit est éventuellement commercialisé, il serait alors possible de développer nos propres librairies d'agents moyennant quelques semaines de travail et un investissement supplémentaire.

Chapitre 4

I3SOFT - INTERFACE ET FONCTIONNALITES

Dans ce chapitre, nous présenterons le logiciel I3Soft (Intelligent Image Interpretation Software), qui a été développé par l'équipe Estritel et qui est au cœur de ce projet de maîtrise. Nous y verrons d'abord l'analyse menée pour bien cerner les besoins et les fonctionnalités de ce système. Ensuite nous nous pencherons sur l'interface utilisateur qui constitue en fait le cœur du système.

La conception d'un logiciel de l'envergure d'I3Soft exige une grande rigueur à toutes les étapes. Pour éviter des erreurs inutiles lors de la phase de codage, une analyse détaillée est de mise au départ. Une bonne analyse doit habituellement se faire selon les trois points de vue suivants : fonctionnel, structurel et dynamique. Dans ce chapitre, nous ne nous intéresserons qu'au point de vue fonctionnel du système. L'architecture (donc l'analyse structurelle) sera vue dans le chapitre 5, et l'analyse dynamique, qui ne s'applique pas encore à ce stade-ci du projet, ne sera pas abordée. La méthodologie employée est UML parce qu'elle est maintenant le standard en vigueur en ce qui a trait aux systèmes programmés dans des langages orientés objet.

4.1 Analyse fonctionnelle

Cette partie est la plus importante du point de vue de l'utilisateur. Dans une telle analyse on se penche d'abord sur tous ses besoins, et de là on peut déterminer toutes les fonctionnalités que l'on devra incorporer pour répondre à ces besoins.

4.1.1 Cas d'utilisation

Cette portion de l'analyse est celle qui vient logiquement en premier lieu, car c'est elle qui illustre le système du point de vue de l'utilisateur. C'est dans cette partie que sont définies toutes les tâches que l'utilisateur pourra commander au système. Elle est directement liée au contenu des fenêtres et boîtes de dialogues qui apparaîtront à l'utilisateur lors d'une session.

Les diagrammes correspondants à cette partie s'appellent des diagrammes de cas d'utilisation (*use case*). Dans ces diagrammes (voir figure 6), les hommes allumettes sont des acteurs. Ils représentent tout ce qui interagit avec le système qui est construit. Les opérations, ou *use case,* illustrées dans des bulles sont des opérations simples et celles dans des rectangles sont des opérations complexes qui seront décomposées dans des diagrammes subséquents (qui ne seront pas illustrés ici). Les flèches simples reliant les acteurs aux opérations sont des relations de communication entre l'acteur et le *use case.* Il existe deux types de flèches existant entre les *use case.* Le premier type représente une relation d'utilisation. Il permet à une opération d'utiliser une fonctionnalité fournie par une autre opération. Ce type est représenté par une flèche avec le mot *uses*. L'autre type est une relation d'héritage qui ressemble à la relation d'utilisation. Elle permet optionnellement à une opération d'étendre la fonctionnalité d'une autre opération. Ce type est représenté par une flèche avec le mot *extends*.

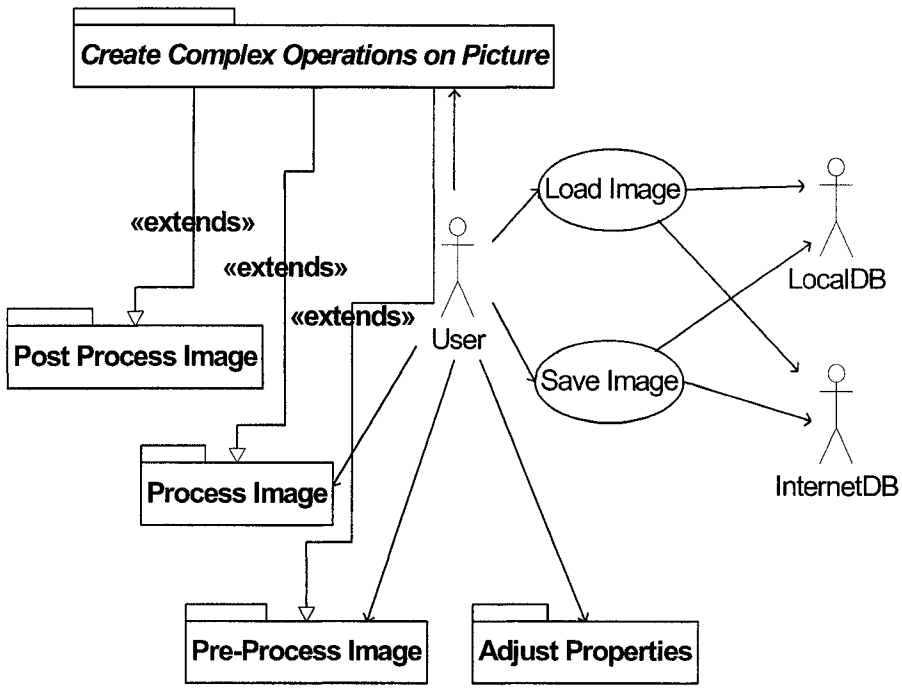


Figure 6. Diagramme général des fonctionnalités

Voici une description des tâches illustrées dans la figure 6.

Load Image : Charge une ou des images à partir d'une base de données locale. (Nous prévoyons la possibilité de le faire à partir d'une base de données sur Internet aussi.) Le chargement de l'image peut se faire de façon classique (ouverture d'un fichier dans un répertoire) ou de façon plus intelligente [K. Rousseau 2002]. Si l'utilisateur ne sait pas quelles images sont à sa disposition dans la base de données, il peut appeler l'agent de sélection de données qui lui fournira automatiquement les images nécessaires pour mener à bien sa requête.

Save Image : Sauvegarde d'une ou des images dans la base de données locale. (Nous prévoyons la possibilité de le faire sur Internet également.)

Pre-Process Image : Les pré-traitements sont des opérations effectuées sur l'image pour la clarifier et en éliminer le bruit. Ces opérations doivent être effectuées avant les opérations de traitement qui extraient des informations de l'image. Le post-traitement permettra d'extraire par la suite de l'information de la meilleure qualité possible.

Process Image : Ces traitements sont des opérations effectuées sur l'image pour en extraire de l'information. Ils constituent l'étape centrale de tout le processus de transformation d'images. C'est l'étape d'analyse des images.

Post Process Image : Ici sont regroupées toutes les fonctions qui permettent de choisir le format des informations que l'on désire, ainsi que le support utilisé.

La fonctionnalité indexée sur la figure 6 par « Create Complex Operations on Picture » regroupe les deux fonctionnalités suivantes :

Launch Automatic Process Selection : Cette fonctionnalité permet à l'utilisateur, lorsque celui-ci veut effectuer un traitement d'image, d'obtenir le meilleur algorithme possible pour satisfaire sa requête. Cette fonctionnalité est ce qui fait toute l'originalité de I3Soft et elle constitue le chapitre 6 de ce mémoire.

Create Project : Cette fonctionnalité n'est pas encore disponible sur I3Soft. La création d'un projet permettra à l'utilisateur débutant d'effectuer un traitement complexe, complet et adéquat sans en avoir nécessairement toutes les connaissances requises. Une opération si complexe nécessitera l'implémentation dans le système d'un planificateur de tâches. Si l'utilisateur crée un projet, le système planifiera d'avance toutes les étapes du traitement

pour extraire l'information finale voulue. Le système prendra bien soin d'expliquer la raison de chaque traitement et le choix de chaque paramètre à l'utilisateur et lui donnera la possibilité d'intervenir dans le processus et d'effectuer des changements s'il le désire. Car bien que le système puisse effectuer toutes les étapes d'un traitement complexe, nous voulons que l'utilisateur puisse conserver son autonomie.

4.1.2 Scénarios

Cette partie se concentre sur les interactions entre les composantes du système pour un traitement donné. C'est-à-dire que chaque opération que le système peut effectuer, peu importe sa complexité, se traduit par des échanges de signaux et d'informations entre les modules concernés, le tout étant ordonné dans le temps. Nous choisissons de décrire ces interactions par un diagramme de séquences.

4.1.2.1 Chargement d'images

Avant d'effectuer toute requête de traitement simple ou complexe, l'utilisateur doit d'abord charger la ou les images dont il aura besoin. Cette opération est simple pour un usager expérimenté et qui possède déjà les images nécessaires. Par contre si on est un utilisateur peu expérimenté et que l'on n'a pas d'idée précise du type d'image dont on aura besoin, ou encore si on ne connaît pas les images disponibles dans la base de données, alors une sélection d'images automatique s'avère plus utile (voir figure 7). Lors de ce procédé, l'utilisateur formule une requête à l'agent de sélection de données par l'entremise de l'interface. L'utilisateur peut fournir des paramètres (régions géographiques des images voulues, type de capteur, date de capture de l'image, résolution de l'image) pour filtrer les images disponibles dans la base de données. Le

système vérifie alors les images disponibles et renvoie à l'utilisateur leur adresse (sur CD). Si certaines images ne sont pas disponibles dans la BD pour satisfaire une requête, le système en suggère l'achat à l'utilisateur.

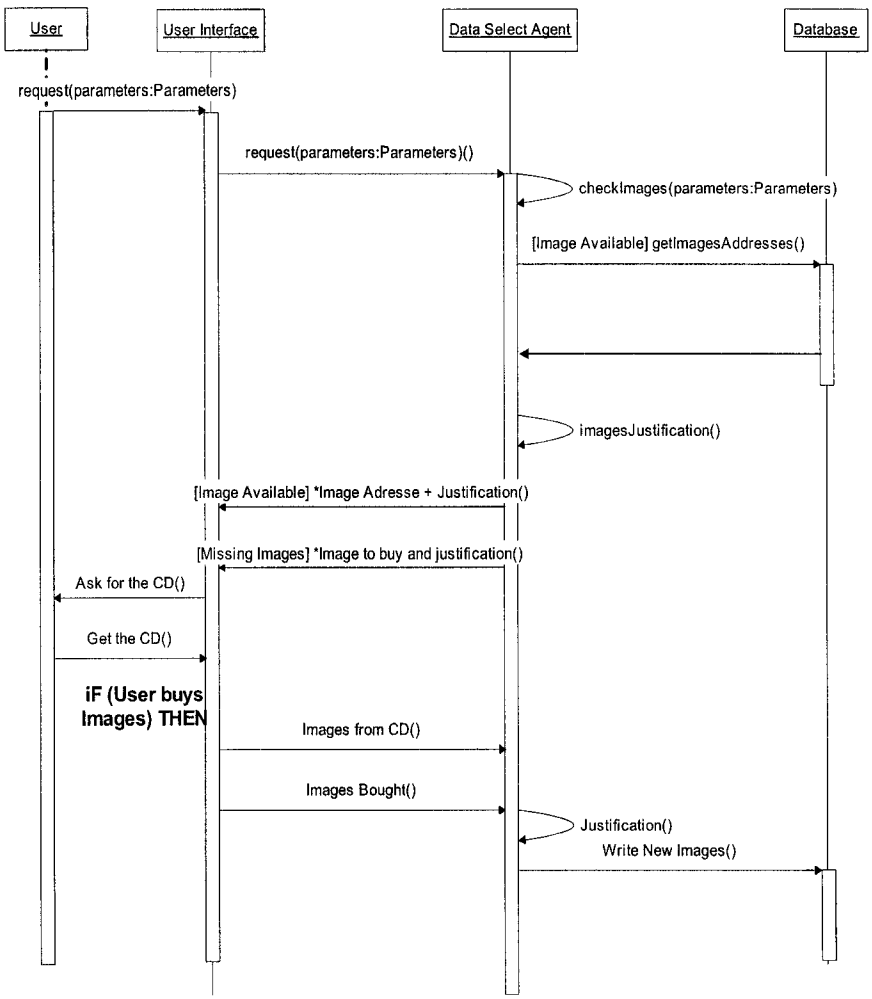


Figure 7. Scénario de sélection automatique de données

4.1.2.2 Traitement simple d'images

Une fois les images chargées, il est possible d'effectuer sur elles les divers traitements qu'offre I3Soft. Bien sûr chaque traitement est différent des autres et requiert un scénario qui lui est propre. Nous nous contenterons d'illustrer ce protocole de façon générale, c'est-à-dire avec les étapes communes à tous les types de traitements d'images disponibles dans I3Soft (figure 8). De plus, ce scénario s'applique aux cas où l'utilisateur effectue ses requêtes de façon classique, c'est-à-dire sans faire appel à un module d'aide à la décision.

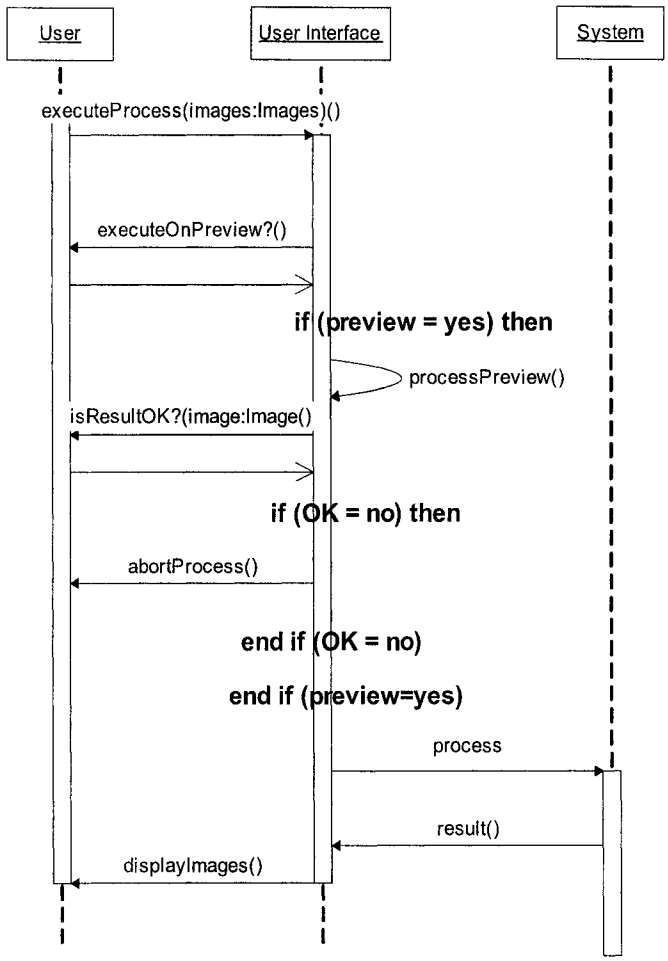


Figure 8. Scénario générique de traitement simple.

La figure 8 illustre une interaction entre l'utilisateur et le système via l'interface. Lorsque l'utilisateur effectue une requête de traitement sur une image, le système doit lui demander s'il veut d'abord effectuer ce traitement sur un échantillon de l'image et la raison est simple : vu que certaines images satellites sont gigantesques et que certains traitements sont longs, il est parfois préférable de vérifier partiellement le résultat d'un traitement avant de le lancer sur toute l'image. Toutefois ce protocole n'est pas encore implémenté.

4.2. Interface homme – machine

L'interface homme machine est le résultat de l'analyse des besoins et des fonctionnalités puisque toutes les fonctionnalités doivent être accessibles depuis cette interface. La règle d'or à suivre lors de l'implémentation de l'interface est la convivialité. Ceci est particulièrement vrai dans notre cas puisque notre logiciel s'adresse avant tout aux débutants. Une bonne interface est la clé du succès d'une application.

> « L'interface utilisateur n'est pas un simple glaçage sur un système informatique, contrairement à ce que pensent bien des informaticiens. L'interface, c'est le système. Tant du point de vue de l'utilisateur que du concepteur, l'interface est la pierre angulaire de tout système. On aura beau avoir un magnifique modèle de données selon la $X^{ème}$ forme, disposer d'un code réutilisable et compact, cela ne rendra pas nécessairement le système utilisable et utile à l'utilisateur dans la réalisation de sa tâche. » [D. Lafrenière, 1995]

L'interface principale de notre système a été mise sur pied pour être le plus simple et intuitif possible (figure 9). Il contient un menu donnant accès à toutes les fonctionnalités, une barre d'outils pour faciliter l'accès aux fonctionnalités les plus souvent utilisées, un espace d'affichage qui contient

toutes les images ouvertes et une barre d'état qui indique des informations pertinentes sur ces images.

Menu : Les fonctionnalités du menu sont regroupées selon leur ordre d'utilisation lors d'un procédé d'extraction et d'interprétation d'informations. Par exemple les menu *Pre-analysis*, *Analysis* et *Information representation* représentent respectivement les fonctions *pre-process*, *process* et *post-process* énumérés dans la section 4.1.1. Le menu *Project* représente la fonction *Create Process* de la même section. Le menu *help* contient les informations disponibles sur le système.

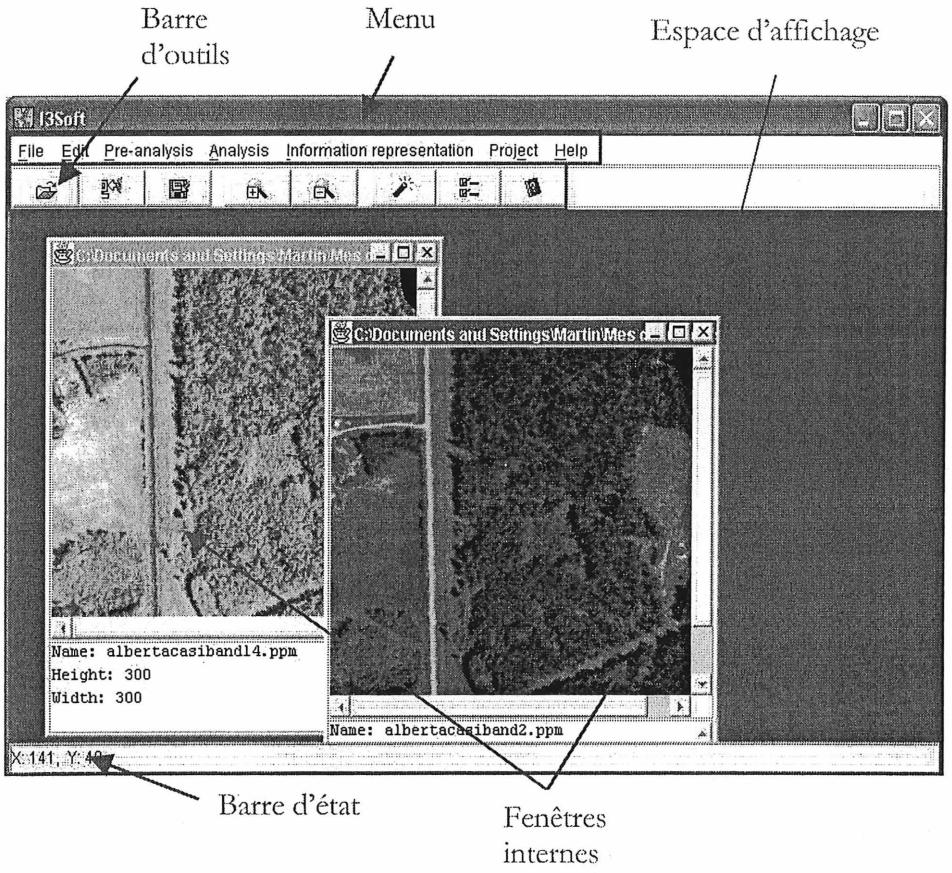


Figure 9. Interface principale d'I3Soft.

Barre d'outils : La barre d'outils contient un raccourci sur toutes les fonctionnalités du menu qui sont utilisées de façon plus courante. Ces fonctionnalités sont : *Open file* (chargement d'image de façon classique), *Import file* (chargement d'image assisté [K. Rousseau 2002]), *Save file* (sauvegarde de l'image), *Zoom in* et *Zoom out* (agrandissement et réduction de la vue de l'image), *Wizard* (lance le module d'aide à la décision expliqué dans le chapitre 6), *Properties* (fonctions de configuration du système) et *Help* (informations pertinentes du système).

Espace d'affichage : Cet espace contient toutes les images ouvertes lors d'une session. De cette façon si on minimise l'interface principale parce qu'on veut travailler sur une autre application, nous n'avons pas à également minimiser toutes les images.

Fenêtre interne : Chaque fenêtre interne contient une image ainsi que les informations pertinentes relatives à cette image : nom, taille, traitements effectués à ce jour sur cette image, etc.

Barre d'état : Cette barre indique des informations relatives à la position du curseur sur l'espace d'affichage, comme la position du pixel sur une image.

Ces éléments ont pour but de faciliter l'accès aux opérations et aux informations relatives aux images sur lesquelles l'utilisateur travaille.

Chapitre 5

I3SOFT - ARCHITECTURE INTERNE

Dans ce chapitre, nous verrons en premier lieu la structure du système. Nous étudierons les arguments qui nous ont poussés vers une architecture multi-agents. Nous discuterons ensuite de ces agents et de leur rôle dans cette structure. Nous verrons finalement de quelle façon ces agents interagissent ensemble pour en arriver à prendre une décision quant au choix d'un traitement à effectuer sur une image.

5.1 Structure globale

Un des objectifs du projet I3Soft est d'intégrer sous une seule application les travaux que plusieurs étudiants du Cartel ont menés dans le domaine du traitement d'images. Comme la plupart de ces travaux ont été développés dans des langages orientés objet (C++ et Java) et puisque l'intégration de plusieurs blocs de code est facilitée dans un système d'architecture modulaire, le choix du langage utilisé doit être orienté objet. Puisque nous envisageons également de faire évoluer le système sur Internet, le choix de Java s'est donc naturellement imposé comme solution pour développer le cœur du système.

La figure 10 montre notre système dans son ensemble au niveau des différents modules qui le composent. Il y a d'abord l'interface utilisateur, les outils, le système expert d'interprétation, les opérations, qui incluent les algorithmes de segmentation vus au chapitre 1 et le module d'aide à la décision.

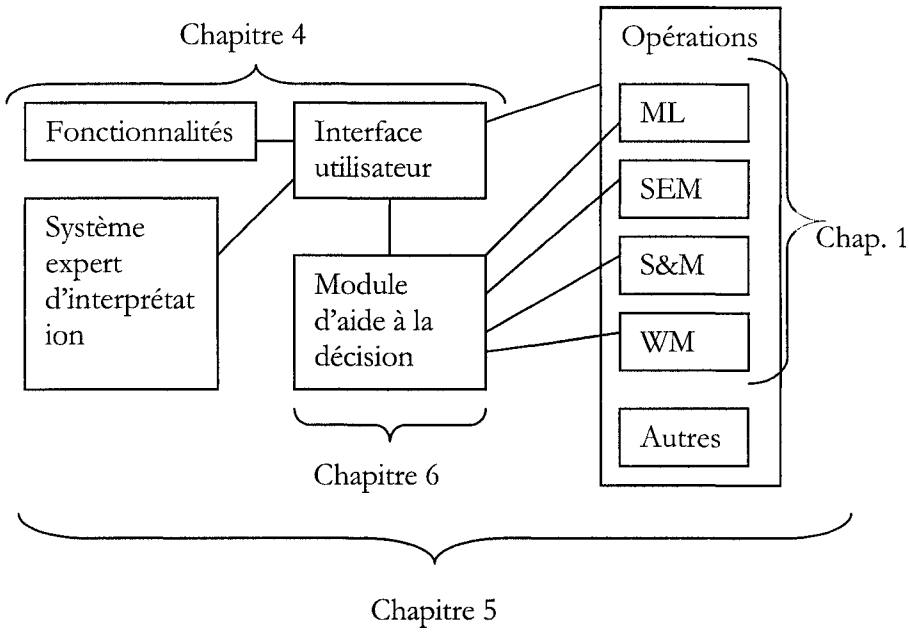


Figure 10. Structure générale d'I3Soft

Comme expliqué dans les chapitres précédents, I3Soft est une application destinée autant aux débutants qu'aux experts en télédétection. De ce fait, nous avons établi comme mandat de lui conférer un caractère intelligent. Plus précisément, nous avons convenu de lui greffer un module spécial qui pourra aider l'utilisateur débutant à choisir les traitements adéquats sur les images satellitaires pour en extraire l'information la plus précise possible. Concrètement, ce module est celui dont nous avons parlé dans l'introduction et qui viendra en aide au forestier qui voudra effectuer sa mise à jour d'une carte forestière.

En ce qui a trait à l'architecture que devrait avoir ce module d'aide à la décision, notre analyse nous a permis d'envisager deux alternatives possibles : le système expert et le système multi-agents. Un système expert est une entité

logicielle qui permet de conseiller, de fournir une expertise dans un domaine donné à une personne, un peu de la même façon qu'un expert dans un tel domaine pourrait le faire. Un système expert est constitué d'une base de connaissances, qui contient toutes les informations spécifiques au domaine. Il est également constitué d'un moteur d'inférence qui contient des règles permettant de répondre aux requêtes des usagers.

Dans notre exemple de l'introduction, le scénario pourrait être le suivant ; Un forestier qui veut obtenir la mise à jour d'une carte forestière fait la requête au système. Il indique qu'il possède la carte forestière et des images Landsat de 1989 du Labrador, et d'autres images Landsat plus récentes de cette région. Pour un système expert, la base de connaissances contiendrait des informations comme :

- une image Landsat est une image optique ;
- une image Landsat de 1989 possède une résolution de 10 mètres ;
- le Labrador est une zone forestière ;
- une zone forestière prise sur une image Landsat est texturée ;
- l'algorithme ML donne de bons résultats avec des images Landsat ;
- l'algorithme SM ne donne pas de bons résultats avec des zones forestières.

Le moteur d'inférence permettrait de déduire de nouveaux faits à partir de la requête et de la base de connaissances, les faits suivants :

- les images fournies sont de Landsat 1989, donc elles possèdent une résolution de 10 m ;

- la région est le Labrador, donc les images sont d'une zone forestière ;

- les images prises sont d'une zone forestière et de Landsat, donc elles sont texturées ;

- les images sont de Landsat, donc ML donnera de bons résultats ;

- les zones sont forestières, donc SM ne donnera pas de bons résultats.

Donc dans ce cas-ci, le système expert pourrait fort bien suggérer à l'usager d'effectuer une segmentation ML sur ses images.

Comme nous l'avons déjà vu au chapitre 3, le système multi-agents est, quant à lui, constitué de plusieurs agents (sous forme de classes) qui ont chacun leur tâche simple à effectuer et qui, lorsqu'ils travaillent de pair, peuvent réaliser des tâches plus complexes comme la prise de décision. Dans le cas d'une telle architecture, la requête du forestier est d'abord prise en main par un agent client, qui assure la communication entre le forestier et le système. Le forestier indique à cet agent client qu'il désire obtenir la mise à jour d'une carte forestière du Labrador et qu'il possède la carte forestière et des images de 1989 du Labrador, et d'autres images plus récentes de cette région. Le forestier fournit à l'agent d'autres renseignements, comme :

- le traitement voulu est une segmentation ;

- le temps d'exécution de cette segmentation ne doit pas être trop long ;

- les images sont de Landsat, d'une zone forestière et sont plutôt texturées ;

- l'image résultante doit comporter cinq classes distinctes.

L'agent client va alors acheminer cette requête à un autre agent dit courtier (ou *broker*). Ce *broker* réveillera tous les agents concernés par cette requête, c'est-à-dire dans ce cas-ci tous les agents pouvant effectuer une segmentation, et il leur diffusera ensuite la requête avec toutes ses contraintes. Dans cette architecture, il y a un agent pour chaque algorithme de segmentation présenté au chapitre 1. Il y a donc quatre agents de segmentations (pour les algorithmes de ML, SEM, S&M et WM) que nous appelons agents fournisseurs. Ces agents contiennent une base de connaissances de l'algorithme qu'ils contrôlent. Ils connaissent les caractéristiques, les propriétés, les forces et les faiblesses de cet algorithme. Par exemple, l'agent ML, en recevant la requête, observe les contraintes et formule les points suivants :

- le temps d'exécution de cette segmentation ne doit pas être trop long, or nous avons vu dans le chapitre 1 que la durée d'exécution de ML était proportionnelle au carré de la taille de l'image. Comme cette information est disponible pour les agents, on s'en servira pour déterminer si ML est efficace. Si l'image est de petite taille, ce critère sera favorable à ML ;

- les images sont de Landsat, d'une zone forestière et sont plutôt texturées. Or, l'agent ML sait que l'algorithme ML est efficace avec les images optiques Landsat. Ce critère sera donc aussi favorable à ML ;

- l'image résultante doit comporter cinq classes distinctes. Comme ML est une méthode de segmentation supervisée, c'est-à-dire où l'on peut indiquer à l'avance le nombre de classes que l'on veut obtenir, ML sera également

avantagé ici par rapport aux autres méthodes de segmentation non supervisées.

Pour chaque critère précisé, chacun des quatre agents fournisseurs calculera sa capacité à le satisfaire. Cette capacité se traduira par un pointage, et la somme des critères par la somme des pointages. Ces agents renverrons alors leur pointage respectif au *broker,* qui sélectionnera l'agent au pointage le plus élevé.

Pour déterminer laquelle de ces architectures nous devions adopter, nous les avons évaluées sur les contraintes que nous nous étions imposées sur le système, c'est-à-dire la modularité et la performance. Nous avons également tenu compte du fait que le système devra éventuellement héberger un planificateur de flux d'images. Est-ce que l'utilisation d'une architecture multi-agents est avantageuse lorsqu'on la compare avec un système expert qui pourrait effectuer les mêmes fonctionnalités? Voici quelques points exposant les différences entre les deux méthodes ainsi qu'un tableau récapitulatif des comparaisons (tableau 3).

Modularité : Le système multi-agents est avantagé sur ce plan en ce sens que chacun des agents est une entité bien distincte. Cette caractéristique est avantageuse pour plusieurs raisons. Tout d'abord on prévoit éventuellement distribuer ce système sur plusieurs machines, et si tout le code est assemblé en un seul bloc comme avec un système expert, le passage à un système distribué sera très ardu et nécessitera un remaniement majeur du code. Un autre avantage que procure cette modularité est la facilité de modification du code. Par exemple si un développeur veut éventuellement greffer un nouvel algorithme de traitement au système, tout ce qu'il doit faire, c'est y ajouter un agent de traitement, car chaque traitement possède déjà son propre agent. Par

ailleurs avec un système expert, l'ajout d'un algorithme nécessite de modifier ou ajouter certaines règles de la base de connaissances. Si cette base de connaissances est de grande taille, cette opération risque d'être complexe et ardue. De plus, comme nous le verrons plus loin le système multi-agents que nous avons choisi a une architecture orientée objet, et permet donc une meilleure homogénéité avec le reste de I3Soft qui est aussi orienté objet.

Performance : Cet aspect est un autre avantage de l'architecture multi-agents par rapport à un système expert. De par leur nature, les agents héritent de toutes les fonctionnalités inhérentes aux *threads*. Ceci a pour conséquence que le système permet l'exécution en parallèle de plusieurs calculs et traitements. Par exemple, lorsqu'une requête utilisateur sera envoyée aux agents de traitement, chacun d'eux effectuera son traitement de l'information simultanément aux autres. Un système expert obtient des performances plus modestes pour plusieurs raisons. Tout d'abord JESS est un moteur d'inférence relativement lent. Ensuite l'exécution des traitements et des calculs se fait de façon séquentielle.

Compatibilité au planificateur : Un autre point très important unissant les deux caractéristiques précédentes et qui procure un avantage considérable à un système multi-agents apparaîtra lors de l'élaboration du planificateur de flux d'image (*image-processing flows planner*). Ce planificateur aura pour tâche de trouver une séquence optimale de traitements de bas niveaux à effectuer sur une image ou une série d'images pour satisfaire à une requête de traitement complexe de haut niveau. Par exemple, pour satisfaire la requête du forestier qui veut obtenir la mise à jour d'une carte forestière, le planificateur préparera une séquence d'opérations, allant des filtres et des corrections à l'interprétation, en passant par la segmentation et la classification des images

fournies, le tout en prenant soin de choisir les opérations les plus appropriées pour ces images. S'il y a des traitements à effectuer sur plusieurs images différentes, le planificateur pourrait alors lancer des traitements en parallèle pour optimiser le temps de traitement (dans le cas d'un système distribué). De par sa nature modulaire, un système multi-agents permettra un séquençage plus facile des traitements de bas niveau. Il est également à souligner que chaque agent de traitement connaît les caractéristiques, les forces et les faiblesses qui lui sont propres (comme vu au chapitre 1 pour les agents de segmentation). Ceci simplifiera énormément le planificateur, car autrement ce dernier aurait été forcé de tenir compte lui-même de toutes ces données. Lorsque la planification suggérera plusieurs traitements simultanés, encore une fois, c'est l'architecture multi-agents qui marquera des points en permettant une telle simultanéité, contrairement à un système expert, qui ne le permet pas.

Tableau 3. Comparaison d'un système multi-agents avec un système expert

	Système multi-agents	Système expert
Modularité	Oui	Non
Performance	Bonne	Moyenne
Compatibilité au planificateur	Oui	Non

Nous avons décidé d'adopter un système multi-agents parce qu'il répondait mieux qu'un système expert aux objectifs à remplir et aux contraintes imposées. Puisqu'un système multi-agents servira de guide à l'utilisateur

quant au choix des traitements à effectuer, nous pouvons le situer dans le système comme une entité servant d'intermédiaire entre l'utilisateur et les fonctionnalités. La figure 11 situe un système multi-agents dans l'architecture de l'application et nous indique l'apport de ce module comparativement à une application classique.

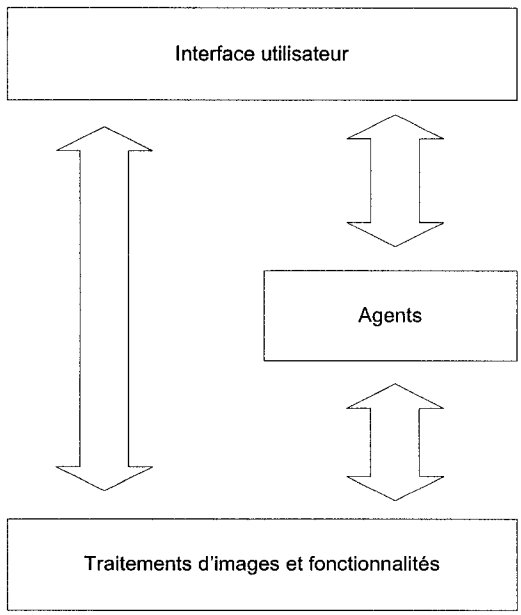


Figure 11. Le système multi-agents dans l'architecture de l'application

La figure 12 schématise le système de façon un peu plus détaillée. On y voit le système au niveau de l'implémentation. Puisque nous avons choisi un langage orienté objet, la structure du diagramme reposera sur la notion de classes. Bien que statique, ce genre de diagramme met l'emphase sur les modules du système et les relations entre eux. Les classes du diagramme doivent être vues comme des boîtes noires avec pour seule partie visible les entrées et les sorties.

La façon dont la programmation et la structure sont faites à l'intérieur de chaque boîte importe peu pour d'autres personnes que le développeur.

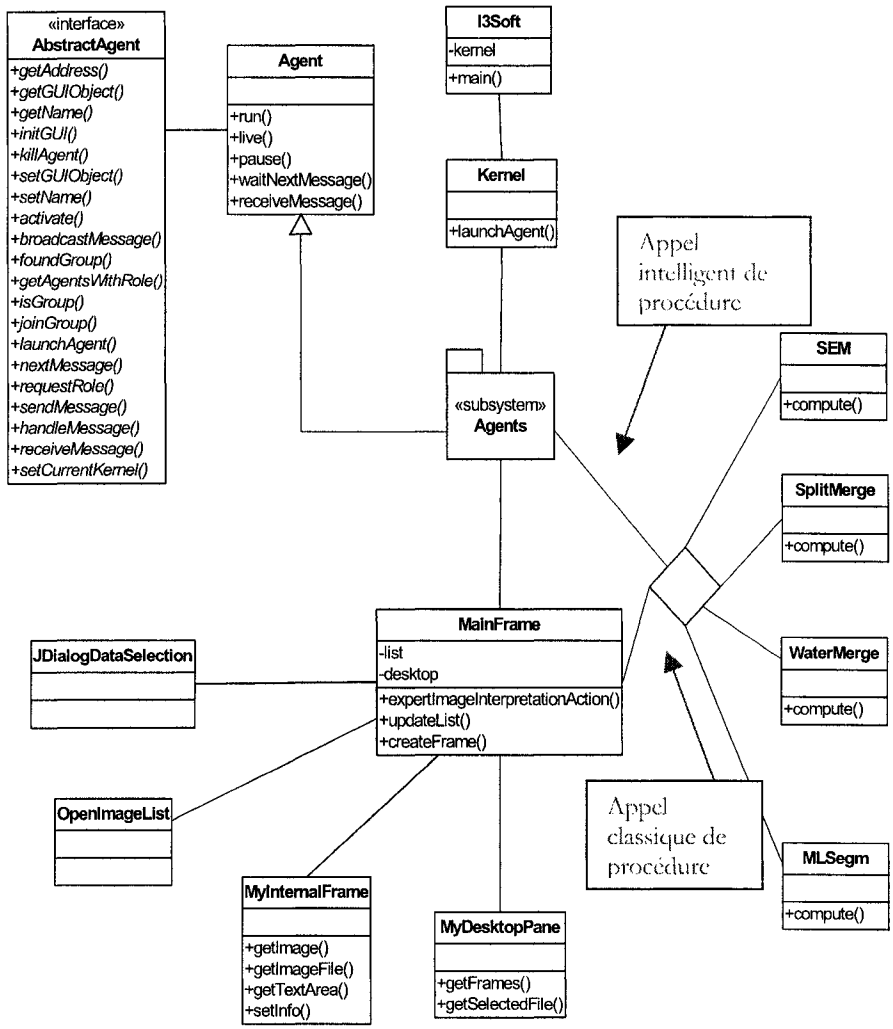


Figure 12. Classes principales du système

Le lancement de l'application se fait à partir de la classe I3Soft. On y crée un noyau, qui lancera la structure d'agents. Cette partie sera vue en détail dans la section suivante. Cette structure d'agents se chargera entre autres de lancer l'interface utilisateur principal (classe *MainFrame*) et d'y observer les

comportements de l'utilisateur. C'est à partir de ce *MainFrame* que l'utilisateur pourra effectuer toutes les fonctionnalités énumérées dans le chapitre précédent.

On observe à la droite de la figure 12 les fonctions de traitement d'images qui ont été implémentés : *SEM*, *SplitMerge*, *WaterMerge* et *MLSegm*. Ces procédures peuvent être accédées et exécutées d'une façon classique directement via le *MainFrame*, ou bien encore via l'architecture d'agents, qui apporte en plus une aide à la décision.

5.2 Architecture d'agents

Tous les traitements et les fonctionnalités inhérents au système sont des outils qui existent déjà dans la plupart des logiciels de traitement d'images disponibles sur le marché (voir chapitre 1). Il est possible d'exécuter ces fonctionnalités en codant des appels simples de fonctions de traitement d'images. Pour l'utilisateur, ces procédures sont habituellement appelées à partir du menu ou de boutons sur l'interface. Cette méthode fonctionne, certes, mais elle ne répond pas aux objectifs de notre projet. Ce qui distingue I3Soft des autres logiciels, ce qui fait sa force et son originalité, c'est son module d'aide à la décision. Ce module est constitué d'une architecture multi-agents, dont chaque agent est une entité qui peut communiquer avec les autres agents et qui peut accomplir une ou des tâches bien définies. Lorsque ces agents travaillent conjointement, il en résulte un comportement considérablement intelligent.

L'architecture du système multi-agents fonctionne de pair avec l'interface, c'est-à-dire que chaque fenêtre, chaque boîte de dialogue permettant des échanges entre l'usager et le système sera gérée par un agent. Donc le système

multi-agents ainsi que l'interface qu'il contrôle constituent le noyau du système.

5.2.1. Présentation des agents

Nous avions constaté dans la figure 12 que tous les agents (inclus dans la boîte « subsystem » agents), excepté le noyau (ou *Kernel*), sont des descendants de la classe *Agent* qui, elle, implémente l'interface *AbstractAgent,* toutes deux faisant partie de la librairie MADKit et définies au chapitre 3. Pour mettre sur pied le module d'aide à la décision nous avons créé, à partir de cette librairie MADKit, un certain nombre d'agents ayant chacun leur tâche précise et qui, dans l'ensemble, font émerger un comportement intelligent (figure 13). Voici une description de ces différents agents que nous avons créés.

Agent Client : Cet agent gère une interface utilisateur (figure 14), qui permettra l'entrée de données dans le système, comme les images sur lesquelles l'utilisateur veut effectuer un traitement et les données rattachées à chaque image, mais également le type de traitement à effectuer et les contraintes liées à ce traitement (temps d'exécution, qualité du résultat.)

Agent Broker : Cet agent s'occupe de la messagerie entre chaque agent client et de tous les agents de traitement qui peuvent satisfaire leurs requêtes. Lorsque le meilleur agent de traitement est trouvé pour effectuer la requête d'un client, le broker les unit par un contrat.

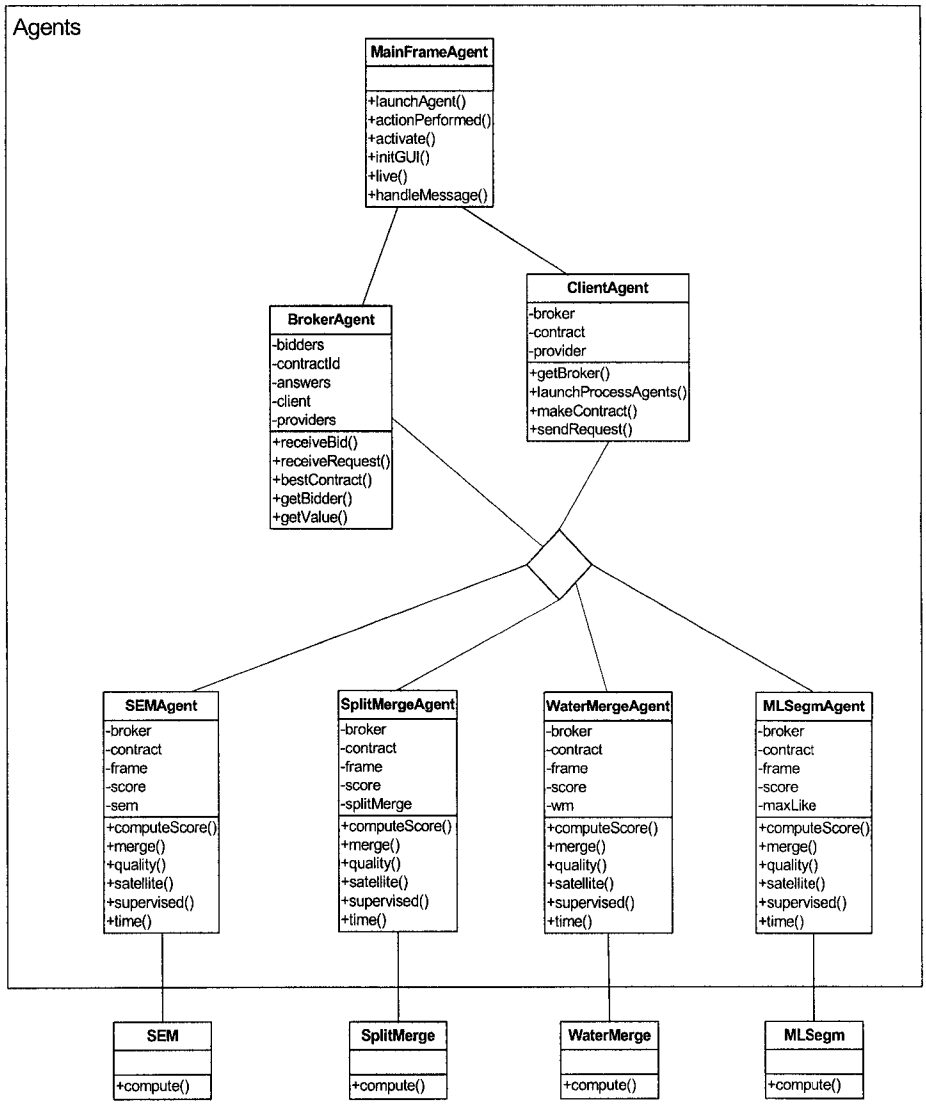


Figure 13. Liens, attributs et fonctionnalités des agents

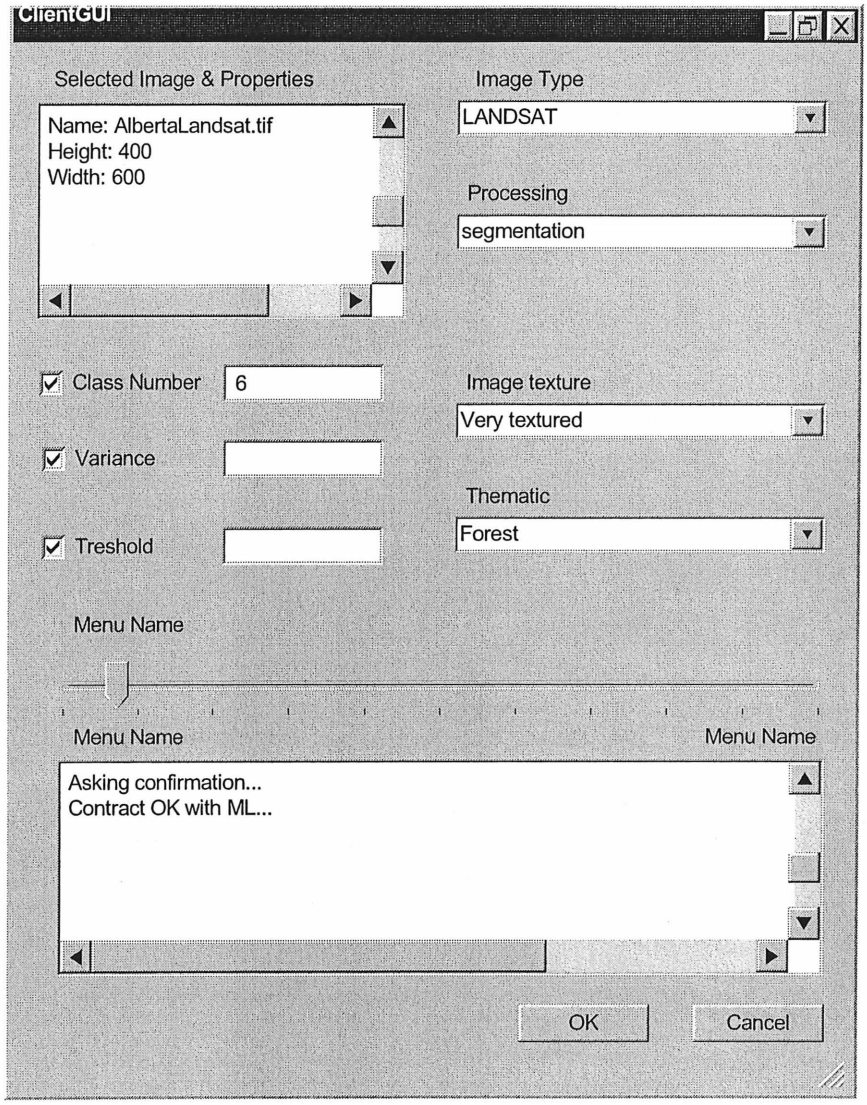


Figure 14. L'interface de l'agent client

Agent de Traitement : Il existe dans un système non seulement un agent pour chaque type de traitement, mais bien un pour chaque façon d'effectuer chaque traitement. Par exemple si l'on prend le cas de la segmentation, on peut l'effectuer avec différents algorithmes qui différeront selon le temps de

traitement et la qualité du résultat. L'agent connaît pour chaque contrainte qu'on lui envoie en entrée sa capacité à respecter cette contrainte.

Les agents de traitement développés jusqu'à maintenant sont au nombre de quatre et chacun implémente un des algorithmes de segmentation vus au chapitre 1. Chacun de ces agents commande l'exécution de la méthode du même nom par l'entremise de la classe du même nom. Par exemple, l'agent *MLSegmAgent* exécutera, s'il est sélectionné, une segmentation du type *Maximum Likelihood* présente dans la méthode *compute* de la classe *MLSegm*.

5.2.2. Groupes et rôles

Notre système d'agents comporte trois groupes principaux. Il y a tout d'abord le groupe process-client, qui contient tous les agents ayant une interaction avec le ou les utilisateurs. En second lieu, nous retrouvons le groupe process-provider, où se trouvent tous les agents reliés aux traitements d'images. Enfin, le groupe contract-X qui est créé lorsqu'une entente survient entre un client et un traitement. Autrement dit si un client lance une requête et que le module lui suggère par exemple l'algorithme du « maximum de vraisemblance (*Maximum Likelihood*) », l'agent *Broker* ajoutera l'agent *Client* et l'agent *MLSegm* au groupe *contract-1* s'il s'agit du premier contrat de la session d'utilisation, au groupe *contrat-2* s'il s'agit du deuxième contrat et ainsi de suite (figure 15).

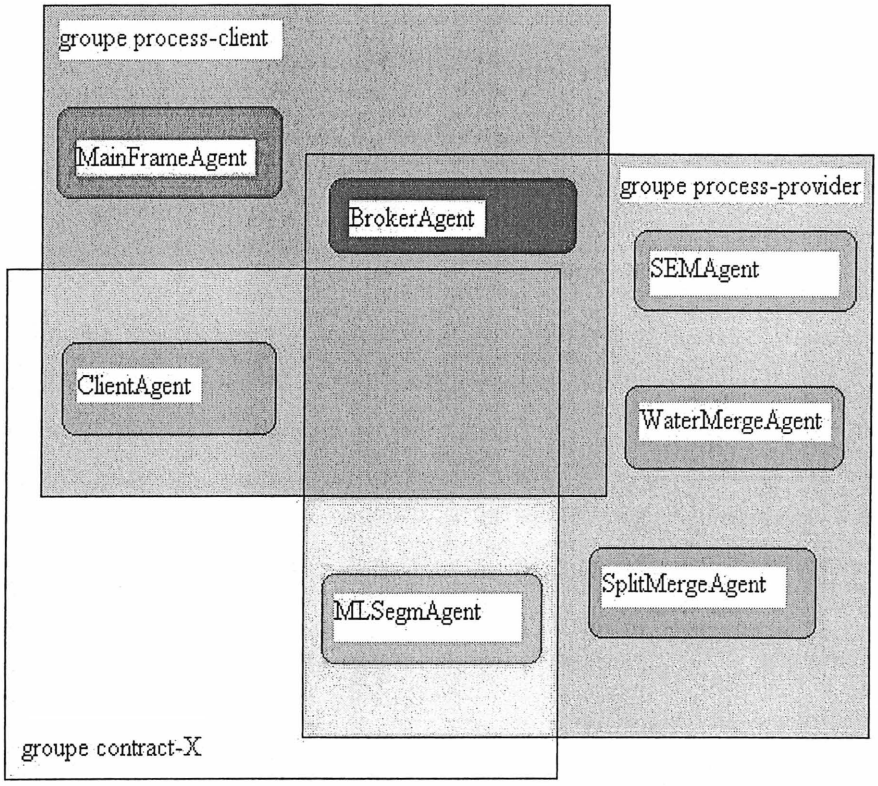


Figure 15. Les agents et leur appartenance aux groupes

Chapitre 6

LE MODULE D'AIDE À LA DÉCISION

Ce chapitre constitue une combinaison des deux chapitres précédents. Nous y verrons d'abord de quelle façon le système multi-agents peut parvenir à effectuer ses choix de traitements en fonction des règles que nous lui avons soumises. Ces règles sont élaborées à partir des caractéristiques de chaque type de traitement. Nous vérifierons ensuite les résultats obtenus.

6.1 Le mécanisme de prise de décision

La prise de décision est bien souvent un procédé complexe. Chaque fois que nous devons faire un choix, nous devons tenir compte de chaque facteur qui influence ou qui sera influencé par ce choix. De plus, nous devons tenir compte du degré d'influence ou d'importance de chacun de ces facteurs. Si l'on veut automatiser une prise de décision il est important d'adopter un certain formalisme pour bien poser le problème. Vient ensuite une analyse rigoureuse de ce problème pour implanter les mécanismes qui effectueront un choix avec les bons algorithmes.

6.1.1. Scénario d'utilisation

La figure 16 fait état des communications qui s'effectuent entre les agents lors d'une requête utilisateur. La requête est d'abord acheminée à l'agent client via son interface utilisateur. L'agent client achemine alors la requête au broker. Ce dernier observe les images, les données qui y sont liées, le type de traitement voulu, ainsi que les contraintes exigées de l'utilisateur et achemine la demande

à tous les agents de traitement qui sont susceptibles d'effectuer la tâche demandée. Par exemple, si l'utilisateur veut effectuer une segmentation sur une image, le broker diffuse un message à tous les agents faisant partie du groupe « process-provider » avec le rôle de « segmentation ». Chacun de ces agents de segmentation examine alors les contraintes du message reçu et crée un tableau des résultats dans lequel chaque contrainte est associée à une cote exprimant la capacité de l'agent de satisfaire cette contrainte. L'agent de segmentation calcule ensuite un pointage pour chacun des paramètres reçus en entrée. Chacun de ces pointages est basé sur la capacité de l'agent de segmentation à respecter chaque contrainte. Ces contraintes feront l'objet de la prochaine section.

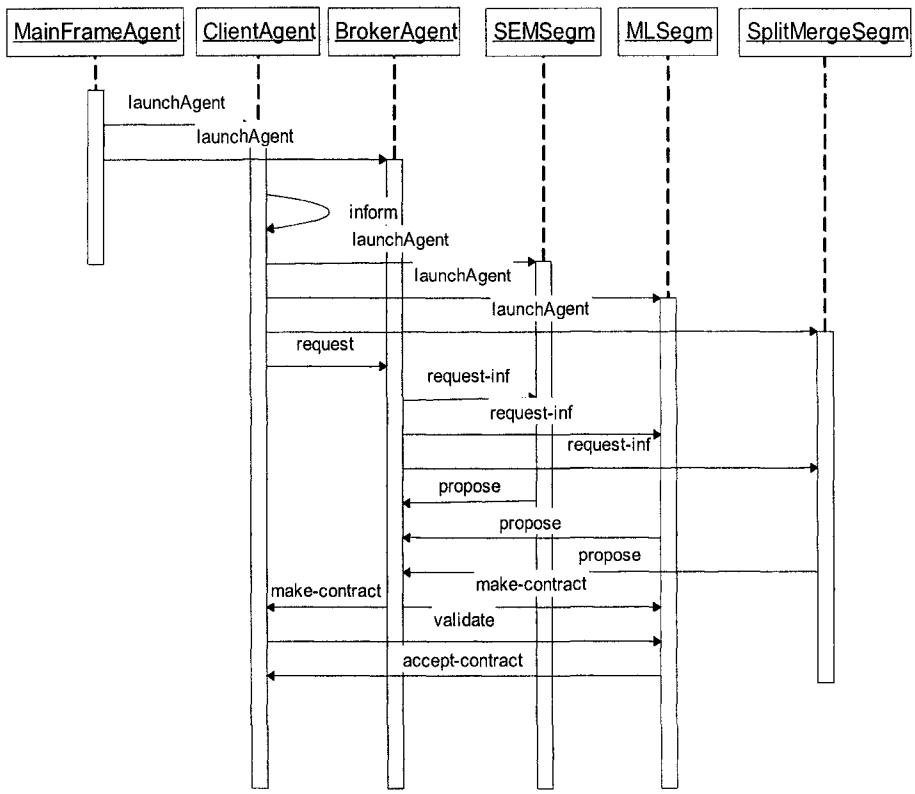


Figure 16. Scénario général de communication entre agents

Suite à ces calculs, chaque agent de segmentation parvient à un certain pointage qu'il propose au broker. Ce broker compare donc ces pointages et retient le plus élevé d'entre eux. Il envoie ensuite une validation au client et à la segmentation choisie et il les ajoute à un groupe commun de contrat. Suite à quoi le client demande une confirmation à la segmentation, qui lui renvoie.

6.2. Règles

Pour calculer leur pointage quand une requête utilisateur leur parvient, les agents de segmentation ont besoin de se baser sur des règles. Il y a beaucoup de variables qui interviennent dans le processus et qui permettent de départager les concurrents.

Il y a tout d'abord les paramètres que l'utilisateur veut lui-même introduire dans l'algorithme. Par exemple, il peut décider d'imposer le nombre de classes voulu dans le résultat d'une segmentation, ou encore la variance minimale nécessaire pour délimiter deux régions. Ces paramètres peuvent influencer le choix de la segmentation, car certains algorithmes donnent de meilleurs résultats avec ces données et d'autres pas. Il y a ensuite des contraintes que l'utilisateur peut imposer à la segmentation, comme la durée de son exécution, et la qualité de l'image résultante. Enfin, il y a les informations que nous fournit directement l'image, comme sa taille, sa texture, sa résolution et également le milieu représenté sur l'image (urbain, forestier).

Le tableau 4 présente tous les paramètres qui entreront dans le calcul du module d'aide à la décision pour choisir la meilleure segmentation à effectuer. La formule est simple. Lorsque la requête utilisateur est acheminée aux agents de segmentation, ces derniers prennent tous les paramètres de la requête et ils

Tableau 4. Paramètres de calcul des algorithmes de segmentation

		ML		SEM		S&M		WM	
		Pd*	Pts**	Pd	Pts	Pd	Pts	Pd	Pts
Type de satellite	Inconnu	0,3	1	0,3	0,5	0,2	0,5	0,2	0,5
	CASI		1		1		0,5		0,5
	IKONOS		1		1		0,5		0,5
	LANDSAT		1		0,8		0,5		0,7
	RADARSAT		0		0		0,7		0,7
	SPOT		1		0		0,7		0,7
Nombre Classes	Vrai	0,3	1	0,3	1	0,1	0,3	0,1	0,3
	Faux		0		0		0,7		0,7
Variance	Vrai	0		0		0,2	1	0	
	Faux						0		
Seuil	Vrai	0		0		0		0,3	1
	Faux								0
Thématique (zone)	Forestière	0,2	0,6	0,2	0,7	0,3	0	0,2	1
	Urbaine		0,4		0,3		1		0
Temps		0,4	f(Θ(ML))	0,4	f(Θ(SEM))	0,4	f(Θ(S&M))	0,4	f(Θ(WM))

* Pondération

** Points

effectuent la somme des produits du pointage de chaque paramètre par la pondération qui lui est assignée.

Voici un exemple : Un utilisateur veut effectuer une segmentation sur une image forestière Landsat. Il sait qu'il veut obtenir six classes, alors il l'indique dans sa requête via l'agent client (voir figure 14). Il indique également que l'image à traiter est une image Landsat plutôt texturée, d'un milieu forestier et que le traitement ne doit pas être trop long. Cette requête est alors acheminée vers tous les agents de segmentation (*MLAgent*, *SEMAgent*, *S&Magent* et *WMAgent*) et chacun de ceux-ci calcule un pointage. Par exemple, *MLAgent* calcule 0,3*1=0,3 pour le type de satellite, 0,3*1 pour le nombre de classes, 0,2*0,6 pour la thématique forestière et 0,4* f(Θ(ML)) pour le temps d'exécution. Il additionne tous ces pointages pour obtenir un pointage global *PG*.

$$PG(ML) = 0,3*1 + 0,3*1 + 0 + 0 + 0,2*0,6 + 0 + 0,4*f(\Theta(ML))$$

Il fait alors parvenir ce *PG* au broker qui le compare avec les PG des autres agents de segmentation pour déterminer le vainqueur.

En ce qui a trait au temps, le pointage est calculé en fonction des formules de durée (voir tableau 1). Par exemple pour *MLAgent*, on sait que la durée d'exécution de l'algorithme est de l'ordre de

$$\Theta(ML)=C_1+xy/s(C_2nk + C_3nsk^2+C_4nkxy/s)$$

Or, comme on le constate le terme le plus significatif est $C_4nk(xy/s)^2$. La durée est proportionnelle au nombre d'itérations *n*, au nombre de classes *k* (=6) et *au carré* de la taille de l'image *xy* divisé par la distance entre les pixels d'échantillons. On peut donc conclure que l'algorithme sera beaucoup plus long si cette image Landsat est de grande taille. Dans ce cas, si l'utilisateur a

spécifié un temps court de traitement et que la taille de l'image est grande, alors la fonction de formule de durée $f(\Theta(ML))$ renverra une plus petite valeur.

Un autre critère significatif pour déterminer les temps des traitements peut être la mesure de la texture de l'image. En effectuant une détection de contour sur l'image traitée, nous pouvons quantifier cette texture et ainsi déterminer un nombre de régions seuil au delà duquel un algorithme dépendant du nombre de régions comme le WM deviendrait désavantagé.

Chapitre 7

RÉSULTATS

Ce chapitre présente les résultats que nous avons obtenus avec l'utilisation de notre module d'aide à la décision. Nous y verrons en détail quelques cas d'utilisation et les réponses du système pour ces cas. Le nombre de combinaisons de paramètres différents que l'usager peut entrer dans une requête au système est presque infini, donc il est impensable d'étudier tous les cas possibles. Nous ne nous sommes concentrés que sur un échantillon de possibilités de requêtes. Les paramètres concernés dans une requête sont ceux que nous avons vus dans le tableau 4. Nous ferons des tests comparatifs du point de vue des critères suivants :

- nature de l'image (optique VS radar) ;
- thématique (forêt VS urbain) ;
- temps d'exécution.

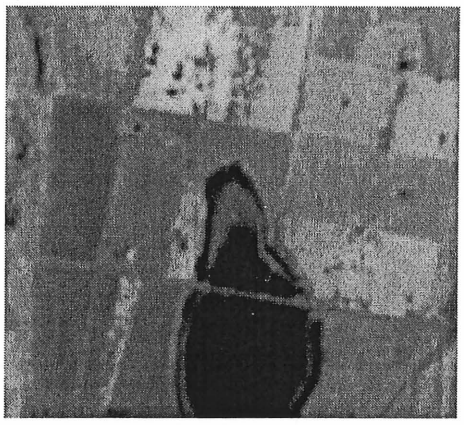
Reprenons l'exemple du traitement d'une image optique Landsat du Labrador du chapitre 1 (figure 4). Nous y avions vu que ML était performant pour ce type d'image, que WM donnait aussi de bons résultats, que SEM était précis mais comportait quelques lacunes et que SM était plutôt déconseillé. Nous avons lancé dans le système une requête typique pour une image optique et nous avons obtenu les résultats présentés dans le tableau 5.

Tableau 5. Résultats avec image optique

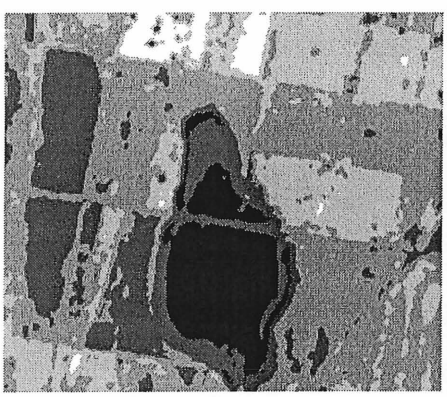
	ML	SEM	S&M	WM
Satellite LANDSAT	0.3*1 = 0.3	0.3*0.8 = 0.24	0.2*0.5 = 0.1	0.2*0.7 = 0.14
Nombre de classes est connu	0.3*1 = 0.3	0.3*1 = 0.3	0.1*0.3 = 0.03	0.1*0.3 = 0.03
La variance est connue	0	0	0.2*1 = 0.2	0
Le seuil est connu	0	0	0	0.3*1 = 0.3
Thématique Forestière	0.2*0.6 = 0.12	0.2*0.7 = 0.14	0.3*0 = 0	0.2*1 = 0.2
Total	**0.72**	**0.68**	**0.33**	**0.67**

Ces résultats concordent plutôt bien avec la théorie.

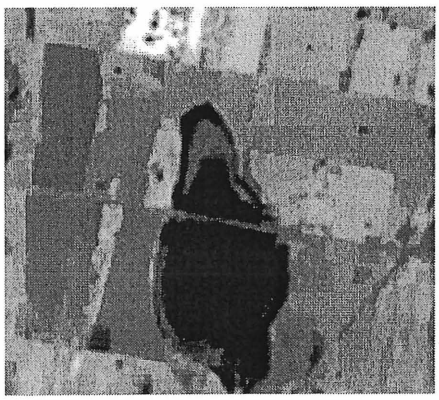
Dans le cas d'images radar, notre théorie stipulait que ML et SEM donnaient des résultats plutôt mauvais. Comme l'indique la figure 17, l'étude de l'application des quatre algorithmes sur une image Radarsat de l'Alberta (a) donne le constat suivant: Le ML (b) et le SEM (c) donnent des résultats trop segmentés, avec du bruit, de la texture dans certaines zones supposément homogènes. Par contre, du côté de S&M (d) et de WM (e), les résultats sont plutôt satisfaisants, puisque l'homogénéité de certaines zones est respectée et les délimitations sont fiables. Nous avons donc lancé cette fois dans le système une requête pour une image radar et nous avons obtenu les résultats présentés dans le tableau 6.

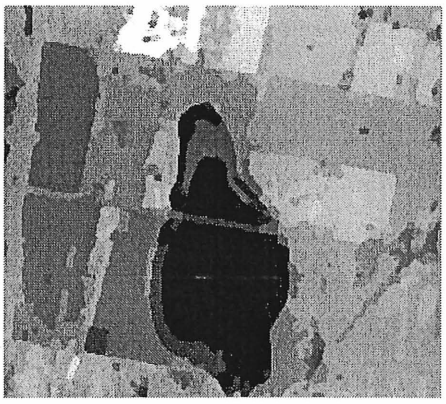


(a)



(b) (c)





(d) (e)

Figure 17. Segmentation sur une image Radarsat

Tableau 6. Résultats avec image radar

	ML	SEM	S&M	WM
Satellite RADARSAT	0.3*0 = 0	0.3*0= 0	0.2*0.7 = 0.14	0.2*0.7 = 0.14
Nombre de classes n'est pas connu	0.3*0 = 0	0.3*0 = 0	0.1*0.7 = 0.07	0.1*0.7 = 0.07
La variance n'est pas connue	0	0	0	0
Le seuil n'est pas connu	0	0	0	0
Thématique forestière	0.2*0.6 = 0.12	0.2*0.7 = 0.14	0.3*0 = 0	0.2*1 = 0.2
Total	**0.12**	**0.14**	**0.21**	**0.41**

Ces résultats concordent bien avec la théorie du chapitre 1, c'est-à-dire que les algorithmes de S&M et de WM seront avantagés et qu'ils récolteront plus de points si l'image à traiter est de type radar.

La thématique est un paramètre que nous avons inclus à la requête utilisateur parce que nous avons constaté avec les expérimentations que d'une part ML, SEM et WM étaient plus adaptés aux images forestières et qu'ils représentaient les zones un peu plus fidèlement sous cette thématique. D'autre part, le S&M s'est avéré plus performant pour la précision de la détection de contour quand la thématique était de type urbaine. Comme l'illustre la figure 18, pour une image urbaine donnée (a), avec des zones plutôt rectangulaires, on constate que le ML (b) tend à trop texturer certaines zones homogènes, comme la route verticale. On retrouve le même problème avec le SEM (c). Par ailleurs, on observe que le résultat le plus net vient de S&M (d). En effet, il respecte l'unicité des zones homogènes et la précision des contours des grands objets comme des petits. Le WM (e) donne également des zones plus homogènes, mais les contours sont moins biens respectés, car WM a tendance

à arrondir les coins. Voilà pourquoi notre système multi-agents aura tendance à privilégier S&M pour les zones urbaines.

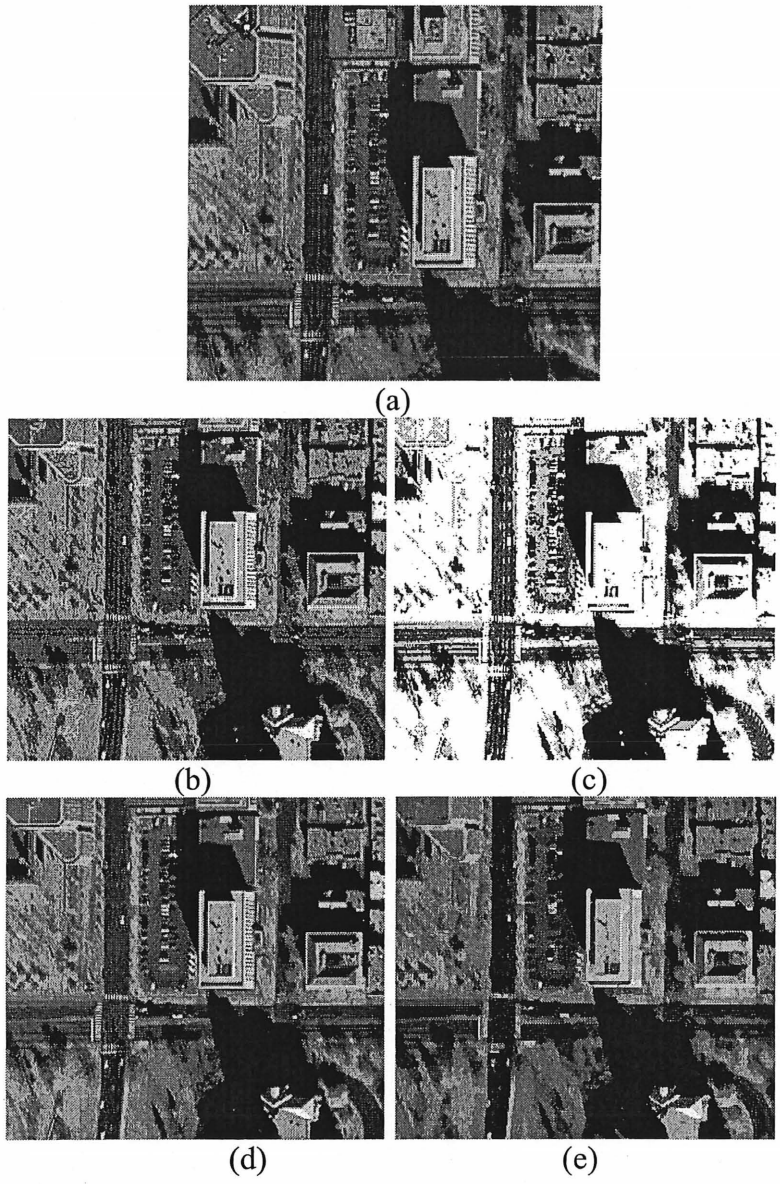


Figure 18. Résultats de segmentation sur une zone urbaine

Pour ces exemples, nous n'avons tenu compte que des critères de qualité et nous avons négligé les contraintes de temps. Si l'usager désire contraindre le temps maximal que le système doit accorder au traitement il a la liberté de le faire. Nous avons évalué dans le chapitre 1 le temps de traitement de chaque algorithme dans divers cas. Les équations de durée d'exécution que nous avons tirées de ces algorithmes nous ont permis de connaître les facteurs qui influençaient la durée des algorithmes et en fonction de quoi elle variait. Toutefois ces équations comportent encore trop d'inconnues pour que le système multi-agents puisse se baser uniquement sur elles dans le pointage en ce qui a trait au temps. Nous avons dû donc évaluer les temps d'exécution expérimentalement sur un échantillon de cas pour les différents algorithmes (voir section 1.2.1 du chapitre 1). Nous avons travaillé les fonctions de temps d'exécution $f(\Theta(ML))$, $f(\Theta(SEM))$, $f(\Theta(S\&M))$ et $f(\Theta(WM))$ pour qu'elles renvoient un pointage proportionnel au respect de la requête utilisateur. Par exemple, si le forestier spécifie dans sa requête que le temps de traitement peut être long, les différentes fonctions de durée renverront à peu près le même pointage pour les quatre algorithmes, car il importe peu au forestier d'avoir à attendre ou non. Par contre, pour une requête où le temps de traitement doit être court, on allouera un certain pointage de base (élevé pour S&M et bas pour SEM), car les expérimentations nous ont montré que S&M était un algorithme généralement rapide, à l'inverse de SEM, qui est généralement lent. À ce pointage de base, on apporte certaines modifications en fonction de certains paramètres, comme la taille de l'image, le nombre de classes ou la texture de l'image.

CONCLUSION

Les travaux que nous avons menés au cours de ce projet de maîtrise visaient à remplir des objectifs clairs :

- développer un système de traitement d'images satellitaires comportant une interface utilisateur conviviale et adaptée au niveau de connaissance en télédétection de l'utilisateur ;

- bâtir ce système sur une architecture modulaire pour simplifier l'ajout de fonctionnalités et d'opérations de traitement d'images développées par le reste de notre équipe ;

- doter ce système d'une forme d'intelligence pour assister l'utilisateur.

Nous avons effectivement travaillé à la convivialité de l'interface de plusieurs façons. Nous avons regroupé des commandes de façon logique dans le menu selon leur ordre d'utilisation et de sorte que l'usager les retrouve instinctivement. De plus, la visualisation d'images est simple et complète car le système permet de les afficher avec les informations et les caractéristiques qui leur sont propres. L'interface est adaptée aux utilisateurs experts et intermédiaires en télédétection. Les experts effectuent à leur guise toutes les sortes d'opérations de filtrage, de correction, de segmentation, de classification et d'interprétation d'images car ils y ont accès directement par le menu. Les utilisateurs intermédiaires, qui savent ce qu'est une segmentation d'image, mais

qui n'en connaissent pas les différents types, peuvent se prévaloir du module d'aide à la décision pour diriger leur choix.

Par ailleurs, nous avons également implémenté ce système sur une architecture modulaire. Comme le langage utilisé est orienté objet et que chaque fonctionnalité est encapsulée dans une classe, il devient alors trivial pour un développeur de greffer de nouvelles fonctionnalités au système en ajoutant manuellement un item du menu qui lance la méthode exécutant la fonctionnalité en question.

Pour assister les utilisateurs, nous avons développé un module d'aide à la décision. C'est ce module qui fournit l'intelligence du système et qui nous a permis d'atteindre notre troisième objectif. Nous le qualifions d'intelligent parce qu'il est constitué d'agents qui répondent à la définition d'agents intelligents que nous avons introduite au premier chapitre. Ces agents prennent des décisions et exécutent des actions à la place de l'usager en fonction de ce qu'ils perçoivent dans leur environnement. Plus précisément, ils choisissent et exécutent le meilleur algorithme de segmentation parmi ceux disponibles dans le système en fonction des paramètres de type de satellite, de thématique, de texture de l'image et du temps d'exécution, spécifiés par l'usager et qui constituent l'environnement de ces agents.

Bien que l'état d'avancement des travaux soit satisfaisant, il reste encore beaucoup de travail à faire pour que notre système soit vraiment performant, original et compétitif.

Entre autres améliorations possibles, il serait intéressant de doter I3Soft d'un agent assistant l'utilisateur voulant ajouter un algorithme de traitement d'images au système. De cette façon, même les utilisateurs sans expérience en

informatique pourraient greffer leurs propres algorithmes de traitement d'images à I3Soft.

De plus, I3Soft est maintenant adapté aux experts et aux intermédiaires en télédétection, mais il reste encore une lacune au niveau des débutants. Pour les assister, on pourrait doter I3Soft d'un agent planificateur qui, conjointement à notre module d'aide à la décision, planifierait la production d'une tâche complexe, comme la mise à jour d'une carte forestière, du début à la fin. C'est-à-dire qu'à partir des images disponibles de la région étudiée, le planificateur établirait pour chaque image une chaîne de tâches à effectuer, partant des opérations de pré-traitements, comme les différents filtres et corrections, aux traitements, comme la segmentation, et enfin aux post-traitements, comme l'interprétation de ces images.

Annexe A

CODE DES AGENTS

Agent Client

```
package siti;

/**
 * Title:       ClientAgent
 * Description: Agent that Introduce to the system the user's request to select
 *              an image process.
 * Copyright:   Copyright (c) 2001
 * Company:
 * @author
 * @version 1.0
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import madkit.kernel.*;
import madkit.lib.messages.ACLMessage;
import madkit.lib.messages.ObjectMessage;
import java.io.File;

public class ClientAgent extends Agent implements ActionListener {

ClientGUI clientGUI;
String treatment = "";
int tempsQualite = 0;
boolean preview = false;
MyInternalFrame inFrame;
File image = null;
AgentAddress broker = null;
AgentAddress provider = null;
String contract;
Request request;

private MainFrame frame;
boolean segmentationLaunched = false;
boolean classificationLaunched = false;

 public ClientAgent(MainFrame _frame) {
  frame = _frame;
 }
```

```
 public void activate() {
  if (isGroup("process-client"))
   joinGroup("process-client");
  else
   foundGroup("process-client");
  requestRole("process-client","client");
  println("activated");
  initGUI();
 }

 public void initGUI() {
     clientGUI = new ClientGUI(this, frame);
     setGUIObject((Object) clientGUI);
 }


 public void live () {
  while (true) {
   Message m = waitNextMessage();
   if (m instanceof ACLMessage)
         handleMessage((ACLMessage)m);
   else
         System.err.println("ERROR: invalid message type: "+ m);
  } //while
 } //live

 protected void handleMessage(ACLMessage e) {
  if (e.getAct().equals("inform")) { //the client sends himself a message to record the user
request ad send it to the broker
   sendRequest(); //sends a request to the broker
  }
  else if (e.getAct().equals("activate")) {//set the clientGUI object visible
   this.clientGUI.setVisible(true);
  }
  else if (e.getAct().equals("make-contract")) {//creates a direct link to the provider
   makeContract(e);
  } //else if
  else if (e.getAct().equals("accept-contract")) { //msg confirmation
   println("Contract OK with " +/*e.getSender().getName()*/provider.getName());
   pause (500);
   sendMessage(provider,new InternalFrameMessage(this.inFrame));
  }
 }

 private void sendRequest() {
    println("Sending request in : " + treatment);
//     ACLMessage req = new ACLMessage("request", treatment + "," + tempsQualite + "," +
preview);
//     sendMessage(getAgentWithRole("process-client", "broker"), req);
```

```
    sendMessage(getAgentWithRole("process-client", "broker"),new
RequestMessage(request));
  }

  private void makeContract(ACLMessage e)  {
    contract = e.getContent().toString();
    println("received contract confirmation. Signing " + contract);
    joinGroup(contract);
    requestRole(contract,"client");
    while (provider == null)  {
      provider = getAgentWithRole(contract,"provider");
          pause(100);
    }  //while
    println("Asking confirmation");
    sendMessage(provider, new ACLMessage("validate"));
  }

  public AgentAddress getBroker()  {
   return broker;
  }

  public void setBroker(AgentAddress add)   {
   broker = add;
  }

  private void launchProcessAgents()  {
   //structure à changer pour permettre l'ajout automatique de nouvelles fonctions
   if (treatment.equalsIgnoreCase("Segmentation"))  {
    if(!(segmentationLaunched))  {
     launchAgent((AbstractAgent) new SEMAgent(), "SEM",  true);
     launchAgent((AbstractAgent) new MlsegmAgent(), "MaxLikely",  true);
     launchAgent((AbstractAgent) new SplitMergeAgent(), "SplitMerge",  true);
     launchAgent((AbstractAgent) new WaterMergeAgent(), "WaterMerge",  true);
     segmentationLaunched = true;
    }
   }
   else if (treatment.equalsIgnoreCase("Classification"))  {
    if (!(classificationLaunched))  {
     classificationLaunched = true;
    }
   }
   else
    println("No " +treatment + " agent available.");
  }

  public void actionPerformed(ActionEvent e)  {
   println("action performed");
   provider = null;
   if (e.getSource() == clientGUI.jButtonValider)  {
    request = clientGUI.getRequest();
```

```
    if (request.traitement==Request.PROCESS_UNKNOWN){
      JOptionPane jop = new JOptionPane("You must choose a process");
      jop.setVisible(true);
    }
    treatment = clientGUI.getTreatment();
    tempsQualite = clientGUI.getTime();
    preview = clientGUI.getPreview();
    inFrame = clientGUI.getFrame();
    //      image = clientGUI.getImage();
    launchProcessAgents();
    sendMessage(this.getAddress(), new ACLMessage("inform", "action-pressed"));

  } //e.getSource() == jButtonValider
 }


}
```

Agent Broker

```
/**
 * Title:       BrokerAgent
 * Description:  Agent that controls the communications and
 *               negotiations between the ClientAgent and each process agent.
 *               It also finds the best process agent for each client request.
 *               It doesn't have an interface.
 * Copyright:    Copyright (c) Martin Labrie<p>
 * Company:      <p>
 * @author Martin Labrie
 * @version 1.0
 */
package siti;

import madkit.kernel.Agent;
import madkit.lib.messages.ACLMessage;
import madkit.kernel.Message;
import madkit.kernel.AgentAddress;
import java.util.Vector;

/*
* Subclass of BrokerAgent
*/
class BidAnswer {
        AgentAddress bidder;
        double value;
        BidAnswer(AgentAddress b, double v){bidder = b; value = v;}
        double getValue(){return value;}
        AgentAddress getBidder(){return bidder;}
}

public class BrokerAgent extends Agent {

static int contractId = 1;
Vector bidders = new Vector();
private AgentAddress[] providers;
private AgentAddress client;
private BidAnswer[] answers;
static int cpt;

 public BrokerAgent() {
 }

 public void activate()  {
  if (isGroup("process-client"))
   joinGroup("process-client");
  else
```

```
    foundGroup("process-client");
   requestRole("process-client","broker");
   if (isGroup("process-provider"))
    joinGroup("process-provider");
   else
    foundGroup("process-provider");
   requestRole("process-provider","broker");
   println("activated");

  }

  public void live () {
   while (true) {
    Message m = waitNextMessage();
    if (m instanceof ACLMessage)
          handleMessage((ACLMessage)m);
    else if (m instanceof RequestMessage)
          handleMessage((RequestMessage)m);
    else
          System.err.println("ERROR: invalid message type: "+ m);
   } //while
  } //live

  public void handleMessage(ACLMessage e) {
   if (e.getAct().equals("request")) {       // the client request a process
//     receiveRequest(e);
   }
   else if (e.getAct().equals("propose")) { //the process agents give their performances
    receiveBid(e);
   }
  }
  public void handleMessage(RequestMessage e) {
   receiveRequest(e);
  }


  protected void receiveRequest(RequestMessage e) {
     client = e.getSender();
     println("agent client name: " + e.getSender().getName());
     // extract the process type from the content of the message
/*     int separator = e.getContent().toString().indexOf(",");
     String process = e.getContent().toString().substring(0, separator);
     String content = e.getContent().toString().substring(++separator);*/
     String process = e.getContent().getProcessString(e.getContent().traitement);
     AgentAddress[] bidders = getAgentsWithRole("process-provider", process);
     cpt = bidders.length;
     answers = new BidAnswer[cpt];
//     broadcastMessage("process-provider", process, new ACLMessage("request-inf",
content));
     broadcastMessage("process-provider", process, new RequestMessage(e.getContent()));
```

```
}

 protected void receiveBid(ACLMessage m){
          println("Received an offer of " + m.getContent() + " from " + m.getSender());
          cpt--;
//       Integer temp = new Integer(m.getContent().toString().substring(0,5));
//       println("temp: " + temp);
//       double temp2 = temp.doubleValue();
//       println("temp2: " + temp2);
          answers[cpt]=new BidAnswer(m.getSender(), new
Double(m.getContent().toString()).doubleValue());
          if (cpt <= 0)
                    bestContract();
  }

  void bestContract()
    {
     AgentAddress best = null; // best process provider
     double bestoffer = 0;
     System.out.println("Selecting best offer from " + answers.length + " proposals");
     println("Selecting best offer from " + answers.length + " proposals");
     for (int i=0;i<answers.length;i++){









     if (best != null)
          {
           println("Chosen provider:"+best.getName());
           println("  with "+bestoffer);
           System.out.println("Sending provider confirmation");
           println("Sending provider confirmation");
           sendMessage(best, new ACLMessage("make-contract","contract-"+contractId));
           pause(100);
           println("Sending client confirmation");
           sendMessage(client, new ACLMessage("make-contract","contract-"+contractId));
       contractId++;
          }
    }


}
```

Agent MLSegm (fournisseur de traitement)

```
package siti;

/**
 * Title:       MlsegmAgent
 * Description:  This is a process provider agent that represents the MlSegm
 *              process. When a user request is sent by the client agent via
 *              the broker agent, it is received by all process providers like
 *              MlSegmAgent. Then this provider, like the others, checks out
 *              the characteristics of its process and compute if it fits well
 *              with the request. It calculates a score that is returned to
 *              the broker. If the process is the most adequate then the agent
 *              executes it. (see SEMAgent, SplitMergeAgent, WaterMergeAgent)
 * Copyright:    Copyright (c) 2001
 * Company:
 * @author
 * @version 1.0
 */

import madkit.kernel.*;
import madkit.lib.graphics.*;
import madkit.lib.messages.ACLMessage;
import madkit.lib.messages.ObjectMessage;
import madkit.kernel.Message;
import javax.swing.*;
import java.awt.*;


public class MlsegmAgent extends Agent  {

 JFrame frame;
 private Mlsegm maxLike;
 AgentAddress broker = null;
 private double _time;
 private boolean preview;
 private double score = 0;
 static int frameCount = 2;
 String contract;


 public MlsegmAgent() {

 }
 public void activate()  {
  if (isGroup("process-provider"))
   joinGroup("process-provider");
  else
```

```
    foundGroup("process-provider");
    requestRole("process-provider","segmentation");
    println("activated");
    initGUI();
  }

  public void initGUI()  {
    frame = new JFrame("Maximum Likelyhood");
    JPanel p = new JPanel(new BorderLayout());
    frame.getContentPane().add(p);
    OPanel o = new OPanel();
    p.add(o);
    this.setOutputWriter(o.getOut());
    setGUIObject(frame);
    setRandomTime();
    setRandomPreview();
    frame.setSize(200, 150);
    frame.setLocation(0,150 * frameCount);
    frame.validate();
    println("init Maximum Likelyhood");
  }

  public void live ()  {
    while(true)  {
      Message m = waitNextMessage();
      System.out.println("live: " + m.getReceiver().getClass().toString());
      if (m instanceof ACLMessage)
            handleMessage((ACLMessage)m);
      else if (m instanceof RequestMessage)
            handleMessage((RequestMessage)m);
      else if (m instanceof FileMessage)
            handleMessage((FileMessage)m);
      else if (m instanceof InternalFrameMessage)
            handleMessage((InternalFrameMessage)m);
      else
            System.err.println("ERROR: invalid message type: "+ m);
/*      if (m.getClass().toString().endsWith("ACLMessage")) {
       handleMessage((ACLMessage) m);
      }
      else if (m.getClass().toString().endsWith("FileMessage")) {
       handleMessage((FileMessage) m);
      }
      else if (m.getClass().toString().endsWith("InternalFrameMessage")) {
       handleMessage((InternalFrameMessage) m);
      }*/
    }  //while
  }

  protected void handleMessage(ACLMessage e)  {
```

```
  if (e.getAct().equals("request-inf")) { //request to evaluate the capability of this agent to do
the task
    frame.show();
    computeScore(e);
    sendMessage(e.getSender(),new ACLMessage("propose", String.valueOf(score)) );
   }
   else if (e.getAct().equals("make-contract")) {
    contract = e.getContent().toString();
    println("received contract confirmation. Signing " + contract);
    joinGroup(contract);
    requestRole(contract,"provider");
   }
   else if (e.getAct().equals("validate")){
    println("Validating contract OK");
    sendMessage(e.getSender(), new ACLMessage("accept-contract"));
   }
  }
  protected void handleMessage(RequestMessage e) {
    println("Request reception...");
    frame.show();
    computeScore(e);
    sendMessage(e.getSender(),new ACLMessage("propose", String.valueOf(score)) );
  }
  protected void handleMessage(FileMessage m) {
    println("initiate processing..");
    maxLike = new Mlsegm(m.getContent());
  }
  protected void handleMessage(InternalFrameMessage e) {
   println("initiate processing..");
   maxLike = new Mlsegm(e.getContent());
  }


  public void setRandomTime() {
   _time = 0.1/*Math.random()*/;
  }

  public void setRandomPreview() {
   if(Math.random() > 0.5)
    preview = true;
   else
    preview = false;
  }

  public void computeScore(ACLMessage e) {
    int separator = e.getContent().toString().indexOf(",");
    String msgTimeStr = e.getContent().toString().substring(0,separator);
    Integer integer = new Integer(msgTimeStr);
    double msgTime = (integer.doubleValue())/100;
    String preview = e.getContent().toString().substring(++separator);
```

```
    println("wanted time: " + msgTime);
//    println("real time : " + time );
//    score = Math.abs(msgTime - time);
    println("score: " + score);
  }
  public void computeScore(RequestMessage e)  {
   float satellite = 0.3f;
   float supervised = 1.0f;
   float merge = 0.1f;
   float time = 0.3f;
   float quality = 0.3f;

   float sa = satellite(e);
   float su = supervised(e);
   float me = merge(e);
   float ti = time(e);
   float qu = quality(e);
   score = sa*satellite+su*supervised+me*merge+ti*time+qu*quality;
   println("score: " + score);
   }
  private float satellite(RequestMessage e)  {
   float sat=0.0f;
   int type = e.getContent().type;
   switch (type) {
   case Request.TYPE_CASI:sat=1.0f;
   break;
   case Request.TYPE_IKONOS:sat=1.0f;
   break;
   case Request.TYPE_LANDSAT:sat=1.0f;
   break;
   case Request.TYPE_RADARSAT:sat=0.0f;
   break;
   case Request.TYPE_SPOT:sat=1.0f;
   break;
   case Request.TYPE_UNKNOWN:sat=0.5f;
   break;
   }
   return sat;
  }
  private float supervised(RequestMessage e)  {
   boolean sup = e.getContent().classe;
   if (sup) {
    return 1.0f;
   }
   else {
    return 0.0f;
   }
  }
  private float merge(RequestMessage e)  {
   boolean mer = e.getContent().merge;
```

```
    if (mer) {
      return 0.5f;
    }
    else {
      return 0.5f;
    }
  }
  private float time(RequestMessage e)  {
    float tim = (float) e.getContent().time;
    return tim/100;
  }
  private float quality(RequestMessage e)  {
    float qual = (float) e.getContent().quality;
    return qual/100;
  }
}
```

BIBLIOGRAPHIE

D. Lafrenière, *Créez des interfaces gagnantes*, Montréal, Les Éditions Logiques, 1995

P. Maes, *Artificial Life Meets Entertainment: Life like Autonomous Agents,* Communications of the ACM, 1995

P. Masson, W. Pieczynski, *SEM algorithm and unsupervised statistical segmentation of satellite images,* IEEE Transactions on geoscience & remote sensing, 1993

K. Rousseau, *Agent de sélection de données,* Mémoire, U. de Sherbrooke, 2002

S. Russell, P. Norvig, *Artificial Intelligence : A Modern Approach*, New Jersey: Prentice-Hall, 1995

M. Wooldridge, N. Jennings, *Agent Theories, Architectures, and Languages: a Survey,* Berlin: Springer-Verlag, 1995

ESRI (Environmental Systems Research Inc), http://www.esri.com/index.html

Research Systems Inc, http://www.rsinc.com/

PCI Geomatics, http://www.pcigeomatics.com/