# Analyse des documents graphiques : une approche par reconstruction d'objets

15 juin 2007

# Chapitre 1

# Reconstruction d'objets : du formalisme au système

## 1.1 Introduction

Au cours du chapitre précédent nous avons présenté nos différents opérateurs d'extraction de primitives graphiques. Dans ce chapitre nous développons notre approche de reconstruction d'objets (introduite en début de partie) basée sur l'utilisation de ces opérateurs. Nous présentons tout d'abord dans la section (1.2) notre approche pour la gestion des connaissances graphiques. Nous y développons un formalisme objet permettant la multi-représentation des formes graphiques. Celui-ci est opérationalisé au sein de nos différents opérateurs permettant leur interopérabilité et donc la reconstruction d'objets. Nous développons alors les principes généraux de la reconstruction d'objets dans la section (1.3). Nous introduisons préalablement les concepts de planification de programmes et de taxinomie de représentations. Nous présentons alors une formalisation de la reconstruction d'objets. La section (1.4) présente enfin notre système de reconstruction d'objets à travers ses éléments principaux : modélisation orientée objet et système de contrôle.

## 1.2 Gestion des connaissances graphiques

Cette section décrit notre approche pour la gestion des connaissances graphiques. Dans la sous-section (1.2.1) nous présentons notre formalisme objet permettant la multi-représentation des formes graphiques. Nous présentons ensuite dans la sous-section (1.2.2) comment nous représentons et opérationalisons ce formalisme particulièrement pour assurer l'interopérabilité entre les opérateurs d'extraction. Dans la sous-section (1.2.3) nous présentons quelques cas d'usage d'interopérabilité de notre approche. Finalement dans la sous-section (1.2.4) nous concluons.

### 1.2.1 Formalisme objet pour la multi-représentation

Notre formalisme est basé sur une approche objet[1] pour la gestion dynamique des objets graphiques. Notre connaissance graphique $(k_g)$[2] est représentée (équation (1.1)) à l'aide d'un seul objet graphique $(o)$. Cet objet graphique est une instance $(i)$ d'une classe graphique générique (ou méta-classe) abstraite $(o_g)$. Cette classe $(o_g)$ se spécialise en différentes classes $(1, ., u)$ selon une relation d'héritage $(I)$. De cette façon, cette représentation exploite deux propriétés importantes de la conception orientée objet : la spécialisation et le polymorphisme. Nos objets graphiques sont composés (équation (1.2)) d'un ensemble de données $(D)$ et de méthodes $(M)$. Ces données peuvent être des données de types primitifs $(d_i)$, ou d'autres objets graphiques $(o_i)$ inclus via une relation de composition.

$$\exists\ k_g = \{o\} \qquad o \overset{i}{\leftarrow} o_g \overset{I}{\rightarrow} \{o_{g_1}, ., o_{g_u}\} \tag{1.1}$$

$$o = \{D, M\} \qquad D = \{\{d_0, ., d_u\}, \{o_0, ., o_v\}\} \quad M = \{P, \{r, w\}\} \tag{1.2}$$

$$o_{g_i} \in \{\{p_1, ., p_u\}, \{l, g\}\} \tag{1.3}$$

Nous avons implicitement décomposé nos objets graphiques selon deux niveaux de formalisme (équation (1.3)). Le premier niveau exploite des formalismes “bas-niveau" pour la représentation des primitives graphiques $(p_i)$. Il s'agit donc de formalismes à base de vecteurs et de pixels. Nous avons considéré quelques primitives graphiques standards comme : les raster, les points, les composantes connexes, les lignes, les arcs, les courbes, les quadrilatères, .... Cependant celles-ci peuvent être facilement étendues grâce aux propriétés de spécialisation et de polymorphisme de notre approche (équation (1.1)). Le second niveau de formalisme (équation (1.3)) exploite des formalismes “haut-niveau" pour la structuration de nos objets graphiques. Plus particulièrement nous exploitons les formalismes de type liste $(l)$ et graphe $(g)$. Nous les détaillons par la suite.

Le formalisme liste $(l)$ définit (équation (1.4)) un ensemble d'objets graphiques $(O)$ et d'attributs graphiques $(A)$ ordonnés. Ces attributs graphiques sont en fait des objets graphiques utilisés de façon particulière au sein du formalisme liste. Un attribut graphique $(a_i)$ définit une relation entre deux objets graphiques successifs $(o_i)$ et $(o_j)$ de la liste $(l)$ (équation (1.5)). Selon que la liste est fermée ou non (premier objet graphique connecté au dernier), sa taille (équation (1.4)) peut être de $(u)$ ou $(u-1)$. Nous avons considéré quelques attributs graphiques standards comme : les étiquettes, les angles, les longueurs, ....

[1] Nous reportons le lecteur à l'Annexe B pour une introduction sur les langages O.O

[2] Notation anglophone conforme à notre librairie d'opérationalisation (sous-section (1.2.2)).

$$l = \{O, A\} = \{\{o_0, ., o_u\}, \{a_0, ., a_v\}\} \quad v = \{u, u-1\} \tag{1.4}$$

$$\forall\ a_i \quad \exists\ \{o_i, o_j\} \quad f(o_i, o_j) = a_i \tag{1.5}$$

$$a = \{D, M\} \qquad D = \{\{d_0, ., d_u\}, \{o_0, ., o_u\}\} \quad M = \{P, \{r, w\}\} \tag{1.6}$$

$$g = \{O, E\} = \{\{o_0, ., o_u\}, \{e_0, ., e_v\}\} \tag{1.7}$$

$$\forall\ e_{i,j} = \{\{i, j\}, a_{i,j}\} \quad \exists\ \{o_i, o_j\} \quad f(o_i, o_j) = a_{i,j} \tag{1.8}$$

Le formalisme graphe ($g$) définit (équation (1.7)) un ensemble d'objets graphiques ($O$) et d'objets arcs ($E$). Un objet arc ($e_{i,j}$) donné décrit une relation directe (ou indirecte) entre deux objets graphiques non ordonnés ($o_i$) et ($o_j$). Cette relation est définie (équation (1.8)) selon un attribut graphique ($a_{i,j}$).

Via ses propriétés objet notre formalisme est particulièrement adapté pour la multi-représentation des objets graphiques. En effet, grâce à la propriété de polymorphisme nos différents objets graphiques sont manipulés de façon générique au travers des références de type ($o_g$) (équation (1.1)). Cette généricité permet une structuration hiérarchique des connaissances graphiques. Les objets graphiques ($o$) (équation (1.1)) inclus dans les objets listes ($l$) et graphes ($g$) (équations (1.4) et (1.7)) peuvent être eux-même des objets listes ($l$) et des objets graphes ($g$). Ces différents aspects permettent alors une variation aisée des modèles employés pour la représentation des objets graphiques. Nous illustrons cette propriété à travers un cas d'usage de modélisation présenté sur la figure (1.1) (a) : celui des carrés-liés.

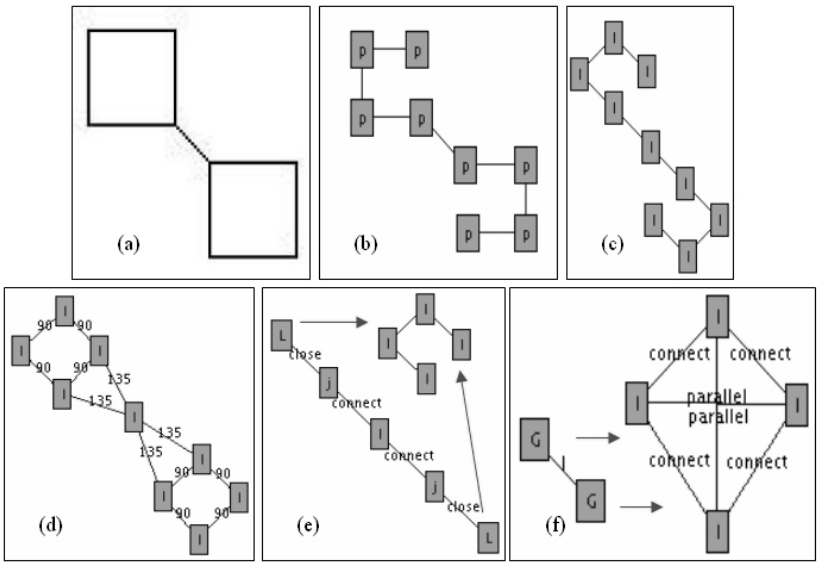



FIG. 1.1 – (a) carrés-liés (b) liste de points (c) liste de lignes (d) graphe de lignes (e) listes hiérarchiques (f) graphes hiérarchiques

Les figures (1.1) (b), (c) et (d) présentent trois premières modélisations des carrés-liés basées sur notre formalisme. Les modélisations (b) et (c) sont basées sur le formalisme liste exploitant respectivement les objets graphiques points ($p$) et lignes ($l$). En effet, il est possible de décrire les carrés-liés selon une succession de points ou de lignes. La modélisation (d) est basée elle sur le formalisme graphe exploitant des objets graphiques lignes ($l$). Dans ce graphe, les arcs décrivent les relations de connexion entre les objets graphiques lignes ($l$). Ces relations de connexion sont attribuées par les rapports angulaires entre les lignes connectées.

Les figures (1.1) (e) et (f) présentent deux modélisations hiérarchiques des carrés-liés. La modélisation (e) est basée sur le formalisme liste exploitant des objets graphiques de type ligne ($l$), jonction ($j$) et ($L$). Les objets ($L$) représentent des sous-listes d'objets lignes ($l$). Chaque sous-liste correspond à un carré. Les objets graphiques ($l$), ($j$) et ($L$) sont reliés par des arcs décrivant des relations de boucle (*close*) et de connexion (*connect*). La modélisation (f) est basée sur un formalisme graphe exploitant des objets graphiques de type ligne ($l$) et ($G$). Les objets ($G$) représentent des sous-graphes composés d'objets graphiques lignes ($l$). Chaque sous-graphe correspond à un carré. Dans ces sous-graphes, les objets graphiques lignes ($l$) sont reliés par des arcs décrivant des relations de connexion (*connect*) et de parallélisme (*parallel*). Les objets graphiques ($G$) sont reliés par un arc attribué représentant la ligne reliant les carrés.

Dans cette sous-section nous avons présenté notre formalisme objet pour la représentation des connaissances graphiques. Nous avons illustré à travers le cas d'usage des carrés liés les propriétés de multi-représentation de ce formalisme. De cette façon, un système exploitant ce formalisme peut aisément modifier les représentations des objets graphiques qu'il manipule. Nous présentons dans la sous-section suivante comment nous déclarons et opérationalisons ce formalisme au sein d'un système afin d'assurer l'interopérabilité entre ses opérateurs.

### 1.2.2 Déclaration, opérationalisation et interopérabilité

#### Introduction

Nous présentons dans cette sous-section la déclaration et l'opérationalisation de notre formalisme au sein des systèmes. La figure (1.2) illustre le principe de notre approche. Les connaissances graphiques sont représentées via notre formalisme de façon déclarative dans une base propre à un système donné. Elles sont par la suite opérationalisées par les différents opérateurs d'extraction de ce système. Ceux-ci utilisent pour cela notre librairie de modélisation des objets graphiques basée sur notre formalisme : la GOMLib[3]. Cette dernière permet aux opérateurs à la fois d'interagir avec la base de connaissances graphiques mais aussi de représenter les connaissances graphiques au niveau procédural. Nous développons les aspects déclaration et opérationalisation de notre approche dans la suite de cette sous-section.

[3]Graphical Object Modelling Library

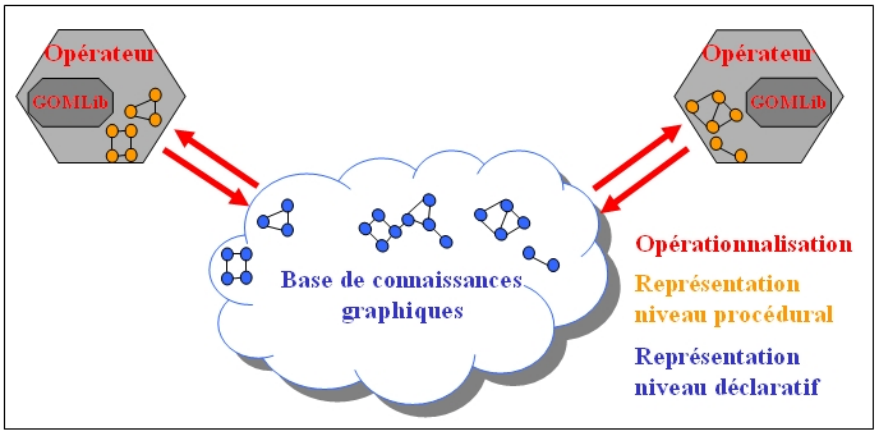



FIG. 1.2 – Déclaration et opérationalisation des connaissances graphiques

### Déclaration

Nos connaissances graphiques sont représentées au niveau déclaratif (dans la base) et au niveau procédural (au sein des opérateurs) dans un système. Leur représentation au niveau déclaratif est basée sur l'utilisation du langage XML[4]. La figure (1.3) en donne deux exemples avec la déclaration d'un objet point (a) et ligne (b). Le passage entre les niveaux de représentation déclaratif vers procédural pour des objets est assuré par l'utilisation de leurs méthodes ($M$) (équations (1.2)). En effet, ces dernières sont composées de méthodes de traitement ($P$) applicables aux données de l'objet, et de méthodes de lecture ($r$) et d'écriture ($w$) de ces données. De cette manière chaque objet prend en charge lui-même ses mécanismes d'externalisation et de chargement. Cette caractéristique présente différents avantages. Premièrement les propriétés d'externalisation et de chargement d'objets donnés peuvent être ré-utilisées par d'autres objets à travers les relations de composition (équation (1.2)). Ensuite, la rencontre de types d'objet inconnus à un opérateur donné dans la base de connaissances peut être aisément traitée. Ces objets sont simplement ignorés, les balises XML sont alors utilisées pour localiser les objets suivants dans la base de connaissances.

```
<OPoint x= "10" y= "10" />
```

(a)

```
<OLine length="10" direction="0">
    <OPoint x="10" y="10" />
    <OPoint x="20" y="10" />
</OLine>
```

(b)

FIG. 1.3 – Déclaration XML (a) point (b) ligne

[4] eXtensible Markup Language

### Opérationalisation et interopérabilité

L'opérationalisation de nos connaissances graphiques par les opérateurs se décline sous deux aspects principaux. Premièrement ceux-ci manipulent directement les objets graphiques à travers leurs différentes méthodes ($M$) (équation (1.2)). Ceci permet une conception aisée des opérateurs, une large partie du traitement des primitives graphiques est effectuée en majeure partie à travers ces méthodes ($M$). Ensuite, les opérateurs extraient (et insèrent) de (dans) la base de connaissances graphiques des objets graphiques. Ce dernier aspect constitue le mécanisme central de l'interopérabilité entre nos opérateurs. Nous utilisons pour cela deux méthodes principales, une pour l'extraction des objets graphiques et une autre pour leur substitution. Ces méthodes exploitent des techniques de parcours récursif des graphes et des listes. Nous les détaillons dans la suite de cette sous-section.

Notre méthode d'extraction est présentée dans le pseudo-algorithme (1.2.1). Elle recherche les objets graphiques selon une contrainte (*classes*) qui décrit les types d'objet à extraire. Cette recherche emploie un parcours récursif des objets liste et graphe. Ce dernier est illustré sur la figure (1.4) (a). Les objets graphiques sont extraits prioritairement de gauche en bas puis de haut à droite.

**Pseudo-algorithme 1.2.1:** EXTRAIRE($classes, base$)

**globale** *extrait*

**fonction** EXTRAIRE($classes, base$)
**pour chaque** *objet* de *base* de *début* à $fin$
**faire** { **si** *objet* est instance des *classes*
**alors** ajouter *objet* à *extrait*
**si** *objet* est instance de *liste* **ou** *graphe*
**alors** EXTRAIRE($classes, objet$)

Notre méthode de substitution est détaillée elle dans le pseudo-algorithme (1.2.2). Celle-ci utilise, suite au traitement des objets extraits, la contrainte (*classes*). De cette façon, la contrainte utilisée pour l'extraction des objets est de nouveau utilisée pour leur substitution. Le parcours récursif employé est cette fois ci de sens inverse à l'extraction. Celui-ci est illustré sur la figure (1.4) (b). En effet il est indispensable de substituer prioritairement les objets graphiques se situant dans les niveaux hiérarchiques inférieurs. L'insertion de nouveaux objets graphe ou liste peut modifier la structure hiérarchique des objets graphiques. Ceci rend alors impossible la recherche postérieure d'objets graphiques dans les niveaux inférieurs.

**Pseudo-algorithme 1.2.2:** SUBSTITUER($classes, base$)

**globale** $extrait, substitut, k, t$

**fonction** SUBSTITUER($classes, base$)
**pour chaque** *objet* de *base* de *fin* à *début*
**faire**
- **si** *objet* est instance de *liste* **ou** *graphe*
  - **alors** SUBSTITUER($classes, objet$)
- **si** *objet* est instance des *classes*
  - **alors**
    - **si** *objet* est différent $substitut[t - k]$
      - **alors** substituer $substitut[t - k]$ à *objet* dans *base*
    - $k \leftarrow k + 1$

**principal**
$classes \leftarrow$ liste des classes requises
$extrait \leftarrow \emptyset$
$base \leftarrow$ LECTURE($adresse$)
EXTRAIRE($classes, base$)

$substitut \leftarrow$ TRAITER($extrait$)
$t \leftarrow$ taille de *substitut*
$k \leftarrow 0$
SUBSTITUER($classes, base$)

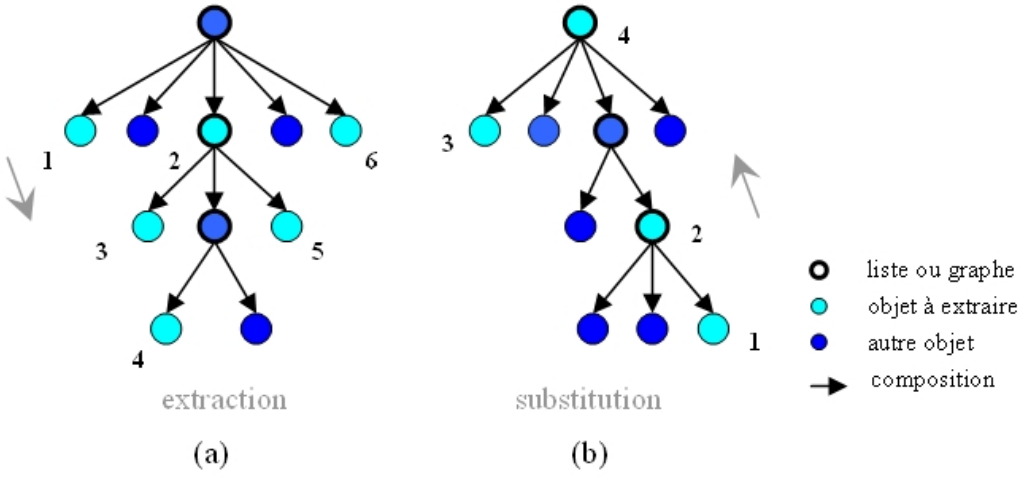



FIG. 1.4 – Parcours récursif des algorithmes d'extraction/substitution

Les méthodes d'extraction/substitution seules ne suffisent pas à gérer les interactions des opérateurs avec la base de connaissances graphiques. En complément ceux-ci utilisent des méthodes de filtrage. Ces dernières permettent d'identifier des objets extraits de la base, les objets à traiter par les opérateurs. Elles permettent également de déterminer de quelles façons ces objets vont être traités puis substitués. Elles sont donc propres à chacun des opérateurs. De façon à présenter ces méthodes de filtrage nous les avons formalisées sous la forme de spécifications de contraintes[5]. Nous présentons dans les tableaux (1.1) et (1.2) les notations que nous utilisons pour définir ces spécifications de contraintes. Ces dernières ne sont qu'une spécialisation à notre problème des notations logiques usuelles [Paulson 99].

Le tableau (1.1) présente les notations utilisées pour les types, objets et contraintes. Les types se regroupent en différents ensembles : les listes et les graphes $\{L, G\}$, les types graphiques $\{r, c, o, p, j, l, q, t, s\}$ et les attributs $\{v_g, i, d, x, f\}$. Chacun de ces types dans notre formalisme est spécialisé du type générique graphique $(g)$. Nous avons ensuite défini une notation $(\theta)$ pour les objets. Parmi ces objets nous distinguons les objets particuliers base $(\theta_\theta)$ et résultat d'opérateur $(\theta_\vartheta)$. Nous utilisons enfin différentes notations pour les contraintes. Celles-ci concernent à la fois les types et les objets. Concernant les types elles sont d'ordre arithmétique $(=)$, logique $(\wedge)$ $(\vee)$ $(\oplus)$ $(\overline{T})$ et de composition $(\sqsupset)$ $(\sqsupseteq)$. Une variable $(T_n)$ est alors utilisée pour représenter les expressions : types + contraintes. Cette dernière sert également en paramètre de certains opérateurs afin d'adapter leurs spécifications. Concernant les objets les contraintes sont d'ordre ensembliste $(\subset)$ $(\supset)$, d'instanciation $(\models)$ et d'équivalence structurelle $(\equiv)$.

| | | | |
|---|---|---|---|
| $g$ | type (g)raphique | $\theta_n$ | objet avec $n \in \emptyset, \alpha, \beta, ., \zeta$ |
| | | $\theta_\theta$ | objet base |
| $L$ | type (L)iste | $\theta_\vartheta$ | objet résultat opérateur |
| $G$ | type (G)raphe | | |
| | | $T_n$ | variable type + contrainte |
| $r$ | type (r)aster | | avec $n \in \emptyset, 1, 2, \ldots$ |
| $c$ | type (c)omposante | $T_1 = T_2$ | $T_1$ égal à $T_2$ |
| $o$ | type (o)cclusion | $T_1 \wedge T_2$ | $T_1$ et $T_2$ |
| $p$ | type (p)oint | $T_1 \vee T_2$ | $T_1$ ou $T_2$ |
| $j$ | type (j)onction | $T_1 \oplus T_2$ | soit $T_1$ soit $T_2$ |
| $l$ | type (l)igne | $\overline{T}$ | non $T$ |
| $q$ | type (q)uadrilatère | $T_1 \sqsupset T_2$ | $T_1$ est composé de $T_2$ |
| $t$ | type carac(t)éristiques | $T_1 \sqsupseteq T_2$ | $T_1$ est strictement composé de $T_2$ |
| $s$ | type (s)ymbole | | |
| $v_g$ | type (v)oisinage de $g$ | $\theta \models T$ | $\theta$ est instance de $T$ |
| $i$ | type (i)nclusion | $\theta_\beta \subset \theta_\alpha$ | $\theta_\beta$ appartient à $\theta_\alpha$ |
| $d$ | type (d)istance | $\theta_\alpha \supset \theta_\beta$ | $\theta_\alpha$ contient $\theta_\beta$ |
| $x$ | type conne(x)ion | $\theta_\beta \equiv \theta_\gamma$ | $\theta_\beta$ de structure équivalente à $\theta_\gamma$ |
| $f$ | type (f)ermé | | |

TAB. 1.1 – Notations des types, objets et contraintes

[5] Une présentation exhaustive de chacun des algorithmes sortirait du cadre de ce manuscrit.

Le tableau (1.2) présente les notations des différentes opérations pour la mise en oeuvre des types, objets et contraintes. Nous utilisons tout d'abord les opérations relatives à nos méthodes d'extraction ($\theta_\alpha[\theta_\beta]$) et de substitution ($\theta_\alpha[\theta_\beta/\theta_\gamma]$). Ces opérations permettent d'extraire et de substituer des objets ($\theta_\gamma$) sous-ensemble d'objet ($\theta_\alpha$). Nous utilisons en complément deux opérations de navigation ($\top$) ($\bot$) afin de basculer entre les ensembles et sous-ensembles. Les opérations ($\sqcup$) ($\sqcap$) correspondent enfin aux différentes reconstructions effectuées par les opérateurs.

| | |
|---|---|
| $\theta_\beta \leftarrow \theta_\alpha$ | $\theta_\beta$ est affecté de $\theta_\alpha$ |
| $\theta_\beta \leftarrow \theta_\alpha[T]$ | extraire les objets $T$ de $\theta_\alpha$ et affecter $\theta_\beta$, $\theta_\beta \subset \theta_\alpha$ |
| $\theta_\alpha[\theta_\beta/\theta_\gamma]$ | substituer $\theta_\beta$ par $\theta_\gamma$ dans $\theta_\alpha$, sachant que $\theta_\beta \subset \theta_\alpha$ et $\theta_\beta \equiv \theta_\gamma$ |
| $\theta_\alpha \leftarrow \top[\theta_\beta]$ | retourner l'objet $\theta_\alpha$ tel que $\theta_\alpha \supset \theta_\beta$ |
| $\theta_\beta \leftarrow \bot[\theta_\alpha]$ | retourner l'objet $\theta_\beta$ tel que $\theta_\beta \subset \theta_\alpha$ |
| $\theta_\alpha \cup [\theta_\beta]$ | insérer $\theta_\beta$ dans $\theta_\alpha$ |
| $\theta_\alpha \cap [\theta_\beta]$ | supprimer $\theta_\beta$ dans $\theta_\alpha$, sachant que $\theta_\beta \subset \theta_\alpha$ |
| $\theta_\alpha \sqcup [\theta_\beta]$ | reconstruire $\theta_\alpha$ par ajout de $\theta_\beta$ |
| $\theta_\alpha \sqcap [\theta_\beta]$ | reconstruire $\theta_\alpha$ par suppression de $\theta_\beta$, sachant que $\theta_\beta \subset \theta_\alpha$ |

TAB. 1.2 – Notations des opérations

Basé sur les notations des tableaux (1.1) et (1.2) nous présentons dans l'équation (1.9) un exemple de spécification de contraintes relatif à l'opérateur de marquage de composantes. Cette spécification décrit l'extraction de l'objet raster ($r$) et sa substitution dans ($\theta_\theta$) par la liste de composantes ($\theta_\vartheta$) extraite par l'opérateur.

$$\left.\begin{array}{ll} T_1 & = r \\ \theta_\alpha & \leftarrow \theta_\theta[T_1] \\ T_2 & = L \sqsupseteq c \\ \theta_\vartheta & \models T_2 \\ \theta_\theta & [\theta_\alpha/\theta_\vartheta] \end{array}\right\} \theta_\theta[\theta_\theta[r]/\theta_\vartheta \models (L \sqsupseteq c)] \tag{1.9}$$

Nous présentons dans les tableaux (1.3), (1.4), (1.5), (1.6) et (1.7) les spécifications de contraintes[6] de chacun de nos opérateurs présentés dans le chapitre (**??**). D'autres opérateurs complémentaires aux nôtres et basés sur notre formalisme sont également présentés dans le tableau (1.8). Pour chacun des opérateurs présentés dans ces tableaux les spécifications de contraintes sont également exprimées en langage naturel. Nous détaillons chacun de ces tableaux dans la suite de cette sous-section.

[6] Ces spécifications de contraintes présupposent que ($\theta_\theta$) est définie pour une seule image.

| **Opérateur** | **Spécification de contraintes** | **Description** |
|---|---|---|
| Marquage composantes | $\theta_\theta[\theta_\theta[r]/\theta_\vartheta \models (L \sqsupseteq c)]$ | Extraire le raster, le substituer par une liste de composantes. |
| Extraction occlusions (1) | $\theta_\theta[\theta_\theta[r]/\theta_\vartheta \models (L \sqsupseteq o)]$ | Extraire le raster, le substituer par une liste d'occlusions. |
| Extraction occlusions (2) | $\theta_\theta[\theta_\theta[c]]$<br>$\theta_\theta \cup [\theta_\vartheta \models (L \sqsupseteq o)]$ | Extraire les composantes, insérer la liste d'occlusions correspondante dans la base. |
| Extraction contours | $\theta_\theta[\theta_\theta[T = (c \vee o)]/\theta_\vartheta \models (L \sqsupseteq p)]$ | Extraire les composantes ou les occlusions, les substituer par des listes de points. |

TAB. 1.3 – Spécifications de contraintes des opérateurs (1)

Les opérateurs *marquage composantes*, *extraction occlusions*, *extraction contours* exploitent des spécifications de contraintes purement basées sur les opérations d'extraction/substitution. Ils extraient de la base ($\theta_\theta$) des objets graphiques de types raster ($r$), composante ($c$) ou occlusion ($o$). Ces objets sont alors substitués par les objets résultats des opérateurs ($\theta_\vartheta$). L'opérateur *extraction contours* exploite une variable ($T$) en paramètre de l'opération d'extraction[7] permettant de spécifier si l'extraction doit s'effectuer sur les composantes, sur les occlusions, ou sur les deux. L'opérateur *extraction occlusions* utilise deux spécifications de contraintes (1) (2). En effet, dans le cas où l'objet raster ($r$) est absent de la base ($\theta_\theta$) les objets composantes ($c$) sont exploités. La liste d'occlusions extraite est alors insérée dans la base ($\theta_\theta$).

| **Opérateur** | **Spécification de contraintes** | **Description** |
|---|---|---|
| Squelettisation | $\theta_\theta[\theta_\theta[T = (r \vee c \vee o)]/\theta_\vartheta \models T]$ | Extraire le raster ou les composantes ou les occlusions, les substituer par leurs squelettes. |
| Graphe squelette | $T_1 = (r \vee c \vee o)$<br>$T_2 = G \sqsupseteq ((L \sqsupseteq p) \wedge j \wedge x)$<br>$\theta_\theta[\theta_\theta[T_1]/\theta_\vartheta \models T_2]$ | Extraire le raster ou les composantes ou les occlusions, les substituer par des graphes de squelette. |

TAB. 1.4 – Spécifications de contraintes des opérateurs (2)

Les opérateurs de *squelettisation* et *graphe squelette* procèdent également par extraction/substitution. Cependant en ce qui concerne l'opérateur *squelettisation* les objets substitués sont de mêmes types que les objets extraits. Celui-ci extrait les objets graphiques squelettisables (raster ($r$), composante ($c$), occlusion ($o$)) puis les substitue par leurs squelettes. Un marquage préliminaire de l'image permettra donc une segmentation de l'image en différents squelettes connexes. L'opérateur *graphe squelette* extrait les mêmes objets (raster ($r$), composante ($c$), occlusion ($o$)). Il les substitue alors par des graphes de squelette.

[7] Le paramètrage est également utilisé pour d'autres spécifications dans les tableaux suivants.

| **Opérateur** | **Spécification de contraintes** | **Description** |
|---|---|---|
| Graphe voisinage (1) | $T_2 = L \sqsupseteq (T_1 = (c \oplus o))$<br>$T_3 = G \sqsupseteq (T \wedge v_{T_1})$<br>$\theta_\theta[\theta_\theta[T_2]/\theta_\vartheta \models T_3]$ | Extraire les listes composées soit de composantes soit d'occlusions, les substituer par des graphes de voisinage. |
| Graphe voisinage (2) | $T_2 = G \sqsupset (T_1 = (c \oplus o) \wedge \overline{v_{T_1}})$<br>$\theta_\theta[T_2] \sqcup [\theta_\vartheta \models v_{T_1}]$ | Extraire les graphes composés soit de composantes soit d'occlusions, y reconstruire les relations de voisinage. |

TAB. 1.5 – Spécifications de contraintes des opérateurs (3)

L'opérateur *graphe voisinage* utilise deux spécifications de contraintes (1) et (2). La spécification (1) présuppose que le modèle en cours dans la base ($\theta$) n'est pas encore formalisé sous forme de graphe. Les graphes de voisinage sont alors construits à partir des listes composées, soit de composantes ($c$), ou soit d'occlusions ($o$) extraites de ($\theta$). La spécification (2) présuppose elle que le modèle en cours dans ($\theta$) est déjà formalisé sous forme de graphe. Dans ce cas les relations de voisinage ($v_g$) sont construites, soit à partir des composantes ($c$), ou soit à partir des occlusions ($o$) existantes dans ces graphes. Elles sont ensuite ajoutées dans chacun de ces graphes.

| **Opérateur** | **Spécification de contraintes** | **Description** |
|---|---|---|
| Graphe inclusion (1) | $\theta_\alpha \leftarrow \theta_\theta[(L \sqsupseteq c) \wedge (L \sqsupseteq o)]$<br>$\theta_\theta \cup [\theta_\vartheta \models G \sqsupseteq (c \wedge o \wedge i)] \cap [\theta_\alpha]$ | Extraire les listes composées de composantes et d'occlusions, les remplacer par un graphe d'inclusion. |
| Graphe inclusion (2) | $(T_1 = c \wedge T_2 = o) \oplus (T_1 = o \wedge T_2 = c)$<br>$\theta_\alpha \leftarrow \theta_\theta[L \sqsupseteq T_1]$<br>$\theta_\theta[G \sqsupset (T_2 \wedge \bar{i} \wedge \overline{T_1})] \sqcup [\theta_\alpha] \sqcup [\theta_\vartheta \models i]$<br>$\theta_\theta \cap [\theta_\alpha]$ | Extraire soit la liste de composantes soit celle d'occlusions, reconstruire la liste et les relations d'inclusion dans le graphe complémentaire, supprimer la liste de la base. |
| Graphe inclusion (3) | $(T_1 = c \wedge T_2 = o) \oplus (T_1 = o \wedge T_2 = c)$<br>$\theta_\alpha \leftarrow \theta_\theta[G \sqsupset (T_1 \wedge \bar{i} \wedge \overline{T_2})]$<br>$\theta_\theta[G \sqsupset (T_2 \wedge \bar{i} \wedge \overline{T_1})] \sqcup [\theta_\alpha] \sqcup [\theta_\vartheta \models i]$<br>$\theta_\theta \cap [\theta_\alpha]$ | Extraire les deux graphes complémentaires, les fusionner et y reconstruire les relations d'inclusion. |

TAB. 1.6 – Spécifications de contraintes des opérateurs (4)

L'opérateur *graphe inclusion* utilise différentes spécifications de contraintes (1), (2) et (3). La spécification (1) présuppose qu'aucun graphe de composantes ($c$) ou d'occlusions ($o$) n'est existant dans la base ($\theta$). Dans ce cas les deux listes de composantes ($c$) et d'occlusions ($o$) sont recherchées afin de construire le graphe d'inclusion. La spécification (2) présuppose que au moins un graphe de composantes ($c$) ou d'occlusions ($o$) est existant dans la base ($\theta$). Dans ce cas la liste complémentaire est utilisée afin de reconstruire ce graphe. Les relations d'inclusion y sont par la suite reconstruites. Enfin, la spécification (3) présuppose que deux graphes composés de composantes ($c$) et d'occlusions ($o$) sont existants dans la base ($\theta$). Ces graphes sont alors fusionnés et les relations de voisinage reconstruites.

| **Opérateur** | **Spécification de contraintes** | **Description** |
|---|---|---|
| Graphe complet (1) | $\theta_\theta[\theta_\theta[L \sqsupseteq g \wedge \overline{d}]/\theta_\vartheta \models (G \supseteq (g \wedge d))]$ | Extraire les listes composées d'objets graphiques, les substituer par des graphes complets. |
| Graphe complet (2) | $\theta_\theta[G \sqsupset (g \wedge \overline{i})] \cup [\theta_\vartheta \models d]$ | Extraire les graphes comportants des objets graphiques, y insérer les relations de distance. |
| Filtrage distance | $\theta_\theta[G \sqsupset (g \wedge \overline{i})] \cap [\theta_\vartheta \models d]$ | Extraire les graphes comportant des objets distance, les filtrer. |

TAB. 1.7 – Spécifications de contraintes des opérateurs (5)

Les spécifications de contraintes utilisées par les opérateurs *graphe complet* et *filtrage distance* sont comparables à celles de l'opérateur *graphe voisinage*. Elles ne sont pas limitées par contre par des types d'objet graphique particuliers. Les opérateurs *graphe complet* et *filtrage distance* peuvent donc s'exécuter sur n'importe quel ensemble d'objets graphiques afin d'en calculer ou d'en filtrer les relations de distance ($d$). Ceci fait que ces opérateurs sont parmi les plus inter-opérables. En effet, dans notre formalisme les objets graphiques disposent de méthodes afin de calculer leurs caractéristiques topologiques (centre de gravité, rectangle englobant, ...). Ces méthodes existent aussi pour les graphes et les listes, ces derniers étant également des objets graphiques. La figure (1.5) illustre ces propriétés de calculs topologiques à travers un exemple de rotation d'un symbole. Ce dernier est représenté dans notre formalisme sous la forme de graphe de vecteurs. Les calculs des centres de gravité et rotations sont donc alors directement effectués à partir de l'objet graphe.

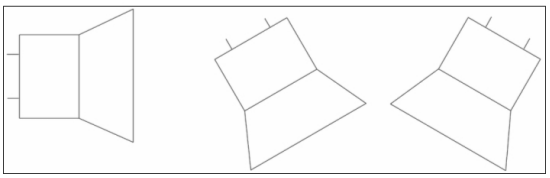


FIG. 1.5 – Illustration du calcul topologique sur un objet graphe

Le tableau (1.8) présente enfin différentes spécifications de contraintes d'opérateurs[8] usuels en analyse des documents graphiques. Il s'agit des opérateurs *extracteur statistique*, *kppv*, *appariement graphe*, *polygonalisation*, *appariement arc*, *appariement contour*. Leurs spécifications de contraintes sont proches[9] de celles présentées pour nos opérateurs. Ces opérateurs usuels nous permettent de compléter les nôtres à l'aide de primitives graphiques de plus haut niveau comme les symboles (ou étiquettes), les primitives à base de vecteurs (vecteurs, arcs, courbes, ...), ....

[8] Ces derniers sont exploités de la librairie PSI présentée en Annexe C de ce manuscrit.
[9] Nous reportons le lecteur aux explications précédentes pour leur interprétation.

| **Opérateur** | **Spécification de contraintes** | **Description** |
|---|---|---|
| Extracteur statistique | $\theta_\theta[\theta_\theta[T = (c \vee o)]/\theta_\vartheta \models t]$ | Extraire les composantes ou les occlusions, les substituer par des caractéristiques. |
| kppv | $\theta_\theta[\theta_\theta[t]/\theta_\vartheta \models s]$ | Extraire les caractéristiques, les substituer par des symboles. |
| Appariement graphe | $\theta_\theta[\theta_\theta[G \sqsupseteq T]/\theta_\vartheta \models s]$ | Extraire les graphes d'un type donné, les substituer par des symboles. |
| Polygonalisation | $\theta_\theta[\theta_\theta[L \sqsupseteq p]/\theta_\vartheta \models L \sqsupseteq l]$ | Extraire les listes composées de points, les substituer par des listes composées de lignes. |
| Appariement arc | $\theta_\theta[L \sqsupseteq l] \sqcup [\theta_\vartheta \models a]$ | Extraire les listes composées de lignes, y reconstruire les arcs. |
| Appariement contours | $\theta_\theta[\theta_\theta[L \sqsupseteq (l \wedge f)]/\theta_\vartheta \models L \sqsupseteq q]$ | Extraire les listes fermées et composées de lignes, les substituer par des listes de quadrilatères. |

TAB. 1.8 – Spécifications de contraintes des opérateurs (6)[10]

Nous avons présenté ici dans les tableaux (1.3) à (1.8) les différentes spécifications de contraintes de nos opérateurs. Ainsi chaque opérateur exploite, en complément des méthodes d'extraction et de substitution, un/des ensemble(s) de contraintes permettant de spécifier comment les objets graphiques doivent être extraits puis insérés dans la base de connaissances. Notre formalisme sur la base de ces contraintes devient alors pivot et permet de combiner (aisément) les opérateurs. De cette manière notre approche permet une interopérabilité entre les opérateurs. Nous présentons quelques cas d'usage d'interopérabilité dans la sous-section suivante.

### 1.2.3 Cas d'usage d'interopérabilité

Au cours de cette section nous avons présenté notre formalisme objet permettant la multi-représentation des formes graphiques. Nous avons également présenté comment nous déclarons et opérationalisons ce formalisme, particulièrement pour assurer l'interopérabilité entre nos opérateurs. Dans cette sous-section nous présentons quelques cas d'usage d'interopérabilité basés sur notre approche.

Notre premier cas d'usage concerne la reconnaissance statistico-structurelle de symboles. Cette dernière vise à construire des représentations hybrides des symboles combinant à la fois des caractéristiques structurelles (graphe de voisinage) et statistiques (vecteurs de caractéristiques). Nous la présentons plus en détail au travers de la figure (1.6). Afin d'illustrer l'interopérabilité entre les opérateurs, durant les différentes étapes de cette reconnaissance, le tableau (1.9) synthétise l'évolution de la base de connaissances graphiques.

[10] Liste non exhaustive, d'autres opérateurs sont disponibles dans la librairie PSI (voir Annexe C).

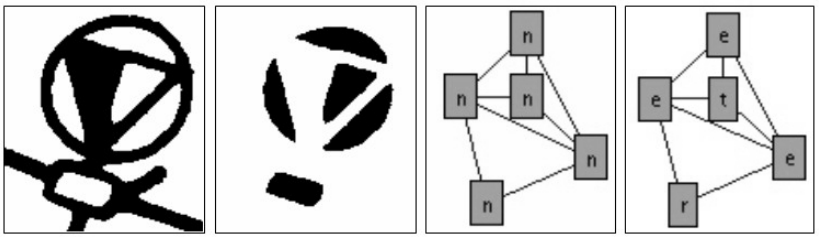



FIG. 1.6 – (a) symbole (b) marquage (c) voisinage/extraction (d) kppv

| **entrée** | **marquage** | **voisinage** | **extraction** | **kppv** |
|---|---|---|---|---|
| $i$ | $L \sqsupseteq o$ | $G \sqsupseteq (o \wedge v)$ | $G \sqsupseteq (t \wedge v)$ | $G \sqsupseteq (s \wedge v)$ |

TAB. 1.9 – Évolution de la base de connaissances graphiques (1)[11]

Dans une première étape la chaîne de reconnaissance de symboles effectue une étape d'extraction des occlusions (b) à partir d'image de symbole (a). À l'issue de cette étape la base de connaissances graphiques est uniquement composée d'une liste d'occlusions ($L \sqsupseteq o$). Deux opérateurs (c) sont alors tour à tour appliqués : celui de graphe de voisinage et un extracteur statistique. Celui de graphe de voisinage construit à partir des occlusions ($o$) extraites des graphes ($G$) des relations de voisinage ($v$). L'extracteur statistique vient lui traiter les occlusions ($o$) au sein des graphes de voisinage ($G$) afin de les substituer par des vecteurs de caractéristiques ($t$). Un dernier opérateur kppv (d) est alors utilisé afin de traiter les caractéristiques ($t$) extraites dans le but de reconnaître les différents étiquettes ($s$) des occlusions. L'interopérabilité dans ce cas d'usage est particulièrement illustrée sur la partie statistique (extraction et kppv) de la chaîne de reconnaissance. En effet, les informations structurelles (graphe et relation de voisinage) présentes dans la base de connaissances graphiques ont été traitées de façon "transparente" par les opérateurs extraction et kppv. La structuration des occlusions en graphe, ainsi que la présence des relations de voisinage, n'ont en aucun cas été prises en compte par ces opérateurs.

Notre deuxième cas d'usage concerne la vectorisation par approche squelette/contour. Cette vectorisation a pour but d'extraire en parallèle des vecteurs issus du squelette et des contours des formes. Nous la présentons plus en détail au travers de la figure (1.7) et du tableau (1.10) qui synthétise l'évolution de la base de connaissances graphiques durant la vectorisation.

[11] Ce tableau utilise nos notations de spécifications de contraintes (tableau (1.1) page 7).

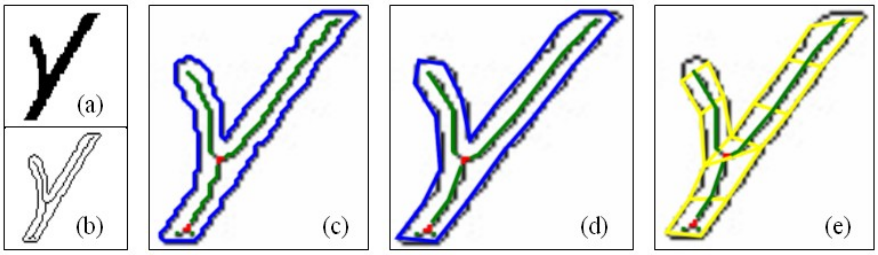



FIG. 1.7 – (a) symbole (b) contours/squelettisation (c) chaînage (d) polygonalisation (e) appariement

| **entrée** | **chaînage** | **polygonalisation** | **appariement** |
|---|---|---|---|
| $i$ | $G \sqsupseteq (j \wedge L \sqsupseteq p)$ | $G \sqsupseteq (j \wedge L \sqsupseteq l)$ | $G \sqsupseteq (j \wedge L \sqsupseteq l \wedge L \sqsupseteq q)$ |

TAB. 1.10 – Évolution de la base de connaissances graphiques (2)[11]

Dans un premier temps la chaîne de vectorisation effectue (b) des opérations jointes de détection morphologique[12] des contours et de squelettisation à partir d'image de plan (a). L'image résultante de ces opérations est ensuite traitée par deux opérateurs successifs de chaînage (c) et de polygonalisation (d). La base de connaissances graphiques est alors constituée d'un graphe ($G$) structurant des jonctions ($j$) et listes de lignes ($L \sqsupseteq l$). Un opérateur (e) vient alors apparier les contours par extraction de la base de connaissances graphiques des listes de lignes ($L \sqsupseteq l$) bouclées. Ces dernières sont alors substituées par des listes de quadrilatères ($L \sqsupseteq q$) dans la base. Cet opérateur ne traite ainsi qu'une partie des listes de lignes correspondant aux contours de l'image. De cette façon la présence d'objets graphiques issus du squelette n'a aucune incidence sur le déroulement de l'appariement de contours.

### 1.2.4 Conclusion

Dans cette section nous avons présenté notre approche pour la gestion des connaissances graphiques. Celle-ci repose sur un formalisme permettant la multi-représentation des objets graphiques. Celui-ci est basé sur des principes de conception orientée objet et exploite les propriétés de spécialisation et de polymorphisme. Nous représentons et opérationalisons ce formalisme à travers une librairie de modélisation utilisée au sein de différents opérateurs d'extraction. Ceux-ci utilisent alors des ensembles de spécifications de contraintes déterminant comment les objets graphiques doivent être extraits puis insérés dans la base de connaissances. Cette approche permet ainsi une interopérabilité entre les opérateurs et donc de plus larges combinaisons. Nous présentons dans la section suivante les principes généraux de notre approche de reconstruction d'objets basées sur l'exploitation de cette interopérabilité.

[12] Ces traitements sont détaillés dans notre librairie PSI (Annexe C).

## 1.3 Reconstruction d'objets : principes

Nous avons présenté précédemment notre formalisme et sa mise en oeuvre pour l'interopérabilité entre opérateurs d'extraction. Dans cette section nous développons les principes généraux de notre approche de reconstruction d'objets basée cette interopérabilité. Nous dressons tout d'abord dans la sous-section (1.3.1) le parallèle existant entre la planification de programmes (particulièrement la combinaison d'opérateurs) et la reconstruction d'objets. La sous-section (1.3.2) présente ensuite la notion de taxinomie de représentations inhérente à notre approche. Nous présentons alors dans la sous-section (1.3.3) une formalisation de la reconstruction d'objets. Finalement, dans la sous-section (1.3.4) nous concluons.

### 1.3.1 Combinaison d'opérateurs

Le combinaison d'opérateurs est utilisé dans les systèmes de planification de programmes[13] (ou opérateurs)[14] [Thonnat 95]. Ces derniers se sont principalement développés dans le courant des années 1990. Ils ont pour but d'automatiser les étapes nécessaires à la construction d'une chaîne d'opérateurs qui résout l'objectif d'un utilisateur, ainsi que de superviser l'exécution de cette chaîne [Moisan 00]. Ils ont été utilisés dans différents domaines d'application (automatisme, calcul scientifique, ingénierie logicielle, ...) mais plus particulièrement en traitement d'images (au sens vision). Différents travaux y sont d'ailleurs consacrés : [Clément 90], [Crubezy 95], [Dalle 98], [Clouard 99], [Cauchard 99], .... Ces systèmes utilisent l'image comme formalisme pivot afin de combiner les différents opérateurs. Les combinaisons sont généralement formalisées sous la forme de graphes[15] d'opérateurs, la figure (1.8) (a) en donne un exemple. Dans ces graphes les noeuds représentent les opérateurs et les arcs leurs combinaisons deux à deux. Ces arcs sont alors orientés, le sens de l'orientation définit l'ordre de la combinaison entre deux opérateurs successifs. Les successions de combinaisons dans le graphe forme alors des combinaisons plus larges. Une combinaison correspond donc à tout chemin dans le graphe d'opérateurs. La figure (1.8) (b) donne quelques exemples de combinaisons ($C_n$) obtenues à partir du graphe (a).

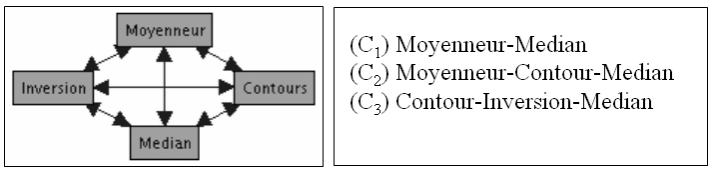



FIG. 1.8 – (a) graphe d'opérateurs (b) combinaisons

[13] Program supervision
[14] Dans le cadre de notre approche de reconstruction d'objets nos préférerons ce terme.
[15] Nous reportons le lecteur à l'Annexe A pour une introduction sur les graphes.

Le nombre de combinaisons est fonction du nombre d'opérateurs ainsi que de leurs relations. Les équations (1.10) et (1.11) donnent deux exemples de nombre de combinaisons $(c_1)$ et $(c_2)$ dans le cas d'un graphe complet d'opérateurs comme celui présenté sur la figure (1.8). Dans un graphe complet chaque noeud est connecté à tous les autres. Le graphe d'opérateurs étant orienté, le degré de chaque noeud est donc de $d = 2 \times (n-1)$ correspondant à $n-1$ arcs entrants et sortants. Le nombre d'arcs du graphe est donc de $m = 2 \times ((n-1)!)$. Le nombre de combinaisons $(c_1)$ ne considère lui que les chemins ne passant au plus qu'une seule fois par opérateur. Celui-ci est alors égal à la somme des arrangements de ces opérateurs. Le nombre de combinaisons $(c_2)$ considère tous les chemins possibles dans le graphe de longueur maximum $(n)$. Il est égal à la somme des puissances des demi degrés des noeuds. Dans le cas du graphe de la figure (1.8) composé de 4 opérateurs $(c_1)$ et $(c_2)$ sont alors respectivement égaux à 64 et 120. D'autres $(c_i)$ peuvent être calculées si on considère des boucles sur les noeuds des graphes et des chemins de longueurs supérieures à $(n)$.

$$\text{Soit } g \text{ un graphe orienté avec } n \text{ noeuds de degré } d = 2 \times (n-1) \tag{1.10}$$

$$c_1 = \sum_{p=2}^{n} A_n^p \qquad c_2 = \sum_{p=1}^{n} \sum_{q=1}^{n} \left(\frac{d}{2}\right)^q \tag{1.11}$$

Selon le nombre d'opérateurs et d'arcs présents dans les graphes le nombre de combinaisons peut donc devenir très important. Les systèmes de planification de traitements d'images ([Clément 90], [Crubezy 95], [Clouard 99], ...) utilisent généralement une "large" librairie d'opérateurs. C'est pour cette raison qu'ils ont recourt à une approche à base de connaissances pour la construction automatique[16] des graphes d'opérateurs. Les connaissances décrivent alors différents aspects : but de l'application, contexte des images, connaissances du domaine, expertise en traitement d'image, .... Les graphes générés sont alors désignés comme des plans dans la littérature ([Crubezy 95], [Dalle 98], [Clouard 99], ...). La figure (1.9) (b) donne un exemple (simple) de plan généré à partir d'une librairie d'opérateurs (a).

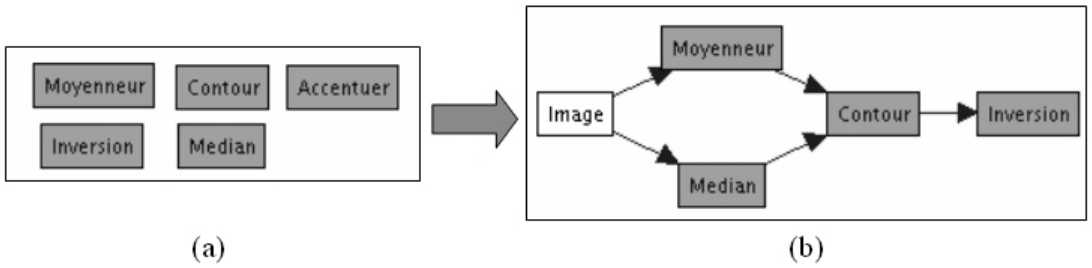



FIG. 1.9 – (a) librairie d'opérateurs (b) exemple de plan

[16] Nous introduisons ces aspects ici et reportons le lecteur aux références citées.

La combinaison d'opérateurs est également utilisée en analyse des documents [Antoine 92], [Bapst 97], [Saidali 02], [Adam 04], [Rendek 04], [Sánchez 04], .... La problématique diffère cependant majoritairement des systèmes de planification de traitements d'image (au sens vision). En effet, dans ce cas les représentations des connaissances varient (fortement) au fil des combinaisons. Ces variations rendent difficile la génération automatique de graphe d'opérateurs à partir de connaissances a priori [Saidali 02]. Les systèmes se limitent donc aux étapes d'acquisition des graphes d'opérateurs et de supervision de leur exécution. De même, ces variations posent un problème de cohérence de représentation du document. Différents systèmes [Adam 04] [Sánchez 04] utilisent pour cela un modèle[17] global de document. Le principe de la combinaison des opérateurs avec ce modèle est illustré sur la figure (1.10). Celui-ci sert de représentation commune au sein du système. Les différents opérateurs viennent alors tour à tour instancier ce modèle par de nouvelles primitives graphiques. De cette façon ce modèle permet une composition cohérente du document par les différents opérateurs.

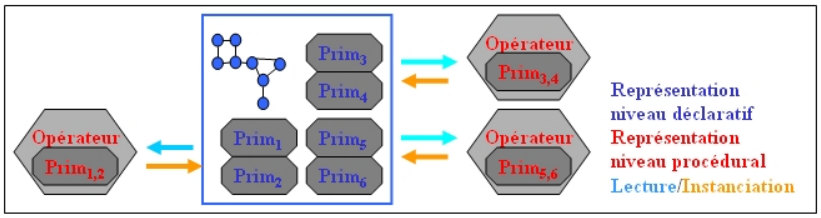



FIG. 1.10 – Combinaison d'opérateurs via un modèle global du document

Cependant dans ces systèmes l'évolution du modèle est "statique" : les opérateurs peuvent l'enrichir (par composition) mais ne peuvent le transformer (par spécialisation). Ils ne permettent donc pas de multi-représentation, le modèle utilisé sert plus de conteneur de données que de formalisme pivot. Les possibilités de combinaisons deviennent alors limitées dès que les représentations différent du formalisme image. La figure (1.11) donne un exemple de graphe d'opérateurs extrait du système de [Adam 04]. Dans ce système l'enchaînement d'un opérateur donné est contraint à un type de primitive graphique précis : image binaire, composante connexe, bloc de texte, ....

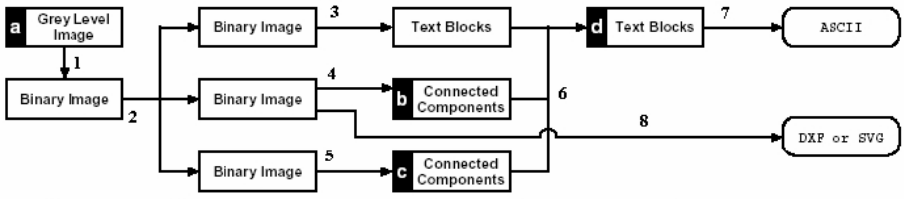



FIG. 1.11 – Graphe d'opérateurs de [Adam 04]

[17] Nous reportons le lecteur page ?? pour une définition du concept de modèle.

Notre approche de reconstruction d'objets est également basée sur la combinaison d'opérateurs. Cependant contrairement aux systèmes précédents notre approche diffère dans le fait que le modèle global du document évolue dynamiquement au fil des combinaisons. Cette capacité d'évolution du modèle est basée sur l'utilisation de notre formalisme objet. Celui-ci sert alors de pivot entre les opérateurs à l'aide de leurs différentes spécifications de contraintes. La figure (1.12) en illustre le principe. Les différents opérateurs viennent extraire les objets composant le modèle en cours. Celui-ci peut être ensuite reconstruit de différentes façons par : ajout/suppression, substitution/fusion, restructuration, .... Il y a donc modification dynamique du modèle global du document au fil des combinaisons.

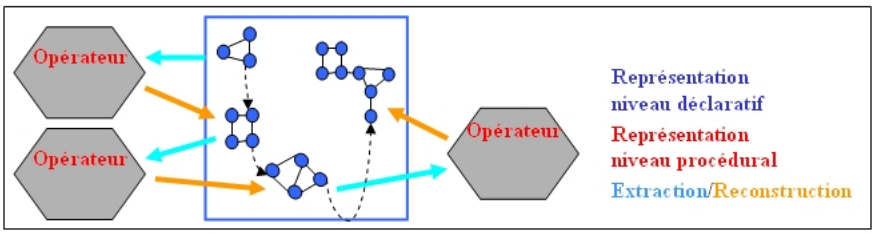



FIG. 1.12 – Combinaison d'opérateurs par évolution dynamique du modèle global

L'utilisation de notre approche de reconstruction d'objets nous assure des possibilités de combinaison plus larges que les systèmes présentés précédemment. Ces possibilités de combinaisons accrues sont liées au fait que nos opérateurs, rendus inter-opérables, peuvent reconstruire à la fois objets tout en modifiant au fil des combinaisons les modèles employés. Elles tendent donc à se rapprocher de celles des systèmes de planification de traitements d'image (au sens vision) où tous les opérateurs peuvent être combinés deux à deux. Cependant, contrairement à ces systèmes nos combinaisons sont soumises à trois contraintes principales : les opérateurs d'entrée, les combinaisons non effectives, et les combinaisons que nous nommons ici par antériorité. Nous développons ces contraintes par la suite.

Premièrement nos différentes combinaisons sont toujours amorcées par des opérateurs d'entrée. Ces derniers sont les opérateurs pouvant traiter les données d'entrée fournies au système (images ou données de plus haut niveau). Ensuite nos graphes d'opérateurs sont naturellement bornés par les spécifications de contraintes. Certaines combinaisons sont non effectives : la combinaison des opérateurs n'induira aucune modification dans la base de connaissances graphiques. Enfin, une combinaison peut être contrainte aux combinaisons précédemment effectuées : nous parlerons ici de combinaisons par antériorité. Celles-ci sont essentiellement liées aux opérateurs agissant sur les formalismes à base de graphes pour la représentation des connaissances graphiques. En effet, dans ce cas ces opérateurs peuvent "accumuler" des objets graphiques résultant d'opérateurs antérieurs.

La figure (1.13) (a) donne un exemple de graphe de nos opérateurs et illustre ces différentes contraintes. Celui-ci est composé de 4 noeuds (opérateurs) et 8 arcs (combinaisons entre deux opérateurs). Dans ce graphe les opérateurs *marquage* et *occlusions* peuvent s'exécuter sur les images d'entrée. Les combinaisons non effectives entre opérateurs n'ont pas été décrites dans ce graphe. La figure (b) exprime les différentes contraintes entre les opérateurs permettant de déterminer ces combinaisons non effectives. Par exemple l'opérateur *contours* est contraint aux résultats des opérateurs *marquage* et *occlusions*. La combinaison donc *contours* vers *occlusions* n'induira aucune modification de la base de connaissances graphiques. L'opérateur *voisinage* dans ce graphe met en oeuvre différentes combinaisons par antériorité. Par exemple, la combinaison {*marquage*, *voisinage*, *occlusion*} sera réalisable. Par contre la combinaison {*marquage*, *contours*, *voisinage*, *occlusion*} produira un résultat sans effet en ce qui concerne l'opérateur *occlusion*. En effet, la combinaison {*marquage*, *contours*} aura pour conséquence une substitution des objets composantes rendant une extraction postérieure des occlusions impossible par notre opérateur.

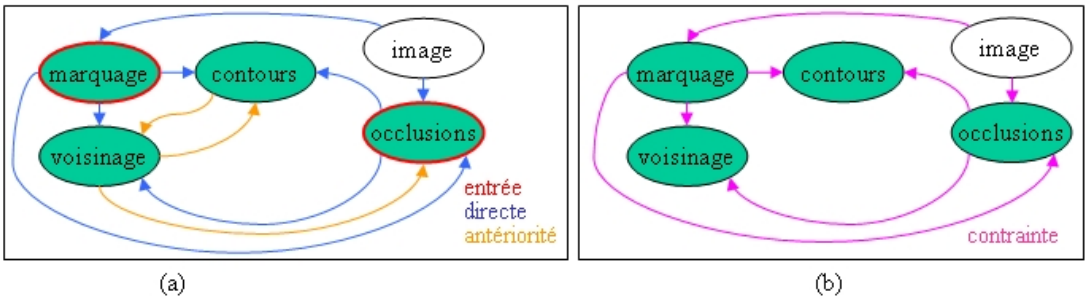



FIG. 1.13 – (a) entrée & antériorité (b) contraintes

Malgré ces différentes contraintes nos possibilités de combinaisons entre opérateurs restent importantes. La figure (1.14) les illustre en donnant notre graphe d'opérateurs. Ce dernier est composé de 9 noeuds (opérateurs) et de 47 arcs (combinaisons entre deux opérateurs). Il ne représente pas l'intégralité des opérateurs exploitables, il peut être encore étendu par les différents opérateurs[18] présentés dans le tableau (1.8) (page 12). Dans ce graphe trois opérateurs d'entrée peuvent s'exécuter sur l'image : *marquage*, *occlusions* et *squelettisation*. À partir de ces opérateurs d'entrée et des 47 arcs de nombreuses combinaisons (correspondant aux chemins dans le graphe)[19] peuvent donc être envisagées. Évidemment, plusieurs[20] de ces combinaisons seront non effectives car par antériorité. De façon à illustrer l'influence de ces combinaisons par antériorité la figure (1.15) présente les différentes contraintes entre nos opérateurs. Cependant malgré ces contraintes d'antériorité nos possibilités de combinaisons restent importantes. Nous les illustrons par la suite à travers quelques exemples.

[18] Nous le limitons ici aux opérateurs liés aux contributions de ce manuscrit.
[19] Ce calcul nécessite un algorithme de recherche des chemins [Bartholdi 99].
[20] Ce calcul nécessite de tenir compte de l'évolution des modèles lors de la recherche des chemins.

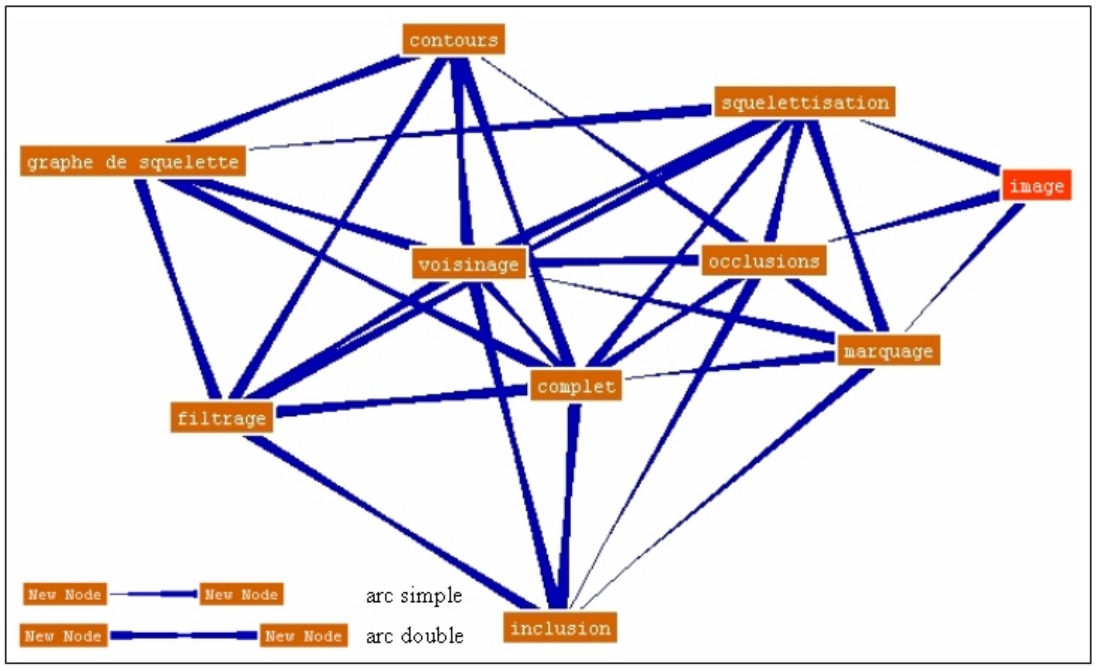



FIG. 1.14 – Graphe d'opérateurs[21,22]

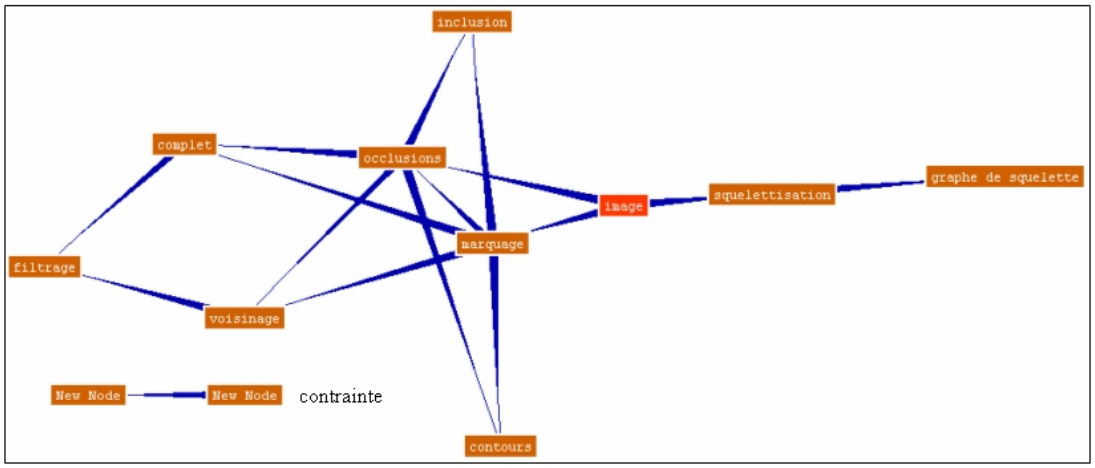



FIG. 1.15 – Contraintes des opérateurs[22]

[21] Ce graphe ne représente que les combinaisons effectives.
[22] Ce graphe a été réalisé via l'application TouchGraph : http ://www.touchgraph.com/

De façon à illustrer nos possibilités de combinaisons entre opérateurs, la figure (1.16) donne celles locales aux opérateurs *marquage* & *voisinage*. Ces combinaisons locales décrivent tous les noeuds adjacents à un noeud donné dans le graphe d'opérateurs. Il est ainsi possible de voir pour chacun des opérateurs les différentes combinaisons possibles.

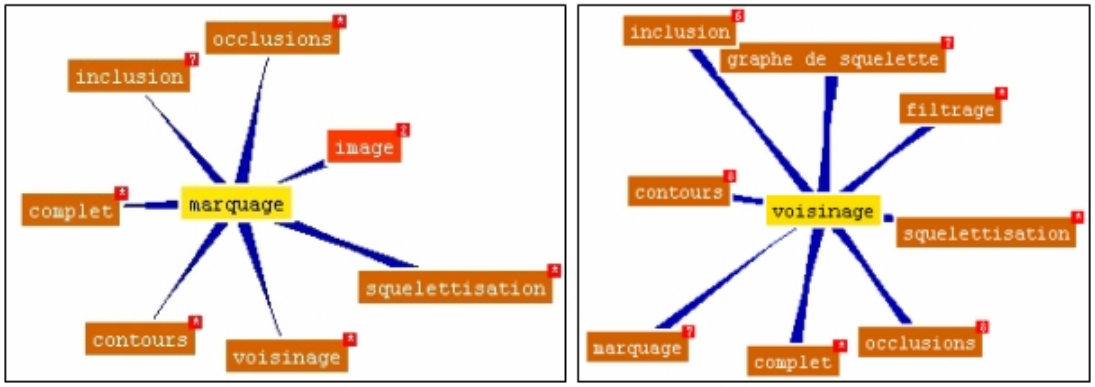



FIG. 1.16 – Combinaison des opérateurs marquage & voisinage[23]

Concernant l'opérateur *marquage* celui peut directement être exécuté à partir de l'image d'entrée ou de l'opérateur *squelettisation*. Suite à son exécution il peut être combiné à 6 opérateurs différents : *occlusions*, *inclusion*, *complet*, *contours*, *voisinage* et *squelettisation*. Parmi ces opérateurs seul l'opérateur *inclusion* est soumis à une combinaison par antériorité. En effet, il nécessaire de procéder au marquage des composantes et à l'extraction des occlusions, avant de construire le graphe d'inclusion.

Concernant l'opérateur *voisinage*, celui-ci peut être exécuté à partir de la totalité de ses opérateurs voisins. Cependant, plusieurs de ces opérateurs sont contraints à des combinaisons par antériorité afin de pouvoir l'exploiter a posteriori. Par exemple, l'opérateur *inclusion* ne doit pas présenter de relations de voisinage sur les occlusion et/ou sur les composantes. L'opérateur *graphe de squelette* lui ne doit avoir été exécuté que sur le fond ou sur la forme, de façon à ce que des objets raster soit encore présents pour l'extraction des relations de voisinage. Suite à son exécution l'opérateur *voisinage* peut être combiné avec la totalité de ces opérateurs à l'exception de l'opérateur *marquage*. En effet, ce dernier ne peut être exploité que sur une image d'entrée comme nous l'avons présenté précédemment. Parmi les opérateurs restants, seuls les opérateurs *inclusion* et *occlusions* sont soumis à des combinaisons par antériorité. L'opérateur *inclusion* ne pourra s'exécuter que si des occlusions ont été préalablement extraites, et inversement pour l'opérateur *occlusions*.

[23] Ce graphe a été réalisé via l'application TouchGraph : http ://www.touchgraph.com/

Au cours de cette sous-section nous avons illustré que notre approche de reconstruction d'objets s'apparentait à une combinaison d'opérateurs (au sens vision). Elle en diffère cependant majoritairement en raison des (fortes) variations des représentations des connaissances au cours de la reconstruction : celles-ci contraignent alors les possibilités de combinaisons. De façon à pallier ce problème, nous utilisons notre formalisme objet comme pivot entre les opérateurs au sein d'un système. Les opérateurs reconstruisent alors les objets tout en modifiant dynamiquement le modèle global du document au fil des combinaisons. Ceci nous permet de multiples possibilités de combinaisons malgré les contraintes liées aux variations des représentations. Cependant ces variations ne sont pas, à foncièrement parlé, des contraintes dans le cadre de notre approche de reconstruction d'objets. Elles sont, tout au contraire, une source précieuse d'information formalisable sous la forme de taxinomies de représentations. Nous introduisons cet aspect dans la sous-section suivante.

### 1.3.2 Taxinomie de représentations

La taxinomie[24] a été introduite au début du 19° siècle dans le cadre de la botanique[25] sous le terme de taxonomie[26,27]. Elle a par la suite dépassé ce cadre, on la définit aujourd'hui comme étant la discipline[28] de la classification d'objets (du monde), le plus souvent sous forme d'arbre[29], sur la base de leur similarité. La taxinomie a été appliquée dans différents domaines comme la biologie ou la linguistique. Dans le cadre de notre approche nous nous intéressons plus particulièrement à la taxinomie dite graphique. Cette dernière est une spécialisation de la taxinomie appliquée aux objets graphiques (ou symboles). Celle-ci est en fait antérieure à la discipline de la taxinomie. En effet, la taxinomie graphique est inhérente aux mathématiques, et en particulier à la géométrie qui est la discipline de base pour la conception de graphique [Wiebe 98]. La figure (1.17) donne un exemple de taxinomie graphique de symboles géométriques.

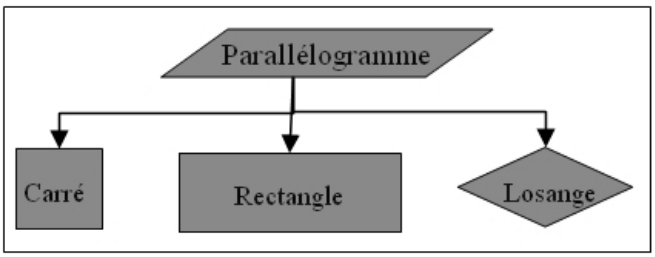



FIG. 1.17 – Taxinomie graphique de symboles géométriques

[24] Taxinomy
[25] A.P. de Candolle, théorie élémentaire de la botanique, 1813.
[26] Taxonomy
[27] La taxonomie est la taxinomie appliquée à la biologie.
[28] On emploie également le terme taxinomie pour désigner une classification spécifique.
[29] Nous reportons le lecteur à l'Annexe A pour une introduction sur les arbres.

La taxinomie graphique a été exploitée initialement en informatique dans le cadre des travaux sur les SIG[30] [Mark 89]. Différents modèles basés sur les formalismes des langages O.O et des schémas formalisent les relations entre objets géographiques (ville, maison, route, ...) sous forme de taxinomies [Egenhofer 92]. Les SIG exploitent alors ces taxinomies dans le but d'optimiser la recherche des objets géographiques au sein des bases de données.

Dans un autre contexte [Messmer 95] a exploité[31] la taxinomie graphique pour la reconnaissance de symboles sur des images de plan technique. La problématique est alors différente de celle des SIG. Il s'agit en effet de structurer la base d'apprentissage, selon une taxinomie graphique, afin de simplifier le problème de reconnaissance des symboles. La figure (1.18) donne un exemple de taxinomie employée par [Messmer 95] pour quatre symboles. Celle-ci se décompose en différents niveaux : de 0 à 5. Chacun des niveaux correspond à un échelon de la taxinomie, et le niveau 0 représente sa racine. [Messmer 95] base sa reconnaissance de symboles sur un modèle de type graphe de vecteurs. Chacun des noeuds dans la taxinomie correspond alors au graphe de vecteurs d'une classe d'un (ou d'une partie de) symbole. L'ordre (ou le nombre de vecteurs) de chacun de ces graphes est égal à la valeur du niveau (+1) auquel il est affecté (de 1 à 6). La taxinomie décrit alors les relations de composition entre les différents graphes de vecteurs : chaque graphe de niveau $n$ est composé de un ou plusieurs sous-graphe(s) des niveaux $\{0,\ldots,n-1\}$. Les symboles correspondant aux graphes d'ordres les plus importants se retrouvent dans les niveaux supérieurs de la taxinomie ($\{3, 4, 5\}$ pour l'exemple de la figure (1.18)).

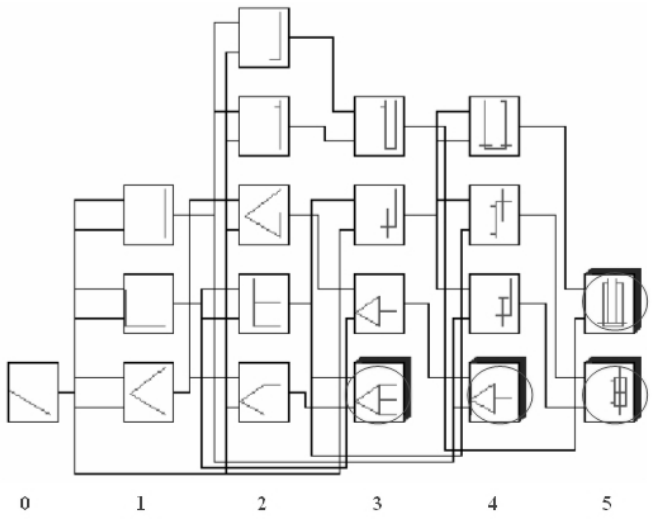



FIG. 1.18 – Réseau de [Messmer 95][32]

[30] Système(s) d'Information Géographique
[31] Son approche a été reprise dans les travaux de [Ah-Soon 01].
[32] Les symboles terminaux sont indiqués par des cercles.

[Messmer 95] exploite[33] ensuite ses bases d'apprentissage, structurées taxinomiquement, à partir d'un module d'appariement de graphes. Celui-ci apparie successivement un graphe d'entrée avec différents graphes d'une base. Chaque résultat d'appariement correspond alors à un cheminement dans la taxinomie, de la racine vers les niveaux supérieurs. Ces résultats permettent également de déterminer les sous-ensembles suivants de graphes à apparier : ces derniers correspondent alors aux noeuds adjacents à celui précédemment apparié. De cette manière, la structuration taxinomique de la base d'apprentissage permet de réduire la complexité de l'appariement, aussi bien en nombre de graphes comparés que dans la taille de ces graphes.

Nous introduisons ici une nouvelle forme de taxinomie graphique (inhérente à notre approche de reconstruction d'objets) dite de représentations. Celle-ci est parallèle à la taxinomie mise en oeuvre par [Messmer 95]. En effet, ce dernier procède par agrégation des différents objets graphiques tandis que nous procédons par spécialisation[34]. Une spécialisation correspond à une transformation d'un objet graphique par enrichissement du modèle le décrivant. La figure (1.19) illustre plus précisément les différences entre ces deux taxinomies. Celles-ci se formalisent toutes les deux sous la forme d'aborescences[35]. Chaque noeud y représente un objet (a) ou ensemble d'objets (b) graphique(s). Les arcs décrivent des relations d'agrégation (a) ou de spécialisation (b) entre ces objets. Dans la taxinomie de représentations chaque ensemble d'objets d'un niveau ($N_n$) de père commun au niveau ($N_{n-1}$) est représenté par un modèle identique. Les niveaux des arborescences ont donc différentes significations : nombre d'agrégations successives depuis l'objet racine (a), nombre de spécialisations successives depuis l'objet racine (b).

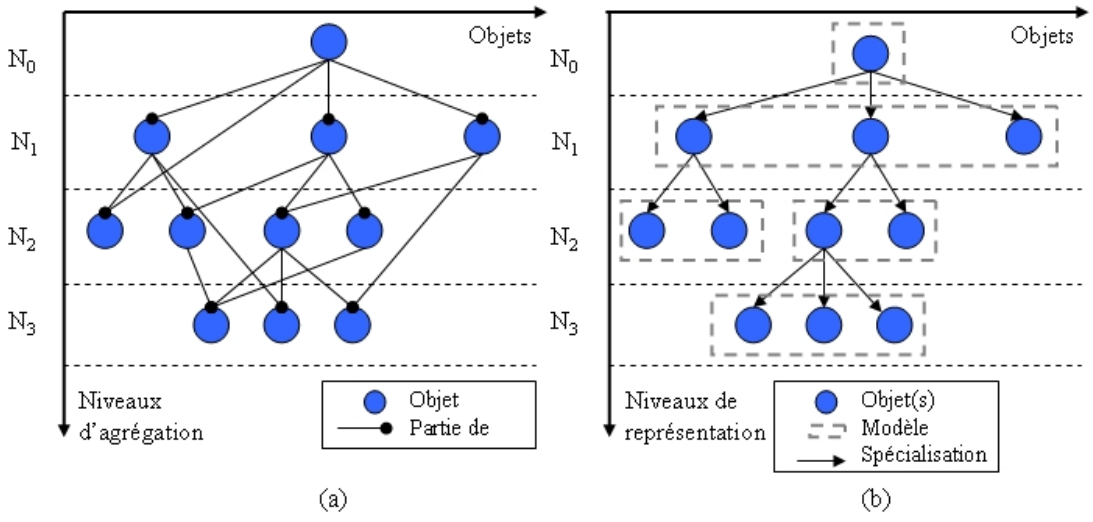



FIG. 1.19 – Taxinomie (a) de [Messmer 95] (b) de représentations

[33] Nous introduisons ces aspects ici et reportons le lecteur à [Messmer 95] pour plus de précisions.
[34] Nous reportons le lecteur à l'Annexe B pour une introduction sur les langages O.O
[35] Plus précisément d'un graphe simple orienté pour la taxinomie de [Messmer 95].

Afin d'illustrer plus clairement la notion de taxinomie de représentations nous présentons un exemple sur la figure (1.20). Celui-ci concerne 6 symboles $\{s_1,s_2,s_3,s_4,s_5,s_6\}$ décomposés en 3 niveaux de représentation. Le niveau $N_1$ repose sur un modèle de graphe de voisinage pour représenter les objets. Ce dernier est basé sur le marquage des composantes connexes des symboles, ainsi que sur l'extraction de leurs relations de voisinage. Les graphes construits selon ce modèle se répartissent en trois objets $o_1$, $o_2$ et $o_3$. Ces objets correspondent respectivement aux symboles $\{s_1\}$, $\{s_2,s_3\}$ et $\{s_4,s_5,s_6\}$. Le modèle graphe de voisinage permet donc la reconnaissance directe du symbole $\{s_1\}$. En effet, parmi les différents objets ($o_1$, $o_2$ et $o_3$) seul $o_1$ est univoque : c'est à dire une seule correspondance parmi l'ensemble des symboles.

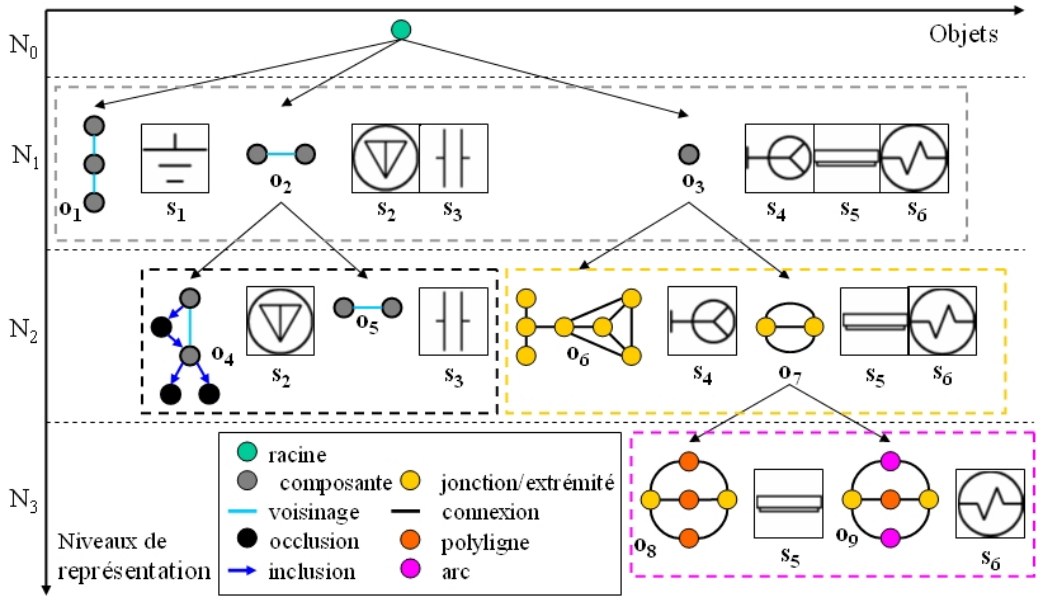



FIG. 1.20 – Exemple de taxinomie de représentations

Le niveau $N_2$ repose sur deux spécialisations du modèle graphe de voisinage. Dans un premier temps $o_2$ est spécialisé selon un modèle de graphe d'inclusion. Ce dernier est basé, sur l'extraction des occlusions des composantes des symboles, et sur la construction de leurs relations d'inclusion. Cette spécialisation permet ainsi de distinguer le symbole $\{s_2\}$ du symbole $\{s_3\}$ qui ne présente pas d'occlusions sur ses composantes. Dans un deuxième temps, $o_3$ est spécialisé selon un modèle de graphe de squelette. Ce dernier est basé sur une squelettisation des composantes des symboles suivi d'un chaînage. Il permet de distinguer les symboles $\{s_5,s_6\}$ du symbole $\{s_4\}$ qui présente une structure de jonctions remarquable. Le niveau $N_3$ repose sur une dernière spécialisation de $o_7$ selon un modèle de graphe de vecteurs. Ce dernier est basé sur une polygonalisation (vecteur et arc) des axes médians des squelettes chaînés. Cette spécialisation permet alors de distinguer le symbole $\{s_5\}$ du symbole $\{s_6\}$ par la détection des arcs composant les axes médians.

Au cours de cette sous-section nous avons présenté les notions de taxinomie puis de taxinomie graphique. Nous avons ensuite introduit une nouvelle forme de taxinomie graphique dite de représentations. Cette dernière est inhérente à notre approche de reconstruction d'objets. Elle met en oeuvre des relations de spécialisation entre les objets graphiques correspondant à leurs transformations par enrichissement des modèles les décrivant. Cette taxinomie formalise ainsi les liens existant entre les différentes représentations de ces objets. Un système peut alors en tirer partie afin d'assister la combinaison de ses opérateurs. Nous développons cet aspect dans la sous-section suivante en formalisant plus précisément notre approche de reconstruction d'objets.

### 1.3.3 Reconstruction d'objets : formalisation

Au cours des sous-sections précédentes nous avons introduit différentes considérations sur notre approche de reconstruction d'objets. Nous avons tout d'abord dressé le parallèle existant avec la combinaison d'opérateurs. Nous avons vu que la reconstruction d'objets en différait majoritairement en raison des (fortes) variations des représentations. Nous avons ensuite illustré que ces variations pouvaient se formaliser sous la forme de taxinomies de représentations. Celles-ci décrivent alors les relations de spécialisation entre les objets. Basé sur ces considérations nous nous proposons, dans cette sous-section, de formaliser notre approche de reconstruction d'objets. Nous argumentons ici que cette formalisation peut se faire au travers d'un graphe biparti que nous qualifierons de graphe de reconstruction. Ce dernier utilise alors deux classes de noeuds pour la représentation des opérateurs et des objets manipulés. Il s'agit donc d'une extension des graphes d'opérateurs, utilisés en traitement d'images (au sens vision), afin de tenir compte des variations de représentation. La figure (1.21) illustre ces différents aspects au travers d'un exemple (basé sur l'exemple de la figure (1.20) précédente) présentant un graphe d'opérateurs (a), un graphe d'objets (b) (la taxinomie de représentations) et le graphe de reconstruction correspondant (c).

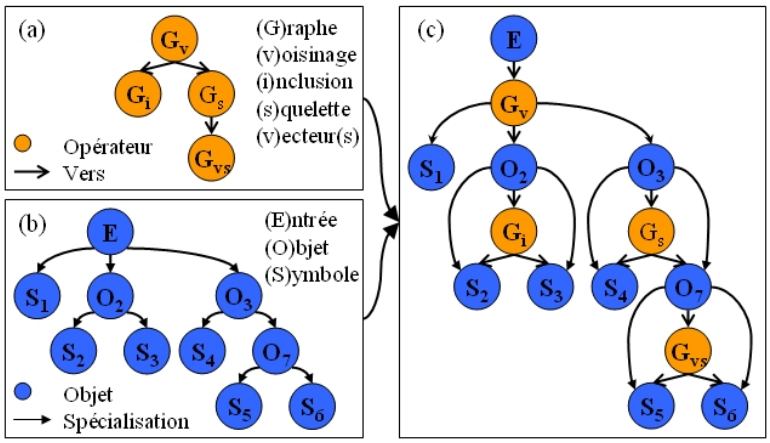



FIG. 1.21 – graphe (a) d'opérateurs (b) d'objets (c) de reconstruction

Le graphe de reconstruction présenté précédemment met en oeuvre des relations de spécialisation entre objets. Ces dernières sont induites par l'action des opérateurs sur les objets : nous parlerons ici de stratégie pour qualifier le comportement de ces opérateurs. Cependant d'autres stratégies sont envisageables selon la nature, le nombre et la configuration des opérateurs employés. Nous proposons ici une typologie de ces différentes stratégies sur la figure (1.22). Celle-ci est basée sur l'étude de différents systèmes (figés a priori)[36] existant dans la littérature : [Lam 95], [Hartog 96], [Okazaki 88], [Tombre 00], [Song 02], [Verma 03] et [Ramel 03]. À partir de cette étude nous avons dégagés cinq cinq types principaux de stratégie : spécialisation (a), simplification (b), composition (c), comparaison (d) et heuristique (e). Nous présentons chacune de ces stratégies par la suite.

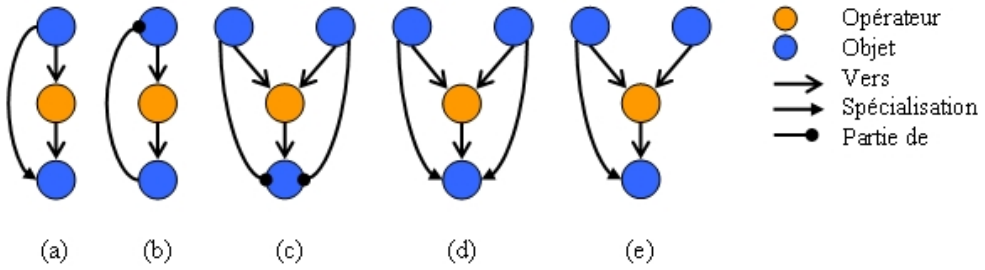



FIG. 1.22 – Typologie des stratégies d'opérateur(s)
(a) spécialisation (b) simplification (c) composition (d) comparaison (e) heuristique

La stratégie par spécialisation est basiquement utilisée par tout système d'analyse des documents graphiques. Elle consiste pour un opérateur à traiter des données d'entrée, afin d'en produire des données de sortie de plus haut niveau. Les opérateurs des systèmes de vectorisation par approche squelette/contours [Tombre 00] en sont des exemples types. La figure (1.23) en donne un exemple : les objets sont tour à tour spécialisés du raster (a) en liste de points (b), des polylignes en quadrilatères (c).

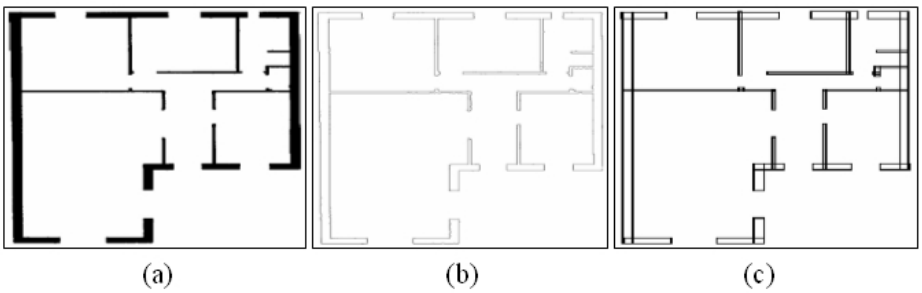
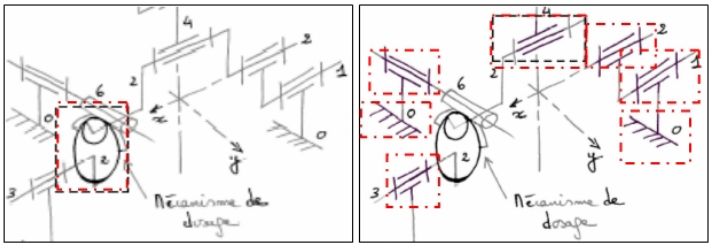



FIG. 1.23 – (a) image (b) contours (c) appariement

[36] Cette remarque est partiellement vrai selon les systèmes.

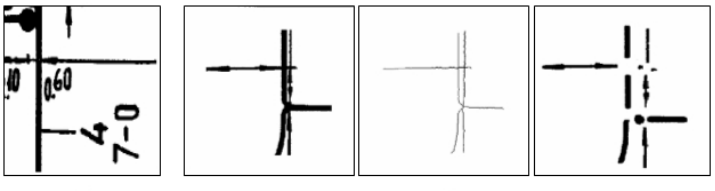
La stratégie par simplification a été introduite initialement par [Pavlidis 87]. Elle a été plus précisément formalisée dans les travaux de [Song 02] et [Ramel 03]. Celle-ci vise à simplifier progressivement les objets graphiques extraits, de façon à assister les différents opérateurs tour à tour. [Ramel 03] exploite cette stratégie dans son système pour la simplification progressive du document par cycle perceptif. La figure (1.24) donne des exemples de résultats obtenus par son système. Celui-ci travaille à partir d'une représentation (haut niveau) du document à base de quadrilatères. À partir de cette représentation son système emploie un ensemble de spécialistes (ou opérateurs) venant, tour à tour, simplifier le document. La figure (a) présente une première étape de détection des courbes. Celles-ci sont alors supprimées, le document simplifié est alors utilisé par le spécialiste de détection de symboles (b). [Song 02] utilise une approche similaire mais basée sur une simplification de l'image.



(a) (b)

FIG. 1.24 – (a) simplification courbe (b) simplification symbole

La stratégie par composition vise à fusionner les résultats de différents opérateurs. [Hartog 96] exploite cette stratégie dans son système, la figure (1.25) illustre son approche. À partir d'une image de plan (a) celui-ci effectue un marquage des composantes connexes suivi d'une séparation texte/graphique. Cette dernière opère par analyse des surfaces, les composantes de grandes tailles sont alors assimilées à des parties graphiques. Celles-ci sont par la suite segmentées (b) via un algorithme de squelettisation par transformée de distances. Le résultat de leur segmentation est ensuite ré-injecté (ou composé) aux composantes de faibles tailles initialement marquées. [Coüasnon 95] procède de façon équivalente dans son système. La composition se fait entre des objets lignes et composantes connexes extraits respectivement par transformée de [Kalman 60] et par marquage.



(a) (b)

FIG. 1.25 – (a) texte & graphique (b) extraction de primitives de la partie graphique

La stratégie par comparaison est proche de la notion de combinaison parallèle de classifieurs [Zouari 02]. Elle vise à exécuter différentes combinaisons d'opérateurs en parallèle dans le but de comparer leurs résultats. [Lam 95] par exemple dans son système extrait et compare différents graphes de squelette. Ces graphes sont basés sur différentes méthodes (dix en tout) de squelettisation parallèles. La figure (1.26) donne des exemples de résultats de ces différents squelettiseurs. Les graphes obtenus sont ensuite appariés par un classifieur structurel dans le but d'être comparés. [Nakajima 99] procède de manière similaire dans son système pour comparer deux squelettiseurs. Cependant, contrairement à [Lam 95], les deux squelettiseurs comparés sont basés sur des approches différentes. Le premier emploie une approche classique de squelettisation parallèle puis de chaînage du squelette. Le second lui utilise une approche par appariement de courbes issues des contours.

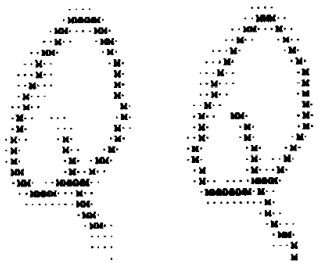


FIG. 1.26 – Combinaison d'opérateurs par comparaison

La stratégie par heuristique est proche de la notion de combinaison séquentielle de classifieurs [Zouari 02]. Elle vise à assister un opérateur à partir de données (ou heuristiques) extraites a priori par un tiers opérateur. [Verma 03] utilise cette stratégie pour la segmentation de mots manuscrits dans son système. L'architecture qu'il utilise est présentée sur la figure (1.27). Dans son système, il exploite l'étape de reconnaissance du cycle antérieur afin d'assister l'étape de segmentation du cycle postérieur. [Okazaki 88] procède de manière équivalente pour la reconnaissance de symboles. Il exploite les résultats de détection d'occlusions comme heuristiques afin de localiser des symboles dans les plans électriques. Cette localisation est par la suite exploitée par un algorithme de reconnaissance.

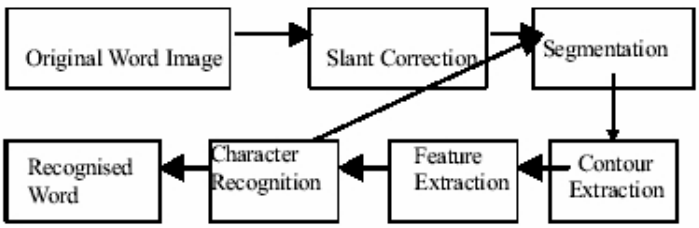



FIG. 1.27 – Combinaison d'opérateurs par heuristique

Les stratégies présentées précédemment sont les plus communes rencontrées dans la littérature. D'autres peuvent être envisagées selon les spécificités de chaque système. L'ensemble de ces stratégies permettent alors de définir des graphes de reconstruction d'objets complexes et propres à chacun des systèmes. Afin d'illustrer ces aspects nous présentons, dans la figure (1.28), un exemple de graphe de reconstruction défini à partir du système OOPSV[37] de [Song 02]. Celui-ci met en oeuvre une stratégie de simplification progressive de l'image. Il extrait successivement différentes primitives graphiques de l'image puis la simplifie par effacement de ces primitives. Il opère donc à la fois par stratégie de spécialisation, d'heuristique et de simplification. La figure (1.29) donne un exemple de résultat de son système. L'image (b) est d'abord obtenue suite à une simplification des traits de l'image initiale (a). L'image (c) est ensuite obtenue suite à une simplification des arcs de l'image (b).

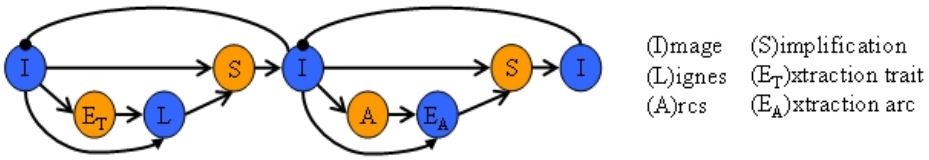



FIG. 1.28 – Graphe d'opérateurs du système de [Song 02]

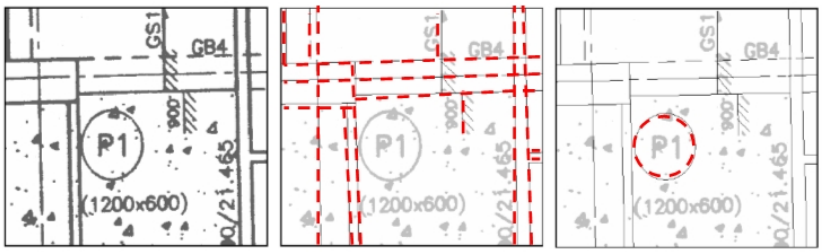



FIG. 1.29 – (a) image (b) simplification des traits (c) simplification des arcs

Dans cette sous-section nous avons proposé une formalisation de notre approche de reconstruction d'objets selon deux aspects. Nous avons tout d'abord illustré que celle-ci se traduisait au travers d'un graphe biparti qualifié de reconstruction. Celui-ci fait intervenir deux classes de noeud pour la représentation, des opérateurs et des objets, ainsi que de leurs relations. Nous avons ensuite illustré que les relations au sein de ces graphes étaient fonction des stratégies employées par les opérateurs. Nous avons alors proposé une typologie des principales stratégies employées dans la littérature. D'autres peuvent être envisagées selon les spécificités de chacun des systèmes. Il est ainsi possible d'utiliser ces différentes stratégies afin de définir des graphes de reconstruction complexes propres à chacun des systèmes.

[37] Object-Oriented Progressive-Simplification based Vectorization

### 1.3.4 Conclusion

Au cours de cette section nous avons présenté les principes généraux de notre approche de reconstruction d'objets. Nous avons tout d'abord dressé le parallèle existant avec la combinaison d'opérateurs. Nous avons vu que la reconstruction d'objets en différait majoritairement en raison des (fortes) variations de représentations mises en oeuvre. Nous avons ensuite illustré que ces variations pouvaient se formaliser sous la forme de taxinomies de représentations. Basé sur ces considérations, nous avons proposé une formalisation de notre approche de reconstruction d'objets en un graphe biparti de reconstruction. Dans la section suivante nous présentons le système de reconstruction d'objets que nous mettons en oeuvre basé sur ces différents principes.

## 1.4 Système de reconstruction d'objets

### 1.4.1 Introduction

Au cours des sections précédentes nous avons présenté notre formalisme objet, ainsi que les principes généraux de notre approche de reconstruction d'objets. Nous avons illustré tout d'abord les propriétés d'interopérabilité de nos opérateurs basés sur notre formalisme. Nous avons montré ensuite que notre approche de reconstruction d'objets s'apparentait à une combinaison d'opérateurs présentant de (fortes) variations de représentations des connaissances. Nous avons alors proposé de la formaliser au travers d'un graphe de reconstruction biparti pour la représentation jointe des opérateurs et des objets. Dans cette section nous présentons notre système de reconstruction d'objets. Son architecture générale est présentée sur la figure (1.30). Celui-ci est basé sur les principes généraux de notre approche de reconstruction d'objets. Pour ce faire il présente deux[38] caractéristiques principales : il utilise une méthode de modélisation orientée objet pour la formalisation des graphes de reconstruction, il exploite un moteur de contrôle couplé à une base de connaissances pour le pilotage automatique des opérateurs. Nous présentons respectivement chacune d'entre elles dans les sous-sections suivantes (1.4.2) et (1.4.3). Dans la sous-section (1.4.4) nous concluons sur notre système.

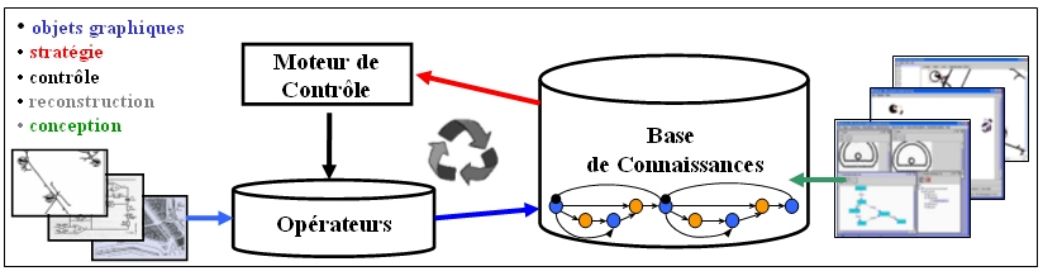



FIG. 1.30 – Architecture de notre système de reconstruction d'objets

[38] Ce système exploite également des interfaces spécialistes présentées en Annexe D.

### 1.4.2 La Méthode de modélisation orientée objet

Les méthodes de modélisation orientées objet (UML, OMT, ...)[39] sont couramment utilisées dans divers domaines pour la conception des systèmes d'information [André 01]. Parmi ces méthodes, nous nous sommes plus particulièrement intéressés à l'OPM[40] [Dori 02]. La principale propriété de l'OPM réside dans sa capacité à modéliser simultanément le comportement (les traitements) et les structures de données (les objets) d'un système. Elle se base pour cela sur un formalisme à base de graphes bipartis : les OPD[41]. Cette méthode et son formalisme associé se prêtent donc particulièrement bien à notre approche de reconstruction d'objets, nous avons donc décidé de les retenir pour la modélisation de nos graphes de reconstruction.

La figure (1.31) résume les notations principales[42] utilisées pour décrire les OPD. Ces derniers sont basés sur deux composantes (building blocks) : les objets (*Object*) et les traitements (*Process*(*ing*)). Différentes relations O.O[39] peuvent être exprimées entre ces composantes comme : l'héritage (*inheritance*), l'agrégation (*aggregation*), résultat (*result*) et déclenchement (*enable*) .... La figure (1.32) illustre ces notation en donnant un exemple d'OPD. Celui-ci peut s'interpréter de la manière suivante : "les objets graphe de région (*regiongraph*) et graphe de squelette (*skeletongraph*) sont spécialisés de l'objet graphe (*graph*), le traitement d'appariement (*matching*) change l'état d'un objet graphe d'inconnu (*unknown*) à reconnu (*recognized*)".

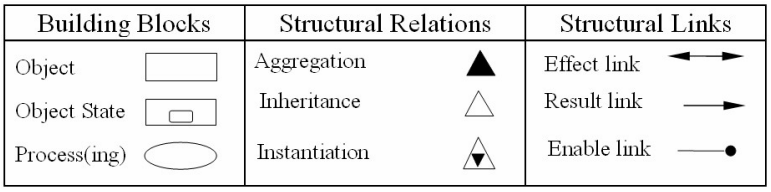



FIG. 1.31 – Notations OPM

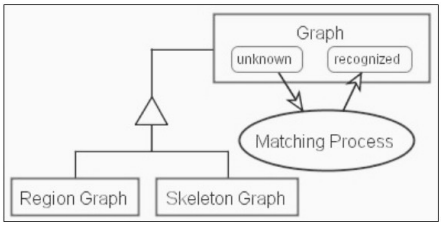



FIG. 1.32 – Exemple d'OPD[43]

[39] Nous reportons le lecteur à l'Annexe B pour une introduction à ces aspects.
[40] Object-Process Methodology
[41] Object-Process Diagram
[42] Nous introduisons l'OPM sommairement ici et reportons le lecteur à [Dori 02].
[43] Cet OPD a été réalisé via l'application Opcat : http ://objectprocess.org/

Basé sur les OPD, l'OPM est particulièrement adaptée pour la modélisation des systèmes d'analyse des documents graphiques [Dori 00]. Certains travaux récents tendent d'ailleurs à l'exploiter pour la définition de briques logicielles ré-utilisables [Wenyin 99]. Dans le cadre de notre système de reconstruction d'objets, nous avons décidé de la retenir pour la formalisation de nos graphes de reconstruction. Plus précisément, nous formalisons des OPD au travers des prédicats utilisés par notre moteur de contrôle. Nous détaillons ces aspects dans la sous-section suivante.

### 1.4.3 Moteur de contrôle

Nous présentons dans cette sous-section notre moteur de contrôle rsOPM[44] employé pour le pilotage automatique de nos opérateurs. Celui-ci est basé sur une approche de type systèmes experts [Farreny 89]. Ces derniers ont été largement utilisés dans des applications de recherche et industrielles. Ils sont basés sur des formalismes logiques, et plus particulièrement sur les prédicats, pour résoudre des problèmes de classification ou de décision. Il nous a donc paru naturel[45] de concevoir rsOPM comme tel pour le contrôle de nos opérateurs. La figure (1.33) présente l'architecture de rsOPM. Celui-ci est basé sur l'utilisation d'interfaces, d'un opérateur de contrôle et d'une base de connaissances (les règles). Les interfaces permettent l'encapsulation de nos différents opérateurs d'extraction de primitives graphiques et leur intégration dans rsOPM. L'opérateur de contrôle exploite un moteur d'inférence à partir de la base de règles (faits + prédicats) afin de piloter les opérateurs via les interfaces. La particularité de rsOPM réside dans la définition faite des prédicats. En effet, ceux-ci sont définis de façon à décrire implicitement un OPD correspondant à un graphe de reconstruction d'objets. Cet OPD est explicitement reconstruit durant le processus d'inférence. Nous détaillons par la suite ces différents aspects au travers des présentations : de l'opérateur de contrôle, des prédicats et faits, puis des interfaces.

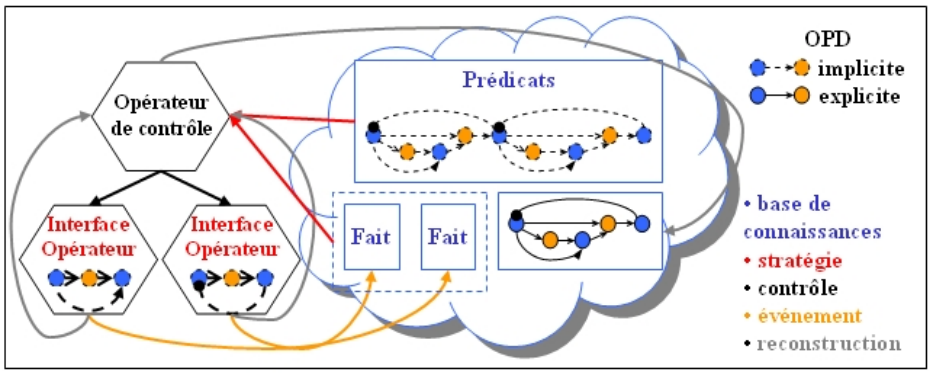



FIG. 1.33 – Architecture de rsOPM

[44] Rules based System for OPM

[45] Nos contributions s'inscrivent ici majoritairement sur l'exploitation faite des connaissances par le système, pas sur les mécanismes de contrôle mis en oeuvre.

L'opérateur de contrôle constitue l'élément central de rsOPM. Il permet d'inférer la base de règles et de charger dynamiquement les opérateurs via leurs interfaces. Il exploite pour cela le moteur Mandarax[46] basé sur l'utilisation du langage RuleML[47] [Wagner 02]. Ce langage utilise différents types de règles [Boley 01] mais plus particulièrement des prédicats[48]. Il est régi selon un triplet $\{P, F, Q\}$ où $\{P\}$ constitue les prédicats, $\{F\}$ les faits et $\{Q\}$ la requête. Le résultat de l'évaluation de $\{P, F\}$ par le moteur d'inférence, suite à la requête $\{Q\}$, détermine alors l'opérateur à exécuter. Les opérateurs sont donc exécutés cycliquement, tant qu'une configuration de faits $\{F\}$ vérifie un prédicat dans la base.

L'opérateur de contrôle est donc essentiellement régi par la base de règles $\{P, F\}$. Dans cette base les prédicats $\{P\}$ sont définis de façon à formaliser implicitement nos OPD de reconstruction d'objets. Pour cela, leur écriture respecte un quadruplet modulaire $\{P, O, S, R\}$. Dans ce quadruplet, $\{P\}$ est l'opérateur à exécuter, $\{O\}$ l'ensemble des objets sur lesquels s'exécute l'opérateur, $\{S\}$ l'ensemble des paramètres, et $\{R\}$ l'ensemble des règles (prédicats & faits) liées à l'opérateur. Ces règles liées sont celles sur lesquelles l'opérateur peut interagir dans la base (par ajout et/ou suppression et/ou modification). La figure (1.34) donne un exemple de prédicat utilisé dans rsOPM. Celui-ci se décompose en deux sections : *head* et *body*. La section *head* définit le quadruplet $\{P, O, S, R\}$ présenté précédemment. La section *body* renseigne le(s) fait(s) $\{F\}$ vérifiant le prédicat. Ces derniers correspondent aux différents états de reconstruction des objets (application antérieure d'opérateurs, évaluation du bruit, résultat de classification, ...). Le prédicat de la figure (1.34) peut donc s'interpréter de la façon suivante : *execute adaptProcess* $\{P\}$ *with* $\{$*imgObject* $\{O\}$, 0.3 $\{S\}$, *adaptRule* $\{R\}\}$ *if adapt image*. En langage naturel on peut le traduire ainsi : Exécuter le traitement d'adaptation sur l'image avec le paramètre 0.3 et mettre à jour la règle d'adaptation si le fait "adapter image" est avéré. Ces prédicats peuvent par la suite se développer, par le nombre de faits et leurs combinaisons logiques associées, et par le nombre d'éléments (ou variables) de chacun des éléments du triplet $\{O, S, R\}$.

```
<imp>
  <_head><atom>
    <_opr><rel>execute</rel></_opr>
    <ind>sgPSI.AdaptProcess</ind>}P
    <ind>imgObject</ind>}O
    <ind>0.3</ind>}S
    <ind>adaptRule</ind>}R
  </atom></_head>
  <_body><atom>
    <_opr><rel>adapt</rel></_opr>
    <ind>image</ind>
  </atom></_body>
</imp>
```

FIG. 1.34 – Exemple de prédicat utilisé dans rsOPM

[46] http ://sourceforge.net/projects/mandarax
[47] Rule Markup Language
[48] Nous reportons le lecteur à [Paulson 99] pour une introduction à la logique des prédicats.

L'opérateur de contrôle exploite ainsi la base de règles $\{P, F\}$ dans le but d'exécuter les opérateurs d'extraction de primitives graphiques. L'intégration de ces derniers dans rsOPM se fait via des interfaces. Celles-ci sont des sur-couches logicielles servant à l'encapsulation des opérateurs. Ces derniers peuvent être encapsulés individuellement au sein des interfaces, ou composés afin de constituer des opérateurs plus complexes. Les interfaces contrôlent également (selon les $\{O, S, R\}$ définis précédemment) les objets produits et utilisés par les opérateurs, ainsi que les interactions de ces opérateurs avec la base de règles $\{P, F\}$.

Nous venons de présenter ici les éléments centraux de rsOPM : l'opérateur de contrôle, la base de règles et les interfaces. Tout au long du processus de contrôle, différentes interactions sont mises en oeuvre entre ces éléments. Ces interactions constituent le cycle de reconstruction d'objets de notre approche, nous le détaillons sur la figure (1.35). Tout d'abord la base de règles $\{P, F\}$ est exploitée (**1**) pour déterminer l'opérateur à exécuter. Celle-ci à son état initial doit donc être définie afin d'amorcer le processus de reconstruction. Le résultat de son inférence, par l'opérateur de contrôle, détermine alors l'opérateur à exécuter. Différentes connaissances sont alors extraites de la base afin d'être transmises (**2**) à cet opérateur comme : des objets (l'image initiale, un objet en cours de construction ou antérieur, un objet résultat, ...), des paramètres (des seuils, des bases d'apprentissage, ...) et des règles (des prédicats $\{P\}$ et/ou des faits $\{F\}$ à écrire et/ou à supprimer de la base $\{P, F\}$). À l'issue de l'exécution de l'opérateur, différentes connaissances (objets, faits $\{F\}$, ...) sont stockées (et/ou supprimées et/ou mises à jour) (**3**) dans la base. Le cycle de reconstruction continue tant qu'une configuration de faits $\{F\}$ vérifie un prédicat pour l'exécution d'un opérateur.

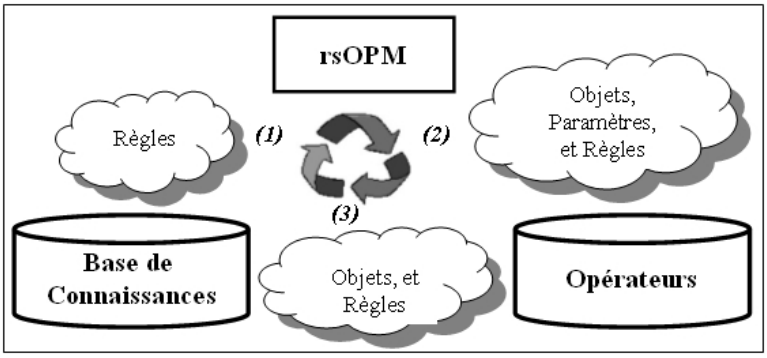



FIG. 1.35 – Cycle de reconstruction d'objets

Durant le cycle de reconstruction d'objets rsOPM reconstruit[49] également un OPD. Ce dernier représente alors explicitement le graphe de reconstruction employé par rsOPM pour le traitement de l'objet d'entrée. Nous illustrons plus en détail cette reconstruction au travers de l'exemple de la figure (1.36). Celle-ci est basée tout d'abord sur les

[49] Reconstruction basée sur l'utilisation de la librairie JGraphT : http ://jgrapht.sourceforge.net/

prédicats $\{P\}$ de la base de règles (a). Ceux-ci définissent en effet (de façon implicite) un OPD correspondant au graphe de reconstruction global de l'application. Tout au long du processus de reconstruction d'objets, cet OPD est parcouru au travers des différents résultats d'inférence de la base règles. Chaque étape de ce parcours correspond alors au chargement d'un opérateur via son interface. Chaque interface d'opérateur définie des règles de reconstruction de l'OPD fonction de l'opérateur qu'elle encapsule. Les figures (b) (d) (f) donnent des exemples de règles de trois opérateurs. La reconstruction de l'OPD s'effectue alors selon deux paramètres : le parcours de l'OPD de la base de règles (a) et la mise en correspondance de ce parcours avec les règles des opérateurs (b) (d) (f). Les figures (c) (e) (g) donne les différents états de reconstruction d'un OPD par trois opérateur successifs (b) (d) (f) à partir du graphe d'opérateurs (implicite) de la base de règles (a).

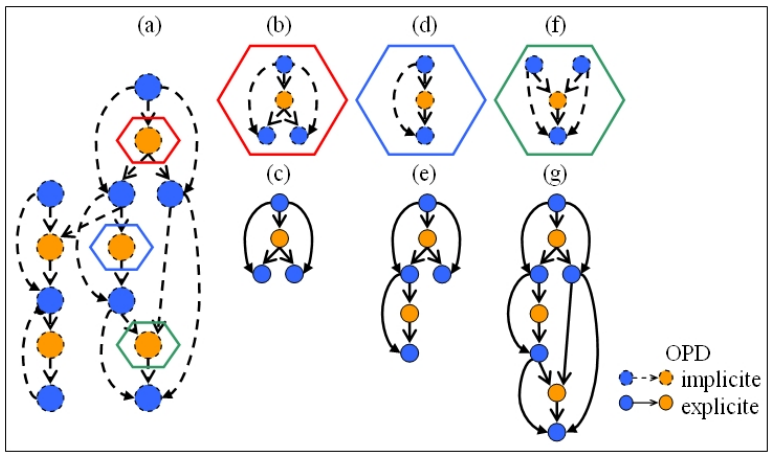



FIG. 1.36 – Reconstruction de l'OPD

Dans cette sous-section nous avons présenté notre moteur de contrôle rsOPM employé pour le pilotage automatique de nos opérateurs. Ce dernier est basé sur une approche de type systèmes experts. Il se compose de trois éléments principaux : l'opérateur de contrôle, la base de règles et les interfaces. Le premier est spécifique à rsOPM tandis que les deux autres sont liés à l'application mise en oeuvre. De cette façon rsOPM se veut le plus générique et adaptatif : la conception d'une application donnée se fait via les définitions de sa base de règles et interfaces associées. Nous avons illustré que la particularité de rsOPM était la définition des prédicats selon un quadruplet $\{P,O,S,R\}$. Cette définition permet ainsi de formaliser implicitement les OPD correspondant aux graphes de reconstruction d'objets. Nous avons également présenté les interactions mises en oeuvre entre les éléments principaux de rsOPM au travers du cycle de reconstruction d'objets. Nous avons enfin détaillé de quelle façon les OPD étaient reconstruits durant ce cycle à partir de la base de prédicats et des interfaces des opérateurs.

### 1.4.4 Conclusion

Dans cette section nous avons présenté notre système de reconstruction d'objets. Celui-ci est basé sur les principes généraux, dégagés au cours de la section précédente, de notre approche de reconstruction d'objets. Il utilise tout d'abord une méthode de modélisation orientée objet : l'OPM. Cette dernière, via le formalisme de graphe biparti des OPD, se prête particulièrement bien à notre approche de reconstruction d'objets. Nous utilisons ensuite un moteur de contrôle, basé sur une approche de type systèmes experts, pour le pilotage de nos opérateurs. De cette façon notre approche se veut générique et adaptative : la conception d'une application donnée se fait principalement via la définition de sa base de règles. Ce moteur formalise implicitement les OPD de reconstruction au travers des bases de prédicats. Ces derniers sont spécialisés afin de décrire ces OPD. Le moteur de contrôle reconstruit au cours du cycle de reconstruction d'objets l'OPD correspondant au chargement des opérateurs.

## 1.5 Conclusion

Dans ce chapitre nous avons présenté notre approche de reconstruction d'objets : du formalisme au système.

Nous avons tout d'abord présenté notre approche pour la gestion des connaissances graphiques. Celle-ci repose sur un formalisme orienté-objet permettant la multi-représentation des objets graphiques. Ce formalisme est opérationalisé au sein des opérateurs d'extraction de primitives graphiques. Ceux-ci utilisent alors des ensembles de spécifications de contraintes déterminant comment les objets graphiques doivent être extraits puis insérés dans la base de connaissances. Cette approche permet ainsi une interopérabilité entre les opérateurs et donc de plus larges combinaisons. Nous avons ensuite présenté les principes généraux de notre approche de reconstruction d'objets. Nous avons vu que la reconstruction d'objets différait majoritairement d'une combinaison d'opérateurs en raison des (fortes) variations des représentations. Nous avons ensuite illustré que les opérateurs, et les variations de représentation, pouvaient se formaliser sous la forme de graphe biparti. Nous avons alors présenté notre système de reconstruction d'objets basé sur ces principes généraux. Celui-ci utilise l'OPM, et plus particulièrement les OPD, pour la formalisation des graphes de reconstruction. Il met en oeuvre ces OPD via un moteur de contrôle, basé sur une approche de type systèmes experts, pour le pilotage de nos opérateurs. Ce dernier formalise implicitement les OPD de reconstruction au travers des bases de prédicats. De cette façon notre approche se veut générique et adaptative : la conception d'une application se fait principalement via la définition de sa base de règles. Le moteur de contrôle reconstruit alors, au cours du cycle de reconstruction d'objets, l'OPD correspondant au chargement des opérateurs.

Ainsi, les différentes connaissances dans notre système sont structurées selon une méthodologie de construction d'objets (formalisée sous la forme d'OPD) fonction de l'application envisagée. Cette structuration originale des connaissances permet au système, de simplifier le problème d'analyse des documents, et d'accroître ses performances de traitement. De plus, en raison des facilités de combinaison de nos opérateurs (interopérabilité), et celles de leur mise en oeuvre et contrôle (approche type systèmes experts), notre système présente différentes propriétés de généricité et d'adaptabilité. Nous illustrons ces différents aspects dans notre chapitre cas d'usage qui suit.

# Bibliographie


[Adam 04] E. Clavier S. Adam, P. Héroux, M. Rigamonti & J.M. Ogier. *Docmining : Une Plate-Forme de Conception de Systèmes d'Analyse de Documents.* In Colloque International Francophone sur l'Ecrit et le Document (CIFED), pages 97–102, 2004.

[Ah-Soon 01] C. Ah-Soon & K. Tombre. *Architectural Symbol Recognition Using a Network of Constraints.* Pattern Recognition Letters (PRL), vol. 22, no. 2, pages 231–248, 2001.

[André 01] P. André & A. Vailly. Conception des systèmes d'information. Editions Ellipses, ISBN : 272980479X, 2001.

[Antoine 92] D. Antoine, S. Collin & K. Tombre. *Analysis of Technical Documents : The REDRAW System.* Structured Document Image Analysis, pages 385–402, 1992.

[Bapst 97] F. Bapst, R. Grugger, A. Zramdini & R. Ingold. *A Scenario Model Advocating User-Driven Adaptive Document Recognition Systems.* In International Conference Document Analysis and Recognition (ICDAR), pages 745–748, 1997.

[Bartholdi 99] L. Bartholdi. *Counting Paths in Graphs.* Enseignement Math, vol. 45, pages 83–131, 1999.

[Boley 01] H. Boley, S. Tabet & G. Wagner. *Design Rationale of RuleML : A Markup Language for Semantic Web Rules.* In Semantic Web Working Symposium (SWWS), 2001.

[Cauchard 99] V. Ficet Cauchard, C. Porquet & M. Revenu. *CBR for the Management and Reuse of Image Processing Expertise : a Conversational System.* Engineering Applications of Artificial Intelligence, vol. 12, no. 6, pages 735–747, 1999.

[Clément 90] V. Clément & M. Thonnat. *Integration of Image Processing Procedures, OCAPI : a Knowledge-Based Approach.* Rapport technique RR-1307, INRIA, Sophia Antipolis, France, 1990.

[Clouard 99] R. Clouard, A. Elmoataz, C. Porquet & M. Revenu. *Borg : A Knowledge Based System for Automatic Generation of Image Processing Programs.* Pattern Analysis and Machine Intelligence (PAMI), vol. 21, no. 2, pages 128–144, 1999.

[Coüasnon 95] B. Coüasnon. *Segmentation et Reconnaissance de Document Guidée par la Connaissance à Priori : Application aux Partitions Musiclales.* Thèse de Doctorat, Université de Rennes, France, 1995.

[Crubezy 95] M. Crubezy, S. Moisan, F. Aubry, J. Elst & V. Chameroy. *Program Supervision in Medical Imagery.* In Australian Joint Conference on Artificial Intelligence (AI), pages 32–40, 1995.

[Dalle 98] P. Dalle & P. DeJean. *Planification en Traitement d'Images : Approche Basée sur Les Données.* In Congrès Francophone de Reconnaissance de Formes et Intelligence Artificielle (RFIA), volume 2, pages 75–84, 1998.

[Dori 00] D. Dori. *Syntactic and Semantic Graphics Recognition : The Role of the Object-Process Methodology.* In Workshop on Graphics Recognition (GREC), volume 1941 of *Lecture Notes in Computer Science (LNCS)*, pages 277–287, 2000.

[Dori 02] D. Dori. Object-process methodology, a holistic systems paradigm. Springer Verlag Publisher, ISBN : 3540654712, 2002.

[Egenhofer 92] M. Egenhofer & A. Frank. *Object-Oriented Modeling for GIS.* Journal of the Urban and Regional Information Systems Association, vol. 4, no. 2, pages 4–19, 1992.

[Farreny 89] H. Farreny. Les systèmes experts, principes et exemples. Editions Cepadues, ISBN : 2854281306, 1989.

[Hartog 96] J.E. Den Hartog. *Knowledge Based Interpretation of Utility Maps.* Computer Vision and Image Understanding (CVIU), vol. 63, no. 1, pages 105–117, 1996.

[Kalman 60] R.E. Kalman. *A New Approach to Linear Filtering and Prediction Problems.* ASME Journal of Basic Engineering, pages 35–45, 1960.

[Lam 95] L. Lam & C.Y. Suen. *An Evaluation of Parallel Thinning Algorithms for Character Recognition.* Pattern Analysis and Machine Intelligence (PAMI), vol. 17, no. 9, pages 914–919, 1995.

[Mark 89] D. Mark & al. *Working Bibliography on "Languages of Spatial Relations".* Rapport technique 89-10, National Center for Geographic Information and Analysis, Univeristy of California, Santa Barbara, USA, 1989.

[Messmer 95] B.T. Messmer. *Efficient Graph Matching Algorithms for Preprocessed Model Graphs.* Thèse de Doctorat, Bern University, Switzerland, 1995.

[Moisan 00] S. Moisan & D. Ziébelin. *Résolution de Problèmes en Pilotage de Programmes.* In Congrès Francophone de Reconnaissance de Formes et Intelligence Artificielle (RFIA), pages 387–396, 2000.

[Nakajima 99] Y. Nakajima, S. Mori, S. Takegami & S. Sato. *Global Methods for Stroke Segmentation.* International Journal on Document Analysis and Recognition (IJDAR), vol. 2, pages 19–23, 1999.

[Okazaki 88] A. Okazaki, S. Tsunekawa, T. Kondo, K. Mori & E. Kawamoto. *An Automatic Circuit Diagram Reader with Loop-Structure-Based Symbol Recognition.* Pattern Analysis and Machine Intelligence (PAMI), vol. 10, no. 3, pages 331–341, 1988.

[Paulson 99] L.C. Paulson. *Logic and Proof.* Rapport technique, University of Cambridge, England, 1999.

[Pavlidis 87] T. Pavlidis. *Hybrid Vectorization Algorithm.* In International Conference on Pattern Recognition (ICPR), pages 490–492, 1987.

[Ramel 03] J.Y. Ramel & N. Vincent. *Strategies for Line Drawing Understanding.* In Workshop on Graphics Recognition (GREC), volume 3088 of *Lecture Notes in Computer Science (LNCS)*, pages 1–12, 2003.

[Rendek 04] J. Rendek, G. Masini, P. Dosch & K. Tombre. *The Search for Genericity in Graphics Recognition Applications : Design Issues of the Qgar Software System.* In Lecture Notes in Computer Science (LNCS), volume 3163, pages 366–377, 2004.

[Saidali 02] Y. Saidali. *Modélisation et Acquisition Incrémentale de Connaissances Traiteurs d'Images*. Thèse de Doctorat, Université de Rouen, France, 2002.

[Sánchez 04] G. Sánchez, E. Valveny, J. Lladós, J. Mas & N. Lozano. *A Platform to Extract Knowledge from Graphic Documents. Application to an Architectural Sketch Understanding Scenario*. In Workshop on Document Analysis Systems (DAS), volume 3163 of *Lecture Notes in Computer Science (LNCS)*, pages 389–400, 2004.

[Song 02] J. Song, F. Su, C. Tai & S. Cai. *An Object-Oriented Progressive-Simplification based Vectorization System for Engineering Drawings : Model, Algorithm and Performance*. Pattern Analysis and Machine Intelligence (PAMI), vol. 24, no. 8, pages 1048–1060, 2002.

[Thonnat 95] M. Thonnat & S. Moisan. *Knowledge Based System for Program Supervision*. In Workshop on Knowledge-Based systems for the (re)Use of Program Libraries, pages 4–8, 1995.

[Tombre 00] K. Tombre & S. Tabbone. *Vectorization in Graphics Recognition : To Thin or not to Thin*. In International Conference on Pattern Recognition (ICPR), volume 2, pages 91–96, 2000.

[Verma 03] B. Verma. *A Contour Code Feature Based Segmentation For Handwriting Recognition*. In International Document Analysis and Recognition (ICDAR), pages 1203– 1207, 2003.

[Wagner 02] G. Wagner. *How to Design a General Rule Markup Language*. In Workshop on XML technologien für das Semantic Web (XSW), pages 19–37, 2002.

[Wenyin 99] L. Wenyin & D. Dori. *Object-Process Based Graphics Recognition Class Library : Principles and Applications*. Software : Practice and Experience (SPE), vol. 15, no. 29, pages 1–24, 1999.

[Wiebe 98] E.N. Wiebe. *The Taxonomy of Geometry and Graphics*. Journal for Geometry and Graphics, vol. 2, no. 2, pages 189–195, 1998.

[Zouari 02] H. Zouari, L. Heutte, Y Lecourtier & A. Alimi. *Un Panorama Des Méthodes de Combinaison de Classifieurs en Reconnaissance de Formes*. In Congrès Francophone de Reconnaissance de Formes et Intelligence Artificielle (RFIA), pages 499–508, 2002.

# Table des figures

# Liste des tableaux

# Liste des pseudo-algorithmes

# Table des matières