UNIVERSITÉ DU QUÉBEC À MONTRÉAL

UN SYSTÈME DE RECONNAISSANCE DES ESQUISSES DE MODÈLES CMMN DESSINÉES À LA MAIN

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

SARA AMIRSARDARI

JANVIER 2018

REMERCIEMENTS

Je suis très heureuse d'exprimer ma gratitude à tous ceux qui m'ont soutenu et qui ont contribué à rendre ce travail possible.

Tout d'abord et avant tout, Je veux remercier mon conseiller, Professeur Hafedh Mili, pour ses conseils, son soutien, sa motivation, son inspiration, son enthousiasme et son immense savoir.

Je veux également remercier Renata Carvalho, deuxième lectrice de mon mémoire, et je lui suis reconnaissant pour ses commentaires très précieux.

Je suis également reconnaissante envers mes amis et collègues pour fournir un environnement stimulant dans lequel je pourrais apprendre et grandir, surtout je remercie mes amis, Imen, Anis, Amani et Golrokh.

Je veux remercier tous les membres du personnel du département d'informatique à l'UQAM pour leurs assistants directs et indirects durant mes études à l'UQAM.

Enfin et surtout, Je voudrais exprimer ma très profonde gratitude à mes parents, Mohammad et Akram, et à Marco pour m'avoir fourni un soutien sans faille pour leur amour, leurs encouragements, leurs conseils tout au long de mes années d'études et à travers le processus de recherche et d'écriture de ce mémoire. Cette réalisation n'aurait pas été possible sans eux. Je vous remercie.

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

RÉSUMÉ

Les premières activités de spécification des exigences nécessitent une approche de modélisation plus flexible par rapport à celle fournie par les outils de modélisation traditionnels. Il existe une variété d'outils de modélisation pour capturer les processus métier sous une forme structurée. En dépit de leurs avantages, de tels outils ne sont pas naturels pour l'utilisateur humain et ne sont pas utilisés dans les premières étapes de modélisation et de développement. En comparaison avec d'autres approches flexibles, telles que les outils Office et le tableau blanc qui sont fréquemment utilisés dans les premières étapes de modélisation des systèmes à cause de leur utilisation plus naturelle par l'humain. Néanmoins, ces outils informels offrent plus de flexibilité et de liberté au détriment de la gestion de la consistance, la gestion des changements et l'interchangeabilité des modèles.

Étant donné que ni les approches flexibles ni les outils de modélisation traditionnels sont idéals, nous proposons dans ce mémoire une nouvelle approche intermédiaire dans le but de réduire l'écart entre ces deux approches. Nous proposons un outil qui reconnaît des esquisses des modèles CMMN faits à la main et les transforme en un format qui peut être importé par un outil formel. Dans cette approche, nous utilisons la reconnaissance des formes et la correspondance des patrons pour reconnaître les modèles CMMN faits à la main et les traduire en des modèles CMMN formels. Par la suite, ces modèles formels sont sérialisés dans un fichier XML conforme au format d'échange des modèles CMMN. Ce fichier peut être importé dans un outil CMMN conforme. L'efficacité de notre approche a été testée sur plus de 500 dessins faits à la main. Les résultats confirment l'efficacité de notre approche.

Mots clés : pré-analyse des exigences, modélisation formelle, approches de formes libres, esquisses faites à la main, traitement d'images, vision par ordinateur, modèles CMMN, outils de modélisation CMMN.

INTRODUCTION

Avant l'invention des ordinateurs, le papier et le crayon étaient indispensables dans la réalisation de différentes activités. En effet, les feuilles de papiers ou les tableaux blancs simples étaient parmi les premiers moyens utilisés pour documenter les idées. De nos jours, avec l'évolution technologique dans le domaine de l'informatique, il y a de plus en plus une tendance à utiliser des appareils numériques au lieu du papier traditionnel ou des tableaux blancs. L'invention des ordinateurs et des accessoires, comme la souris, le clavier et le moniteur, a présenté une alternative très appropriée pour les outils primaires. Les feuilles de papier et les outils à base de crayon étaient, et sont toujours utilisés dans l'étape de pré-exigences du cycle de vie du développement logiciel.

La plupart des activités au cours du cycle de développement du logiciel impliquent un processus qui garantit la construction d'un bon logiciel. L'ingénierie des exigences est considérée comme une phase essentielle de ce processus (Chakraborty *et al.*, 2012). Avant d'entrer dans cette phase critique, une phase de pré-exigences est exécutée afin de déterminer si un nouveau développement est requis (Ossher *et al.*, 2010). «Dans cette phase, avant que les exigences ne soient formulées, un analyste métier a besoin de collecter des informations, les organiser , envisager des futurs alternatifs et présenter des idées et des recommandations aux parties prenantes» (Ossher *et al.*, 2010).

Par conséquent, les analystes métier doivent utiliser une manière simple d'interagir avec les parties prenantes afin d'expliquer la structure, les politiques, les problèmes, les besoins, les exigences et les opportunités d'amélioration à tous les niveaux d'un projet.

Toutefois, actuellement, avec la popularité croissante de la "technologie d'écran tactile", tous les types d'utilisateurs, y compris les analystes commerciaux, ont généralement la possibilité d'entrer des informations directement à l'écran en utilisant leurs doigts ou un stylo plutôt que d'utiliser une souris. Ils peuvent également faire des choses comme défiler une liste ou déplacer des objets sur l'écran, les agrandir et les rétrécir. Par conséquent, en ayant une tablette, et en utilisant des outils bureautiques tels que des outils de traitements de texte, de dessin ou de présentation, les analystes pourraient interagir avec les parties prenantes beaucoup plus facilement lors des premières étapes de l'ingénierie des exigences. En ce qui concerne cette interaction, elle n'est pas statique, car les besoins et les demandes sont modifiables et les analystes. Les parties prenantes peuvent modifier ou supprimer des parties, comme ils peuvent créer ou étendre d'autres.

0.1 Problématique

Les approches flexibles telles que l'utilisation des feuilles de papier et du crayon, des tableaux blancs ou des outils bureautiques, généralement connues sous le nom d'approches "free-form" (Ossher *et al.*, 2011), sont utilisées au début du système de modélisation pour avoir des dessins à main libre ou des écritures (esquisses). Selon (Coyette *et al.*, 2007), les approches libres ont plusieurs avantages tels que:

- Le dessinateur est libre de l'adopter comme une approche à n'importe quelle étape de la conception, sans qu'il aura besoin de suivre une chaîne d'étapes spécifiques ou un cadre bien défini (Newman *et al.*, 2003);
- Le dessinateur n'a pas besoin d'un cours de formation pour dessiner ou écrire des esquisses, et le résultat peut être produit rapidement (Duyne *et al.*, 2002);
- Cette approche permet au dessinateur de se concentrer sur des problèmes structurels de base, plutôt que sur des détails triviaux (alignement exact, typographie et couleurs, par exemple) (Landay et Myers, 2001);

- Cette approche encourage la créativité et permet au dessinateur d'élaborer ses idées sur le papier sans aucune limitation (Landay et Myers, 2001), et

- L'exécution collaborative des esquisses entre l'analyste métier et les parties prenantes, permet d'améliorer la conception à travers les discussions et la modification continue des esquisses et des annotations. (Plimmer et Apperley, 2003b).

Malgré ses forces, les approches "free-form" ont également des faiblesses. En utilisant ces approches, la forme structurée de l'information documentée ne serait pas disponible et les possibilités de changement et de post-traitement pourraient être limitées. Ainsi, il est nécessaire de mettre du temps et de remodeler les esquisses précoces pour faire d'autres modifications possibles (Wüest *et al.*, 2012).

Contrairement aux approches "free-form", c'est "**formal modeling tool**" qui utilise un modèle de processus métier sous une forme structurée (Ossher *et al.*, 2011). La modélisation formelle est plus artificielle pour les humains et plus compréhensible a les appareils. Ils ont une variété d'avantages tels que:

- « Soutenir plusieurs vues sur le même modèle pour la visualisation et la commodité de manipulation » (Ossher *et al.*, 2010);

- « Faciliter la gestion de la cohérence du modèle » (Ossher *et al.*, 2010);
- « Fournir une assistance spécifique au domaine (par exemple, "assistance au contenu") basée sur la structure du modèle » (Ossher *et al.*, 2010);

- Préparer la documentation des décisions de modélisation (par exemple, raisons) (Ossher *et al.*, 2011);

- « Fournir la syntaxe, le modèle sémantique et la cartographie sémantique » (Ossher *et al.*, 2010), et

- « Intégration avec d'autres outils et processus formels, tels que l'ingénierie dirigée par les modèles (MDE) et la vérification des modèles » (Ossher *et al.*, 2010).

Malgré ces avantages, les outils de modélisation formelle ne sont pas efficaces dans les premières étapes de la modélisation et du développement. En effet, il est commun d'utiliser des mécanismes informels tels que des approches "free-form" (Ossher *et al.*, 2011).

(Ossher *et al.*, 2010), a soutenu que l'entrée qu'ils ont reçue de nombreux pratiquants indique, clairement, que ni la modélisation informelle ni la modélisation formelle sont idéales. Dans ce contexte, nous croyons qu'une nouvelle classe d'outils est nécessaire pour réduire l'écart entre les deux approches. Ces approches intermédiaires devraient être capables de gérer des esquisses qui sont produites dans les premières étapes de la modélisation, et de transférer ces esquisses informelles dans des modèles formels. Cela permettrait aux analystes métier de migrer facilement vers un outil formel dans une étape ultérieure.

0.2 Objectifs

Le but de ce mémoire est de développer un outil qui migre des esquisses dessinées à la main de modèles CMMN (OMG) vers des modèles formels qui peuvent être importés dans un outil de modélisation CMMN. CMMN est une norme OMG pour représenter des modèles de cas, c.-à-d., modèles de processus d'affaires qui impliquent beaucoup de tâches à forte intensité de connaissances comme un diagnostic médical, ou un cas de droit. Pour ce projet, nous générons des modèles pour CMMN Modeler de Trisotech (Trisotech).

À cette fin, nous aurons besoin de:

- Définir une méthode pour accepter l'entrée totalement non structurée et non claire, y compris le dessin à main levée de modèles CMMN;
- Développer une technique pour faire correspondre les entrées avec les modèles par défaut des modèles CMMN;
- Construire les fragments de modèle reconnus dans un modèle CMMN qui

peuvent être sérialisés dans le format d'échange XML pour CMMN, soutenir l'échange du modèle et son importation dans des outils conformes au CMMN.

0.3 Méthodologie

La méthodologie de recherche s'appuie sur la littérature de traitement d'image, la reconnaissance des formes, la correspondance des modèles et la description sémantique des formes dessinées à la main. Après avoir évalué plusieurs architectures, nous avons décidé de baser cette étude sur les esquisses à main levée qui peuvent être reconnues et converties dans les formes habituelles de l'utilisateur. Ainsi, l'approche évalue la séquence brute de points en tant qu'entrée selon les modèles par défaut des modèles CMMN et détermine si les points correspondent à une entrée significative. De plus, dans le cadre du traitement de l'image et de la vision par ordinateur, la reconnaissance des esquisses couvre beaucoup de détails de calculs. Ainsi, l'extension, le développement et la proposition d'une approche générale basée sur la bibliothèque OpenCV et les règles de correspondance de modèles sont adoptés pour convertir l'entrée de l'utilisateur en des modèles CMMN qui sont, à leur tour, transformés en un format accepté par un outil de modélisation CMMN. À la fin, des tests de simulation sont effectués pour évaluer la facilité maximale d'utilisation de l'application.

0.4 Organisation du Mémoire

Le premier chapitre de ce mémoire introduit les concepts essentiels de la reconnaissance de formes en ligne et hors ligne, et les méthodes que chaque approche devrait suivre pour étudier la saisie des données. De plus, les avantages et les inconvénients des outils de prototypage à basse-fidélité et des outils de prototypage à haute-fidélité sont comparés. Le deuxième chapitre décrit les modèles CMMN et leurs structures. Le fondement de notre approche qui consiste à reconnaître et à convertir des esquisses dessinées à la main en un outil de modélisation CMMN est

décrit dans le troisième chapitre. Le quatrième chapitre évalue l'application et finalise le mémoire avec une conclusion et les implications potentielles pour une recherche future.

CHAPTRE I

ÉTAT DE L'ART DES SYSTÈMES DE RECONNAISSANCE DES ESQUISSES

La reconnaissance de formes est une branche de l'apprentissage automatique (Michalski *et al.*, 2013), qui souligne la reconnaissance de modèles de données et de leurs régularités et ce en les classifiant dans un certain nombre de catégories ou classes (Kpalma et Ronsin, 2007). C'est une collection de méthodes mathématiques, statistiques, heuristiques et techniques inductives du rôle fondamental qui permettent de trouver des modèles mathématiques pour des problèmes réels (Liu *et al.*, 2006). Le développement de la reconnaissance de formes augmente très rapidement. De nos jours, la reconnaissance de formes est un vaste domaine de recherche qui peut toucher un large éventail de disciplines telles que l'ingénierie, les mathématiques, l'art et la médecine. Les systèmes de reconnaissance de caractères optiques représentent l'une des applications les plus réussies de la technologie dans le domaine de la reconnaissance de formes et de l'intelligence artificielle. Le but principal de ces applications est de classer le motif d'entrée dans une classe spécifique (Kpalma et Ronsin, 2007). Particulièrement, ce chapitre se concentre sur les concepts essentiels de la représentation des données d'une image composée de graphiques vectoriels et de graphiques rasters. La section suivante présente les approches de reconnaissance de formes qui peuvent être classées en deux types : la reconnaissance en ligne et la reconnaissance hors ligne. Les similarités et les différences de ces deux types sont reportées dans ce qui suit.

1.1 Graphiques Vectoriels et Graphiques Raster

Les infographies sont des images et des films créés à l'aide d'ordinateurs. Dans ce domaine, il y a deux façons substantielles de représenter des données d'une image (voir la figure 1.1): graphiques raster (bitmap) et graphiques vectoriels (Umbaugh, 1997). Les graphiques raster (Umbaugh, 1997), sont définis par des pixels. Ces pixels sont non évolutifs. Par ailleurs, chacun de ces points carrés minuscules représente une couleur. Ainsi, pour réaliser une image, ces points se combinent en motif. Les programmes graphiques raster définissent la couleur de chaque pixel la dimension de l'image. Dans les graphiques rasters, la résolution est définie par le nombre de pixels contenu dans un fichier qui est souvent mentionné comme DPI (points par pouce). Par conséquent, les graphiques rasters dépendent de la résolution. En revanche, les graphiques vectoriels (Umbaugh, 1997) sont définis par une série d'équations mathématiques qui permettent de spécifier des lignes, des courbes, des formes, ainsi que des attributs modifiables tels que la direction, l'épaisseur et la couleur de la ligne. Les fichiers vectoriels n'ont pas besoin de prendre en compte chaque pixel. Par conséquent, les vecteurs peuvent être évolutifs à n'importe quelle taille et ils sont indépendants de la résolution. En outre, ils nécessitent beaucoup moins de mémoire par rapport aux graphiques rasters.

De ce fait, les graphiques vectoriels sont plus avantageux que les graphiques rasters. Globalement, ils consistent à vectoriser les données d'entrée pour un stockage plus efficace et une manipulation plus facile pour analyser et traiter les formes géométriques.

Figure 1.1 Représente les graphiques vectoriels et raster (Vector vs. Raster Graphics)

1.2 Reconnaissance Optique de Caractères

Avec le développement des ordinateurs numériques, une large gamme d'applications dans le domaine de la banque, la sécurité, le traitement postal et l'identification des langues avec les méthodologies de reconnaissance OCR (*Optical Character Recognition*) sont exposées.

La méthodologie OCR est basée sur la reconnaissance de documents imprimés ou écrits à la main (RS et Afseena, 2015). Par conséquent, cette technique utilise une appareil photo numérique ou un scanner pour enregistrer et stocker différents types de documents tels que des documents papier ou des images de caractères. Ensuite, tous ces documents sont traduits dans des formats éditables par la machine comme le code ASCII (RS et Afseena, 2015). Par conséquent, l'espace mémoire pour stocker les documents est réduit et la vitesse de récupération des données est augmentée. Par exemple, l'OCR peut être utilisé dans divers domaines comme la banque, où ils doivent traiter un grand volume de papier. Avec OCR, il pourrait être traité sans intervention humaine.

« OCR peut être classé en deux catégories basées sur le type de texte et l'acquisition des documents » (RS et Afseena, 2015) (voir la figure 1.2).OCR est composé de deux types sur la base du type de texte: HCR (*Handwritten Character Recognition*) et

PCR (*Printed Character Recognition*) (RS et Afseena, 2015). Le HCR reconnaît les entrées écrites à la main telles que les documents papier et le PCR reconnaissent les documents imprimés (RS et Afseena, 2015). Dans la deuxième couche, HCR en tant que type de l'OCR est divisé en systèmes de reconnaissance en ligne et hors ligne basée sur l'acquisition de documents. L'acquisition de documents peut inclure des étapes générales telles que le prétraitement, la segmentation, l'extraction de caractéristiques et la classification (RS et Afseena, 2015).

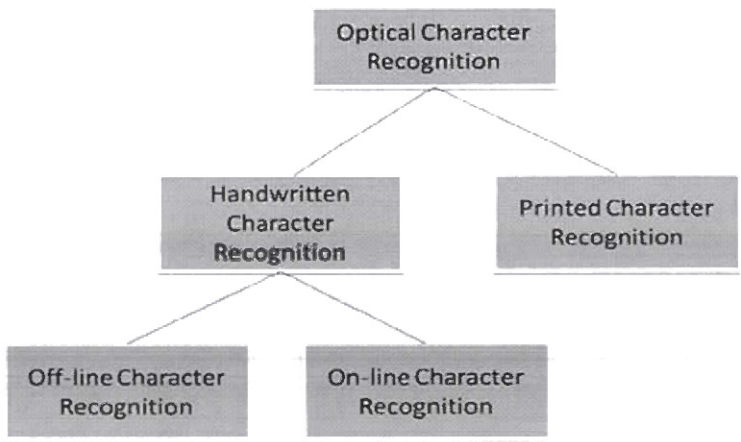


Figure 1.2 Classification de reconnaissance Optique des Caractères (RS et Afseena, 2015)

1.2.1 Reconnaissance hors ligne

Il y a eu beaucoup de recherches dans le domaine de la reconnaissance de caractères hors ligne (Arica et Yarman-Vural, 2001; Mori *et al.*, 1984; Plamondon et Srihari, 2000; RS et Afseena, 2015; Suen *et al.*, 1980; Vinciarelli, 2002). Dans cette approche, une image numérique généralement obtenue par numérisation au moyen d'un scanner ou bien prise en photo par une caméra constitue une entrée pour la reconnaissance (figure 1.3). Par conséquent, dans la reconnaissance hors ligne, avant de commencer la phase de reconnaissance de caractères, plusieurs étapes de prétraitement imparfaites et coûteuses doivent être exécutées. Le but des étapes de

prétraitement est d'exclure les informations non pertinentes et d'inclure des informations pertinentes dans l'entrée (RS et Afseena, 2015).

La première étape du prétraitement est le seuillage. Il est composé de plusieurs techniques (Sezgin, 2004). Le seuil est utilisé pour distinguer les objets de l'arrière-plan d'une image. Il consiste à convertir une image en niveaux de gris en une image binaire, c.-à-d., en noir et blanc (RS et Afseena, 2015; Sezgin, 2004).

Dans la deuxième étape, afin d'améliorer les performances de reconnaissance, une suppression de bruit doit être faite pour extraire la matière textuelle du premier plan d'un arrière-plan texturé par exemple. Cela peut se faire en supprimant les interférences du trait, le bruit impulsif, le bruit gaussien, le bruit de speckle, et le bruit de photons (Cheriet et Suen, 1993; Plamondon et Srihari, 2000; RS et Afseena, 2015). La suppression du bruit d'une image peut être employée dans d'autres applications, qui ont été discutées dans ces papiers (Motwani *et al.*, 2004; Rao et Panduranga, 2006).

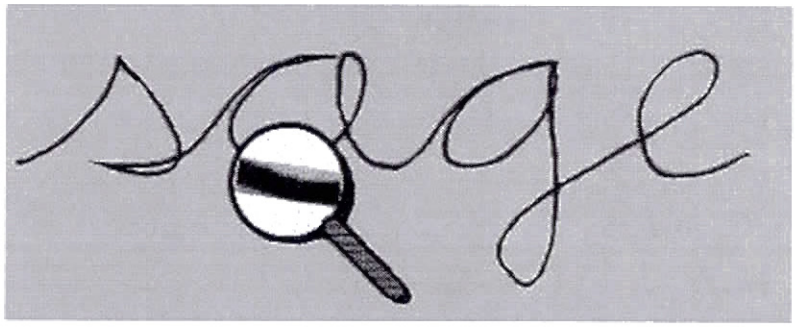


Figure 1.3 Reconnaissance Hors Ligne.L'image du mot est convertie en pixels de niveau gris à l'aide d'un scanner (Plamondon et Srihari, 2000).

La dernière étape du prétraitement consiste à utiliser l'image en noir et blanc comme entrée pour le processus d'amincissement. Ce processus permet de représenter le contenu de l'image avec des lignes fines. Le but de cette technique est de conserver

intactes les propriétés géométriques et topologiques de l'image, ce qui rend l'image appropriée pour l'analyse dans les phases suivantes (Lam *et al.*, 1992). Comme le montre la figure 1.4, les étapes impliquées dans le prétraitement sont affichées.

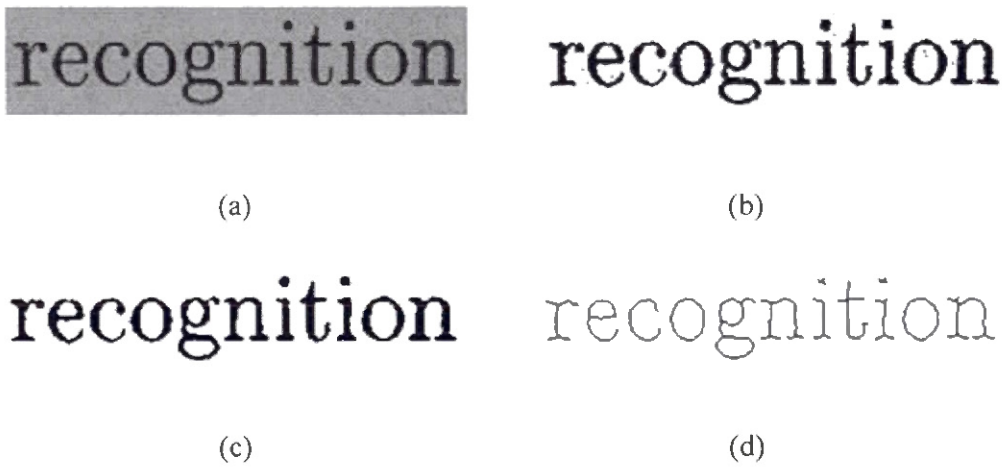


Figure 1.4 Étapes de prétraitement de la reconnaissance hors ligne.: (a) Image numérisée en entrée brute; (b) Seuillage en noir et blanc avec du bruit; (c) Image débruitée; (d) Image amincie (Soisalon-Soininen, 2011)

Les étapes de prétraitement expliquées ci-dessus sont communes à tous les problèmes de reconnaissance hors ligne, et peuvent être utilisées dans les caractères ou les problèmes graphiques. La deuxième phase de reconnaissance hors ligne, qui comprend la segmentation, est le processus de conversion des images d'entrée en texte. Cette étape se fait en trois parties: segmentation de ligne, segmentation de mots et segmentation de caractères (RS et Afseena, 2015). Par conséquent, cette étape sépare les phrases du texte et divise les mots et les lettres de phrases par la suite (Suen *et al.*, 1980).

La dernière phase, extraction de caractéristiques, qui permet d'extraire les fonctionnalités les plus pertinentes, dépend principalement de l'application (RS et Afseena, 2015). Différentes méthodes d'extraction de caractéristiques sont expliquées

dans la littérature (Chen et Kégl, 2010; George et Gafoor, 2014; Majumdar, 2007; Mamatha *et al.*, 2013; Nemmour et Chibani, 2011). Ces méthodes décrivent comment les caractéristiques du caractère segmenté sont extraites. En fonction de ses caractéristiques, chaque caractère est assigné à l'une des classes spécifiées telles que les lettres majuscules et minuscules, les dix chiffres, et les symboles spéciaux (Plamondon et Srihari, 2000).

Du fait qu'il y a une forte relation entre la reconnaissance de caractère et la reconnaissance de forme (Arica et Yarman-Vural, 2001; Lladós *et al.*, 2001), il est utile d'étudier les approches de base, les méthodes et les applications liées à la reconnaissance hors ligne. Par exemple, la tâche de reconnaissance de la formule mathématique implique deux tâches: la reconnaissance des symboles et l'interprétation de la structure en deux dimensions (Garain et Chaudhuri, 2004). Bien que les méthodes de reconnaissance en ligne et hors ligne soient différentes, comprendre les défis de la reconnaissance hors ligne nous amène à découvrir et utiliser les avantages des méthodologies en ligne (Arica et Yarman-Vural, 2001).

1.2.2 Reconnaissance en Ligne

Dans les systèmes en ligne, le processus de reconnaissance de caractère ou de forme est exécuté pendant que l'utilisateur écrit ou dessine (Plamondon et Srihari, 2000). Par conséquent, un trait à main levée est capturé par une souris d'ordinateur, ou un doigt sur un appareil à écran tactile, en se basant sur le mouvement de la souris ou du doigt (figure 1.5). De plus, un trait à main levée dessiné par un utilisateur humain est généralement très cursif, imprécis et contient des imperfections. Par exemple, les lignes censées être droites seront dessinées comme des arcs, et les cercles seront dessinés comme des ellipses avec des formes irrégulières et un bruit significatif.

(Liu, 2003) propose une approche pour la reconnaissance en ligne qui présente un aperçu général à l'utilisateur afin de spécifier les problèmes généraux de reconnaissance graphique en ligne, et trouve des solutions pour convertir la séquence

de points de coordonnées en une séquence similaire à l'entrée spécifiée par l'utilisateur. Cette approche (voir la figure 1.7) se compose de trois étapes: reconnaissance de formes primitives, reconnaissance d'objets graphiques composites et reconnaissance et compréhension de documents. Pour décrire *la reconnaissance de formes primitives* (voir la figure 1.6), quatre sous étapes ont été définies (a) prétraitement de la courbe dessinée, (b) classification de forme (c) ajustement de forme et (d) régularisation de forme.

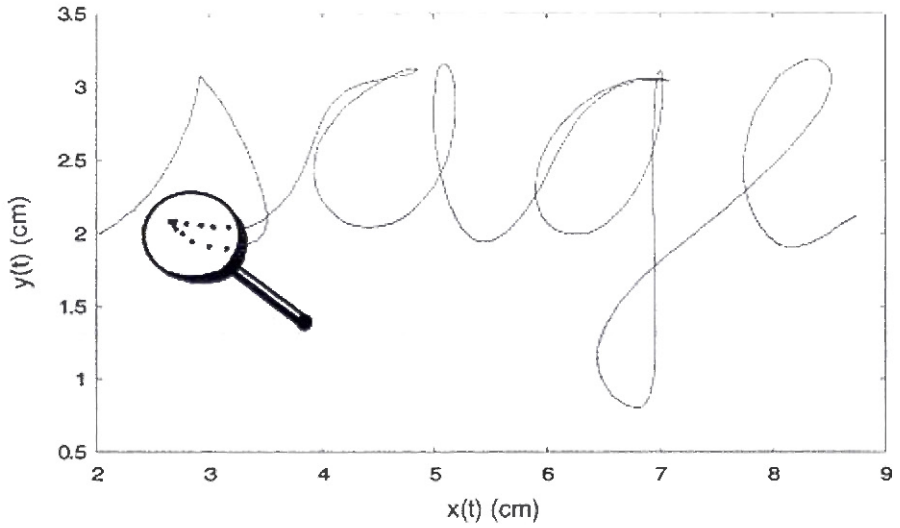


Figure 1.5 Reconnaissance en ligne. Données similaires telles que présentées (figure 1.3) sous forme de données de trajectoire ponctuelle. La coordonnée x, y est enregistrée en fonction du temps avec un numériseur. (Plamondon et Srihari, 2000).

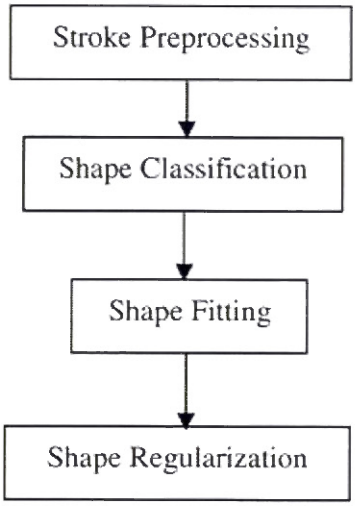


Figure 1.6 Étapes de reconnaissance de la forme primitive (Xiangyu *et al.*, 2002)

Pendant que l'utilisateur dessine un trait à main levée, la préoccupation de *la reconnaissance de formes primitives* est de déterminer le type et les paramètres de la forme primitive, qui peut être, par exemple, une ligne, un triangle, un rectangle ou une ellipse (Liu, 2003). Après avoir reconnu et converti le trait actuel (forme primitive reconnue), il est possible de le combiner avec des formes primitives précédemment reconnues, en se basant sur leurs relations spatiales. C'est l'étape de *reconnaissance d'objet graphique composite* (Liu, 2003). *La reconnaissance et la compréhension des documents constituent* les dernières étapes de la reconnaissance en ligne. Après avoir reconnu et converti les éléments graphiques (formes primitives et objets graphiques composites), nous devons comprendre les connexions et les relations entre les éléments, ainsi que leur sémantique (Liu, 2003). Dans la section suivante, nous expliquons davantage le sous-ensemble de la reconnaissance de forme primitive.

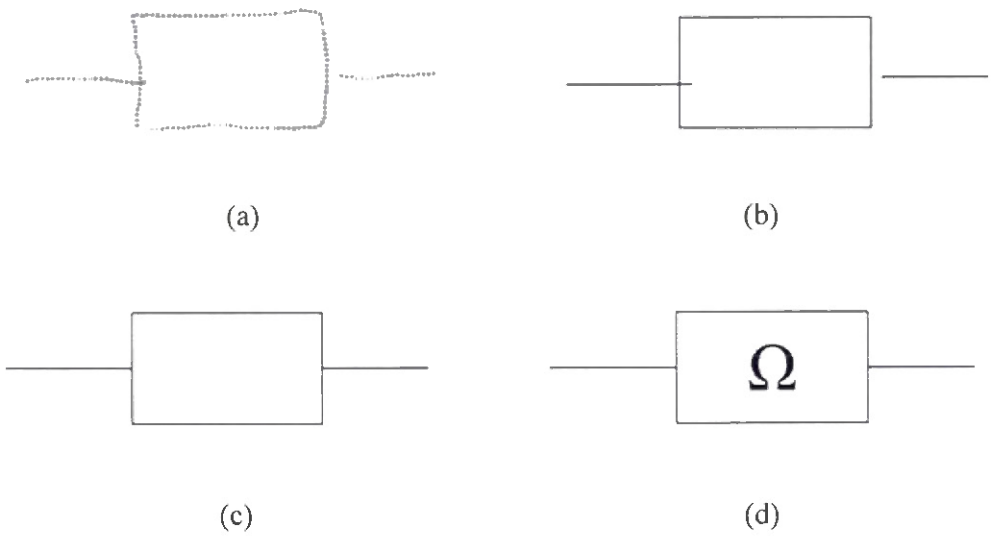


Figure 1.7 Sous-processus de reconnaissance graphique en ligne: (a) forme initialement dessinée; (b) dessin reconnu comme une forme primitive; (c) Formes primitives combinées en un objet composite en fonction de leur relation spatiale; (d) La sémantique de l'objet composite interprétée en utilisant des informations contextuelles. (Soisalon-Soininen, 2011).

1.2.2.1 Prétraitement

Dans la reconnaissance en ligne, Le prétraitement a le même rôle que dans la reconnaissance hors ligne. Il vise à éliminer le bruit et les erreurs mineures afin de rendre les traits le plus proche possible de l'intention de l'utilisateur. Le prétraitement peut être divisé en quatre étapes: *approximation polygonale, filtrage des points d'agglomération, raffinement des points d'extrémité* et *calcul de la coque convexe* (Liu, 2003; Wenyin *et al.*, 2001). *Le calcul de la coque convexe* est utilisé individuellement pour reconnaître les formes fermées et ne sera donc pas discuté ici.

(Xiangyu *et al.*, 2002) explique qu'en général, le matériel d'entrée produit beaucoup plus de points que nécessaires pour définir la forme du trait. En enlevant ces points de la chaîne, la ligne sommaire sera affichée approximativement par une polyligne avec

beaucoup moins de sommets critiques. Par conséquent, en déterminant la mesure dans laquelle cela est contrôlé par le paramètre ε, la polyligne peut conserver la forme originale (figure 1.8). « Si la distance d'un point au segment de droite formé en connectant ses voisins est inférieure à ε, ce point est non critique et doit être supprimé de la polyligne » (Xiangyu *et al.*, 2002). Ainsi, d'une part, plus de sommets sont laissés en diminuant la valeur de ε, d'une autre part, la précision du bord est améliorée. Ce processus est appelé *approximation polygonale*.

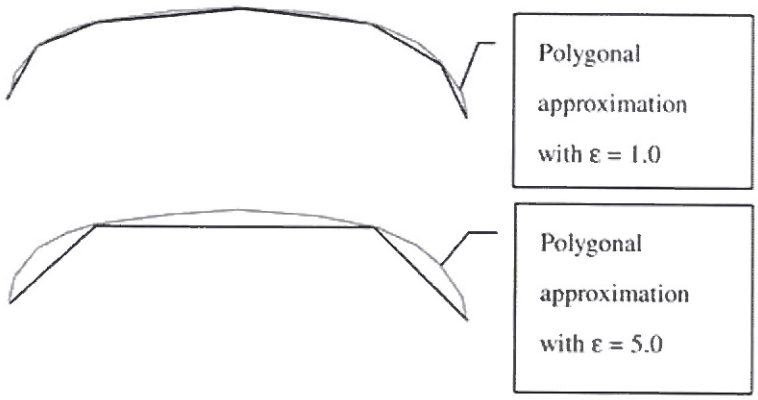


Figure 1.8 Les illustrations du processus d'approximation polygonal (Xiangyu *et al.*, 2002).

L'utilisation d'un stylo ou d'un numériseur peut produire un crochet ou un cercle (voir la figure 1.9) à la fin du trait, qui a généralement des agglomérations ponctuelles beaucoup plus élevées que la valeur moyenne de toute la polyligne.

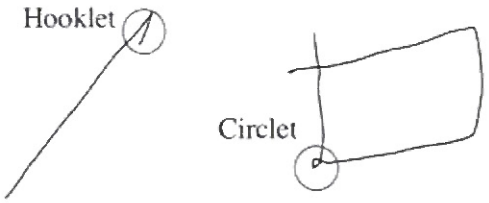
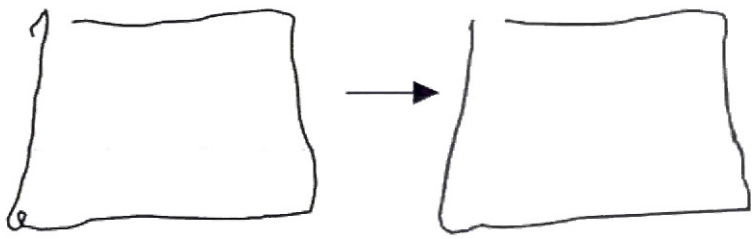


Figure 1.9 Points d'agglomérationcomme un crochet et un cercle (Xiangyu *et al.*, 2002).

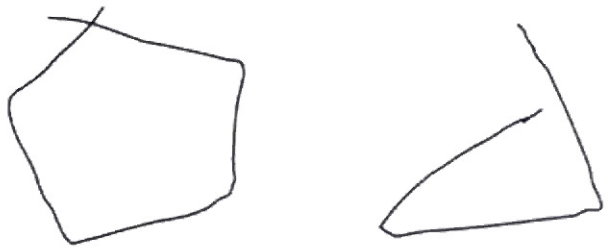
La tâche de *filtrage des points agglomérés* (voir la figure 1.10) examine les agglomérations ponctuelles de la polyligne d'entrée. En trouvant ces segments, il commence à supprimer ces bruits et utilise moins de points pour représenter le segment.



(a) (b)

Figure 1.10 Filtrage des points d'agglomération: (a) La ligne sommaire avant le traitement; (b) La ligne sommaire après le traitement (Xiangyu *et al.*, 2002).

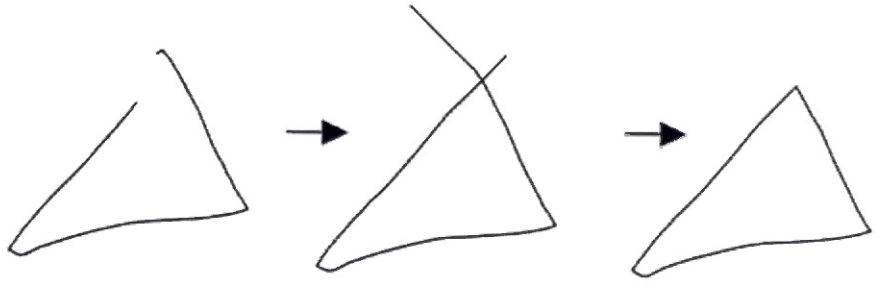
Un autre "bruit" introduit par dessin à la main est le cas où le trait qui est peint par l'utilisateur est destiné à être un polygone, mais finit comme une forme non fermée, ou une forme avec une croix près de ses points d'extrémité (voir la figure 1.11).



(a) (b)

Figure 1.11 Formes avec des points limites inappropriés(a) Un pentagone avec une croix; (b) un triangle non fermé (Xiangyu *et al.*, 2002).

Par conséquent, ces points limites inappropriés apportent suffisamment de barrières pour la classification des formes et la régularisation. En conséquence, l'affinement du point final peut être utilisé pour fermer la partie ouverte en prolongeant les points d'extrémité le long de ses directions d'extrémité (voir la figure 1.12).



(a) (b) (c)

Figure 1.12 Exemples de raffinement de point final: (a) La ligne sommaire avant le traitement; (b) Après avoir tiré les points de terminaison; (c) Après avoir supprimé des points supplémentaires (Xiangyu *et al.*, 2002).

1.2.2.2 Classification de Forme

Après le prétraitement, la classification de forme est la partie la plus essentielle de la reconnaissance de forme. Elle est utilisée pour décider si un trait non défini dessiné par un utilisateur peut représenter une forme prédéfinie telle qu'un carré, un cercle et une flèche, etc. De plus, dans le domaine de la reconnaissance de forme, les articles académiques se concentrent principalement sur la classification de forme. (Liu, 2003).

En général, le but de la classification des formes est d'extraire des informations utiles des données d'entrée pour faciliter la classification. L'étendue des méthodes de classification en raison de la quantité de détails dans le processus et de la capacité de combiner différentes parties d'approches différentes est abondante.

(Lladós *et al.*, 2001) propose la catégorisation traditionnelle de la reconnaissance de formes dans le contexte des méthodes de reconnaissance de symboles. Ces méthodes peuvent être divisées en méthodes *statistiques* et *structurelles*. « Dans la reconnaissance de modèle statistique, chaque motif est représenté comme un vecteur de caractéristique n-dimensionnel extrait de l'image » (Lladós *et al.*, 2001). Par conséquent, cette classification est effectuée en partitionnant l'espace caractéristique en différentes classes. Ainsi, nous devons nous concentrer sur la sélection des fonctionnalités, ainsi que la sélection de la méthode afin de partitionner l'espace des fonctionnalités.

Les propriétés des motifs qui doivent être classées peuvent affecter la sélection de l'espace des caractéristiques. « Le critère principal doit être de minimiser la distance entre les motifs appartenant à la même classe et de maximiser la distance entre les motifs appartenant à différentes classes » (Lladós *et al.*, 2001). Une fois qu'un ensemble de fonctionnalités a été choisi, nous utilisons la classification pour partitionner l'espace caractéristique et assigner chaque vecteur caractéristique à l'une

des classes prédéfinies. Trois des méthodes de sélection les plus courantes sont: *k-plus proches voisins, arbre de décision* et *réseaux de neurones* (Lladós *et al.*, 2001).

La méthode *des k-plus proches voisins* (Larose, 2005; Lladós *et al.*, 2001), est basée sur une mesure de similarité. Nous devons définir une fonction de distance entre les vecteurs de caractéristiques afin d'affecter chaque image d'entrée à la classe avec le représentant le plus proche (Lladós *et al.*, 2001). Une fois que tous les représentants sont définis pour chaque modèle d'entrée et chaque classe, l'ensemble des k-plus proches représentants est construit, et le modèle sera assigné à la classe qui a le plus de représentants dans cet ensemble. (Lladós *et al.*, 2001).

La méthode de *l'arbre de décision* (Freund et Mason, 1999; Lladós *et al.*, 2001), utilise des règles de décision simples. Dans cette méthode, des conditions spécifiques sur la valeur d'une caractéristique particulière sont testées dans chaque nœud (Freund et Mason, 1999; Lladós *et al.*, 2001). La classification sera exécutée en traçant les branches dans l'arbre en fonction du résultat du test de condition (Freund et Mason, 1999; Lladós *et al.*, 2001). En conséquence, l'une des feuilles de l'arbre correspondant aux symboles reconnus est atteinte.

Les *réseaux de neurones* (Bishop, 1995; Lladós *et al.*, 2001) résout des problèmes dans la même fonction du cerveau humain. Par conséquent, la capacité d'apprentissage est l'un des avantages de cette approche. Sur la base de cette capacité, les réseaux de neurones ont obtenu de bons taux de classification dans de différents domaines. Cet avantage leur permet de s'adapter en fonction des propriétés de l'ensemble d'entraînement(Bishop, 1995; Lladós *et al.*, 2001). En apprenant automatiquement, le réseau de neurones optimise ses paramètres afin de reconnaître les symboles dans l'ensemble d'apprentissage (Bishop, 1995; Lladós *et al.*, 2001).

Dans l'approche *structurelle* (Lladós *et al.*, 2001), une description des formes est utilisée comme référence. Par conséquent, cette méthode utilise un ensemble de

primitives géométriques et leurs relations pour représenter des symboles (Lladós *et al.*, 2001). Les lignes droites et les arcs sont les primitives utilisées pour décrire la forme des symboles, ainsi que d'autres primitives géométriques telles que des boucles, des contours ou des formes simples (cercles, rectangles, etc.) (Lladós *et al.*, 2001). Par exemple, un diamant peut être représenté comme un ensemble de quatre lignes avec certaines contraintes (Lladós *et al.*, 2001). Par conséquent, un modèle de forme idéale est construit pour chaque symbole en utilisant ces primitives (Lladós *et al.*, 2001). En conséquence, une image d'entrée est classée sur la base de la meilleure correspondance entre la représentation de l'image et le modèle du symbole. (Lladós *et al.*, 2001).

1.2.2.3 Ajustement de Forme et Régularisation

La mise en forme et la régularisation sont moins compliquées que la classification des formes. Après la classification de l'esquisse, le processus d'ajustement est utilisé pour déterminer si une forme d'une classe est similaire à l'esquisse. Par conséquent, le but est d'utiliser différentes méthodes pour trouver les paramètres de la forme ajustée qui a la distance moyenne la plus basse à l'esquisse (Xiangyu *et al.*, 2002). Plusieurs approches ont été proposées à ce sujet, sont expliquées ci-dessous.

Chen and Xie, proposé une approche qui trouve la forme appropriée à partir d'une analyse de la courbe dessinée avec des modèles de référence (Chen et Xie, 1996). Leur technique utilise des règles de filtrage floues afin de reconnaître le bon modèle et d'éliminer les points indésirables. Il existe différentes méthodes d'ajustement pour les lignes, les cercles, les arcs de cercle, les ellipses et les arcs elliptiques.

Pour générer une ligne droite à partir d'une séquence de points, ils tentent de trouver une ligne avec les meilleures approximations des données dispersées. Ainsi, la méthode des moindres carrés (Marquardt, 1963), est utilisée pour le générer. De plus, trois points non colinéaires peuvent être utilisés pour déterminer si un arc est un cercle ou un arc de circulaire. Ainsi, pour représenter le cercle, des informations

floues sont nécessaires pour obtenir une moyenne pondérée des centres et des rayons en choisissant tous les triplets possibles dans le dessin à main levée. Pour l'ajustement de l'ellipse, la technique du multiplicateur de chaulage est proposée. Plus de méthodes pour l'ellipse et l'ajustement polygonal sont suggérés par Liu (Liu, 2003; Wenyin *et al.*, 2001; Xiangyu *et al.*, 2002).

Liu a discuté un ensemble de règles de régularisation de forme qui sont décrites plus en détail dans son travail (Xiangyu *et al.*, 2002). Ces règles tentent de corriger les défauts du dessin d'un dessinateur. Elles suggèrent un processus comprenant deux sous-processus: *régularisation de forme intérieure* et *régularisation inter-formes*.

La régularisation de forme intérieure (figure 1.13) consiste à apporter des modifications aux formes ajustées selon plusieurs rectifications (Xiangyu *et al.*, 2002). Le premier redressement ajuste les arêtes ou l'axe A et l'axe B des polygones pour égaliser leurs longueurs. La deuxième rectification ajuste les bords des polygones pour les rendre parallèles. La troisième rectification relie les angles internes des polygones à des figures régulières telles que les rectangles. En outre, la rectification horizontale / verticale comme la dernière rectification en régularisation de forme interne est capable de rectifier les bords d'un polygone, ou les axes d'une ellipse ou des diagonales d'un diamant à l'horizontale ou à la verticale.

(a) (b)

Figure 1.13 Régularisation de forme intérieure (a) rectifier un triangle dans un triangle isocèle; (b) rectifiant deux rectangles dans la même taille (Xiangyu *et al.*, 2002).

La régularisation inter-forme (figure 1.14) introduit un groupe de rectifications incluant la taille, la position et les points critiques (Xiangyu *et al.*, 2002). La rectification de taille ajuste un groupe de formes primitives adjacentes qui ont le même type ou approximativement la même taille. Les bords de deux polygones adjacents qui sont presque sur la même ligne horizontale / verticale sont ajustés par rectification de position. À la fin, la rectification des points critiques est appliquée pour rectifier les centres d'ellipses, les sommets des polygones et les points médians des arêtes.

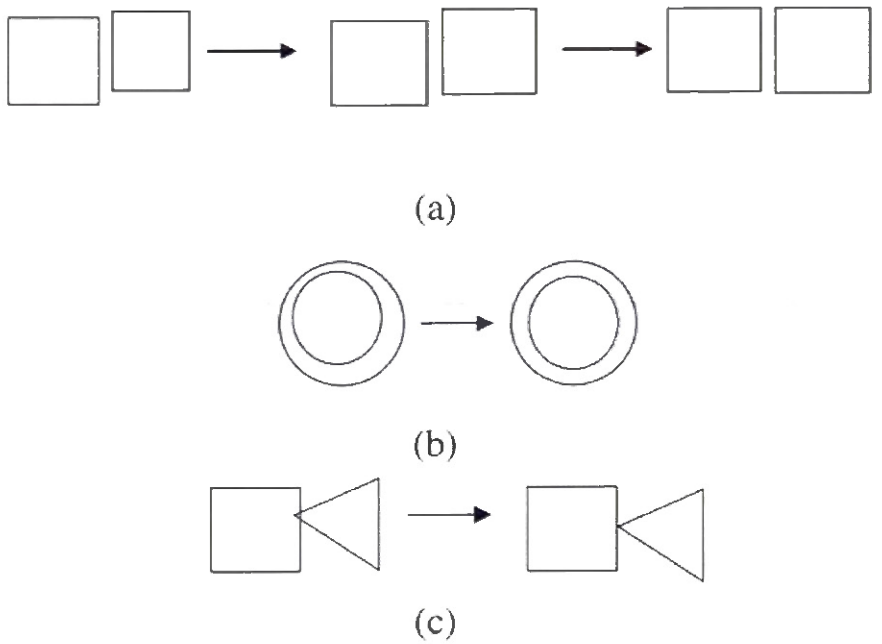


Figure 1.14 La régularisation inter-forme: (a) à l'échelle les deux cases adjacentes pour avoir la même taille; (b) décale les deux cercles adjacents pour avoir la même position; (c) décaler le triangle et le carré, le sommet le plus à gauche du triangle est au centre du bord le plus à droite du carré (Xiangyu *et al.*, 2002).

1.3 Systèmes de Reconnaissance d'Esquisse

Coyette et Vanderdonckt ont proposé trois catégories pour les prototypes d'IU qui ont été définis en fonction de leur degré de fidélité, ce qui renvoie à la précision qui leur permet de présenter la réalité de l'interface utilisateur (Coyette et Vanderdonckt, 2005). Le premier outil prototype UI est la haute-fidélité (Hi-Fi) qui montre le résultat final inapplicable. Il peut être considéré comme un outil de haute-fidélité qui prend en charge la construction d'une interface utilisateur presque complète et utilisable (Coyette et Vanderdonckt, 2005). Par conséquent, ce type de prototype d'interface utilisateur comprend des fonctions d'édition telles qu'annuler, effacer et déplacer, qui peuvent créer une complète IUG (Interface Utilisateur Graphique) pour les concepteurs (Coyette et Vanderdonckt, 2005).

L'approche de la fidélité moyenne (Me-Fi) se situe entre les prototypes de haute-fidélité et de basse fidélité afin de présenter les principales fonctions et les détails (Coyette et Vanderdonckt, 2005). Cette approche maintient les informations pertinentes pour les schémas de couleurs, la typographie ou d'autres détails mineurs (Coyette et Vanderdonckt, 2005). L'approche basse-fidélité (Lo-Fi) est utilisée pour conserver les informations générales afin d'obtenir une compréhension générale de ce qui est désiré (Coyette et Vanderdonckt, 2005). L'esquisse faites à la main sur le papier est bien connue comme l'un des moyens les plus efficaces pour représenter les premières ébauches d'une future UI. (Bailey et Konstan, 2003; Coyette et Vanderdonckt, 2005; Landay et Myers, 2001; Newman *et al.*, 2003). Cette approche illimitée connue sous le nom de prototype d'interface utilisateur basse fidélité présente de nombreux avantages. Par exemple, lors de toute étape de conception, des esquisses peuvent être dessinés sans aucun prérequis (Newman *et al.*, 2003). Cette méthode est rapide à produire (Duyne *et al.*, 2002). Ainsi, au lieu de perturber l'utilisateur avec des détails non essentiels, il permet au dessinateur de se concentrer sur des problèmes structurels essentiels (Landay et Myers, 2001). Un autre avantage,

non seulement il encourage la créativité (Landay et Myers, 2001) mais il augmente également le niveau de collaboration entre les concepteurs et les utilisateurs finaux (Plimmer et Apperley, 2003a). En outre, «la création d'un prototype d'interface utilisateur basse-fidélité (comme les esquisses d'interface utilisateur) est au moins 10 à 20 fois plus facile et plus rapide que son équivalent avec un prototype haute-fidélité telle que produit dans les constructeurs d'interface utilisateur» (Coyette et Vanderdonckt, 2005; Duyne *et al.*, 2002).

En comparant ces prototypes d'interface utilisateur, un prototype Lo-Fi a un ensemble d'avantages pour comparer les autres prototypes (voir un résumé de ces avantages dans le tableau 1.1). En ayant plusieurs écrans qui ont beaucoup de caractéristiques similaires, utiliser copier et coller au lieu de réécrire tout l'écran est plus rentable. Le manque d'assistance dans cette approche est palpable (Coyette et Vanderdonckt, 2005). Par conséquent, en considérant les avantages de Lo-Fi et en combinant ces approches, deux familles d'outils logiciels qui contiennent des esquisses d'interface utilisateur avec ou sans génération de code seront développées (Coyette et Vanderdonckt, 2005).

La section suivante est divisée en deux sous-sections. La section 1.3.1 décrit les outils basés sur l'esquisse pour les diagrammes de classes UML, et la section 1.3.2 explique les outils basés sur l'esquisse dans d'autres domaines.

Table 1. Comparaison des outils de prototypage d'interface utilisateur de fidélité logicielle (Coyette et Vanderdonckt, 2005)

Fidelity	Appearance	Advantages	Shortcomings
Low	- Sketchy - Little visual detail	- Low development cost - Short production time - Easy communication - Basic drawing skills needed	- Is facilitator-driven - Limited for usability tests - Limited support of navigational aspects - Low attractiveness for end users - No code generation
Medium	- Simple - medium level of detail, close to appearance of final UI	- Medium development cost - Average production time - May involve some basic graphical aspects as specified in style guide: labels, icons,... - Limited drawing skills - Understandable for end user	- Is facilitator-driven - Limited for usability tests - Medium support of navigational aspects - No code generation
High	- Definitive, refined - Look and Feel of final UI	- Fully interactive - Serves for usability testing - Supports user-centered design - Serves for prototype validation and contract - Attractive for end users - Code generation	- High development cost - High production time - Advanced drawing and specification skills needed - Very inflexible with respect to changing requirements

1.3.1 Outils basés sur l'esquisse pour les diagrammes de classes UML

Hammond et Davis ont proposé Tahuti comme outil d'esquisse pour les diagrammes de classes UML (Hammond et Davis, 2006b). Tahuti utilise un cadre de reconnaissance multicouche pour reconnaître les esquisses en fonction de leurs propriétés géométriques (Hammond et Davis, 2006b). Il permet aux utilisateurs d'esquisser librement des objets de différentes manières, au lieu de demander à l'utilisateur de dessiner l'objet de manière prédéfinie (Hammond et Davis, 2006b). De plus, Tahuti utilise deux vues différentes, dessiner la vue et la vue interprétée, afin d'afficher les esquisses

de l'utilisateur et le résultat du processus de reconnaissance. En outre, l'utilisateur peut basculer entre les deux vues.

Le cadre de reconnaissance multicouche de Tahuti utilise un langage formel appelé LADDER, qui a été créé par Hammond et Davis (Hammond et Davis, 2006a; Hammond, 2007). « LADDER est le premier langage de description d'esquisse destiné aux développeurs d'interface utilisateur pour décrire comment les diagrammes esquissés dans un domaine sont dessinés, affichés et édités » (Hammond et Davis, 2006a). Par conséquent, le langage employé dans ce cadre met en œuvre le premier système prototype qui a la capacité de générer automatiquement une interface d'esquisse pour un domaine uniquement en considérant la description du domaine (Hammond et Davis, 2006a).

Ce cadre est divisé en trois parties: *Description de domaine, traduction et système de reconnaissance d'esquisse indépendant de domaine* (Hammond, 2007). La figure 1.15 montre un aperçu d'un tel cadre.

Selon la figure 1.15, un certain nombre de formes prédéfinies, de contraintes, de méthodes d'affichage et de comportements d'édition sont fournis dans la description du domaine. De plus, ce domaine comprend une section d'affichage et une section d'édition. « Une section d'affichage spécifie ce qui doit être affiché sur l'écran lorsque la forme est reconnue. La section d'édition spécifie comment la forme peut être éditée. Les commandes d'édition courantes impliquent le déplacement et la suppression de la forme » (Hammond et Davis, 2006a). Après la description du domaine, le processus de traduction est lancé pour analyser les définitions de la forme et générer le code nécessaire pour reconnaître les formes, modifier les déclencheurs et afficher les formes une fois qu'elles sont reconnues (Hammond et Davis, 2006a). Comme dernière partie de ce cadre, le système de reconnaissance d'esquisse indépendant du domaine est utilisé. Tout d'abord, lorsque le trait est dessiné le

système recherche la base de données des formes dessinées afin de vérifier si le traît dessiné est un déclencheur d'édition pour n'importe quelle forme ou non (Hammond et Davis, 2006a). Si le trait est trouvé, ne pas être un geste d'édition, il doit s'agir d'un geste de dessin.

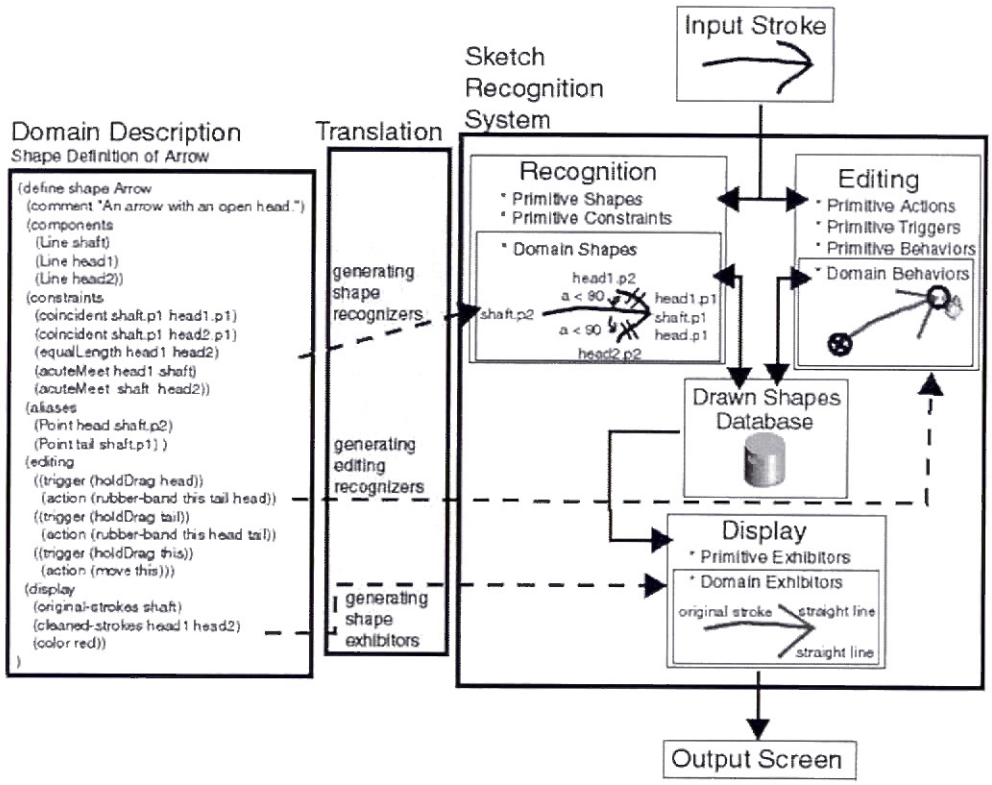


Figure 1.15 Cadre de LADDER (Hammond, 2007)

Ainsi, le système prétraite le trait dans une collection de primitives afin de les ajouter à la base de données de formes dessinées (Hammond et Davis, 2006a). Le module de reconnaissance essaye de construire une forme d'ordre plus haute en examinant la base de données de formes (Hammond et Davis, 2006a). A la fin, le module d'affichage affiche les formes visibles qui sont définies par la description du domaine.

Qiu présente SketchUML comme un outil d'esquisse qui permet aux utilisateurs d'esquisser des diagrammes de classes UML sur un ordinateur avec des capacités d'édition (Qiu, 2007). Par conséquent, il peut reconnaître des esquisses immédiatement, et l'utilisateur n'a pas besoin de basculer à une vue différente pour voir les résultats de la reconnaissance. L'outil prend également en charge la reconnaissance de texte et permet aux utilisateurs d'écrire du texte directement à l'intérieur des objets de classe (Qiu, 2007). SketchUML doit définir la flexibilité d'esquisse et la précision de la reconnaissance. Par conséquent, une approche basée sur la géométrie est utilisée pour reconnaître un élément commun dans le diagramme (Qiu, 2007). En outre, il utilise une approche à base de graffiti pour identifier les gestes spéciaux (Qiu, 2007).

Chen et Grundy présentent SUMLOW comme un outil de diagramme de langage de modélisation unifié (UML) qui utilise un E-tableau blanc et une interface utilisateur basée sur l'esquisse (Chen *et al.*, 2008). Cet outil expérimental propose des techniques d'esquisse pour les exigences de phase précoce et la modélisation de la conception. Ainsi, il permet aux concepteurs d'esquisser des constructions UML en mélangeant différents éléments de diagrammes UML, des annotations de diagrammes et du texte dessiné à la main (Chen *et al.*, 2008). Leur approche présente un certain nombre d'avantages, y compris la conservation d'une copie des esquisses à la main, tandis que d'autres outils de conception UML basés sur l'esquisse convertissent les esquisses à main en modèles formels, et perdent les originaux. En outre, les utilisateurs peuvent utiliser divers traits de stylo pour manipuler des éléments diagrammatiques esquissés. Enfin, SUMLOW peut reconnaître diverses constructions UML dans la vue d'esquisse qui sont dessinées à la main, et des éléments formalisés dans la vue de diagramme.

1.3.2 Outils basés sur l'esquisse dans d'autres domaines

Le DENIM (Lin *et al.*, 2002; Newman *et al.*, 2003) est un exemple d'outil de prototypage Lo-Fi qui permet aux concepteurs d'esquisser rapidement des pages Web et de les présenter dans différents niveaux de détails, y compris une carte du site, un story-board et une page individuelle. Le DENIM a unifié ces niveaux grâce à des vues zoom. Le DEMAIS (Bailey *et al.*, 2001) est un autre prototypage Lo-Fi qui comble le fossé dans la conception multimédia et permet à un concepteur d'esquisser rapidement des idées de conception temporelles et interactives. Par conséquent, il est composé d'un outil de story-board multimédia interactif qui est capable de capturer environ 90% des idées de conception comportementale d'un concepteur (Bailey et Konstan, 2003). Ni DENIM ni DEMAIS ne produisent de code final ou autre sortie. En revanche, les outils de prototypage Hi-fi qui sont discutés ensuite, supportent la génération de code (Coyette et Vanderdonckt, 2005).

Jin et al. propose une approche pour les interfaces utilisateur à stylo qui suivent une classification de roman et de forme rapide et un algorithme de régularisation pour la reconnaissance de graphiques sommaires en ligne (Xiangyu *et al.*, 2002). Ce processus de reconnaissance est divisé en quatre étapes, y compris le prétraitement, la classification des formes, l'ajustement des formes et la régularisation, ce qui a expliqué précédemment. *Le modèle de force* d'attraction est utilisé pour combiner les sommets sur le trait d'esquisse d'entrée basé sur le certain seuil (Xiangyu *et al.*, 2002). Ainsi, le nombre total de sommets est réduit avant que le type de forme puisse être déterminé. Après ce processus, la forme est classée comme une forme primitive selon une approche basée sur des règles (Xiangyu *et al.*, 2002). Finalement, l'ajustement de la forme et la régularisation rectifient graduellement la forme primitive en une forme régulière qui s'adapte à la forme destinée par l'utilisateur (Xiangyu *et al.*, 2002).

Au lieu de travailler sur des connaissances spécifiques au domaine, Yu et Cai se concentrent sur les caractéristiques géométriques de bas niveau afin de réaliser un

système indépendant de domaine pour la reconnaissance des esquisses (Yu et Cai, 2003). Leur approche suit deux étapes. Tout d'abord, le trait est approximé selon une forme primitive ou une combinaison de formes primitives qui peuvent être utilisées dans des applications spécifiques à un domaine. En conséquence, un algorithme récursif segmente le trait en utilisant *la direction* et *des informations de courbure*. De plus, l'algorithme utilise des *caractéristiques géométriques* et *des données sur la direction du trait* pour distinguer entre les primitives graphiques (ligne, arc, cercle et hélice).

La deuxième étape prend en entrée toutes les formes primitives reconnues et analyse la connectivité entre elles en suivant ces segmentations (Yu et Cai, 2003): 1) supprimer les éléments erronés et redondants, 2) ajuster la taille et la position des éléments, et 3) ajuster leur mise en page pour faire correspondre les objets indépendants du domaine prédéfinis.

Leur production hiérarchique est présentée en trois étapes, y compris (a) *Le niveau le plus bas,* qui maintient les informations initiales sur le trait (b) *le niveau intermédiaire,* qui stocke les sommets et les formes primitives et (c) *le niveau le plus élevé,* le niveau sémantique composé des tables de relations qui consistent en des formes primitives reconnues et des objets de base (Yu et Cai, 2003).

Landay et Myers ont proposé un système appelé SILK (Sketching Interfaces Like Krazy) (Landay et Myers, 2001). SILK essaie de reconnaître les composants primitifs qui sont des formes de base telles que des rectangles, des lignes ondulées (représentant du texte), des lignes droites et des ellipses. Cette approche recherche les relations spatiales entre les nouveaux composants et les autres composants de l'esquisse afin de combiner les composants primitifs pour constituer un composant de l'interface utilisateur, tel qu'un bouton, un champ de texte, une barre de menus ou une barre de défilement. En outre, SILK reconnaît les gestes d'édition tels que la suppression et le regroupement ou le dégroupement d'objets.

Le moteur de reconnaissance sous-jacent de SILK utilise l'algorithme de reconnaissance gestuelle de Rubine pour identifier les composants primitifs (Rubine, 1991). Selon cet algorithme, chacun des composants primitifs est formé avec 15 à 20 exemples, et ils varient en taille, et la direction de dessin (Landay et Myers, 2001). Cependant, cet algorithme a certaines limites. Par exemple, les gestes utilisés dans l'algorithme de Rubine sont tous des seuls traits, pour éviter le problème de segmentation de la reconnaissance de caractères multi-traits. En outre, il ne gère pas les variations de taille et de rotation (Landay et Myers, 2001).

Calhoun et al. proposé un système de reconnaissance capable de reconnaître des symboles composés de plusieurs traits (Calhoun *et al.*, 2002). Leur approche applique un système de reconnaissance formable afin de décrire la définition des primitives géométriques, ainsi que les relations géométriques entre elles. Les primitives géométriques sont caractérisées par des propriétés intrinsèques telles que la ligne ou l'arc, la longueur, la longueur relative, la pente (pour les lignes uniquement) et le rayon (pour les arcs uniquement) (Calhoun *et al.*, 2002). Les relations géométriques entre les primitives sont construites sur la base de certains paramètres tels que l'existence d'intersections entre les primitives, la position relative des intersections, l'angle entre les lignes qui se croisent et l'existence de lignes parallèles (Calhoun *et al.*, 2002). L'approche prend en charge deux types de méthodes de reconnaissance. Dans la première méthode, chaque symbole primitif doit suivre la définition spécifiée, qui est apprise en examinant quelques exemples du symbole et de ses relations géométriques. Cependant, cette méthode nécessite une certaine attention de la part du dessinateur (Calhoun *et al.*, 2002). La deuxième méthode utilise une forme de meilleure première recherche basée sur une métrique de qualité spéculative et l'élagage (Calhoun *et al.*, 2002).

1.4 Conclusion

Nous avons étudié des outils basés sur des esquisses dans différents domaines, ainsi que leurs approches pour reconnaître les esquisses dessinés à la main. Cependant, ces approches ont été source d'inspiration pour comprendre le concept de la reconnaissance des esquisses, bien que le contexte de cette mémoire soit basé sur la reconnaissance des esquisses dessinées à la main du CMMN comme des formes primitives et des formes composites. Par conséquent, nous devons mettre en œuvre une nouvelle approche pour reconnaître un dessin à main levée de modèles CMMN. De plus, nous vérifions une approche pour transférer ces éléments reconnus dans un format qui peut être importé dans un outil de modélisation formel. Ainsi, le second problème consiste à sérialiser ces modèles formels dans un fichier XML conforme au format d'échange de modèle CMMN et pouvant ensuite être importé dans des outils compatibles CMMN. Dans les chapitres suivants, la signification de CMMN, ainsi que la définition de chaque modèle de CMMN ont été expliquées.

CHAPITRE II

MODÈLE DE GESTION DE CAS ET NOTATION (CMMN)

En ce qui concerne les systèmes de gestion de cas, afin de définir, créer et gérer des processus d'affaires, il est nécessaire de fournir un dossier de cas comme bâtiment principal, qui contient une collection de documents commerciaux et d'autres informations (Marin *et al.*, 2012). La gestion de cas a été développée pour gérer le travail social et les domaines d'applications connexes tels que les processus de réclamation d'assurance, les processus de soins de santé, les processus de services juridiques, le processus de services sociaux, etc. (Marin *et al.*, 2012).

Le modèle et la notation de gestion de cas (CMMN) est une norme à l'échelle de l'industrie par le groupe de gestion des objets (OMG) pour représenter et gérer les études de cas. Le CMMN soutient la représentation d'un large éventail d'activités de travailleurs du savoir, y compris la planification (p.ex., les initiatives de planification stratégique commerciale), le suivi (p.ex., entretien et réparations), la collaboration (p.ex., développement de spécifications), la tenue de dossiers (p.ex., des vérifications), la prise de décision (p.ex., les affaires judiciaires), et la résolution de problème (p.ex., service client) (Trisotech). Par conséquent, le CMMN est utilisé pour «documenter les processus d'affaires axés sur les cas afin de stimuler les efforts d'amélioration des processus pour le travail de connaissances » (Trisotech).

2.1 La notation

Dans cette section, nous donnons un aperçu de la notation CMMN afin de modéliser les constructions de base d'un cas.

2.1.1 Le cas

Les travailleurs du savoir, qui sont des experts dans leur domaine spécifique, sont capables d'exécuter différentes instances du modèle, ce qu'on appelle un cas dans le domaine de la gestion de cas (de Carvalho *et al.*, 2016). Par exemple, un médecin dans le domaine de la santé peut spécifier un cas qui implique de prendre soin d'un patient, en termes d'antécédents médicaux et de problèmes médicaux actuels (de Carvalho *et al.*, 2016; OMG). Dans un autre exemple, dans le domaine du droit, un juge peut définir un cas impliquant l'application de la loi à un sujet dans une situation particulière (de Carvalho *et al.*, 2016; OMG). En général, un cas est un concept de haut niveau qui combine tous les éléments qui constituent un modèle de cas. Par conséquent, «Un cas est une procédure qui implique des actions prises concernant un sujet dans une situation particulière pour atteindre un résultat souhaité » (OMG). Dans les sous-sections suivantes, nous allons décrire les composants de la modélisation de cas.

2.1.2 Le modèle de plan de cas

Le modèle de plan de cas (OMG), contient toutes les activités pour le cas. Il est composé de tous les éléments qui représentent le plan initial de l'affaire, ainsi que tous les éléments qui soutiennent l'évolution du plan (OMG). Une forme "Dossier", comme indiquer dans la figure 2.1, montre un modèle de plan de cas. Par conséquent, le nom du cas peut être inclus dans le rectangle supérieur gauche.

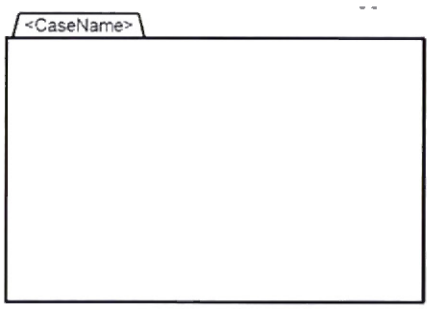


Figure 2.1 Représente la forme du modèle de plan de cas

2.1.3 La tâche

Une tâche, comme montrée par la figure 2.2, est une unité de travail, ainsi qu'une classe de base pour représenter tout le travail qui est fait dans un cas (OMG). Une tâche est un élément central du CMMN et peut avoir cinq types: *la tâche humaine non bloquante, la tâche humaine bloquante, la tâche de processus, la tâche de décision et la tâche de cas*. De plus, *une tâche discrétionnaire* est exécutée selon la discrétion du gestionnaire de cas.

Figure 2.2 Forme de tâche ordinaire et forme de tâche discrétionnaire(gauche à droite)

La tâche humaine non bloquante (OMG), telle qu'illustrée à la figure 2.3, est complétée au moment même où elle est démarrée. Par conséquent, dans le modèle de cas, lorsqu'*une tâche humaine non bloquante* commence, nous n'attendons pas qu'elle se termine: le flux de séquence continue avec la tâche suivante dans la séquence. Par contre, *la tâche humaine de blocage* (OMG), comme indiqué dans la figure 2.4, arrête

le flux de séquence jusqu'à ce qu'il soit complété. Toutes les autres tâches sont par défaut "bloquantes".

Figure 2.3 Non-blocage de formes de tâche humaines

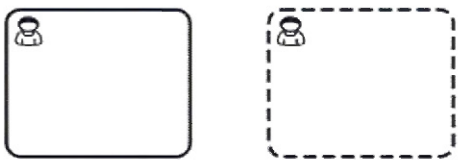


Figure 2.4 Blocage de formes de tâche humaines

La tâche de processus (OMG), illustrée à la figure 2.5, est utilisée pour créer des liens vers un diagramme BPMN. Par conséquent, en cliquant sur le symbole dans le coin supérieur gauche de l'élément, un lien vers un diagramme BPMN sera créé (Signavio).

Figure 2.5 Formes de tâche de processus

La tâche de décision (OMG), comme indiqué dans la figure 2.6, est utilisée pour répéter une tâche qui consiste en une décision représentée par un diagramme DMN.

Par conséquent, en cliquant sur le symbole dans le coin supérieur gauche de l'élément, un lien vers un diagramme DMN sera créé (Signavio).

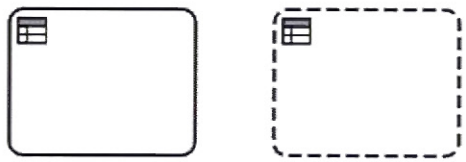


Figure 2.6 Forme de tâche de décision

Enfin, *la tâche de cas* (OMG), illustrée à la figure 2.7, est utilisée pour intégrer un modèle de cas existant dans un autre. Par conséquent, en cliquant sur le symbole dans le coin supérieur gauche de l'élément de tâche case, un lien vers un diagramme CMMN sera créé (Signavio).

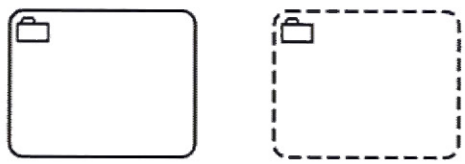
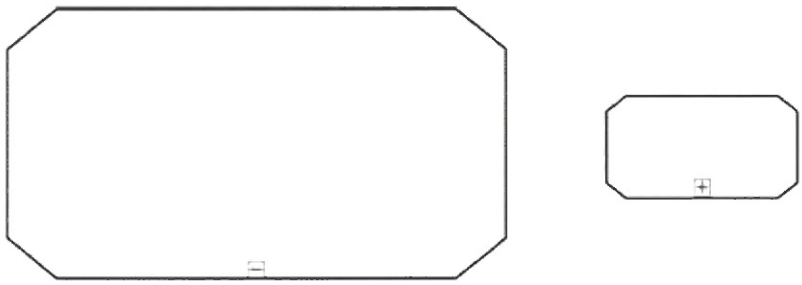


Figure 2.7 Formes de tâche de cas

2.1.4 L'étape

Une étape (OMG) est définie comme un conteneur pour organiser visuellement les tâches et les autres éléments CMMN. Ainsi, un certain nombre de flux de séquences, de tâches et de sous-étapes peuvent être représentées par une étape (Signavio). Comme le montre la figure 2.8, les étapes peuvent être agrandies ou réduites et avoir un - ou + sur le centre inférieur. Dans *une étape étendue*, les éléments qui le constituent deviennent visibles et *une étape effondrée* est liée à un autre diagramme CMMN (Signavio).



(a) (b)

Figure 2.8 Forme d'étape étendue et forme d'étape effondrée (gauche à droite)

2.1.5 L'événement

Un événement (OMG) attend que des événements spécifiques se produisent dans l'incident. Les événements marquent généralement l'activation et l'achèvement des étapes ou des tâches, ou l'atteinte des jalons (OMG). CMMN prend en charge la représentation des écouteurs d'événements, qui attendent qu'un événement particulier se produit. Différents types d'écouteurs existent (OMG), comme le montre la figure 2.9. Il y a *des auditeurs d'événement de minuteur* qui attendent un certain laps de temps pour un certain moment prédéfini. Il existe également *des écouteurs d'événements génériques* qui attendent qu'un événement se produise. Enfin, il y a *des auditeurs d'événements utilisateur* qui attendent l'entrée de l'utilisateur.

Figure 2.9 Les formes d'événement: "Auditeur d'événement", "Auditeur d'événement de minuteur" et " Auditeur d'événement d'utilisateur" (gauche à droite).

2.1.6 Le fichier de cas

Un fichier de cas (OMG), comme indiqué dans la figure 2.10, représente des documents qui contiennent des informations utilisées ou produites par le cas. Ces documents pourraient inclure des éléments d'information structurée ou non structurée qui peuvent être fournis par des sources simples ou complexes. Le contenu d'un fichier de cas peut être défini à l'aide de tout langage de modélisation d'informations (de Carvalho *et al.*, 2016). Les fichiers de cas peuvent être attachés à un autre élément en utilisant un connecteur (Signavio).

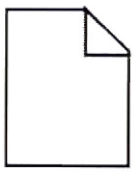


Figure 2.10 Forme du fichier de cas

2.1.7 L'étape importante

Une étape importante (OMG), comme le montre la figure 2.11, représente l'état de l'affaire. Par suite, les étapes importantes en tant que sous-objectifs dans le processus des dossiers indiquent si un certain point ou une étape a été atteinte ou complétée (Signavio). L'étape importante est représentée par une forme rectangulaire avec des extrémités à demi arrondies. De plus, une étape importante peut avoir zéro ou plusieurs critères d'entrée lorsqu'il est atteint.

Figure 2.11 Forme de l'étape importante

2.1.8 La sentinelle

Le diamant, en forme du *critère d'entrée* (OMG) et de *sortie* (OMG), appelé "sentinelle" comme le montre la figure 2.12, permet de préciser les principales conditions qui doivent être remplies pour influencer la poursuite des procédures (OMG). Ces sentinelles peuvent être attachées à des tâches, des étapes, des étapes importantes et des fichiers de cas. De plus, ils n'ont même pas besoin d'être attachés à d'autres éléments. Les sentinelles peuvent rester seules dans un flux de séquence (Signavio).

Une sentinelle utilisée comme un critère d'entrée est dépeinte par "un diamant" peu profond. Ceci indique que le flux de séquence entrant directement attaché à la sentinelle doit être terminé avant que le flux de séquence puisse continuer (Signavio). Une sentinelle utilisée comme critère de sortie est représentée par un «diamant» solide qui se présente lorsqu'un élément du plan est complet. Cela implique que la séquence peut continuer dans quelle direction.

Figure 2.12 Forme du critère d'entrée et Forme du critère de sortie (gauche à droite)

2.1.9 Le connecteur

Connecteurs (OMG), définit les relations entre les éléments CMMN, comme indiqué dans la figure 2.13. Un objet connecteur est représenté par une ligne pointillée qui ne comporte pas de pointes de flèches. Cependant, la direction du flux ou de l'association est déterminée par la présence d'une sentinelle (*le critère d'entrée ou le critère de sortie*) (OMG).

Figure 2.13 Forme du connecteur

Par exemple, le diagramme illustré à la figure 2.14 illustre une situation où le critère d'entrée de la tâche B dépend de l'achèvement de la tâche A (OMG).

Figure 2.14 La sentinelle a basé la dépendance entre deux tâches

2.1.9.1 Utilisation de Connecteur

Les connecteurs peuvent être utilisés pour visualiser les dépendances entre les éléments du plan. Par exemple, la figure 2.15 suivante illustre une situation dans laquelle la tâche C peut être activée uniquement si la tâche A et la tâche B sont terminées (OMG).

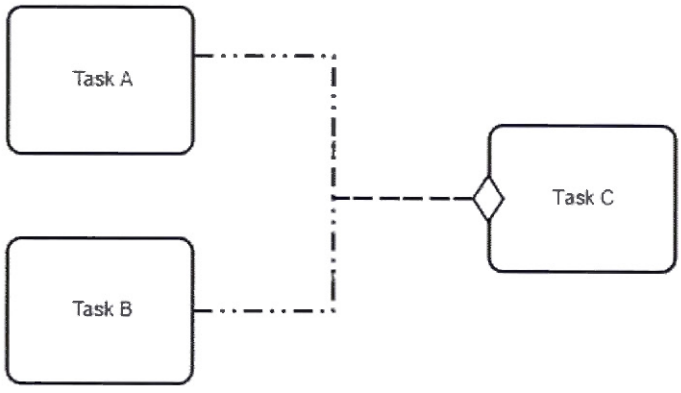


Figure 2.15 L'utilisation de connecteurs pour visualiser "AND"

La Figure 2.16 illustre une situation dans laquelle la tâche C peut être activée si la tâche A ou la tâche B est terminée (OMG).

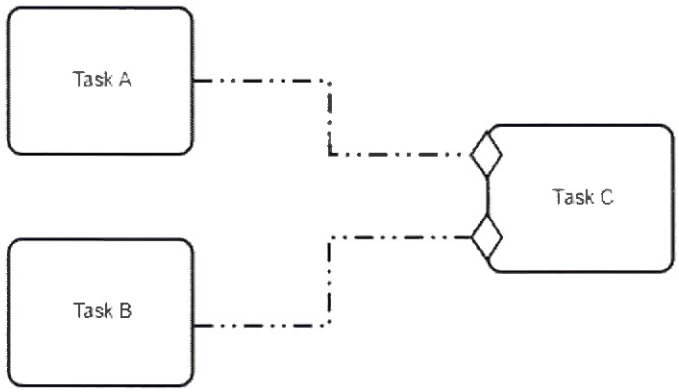


Figure 2.16 L'utilisation de connecteurs pour visualiser "OR"

La figure 2.17 montre une situation où l'étape B dépend du critère de sortie de l'étape A (OMG).

Figure 2.17 Visualiser la dépendance entre des étapes Utilisation de connecteur à base de sentinelle

La figure 2.18 montre une situation où la tâche A dépend de la réalisation du Étape importante A. (OMG).

Figure 2.18 Visualiser la dépendance entre une tâche et une étape importante utilisation du connecteur à base de sentinelle

La figure 2.19 montre une situation où la tâche A dépend de "*auditeur d'événement de minuterie*" recevoir " *événement de minuteur*" (OMG).

Figure 2.19 Visualiser la dépendance entre une tâche et un écouteur d'événement de minuterie utilisation du connecteur à base de sentinelle

La figure 2.20 montre une situation dans laquelle la tâche A dépend d'un élément de fichier de cas (OMG).

Figure 2.20 Visualiser la dépendance entre une tâche et un élément de fichier de casutilisation du connecteur à base de sentinelle

2.2 Exemple de modèle de plan de cas

Dans cette section, comme le montre la figure 2.21, nous illustrons l'utilisation des divers éléments en représentant un exemple de traitement de revendication utilisé dans la norme.

Le modèle de plan de cas ci-dessous contient toutes les activités pour représenter le cas d'un certificat de formation CMMN. Ce cas est organisé en deux étapes distinctes qui comprennent Secrétaire et CMMN-Trainer. Au début, la tâche de "*auditeur d'événement d'utilisateur*" attend les nouveaux employés qui veulent commencer la formation. D'une part, le flux de séquence est arrêté jusqu'à ce que la liste des employés sans formation soit prête. D'un autre côté, selon la discrétion du gestionnaire de cas, la tâche discrétionnaire vérifie que les stagiaires potentiels sont libres. Ces deux tâches doivent être complètes afin d'informer les stagiaires potentiels et le formateur du cours prévu. Dans la suite, l'étape importante définit l'état de l'étape du secrétaire représenté qui indique la formation CMMN.

Dans l'étape CMMN-Trainer, avant de commencer l'entraînement, '*auditeur d'événement de minuteur*' définit un certain laps de temps pour la durée du cours. Dans la suite, le formateur explique les sujets de la formation aux employés. À la fin de la formation, *l'élément de fichier de cas* produit par le dossier contient un certificat de formation CMMN. Une sentinelle utilisée comme critère de sortie se présente lorsqu'un élément du plan est terminé.

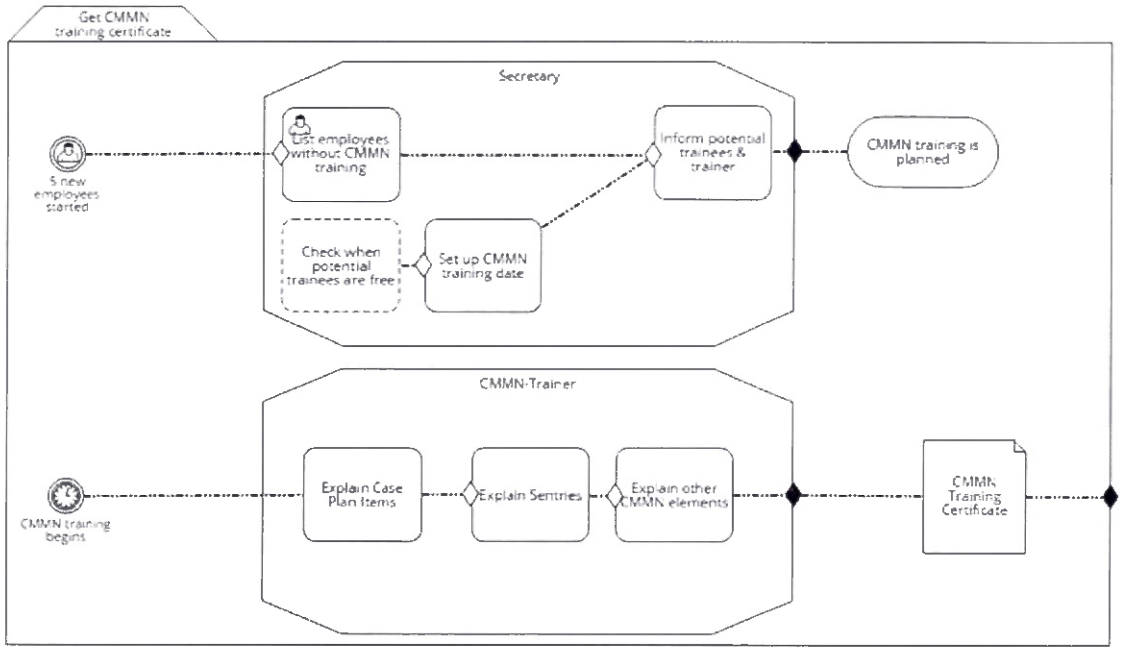


Figure 2.21 Les combinaisons d'éléments divers dans CMMN (Signavio)

2.3 Conclusion

Dans ce chapitre, le concept de CMMN a été décrit. De plus, les notations CMMN et leur signification dans le contexte de CMMN ont été expliquées afin de modéliser les constructions de base d'un cas. Nous avons utilisé un exemple de traitement de revendications pour illustrer l'utilisation des divers éléments de CMMN. Dans le chapitre suivant, les esquisses dessinées à la main des modèles CMMN sont reconnues à l'aide d'extractions de caractéristiques et d'heuristiques. Nous décrirons notre approche en détail et expliquerons les questions posées à chaque étape de la mise en œuvre.

CHAPITRE III

RÉSULTATS EXPÉRIMENTAUX

La reconnaissance d'esquisse est une approche qui reconnaît automatiquement les diagrammes dessinés à la main en utilisant un ordinateur. Ainsi, les recherches dans ce domaine au cours des dernières années se sont développées grâce aux progrès réalisés en intelligence artificielle et en interaction homme-machine. De même, la reconnaissance des dessins faits à la main peut être considérée comme un sous-domaine du domaine de reconnaissance d'esquisse. Ainsi, cela peut faire partie de la capacité d'un ordinateur à recevoir et à interpréter une entrée manuscrite de sources qui peuvent inclure un document papier, des écrans tactiles, une image et plusieurs autres dispositifs. De plus, différents types d'algorithmes de reconnaissance sont utilisés, tels que ceux basés sur les gestes, sur l'apparence, sur la géométrie, ou une combinaison de ceux-ci.

Dans ce chapitre, nous expliquons la mise en œuvre de l'outil qui étudie les différentes étapes de la reconnaissance d'esquisse des éléments du CMMN dessinée à la main, et nous analysons un ensemble de problèmes à chaque étape et nous décrivons leurs solutions en détail.

3.1 Étude de cas

Cette mémoire porte sur la reconnaissance des esquisses des éléments du CMMN dessinées à la main. Pour cela, nous répondons aux deux questions suivantes: 1)

comment identifier des constructions élémentaires de CMMN dans des esquisses dessinés à la main par l'utilisateur, et 2) comment reconnaitre les relations sémantiques entre eux d'une façon compatible avec la sémantique de CMMN. La partie gauche de la figure 3.1 montre à quoi pourrait ressembler une entrée et la partie droite de la figure montre le modèle CMMN correspondant, tel que visualisé par un outil de modélisation CMMN.

En d'autres termes, nous devons créer un prototype de modèle de cas à partir d'une esquisse dessinée à la main afin de l'exporter vers un système de gestion des processus métier ou de gestion de cas pour l'automatisation des cas.

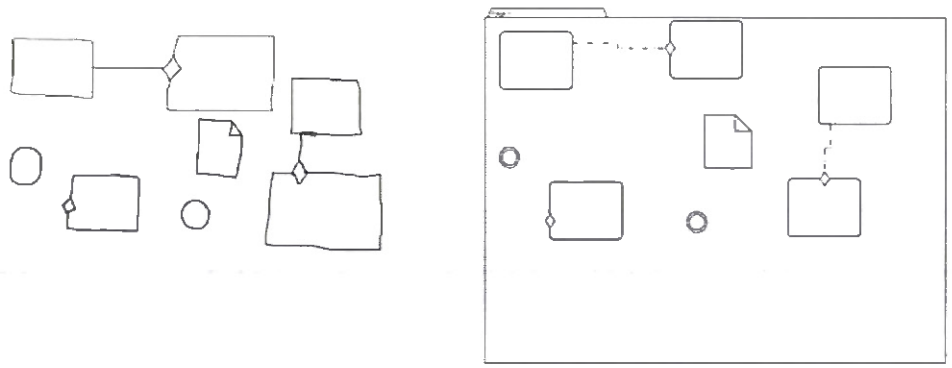


Figure 3.1 La formalisation du modèle de cas à partir d'une esquisse CMMN dessinée à la main.

En effet, afin de reconnaître les éléments CMMN à partir d'esquisses dessinées à la main, nous avons spécifié trois étapes qui incluent:

- La reconnaissance d'une forme primitive;
- La reconnaissance des objets graphiques composites, et
- La reconnaissance et la compréhension des connexions sémantiques.

Au début, l'utilisateur dessine une esquisse qui inclut des éléments CMMN. Par conséquent, la première étape consiste à lire une image brute comme entrée et à reconnaître les contours comme des formes primitives. La seconde étape consist à

reconnaître des formes composites correspondant à des éléments CMMN qui combinent les formes primitives en fonction de leurs relations spatiales. La troisième étape de la reconnaissance consiste à comprendre les connexions sémantiques entre les formes primitives, en fonction des relations spatiales des éléments CMMN.Par la suite, nous pouvons exporter le modèle de résultat dans un fichier XML en utilisant le format d'échange de modèle de CMMN.

Dans la prochaine partie de ce chapitre, chaque étape est expliquée en détail. En particulier, nous allons découvrir les problèmes inhérents à chaque étape, et décrire les algorithmes qui tenteront de les résoudre.

3.2 Technologie Utilisée

Notre prototype a été produit en Java sur l'IDE eclipse, en utilisant la librairie OpenCV (OpenCV). Cette bibliothèque est une bibliothèque C++ open source, (voir figure 3.2), qui est utilisée pour le traitement d'images en temps réel et les applications de la vision par ordinateur. OpenCV a une structure modulaire, ce qui signifie qu'il a des centaines d'algorithmes de traitement d'image et de vision par ordinateur qui rendent le développement d'applications de vision par ordinateur avancées et faciles (OpenCV).

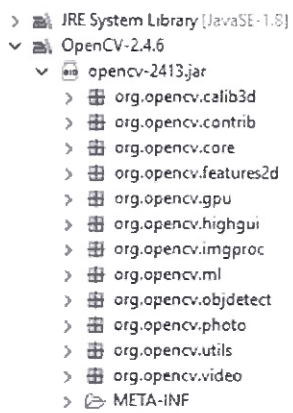


Figure 3.2 Bibliothèque OpenCV sur l'eclipse IDE

Dans OpenCV, les images numériques sont représentées en utilisant des variables numériques pour chaque point. Ainsi, les images sont représentées en utilisant des matrices ainsi que des métadonnées sur l'image. Pour donner une illustration (voir figure 3.3), le miroir de la voiture est une matrice contenant toutes les valeurs d'intensité des points du pixel (OpenCV documentation, 2013).

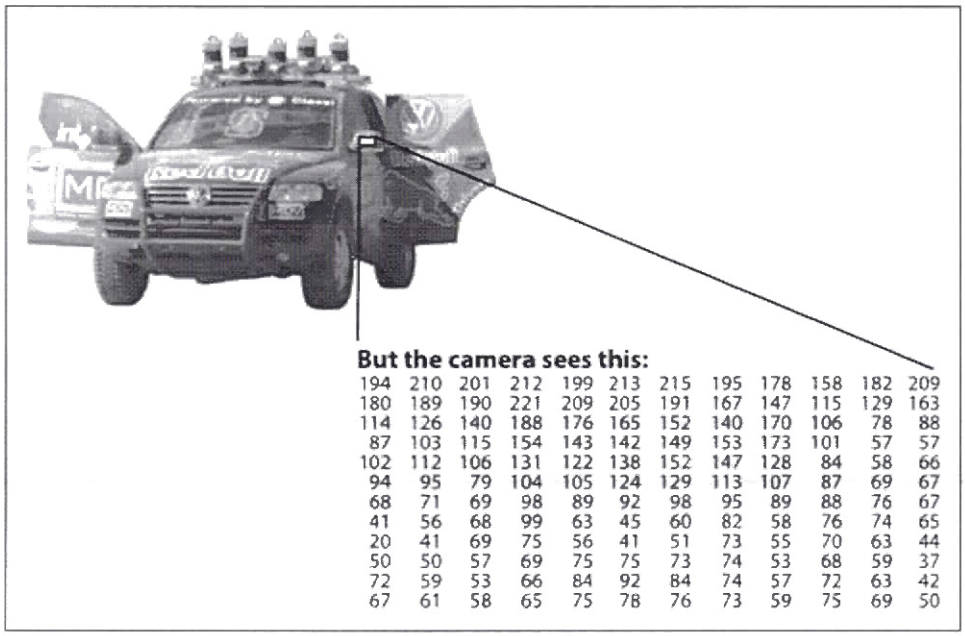


Figure 3.3 Représente la structure de données dans OpenCV (Szeliski, 2010)

3.3 Conception et Implémentation de la Reconnaissance d'Esquisse

Notre approche, malgré sa simplicité, offre une vue d'ensemble du système de reconnaissance et de la connexion des classes afin de mettre en œuvre les trois étapes de reconnaissance mentionnées plus haut. Dans ce qui suit, chaque étape est décrite en détail. La figure 3.4 donne un aperçu du modèle de conception du prototype. Les sections suivantes donneront plus de détails sur les parties pertinentes du modèle de conception.

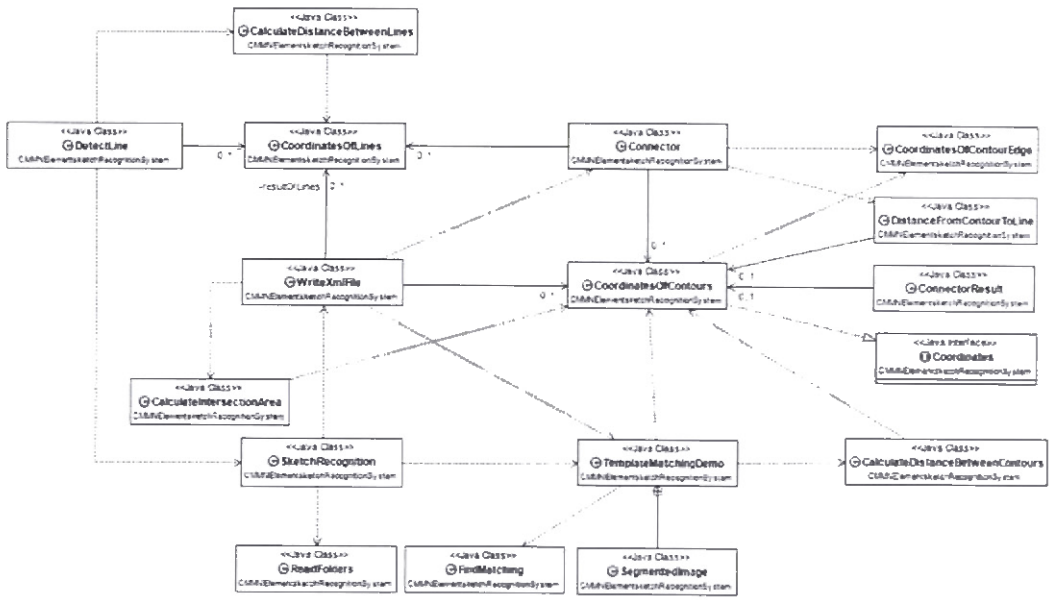


Figure 3.4 Modèle de conception compatible avec l'étude de cas

3.4 Reconnaissance de Forme Primitive

La définition de la forme primitive consiste à la reconnaissance des constructions CMMN élémentaire dans les esquisses faites de l'utilisateur à la main. Dans cette section, nous étudions comment extraire les contours de l'intérieur de l'esquisse et les reconnaître comme les formes primitives qui forment la notation CMMN.

3.4.1 Lecture de l'esquisse dessinée à la main

Afin de reconnaître les formes primitives dans une esquisse faite à la main, le programme doit trouver et lire l'esquisse dessinée à la main en tant qu'entrée. La figure 3.5 montre un exemple de formes primitives CMMN. La définition de chaque forme a été expliquée dans le chapitre II.

Ainsi, nous devons trouver les chemins du répertoire contenant les entrées et les modèles prédéfinis. Pour ce faire, la classe Java "ReadFolders" a été définie, comme indiqué dans l'Annexe A, afin de lire le chemin du répertoire principal et également la

liste de tous les fichiers du répertoire et de ses sous-répertoires. De plus, nous avons défini une condition pour lire uniquement les fichiers avec l'extension JPG ou PNG pour éviter de lire d'autres types de fichiers incompatibles.

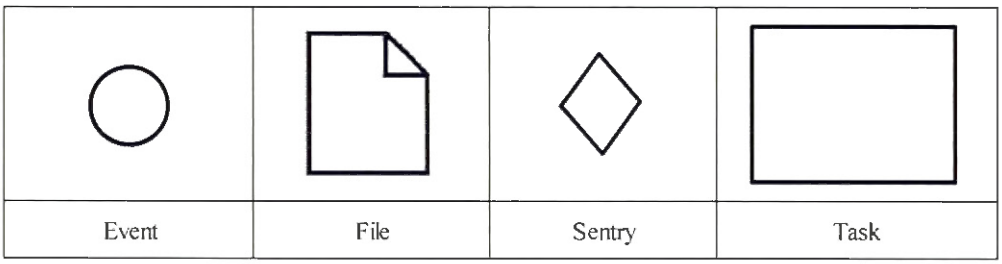


Figure 3.5 Images de modèles

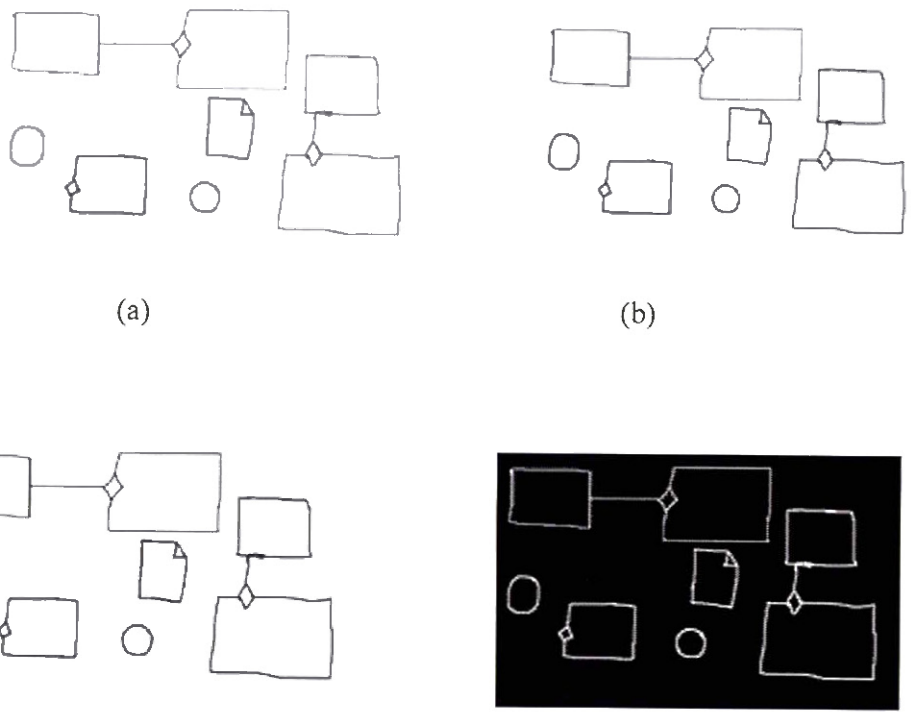
3.4.2 Prétraitement

Pour détecter des objets, comme nous l'avons mentionné au chapitre II, des étapes de prétraitement sont nécessaires. Par conséquent, nous avons utilisé la bibliothèque OpenCV pour implémenter certaines fonctions de prétraitement sur l'image donnée en entrée afin d'améliorer le résultat final (Figure 3.6). Ces fonctions permettent d'effectuer diverses opérations de filtrage linéaires ou non linéaires sur des images 2D représentées par une matrice bidimensionnelle (OpenCV).

La première étape du processus de prétraitement consiste à convertir une image de RGB en niveaux de gris. Cela signifie qu'une image d'entrée sera convertie d'un espace de couleur à un autre (OpenCV documentation, 2013). Le format de couleur par défaut dans OpenCV est BGR, en d'autres termes, les octets sont inversés. Ainsi, une image de couleur standard (24 bits) est composée de 8 bits de bleu, 8 bits vert et 8 bits rouge. Cette séquence est répétée pour chaque pixel (Bleu, Vert, Rouge), et ainsi de suite. (OpenCV documentation, 2013). «Lorsque les images en niveaux de gris sont converties en images couleur, toutes les composantes de l'image résultante sont considérées comme égales; mais pour la transformation inverse, la valeur grise

est calculée en utilisant l'équation pondérée perceptuellement.» (Bradski et Kaehler, 2008):

$$\text{RGB TO GRAY:} \quad Y \leftarrow 0.299\,R + 0.587\,G + 0.114\,B \tag{3.1}$$



(a) (b)

(c) (d)

Figure 3.6 Les étapes de prétraitement du système de reconnaissance: (a)→(b) L'opération « Niveaux de gris » convertit l'image RGB en image en niveaux de gris; (b) →(c) Opération de « réduction de flou » réduit le bruit et le lissage de l'image en niveaux de gris; (c)→(d) L'opération de « seuillage » affecte une valeur (couleur noire) à la valeur de pixel supérieure à la valeur de seuil, sinon affecter une autre valeur (couleur blanc).

La deuxième étape du processus de prétraitement est appelée la réduction du flou de l'image. Elle consiste à réduire le bruit et lisser l'image. Pour d'effectuer une

opération de lissage, nous appliquons un filtre. La performance globale du filtre est que «la valeur d'un pixel de sortie (*c.-à-d. g (i, j)*) est déterminée comme une somme pondérée de valeurs de pixels voisins d'entrée (*c.-à-d. f (i + k, j + l)*) » (OpenCV documentation, 2013), typiquement $-1 \leq k \leq 1$ et $-1 \leq l \leq 1$ (figure 3.7). De plus, le coefficient du filtre appelé noyau est défini par h *(k, l)* (OpenCV documentation, 2013). L'équation ci-dessous montre l'opération de lissage (OpenCV documentation, 2013):

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l) \tag{3.2}$$

La bibliothèque OpenCV est livrée avec plusieurs filtres. Nous avons utilisé le filtre gaussien (OpenCV documentation, 2013; Szeliski, 2010), qui est appliqué en «convolant chaque point dans le tableau d'entrée avec un *noyau gaussien* et en additionnant tous les points qui contribuent à produire le tableau de sortie » (OpenCV documentation, 2013; Szeliski, 2010). Afin de rendre le contexte plus clair, un noyau gaussien 1D est présenté (figure 3.8). Donc, le pixel situé au milieu contient le plus gros poids. Par conséquent, «le poids de ses voisins diminue au fur et à mesure que la distance spatiale entre eux et le pixel central augmente » (OpenCV documentation, 2013; Szeliski, 2010).

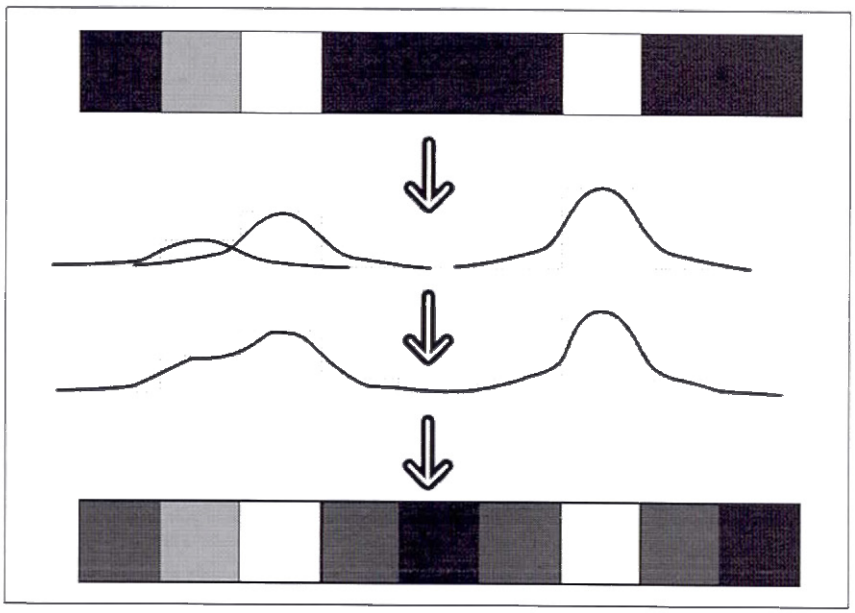


Figure 3.7 Présente un noyau gaussien d'un tableau de pixels à une dimension (Szeliski, 2010)

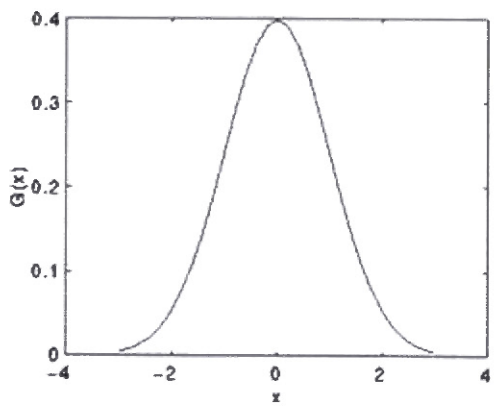


Figure 3.8 1D Le noyau gaussien (OpenCV documentation, 2013)

Donc, un Gaussien 2D peut être représenté en utilisant l'équation suivante (OpenCV documentation, 2013; Szeliski, 2010):

$$G_0(x,y) = Ae^{\frac{-(x-\mu_x)^2}{2\sigma_x^2} + \frac{-(y-\mu_y)^2}{2\sigma_y^2}} \tag{3.3}$$

Où μ est le pic et σ représente la variance (pour chacune des variables x et y) (OpenCV documentation, 2013; Szeliski, 2010).

Seuillage (OpenCV documentation, 2013), La dernière étape du prétraitement de l'image est une opération non linéaire qui segmente les images en niveaux de gris afin de les convertir en images binaires. Le seuillage donne des valeurs de seuil, les pixels dont la valeur est supérieure au seuil sont conservés en blanc et les autres sont colorés en du noir.

Il y a plusieurs opérations de seuillage dans OpenCV, mais nous avons choisi la fonction binaire inversée (figure 3.9) (OpenCV documentation, 2013), pour effectuer une opération de seuil. Par conséquent, si la valeur de pixel est supérieure à une valeur de seuil, une valeur lui est affectée (en noir), sinon, elle est affectée à une autre valeur (en blanc). L'opération de seuillage peut être exprimée comme l'équation suivante (OpenCV documentation, 2013):

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxVal} & \text{otherwise} \end{cases} \tag{3.4}$$

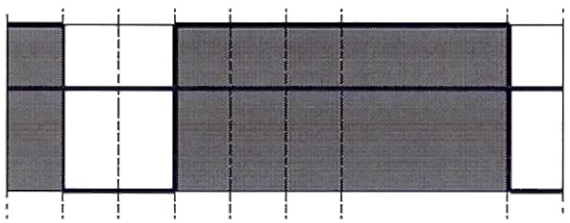


Figure 3.9 Opération de seuillage: Si l'intensité du pixel *src(x,y)* est supérieure au seuil, alors la nouvelle intensité de pixel est définie à *0*. Sinon, il est réglé sur *MaxVal* . La ligne horizontale bleue représente le seuil *thresh* (OpenCV documentation, 2013)

La méthode java "preprocessImage", comme indiqué dans l'Annexe A, montre les détails des opérations du prétraitement dc l'image.

Pour identifier les différentes formes primitives, nous devons comparer chaque forme primitive contenue dans l'esquisse dessinée à la main aux images du modèle prédéfini. Par conséquent, une table sera spécifiée pour classer les images du modèle en fonction de leur nom aussi, et stocker leurs caractéristiques en fonction de leur nom. Notez que le même prétraitement est appliqué aux formes CMMN prédéfinies pour permettre une comparaison équitable. L'annexe A présente les méthodes de prétraitement de tous les modèles avec des détails.

3.4.3 Découverte de Contour

Avant de comparer l'esquisse faite à la main aux modèles CMMN, nous devons convertir à la fois l'image d'entréc et les modèles en contours. Un contour est une liste de points qui représentent une image binaire (Bradski et Kaehler, 2008), Un esquisse dessiné à la main est composé de la séquence de points, qui représentent les formes. Ainsi, la première étape de l'identification des formes dans l'esquisse dessinée à la main et de leurs caractéristiques est de détecter les contours.

La fonction "findcontours" (Bradski et Kaehler, 2008) dans la bibliothèque OpenCV, trouve les contours dans une image binaire. Par conséquent, chaque contour est une séquence de points, chacun représenté par quatre valeurs qui consistent en quatre éléments importants en tant que pointeurs vers d'autres points de la séquence. OpenCV représente le contour en tant que tableau de quadruples (Bradski et Kaehler, 2008):

(h_prev, h_next, v_prev, and v_next)

Comme le montre la figure 3.10, où **h_prev** est la coordonnée horizontale (x) du point précédent dans le contour, **h_next** est la coordonnée horizontale du point

suivant de la séquence, et **v_prev** et **v_next** sont les coordonnées verticales (y) du précédent et le point suivant dans la séquence, respectivement.

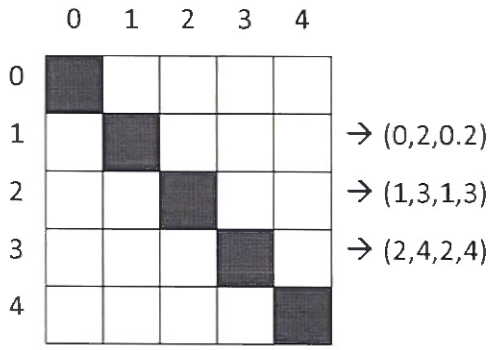


Figure 3.10 La présentation graphique d'une image de pixel et le contour correspondant

Notons que, dans certains cas, certains contours sont à l'intérieur d'autres contours, comme la sentinelle du CMMN, qui est un diamant faisant partie d'une tâche (voir la figure 3.11). Le premier problème qui peut se produire est d'essayer de comprendre comment extraire les contours intérieurs des contours extérieurs.

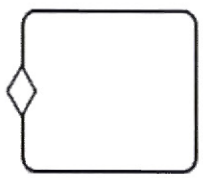


Figure 3.11 Représente un modèle de sentinelle

Dans ce cas, nous pouvons appeler le contour externe comme un *parent* et le contour interne comme un *enfant*. En définissant cette structure (Bradski et Kaehler, 2008), nous pouvons spécifier comment les contours sont connectés les uns aux autres. En d'autres termes, le contour peut être un fils ou un parent d'un autre contour. Ce type de relation s'appelle la *hiérarchie* (Bradski et Kaehler, 2008).

Afin de récupérer la hiérarchie, le mode de récupération de contour qui est un argument de la fonction "findcontours" est utilisé. Il existe quatre types différents de mode de récupération à savoir (Bradski et Kaehler, 2008):

RETR_EXTERNAL, RETR_LIST, RETR_TREE and RETR_CCOMP

RETR_EXTERNAL (Bradski et Kaehler, 2008) renvoie uniquement les contours extérieurs extrêmes. Comme le montre la figure 3.12, il n'y a qu'un seul contour externe. Par conséquent, la figure 3.13- (a) représente les premiers points du contour en tant que séquence la plus externe et il n'y a pas de contours internes.

RETR_LIST (Bradski et Kaehler, 2008) est le mode de récupération le plus simple qui récupère tous les contours sans créer de relation parent-fils et les place dans une liste. En d'autres termes, tous les contours sont au même niveau. La figure 3.13- (c) illustre la liste de l'image de la figure 3.12. Par conséquent, tous les contours sont connectés les uns aux autres par **h_prev** et **h_next**.

RETR_TREE (Bradski et Kaehler, 2008) récupère tous les contours afin de créer une liste hiérarchique complète. Comme le montrent les figures 3.12 et 3.13- (d), le nœud racine de l'arbre est le contour le plus externe. En dessous du nœud racine, chaque nœud est connecté à l'autre trou au même niveau. En outre, «chacun de ces nœuds, à son tour, a des fils, qui sont reliés à leurs parents par des liens verticaux. Cela continue jusqu'aux contours les plus intérieurs de l'image, qui deviennent des feuilles dans l'arbre.» (Bradski et Kaehler, 2008).

Finalement, *RETR_CCOMP* (Bradski et Kaehler, 2008) récupère tous les contours et les organise en une hiérarchie à deux niveaux. Par conséquent, les limites de niveau supérieur sont placées dans la hiérarchie-1 qui est le premier niveau; les nœuds limites sont placés dans la hiérarchie-2 qui représente le deuxième niveau (Bradski et Kaehler, 2008). Comme le montre la figure 3.13- (b), « les nœuds limites sont connectées à leurs limites extérieures correspondantes par **v_next** et **v_prev** »

(Bradski et Kaehler, 2008). De plus, tous les trous sont reliés les uns aux autres par les pointeurs h_prev et h_next (Bradski et Kaehler, 2008).

Rappelons que le mode RETR_CCOMP récupère une hiérarchie à deux niveaux où le premier niveau représente les contours externes qui agissent en tant que parents, et le deuxième niveau contient les contours internes qui agissent comme des fils. Par conséquent, nous devons écrire un algorithme qui ne récupère que les fils des parents. Dans l'algorithme ci-dessous, selon la figure 3.12, nous représentons comment extraire tous les niveaux de la hiérarchie en tant que fils et parent et comment extraire le niveau hiérarchique pertinent pour les contours internes.

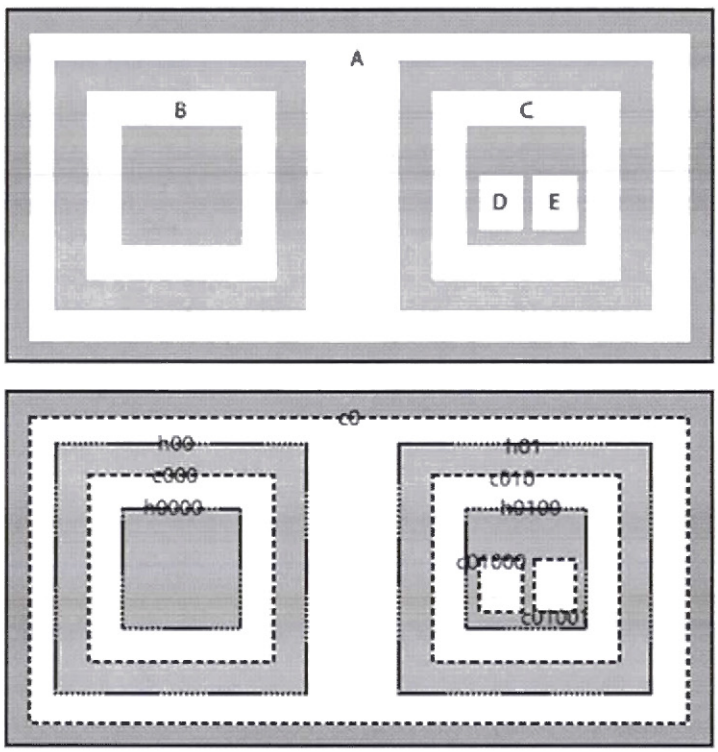


Figure 3.12 Une image de test présente les contours qui pourraient être des contours extérieurs (lignes pointillées) ou des contours intérieurs (Bradski et Kaehler, 2008)

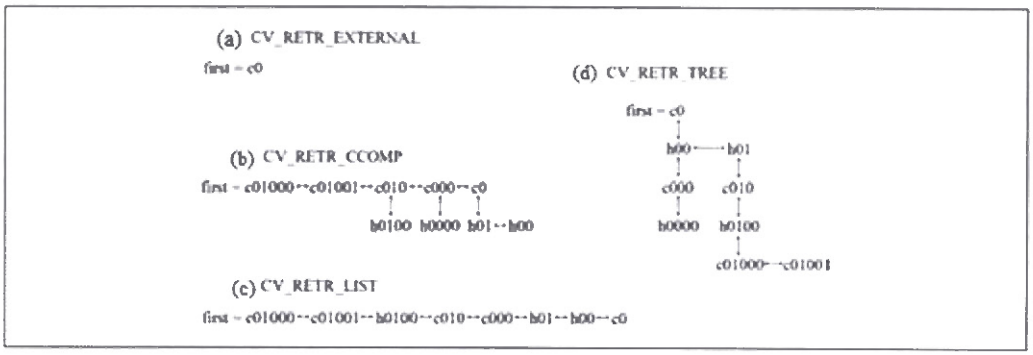


Figure 3.13 Différents types de modes de récupérationpour trouver les contours dans OpenCV (Bradski et Kaehler, 2008)

La classe Java "segmentImage", présentée à l'annexe A, est responsable de la recherche des contours dans l'esquisse dessinée à la main.

3.4.4 Extraction de Caractéristiques

Dans cette étape, après avoir trouvé tous les contours des fils, nous devons extraire les caractéristiques de chaque contour. En trouvant les caractéristiques du contour telles que la longueur, la zone et la forme rectangulaire, nous spécifions les contours en tant que formes indépendantes. En extrayant ces caractéristiques, nous pouvons commencer à rechercher des correspondances entre les formes indépendantes et les images modèles du CMMN.

En utilisant la fonction "boundingRect" dans OpenCV, nous pouvons trouver la forme rectangulaire qui englobe le contour du dessin (voir figure 3.14) et extraire ses caractéristiques telles que la coordonnée en haut à gauche du rectangle comme point de départ du contour, ainsi que sa largeur et hauteur comme la largeur et la hauteur du contour (Bradski et Kaehler, 2008). Ainsi, chaque contour sera présenté en fonction de ces caractéristiques.

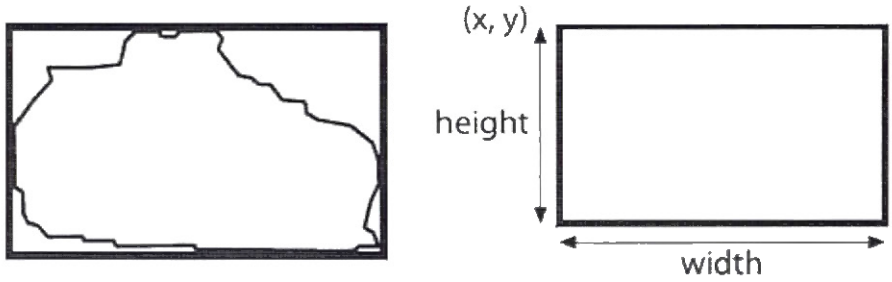


Figure 3.14 Forme rectangulaire qui englobe le contour du dessin (Bradski et Kaehler, 2008)

La classe Java "getShapeSubBitMap", comme indiquée dans l'annexe A, est responsable de la spécification de la forme rectangulaire qui englobe le contour du dessin. L'algorithme suivant montre la fonction principale.

Algorithm 1 : Feature Extraction

Input: A list of contours $C = C_1, ..., C_k$
Output: A list of contours features

```
w: the width of rectangle box
h: the height of rectangle box
x: the start coordinate x of rectangle box
y: the start coordinate y of rectangle box
C_w: the width of contour
C_h: the height of contour
for C = C_1 to C_k do
    C_w ←(y, y+h)
    C_h ←(x, x+w)
return (C_w, C_h)
```

3.4.5 Contour Redimensionnant

Lorsqu'on dessine des esquisses à la main, on s'intéresse rarement à la taille de leur esquisse ainsi que la proportionnalité de leurs figures, c'est-à-dire la taille relative de la hauteur par rapport à la largeur par exemple. Afin de comparer les contours reconnus dans le dessin à la main à ceux des modèles CMMN, nous devons

redimensionner les contours dessinés à la main à la même dimension que le modèle que nous essayons de faire correspondre. En d'autres termes, nous devons redimensionner chaque contour en fonction de la surface de chaque image de gabarit en conservant les proportions. Le code suivant, présenté dans l'annexe A, comme la classe Java "getResizeSize" montre la façon dont nous avons redimensionné le contour en conservant ses proportions.

Algorithm 2 : Resize Contour(C)

Input: A list of contours $C = C_1, ..., C_k$, a list of template's images $T = T_1, ..., T_n$

Output: A list of re-size contours

```
1  C_w: the width of contour
2  C_h: the height of contour
3  T_w: the width of template
4  R_w: the re-size width of template
5  R_h: the re-size height of template
6  R_w ←0
7  R_h ←0
8  for C = C_1 to C_k do
9  |   scale ←C_w / C_h
10 |   R_w ← T_w
11 |   R_h ← R_w / scale
12 return (R_w, R_h)
```

En redimensionnant les contours, nous pouvons passer à l'étape suivante, qui consiste à trouver la correspondance entre chaque contour et chaque image de modèle.

3.4.6 Correspondance de Modèle

Maintenant que tous les contours fermés dans l'image d'entrée ont été récupérés et redimensionnés en fonction de la zone d'image de chaque modèle, nous pouvons commencer à nous concentrer sur la partie principale de la reconnaissance d'esquisse. Template Matching est une méthode pour rechercher et trouver l'emplacement d'une image de modèle dans un contour. Dans ce sens, OpenCV fournit une fonction "matchTemplate" (Bradski et Kaehler, 2008) à cet effet. Cette méthode (figure 3.15,

figure 3.17) commence à faire glisser l'image du gabarit sur le contour en deux dimensions, et compare le gabarit et le contour sous l'image du gabarit, à la recherche d'une correspondance forte (Bradski et Kaehler, 2008).

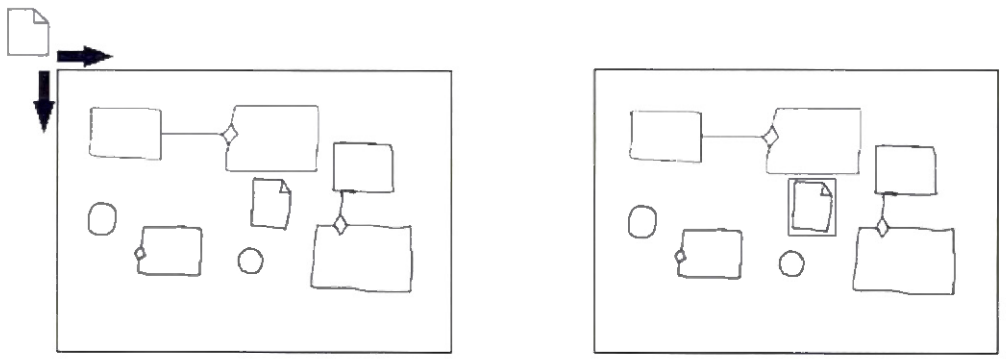


Figure 3.15 Représente la fonction MatchTemplate: commence à superposer l'image de modèle sur l'image d'entrée afin de trouver des correspondances (Mup).

La fonction de correspondance de modèle peut implémenter l'une des trois méthodes de correspondance qui incluent:

(a) Méthode de correspondance des différences carrées (method = TM_SQDIFF)

(b) Méthodes de correspondance de corrélation (method = TM_CCORR)

(c) Méthodes de correspondance des coefficients de corrélation (method = TM_CCOEFF)

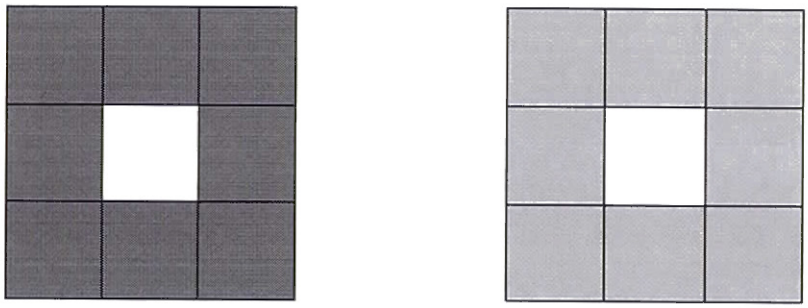
Dans la suite, nous allons expliquer chaque méthode avec sa formule mathématique.

(a) Méthode de correspondance des différences carrées (method = TM_SQDIFF)

« Les résultats de ces méthodes (figure 3.17) correspondent à la différence au carré, donc une correspondance parfaite sera 0 et les mauvaises correspondances seront grandes » (Bradski et Kaehler, 2008).

$$R_{sq\_diff}(x,y)=\sum_{x',y'}[T(x',y')-I(x+x',y+y')]^2 \tag{3.5}$$

Afin de clarifier la signification de la méthode, nous avons défini dans la figure 3.16 deux matrices qui ont les mêmes dimensions. Lorsque les deux matrices ont la même taille, nous aurons une seule valeur à calculer. Donc, normalement, le point avec le score le plus élevé sera toujours (Ø, Ø).



(a) Image d'entrée par matrice 3 par 3 (b) Image de modèle par matrice de 3 par 3

Figure 3.16 Représente l'image d'entrée et l'image de modèle en tant que matrice

(b) Méthodes de correspondance de corrélation (method = TM_CCORR)

« Le résultat de ces méthodes (Figure 3.17) fait correspondre le gabarit à l'image de manière multiplicative, de sorte qu'une correspondance parfaite sera grande et que les mauvaises correspondances seront petites ou égales à 0» (Bradski et Kaehler, 2008).

$$R_{\text{CCORR}}(x,y)=\sum_{x',y'} T(x',y')\cdot I(x+x',y+y') \tag{3.6}$$

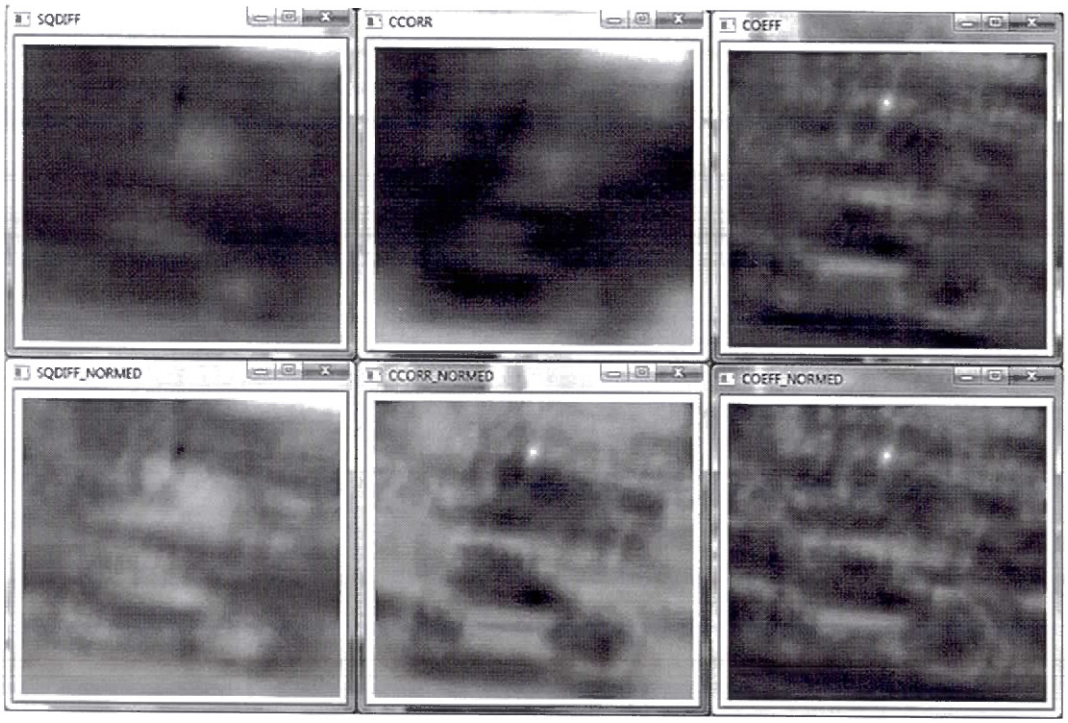
(c) Méthodes de correspondance des coefficients de corrélation (method= TM_CCOEFF)

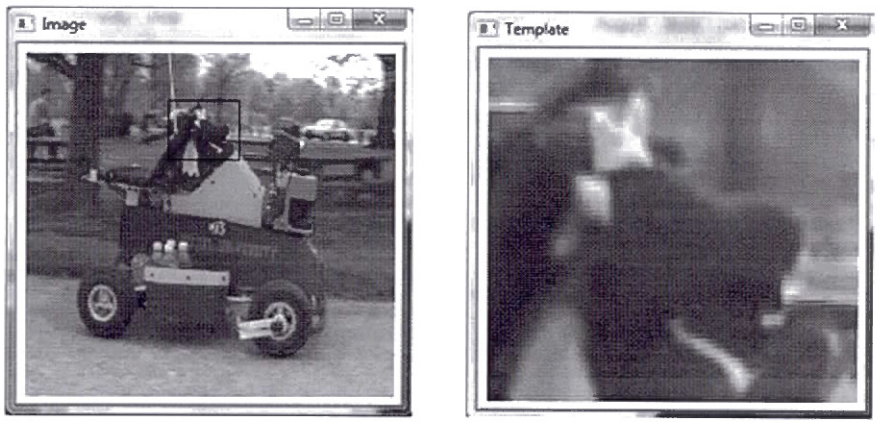
« Le résultat de ces méthodes (figure 3.17) fait correspondre une template relativement à sa moyenne, à une image relativement à sa moyenne, donc une correspondance parfaite sera 1 et une différence parfaite sera -1; une valeur de 0 signifie simplement qu'il n'y a pas de corrélation (alignements aléatoires) » (Bradski et Kaehler, 2008).

$$R_{ccoeff}(x,y)=\sum_{x',y'} T'(x',y')\cdot I'(x+x',y+y') \tag{3.7}$$

$$T'(x',y')=T(x',y')-\frac{1}{(w\cdot h)\sum_{x'',y''} T(x'',y'')} \tag{3.8}$$

$$I'(x+x',y+y')=I(x+x',y+y')-\frac{1}{(w\cdot h)\sum_{x'',y''} I(x+x'',y+y'')} \tag{3.9}$$





a) Image d'entrée b) Image de modèle

Figure 3.17 Représente les résultats de correspondance de six méthodes de correspondance selon les illustrations a) image d'entrée et b) image de modèle (Mup)

La meilleure correspondance pour la différence de carrés est 0 et pour les autres méthodes, c'est le point maximum; ainsi, les correspondances sont indiquées par des zones sombres dans la colonne de gauche et par des points brillants dans les deux autres colonnes » (Bradski et Kaehler, 2008).

En implémentant ces trois méthodes de correspondance, basées sur le résultat de correspondance expérimentale le plus élevé, nous avons sélectionné la méthode TM_CCORR_NORMED qui représente la formule suivante:

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}} \tag{3.10}$$

Le résultat de cette comparaison renvoie une image en niveaux de gris, où chaque pixel définit la correspondance entre le voisinage de ce pixel et chaque image du modèle (OpenCV).

Une matrice vide doit être définie pour stocker le résultat. D'une part, par défaut dans OpenCV, l'image de la zone de saisie est plus grande que la zone du modèle. D'un autre côté, la surface de chaque contour qui est déjà extraite et redimensionnée en conservant le rapport d'aspect peut être plus grande que la taille du gabarit. Par conséquent, avant de définir la matrice vide pour stocker le résultat de correspondance, nous devons spécifier quelle zone (zone de contour ou zone de modèle) est la plus grande. Dans ce qui suit, la taille de la matrice la plus grande est définie comme *(WxH)* et la taille de la matrice plus petite est définie comme *(wxh)*. La taille du résultat qui contient l'image de sortie sera *(W-w+1, H-h+1)*. L'algorithme suivant affiche la fonction.

Algorithm 3 : Result Matrix(R)

Input: A list of resize contours $C = C_1, ..., C_k$, a list of template's images $T = T_1, ..., T_n$

Output: An empty matrix for storing result of matching

```
(W, H): the size of bigger matrix
(w, h): the size of smaller matrix
R_w: the width of re-size contour
R_h: the height of re-size contour
T_w: the width of template
T_h: the height of template
if (R_w > T_w) or (R_h > T_h) then
    (W, H) ←(R_w, R_h) and (w, h) ←(T_w, T_h)
    else
    (W, H) ←(T_w, T_h) and (w, h) ←(R_w, R_h)
return (W − w + 1, H − h + 1)
```

En ayant la comparaison des résultats de chaque contour et de chaque image de modèle, nous devons trouver la meilleure correspondance pour chaque contour. Par conséquent, la méthode "minMaxLoc" dans OpenCV renvoie quatre sorties qui incluent la valeur minimale, la valeur maximale, l'emplacement minimal du point (en deux dimensions) et l'emplacement maximal du point (en deux dimensions). Par conséquent, pour chaque contour, une liste de résultats de tableau est obtenue en

comparant chaque contour avec toutes les images de modèle qui sont récupérées. La meilleure correspondance est composée du résultat avec la valeur maximale. La classe java "MaximumValue" implémente cette fonction, dont l'algorithme est présenté dans la figure suivante.

Algorithm 4 : Best Matching Result(R)

Input: A list of matching result $R = R_1, ..., R_n$
Output: find the best match for each contour C

```
1 find a list of maximum value of matching results
2 find a list of the location of maximum values
3 Max_v ←R_1
4 for R_i = R_2 to R_n do
5 |   if R_i > Max_v then
6 |   |   Max_v ←R_i
7 return Max_v
```

Comme indiqué dans l'annexe A, la classe Java "FindMatching" montre les détails du processus de correspondance de modèle.

Nous devrions noter que même si la méthode d'appariement compare des contours de largeur d'un pixel (voir ci-dessous le problème avec des dessins épais), l'algorithme s'est bien comporté, pour les raisons suivantes.

- Les étapes de prétraitement que nous effectuons avant de calculer les contours nettoient l'image et suppriment le «bruit» qui peut résulter de dessins bancals, en lissant les lignes avant de détecter les contours;

- En réduisant les deux contours à la même boîte englobante, nous éliminons les erreurs dues aux différences dans les rapports de dimensions (par exemple, le rapport hauteur / largeur dans l'esquisse manuelle par rapport au modèle);

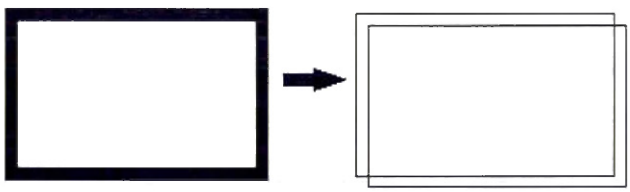
- L'algorithme d'appariement que nous avons choisi (TM_CCORR_NORMED), qui a été validé avec des résultats expérimentaux, considère les moyens par opposition aux points individuels, et enfin

- L'algorithme de correspondance renvoie la meilleure correspondance parmi les modèles disponibles

C'est ce qui nous a permis d'obtenir de bons résultats, en dépit du fait que nous associons des modèles prédéfinis à des formes dessinées à la main d'un pixel.

3.4.7 Contour Distance

Lorsque l'utilisateur dessine une esquisse à la main à l'aide d'un stylo ou d'un pinceau épais, l'étape de reconnaissance du contour peut renvoyer les contours multiples. (Voir la figure 3.18). Nous n'avons aucun moyen facile de détecter les contours multiples. Ainsi, nous nous basons simplement sur la proximité des contours pour ne pas tenir compte de certains d'entre eux.



(a) (b)

Figure 3.18 Représente les doubles du contour: (a) contour dessiné à la main; (b) formes primitives reconnues

Pour résoudre ce problème, nous devons calculer la distance à partir du point de départ de chaque contour avec le reste des contours reconnus à l'intérieur de l'esquisse dessinée à la main et définir le seuil pour comparer la distance entre eux. Si la distance du contour considéré par rapport au reste des contours détectés dans la liste est inférieure au seuil, nous rejetons le contour. Nous répéterons le même calcul pour les contours restants sur la liste. La classe Java "CalculateDistanceBetweenContour", comme indiqué dans l'annexe A, montre les détails.

3.4.8 Détection de Ligne

La méthode d'appariement de templates expliquée précédemment ne fonctionne que pour *les contours*, qui sont représentés par des figures géométriques *fermées* qui ne servent qu'à reconnaître les contours fermés. Alors, comment pouvons-nous reconnaître les lignes? Étant donné une esquisse dessinée à la main, si nous enlevons tous les contours fermés, comme indiqué dans l'algorithme ci-dessous, nous devrions être laissés avec des lignes. Cependant, il convient de souligner que si les lignes sont dessinées avec un stylo épais ou une brosse, comme indiqué sur le côté gauche de la figure 3.19, alors la fonction OpenCV pour déterminer les contours retournera un contour fermé qui correspond aux bords extérieurs des lignes épaisses. (Voir le partie à droite de la figure 3.19). À cette fin, nous avons utilisé un seuil pour le rapport entre les dimensions des contours pour filtrer les contours qui correspondent aux bords des lignes épaisses.

Figure 3.19 Représente les bords détectés de ligne

Algorithm 5 : Detect Line(L)

Input: A list of contours $C = C_1, ..., C_k$ and the input image img
Output: A list of line $L = L_1, ..., L_n$

```
w: the width of rectangle box
h: the height of rectangle box
x: the start coordinate x of rectangle box
y: the start coordinate y of rectangle box
w ← 0, h ← 0, x ← 0, y ← 0
for C = C_1 to C_k do
    define a rectangle box around each C
    initialize Th as a threshold to increase the area of rectangle box
    p_1 ←(x − Th , y − Th)
    p_2 ←(x + w + Th , y − Th)
    p_3 ←(x + w + Th , y + h + Th)
    p_4 ←(x − Th , y + h + Th)
    save the points(p_1, p_2, p_3, p_4) in a list
    paint the list of points with black color in img
return img
```

Il y a deux raisons pour spécifier un seuil:

1. Pour éviter de reconnaître les bords de chaque contour fermé comme une ligne;
2. Une ligne tracée à la main peut être dans une position différente par rapport au contour fermé, comme le montre la figure 3.20. Ainsi, une ligne appropriée pour le reste du processus est créée.

Pour détecter les lignes, nous appliquons d'abord la détection des arêtes aux lignes. Dans ce qui suit, l'utilisation de la méthode "Hough Line Transform" dans OpenCV qui recherche une image binaire pour des lignes droites est requise. Cette méthode examine si un point d'une image binaire peut faire partie d'un ensemble de lignes possibles (Bradski et Kaehler, 2008) .

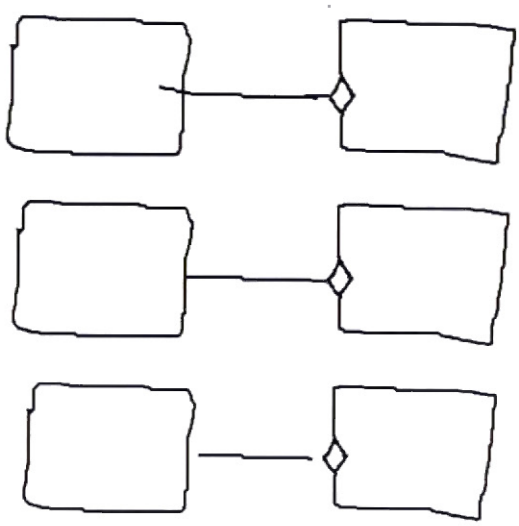


Figure 3.20 Différents styles pour dessiner la ligne

Cette méthode (voir la figure 3.21) exprime la ligne dans le système de coordonnées polaires. Par conséquent, l'équation pour une ligne peut être écrite comme (Bradski et Kaehler, 2008; OpenCV documentation, 2013):

$$P = x \cos\theta + y \sin\theta \tag{3.11}$$

En général, pour chaque point (x_0, y_0), nous pouvons définir une famille de lignes passant par ce point (Bradski et Kaehler, 2008; OpenCV documentation, 2013):

$$P_0 = x_0 \cos\theta + y_0 \sin\theta \tag{3.12}$$

En d'autres termes, chaque paire (P_0, Θ) représente chaque ligne qui passe par (x_0, y_0) (voir la figure 3.21-a) (Bradski et Kaehler, 2008; OpenCV documentation, 2013). Pour chaque point (x_0, y_0), la famille de lignes qui le traverse est définie (voir la figure 3.21-b) (Bradski et Kaehler, 2008; OpenCV documentation, 2013). Ainsi, cette opération sera effectuée pour tous les points d'une image. Si les courbes de deux points différents se croisent, cela signifie que les deux points appartiennent à la même ligne (voir la figure 3.21-c) (Bradski et Kaehler, 2008; OpenCV documentation, 2013).

Pour détecter une ligne, *un seuil* est défini comme le nombre minimal d'intersections. Ainsi, si le nombre d'intersection entre les courbes de chaque point de l'image est supérieur au seuil, alors une ligne avec les paramètres (Θ, p) du point d'intersection est identifiée. (OpenCV documentation, 2013).

OpenCV implémente deux types de Transformations Hough Line qui incluent "HoughLines" (Bradski et Kaehler, 2008) et "*HoughLinesP*" (Bradski et Kaehler, 2008). La première fonction renvoie un vecteur de couples (Θ, P_0) et la seconde fonction, plutôt que de rassembler tous les points possibles, n'en recueille qu'une fraction de ces points. Par conséquent, si le pic va être suffisamment élevé, il renvoie le point de départ et le point final des lignes détectées (x0,y0,x1,y1) (Bradski et Kaehler, 2008; OpenCV documentation, 2013).

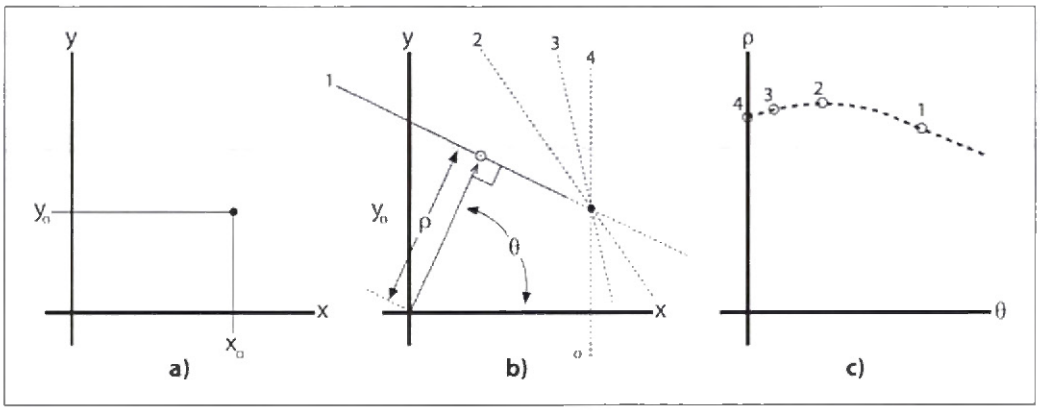


Figure 3.21 Représente le point dans l'image (a) Représente un point (x_0, y_0) dans l'image:: (b) représente les lignes paramétrées par des p and θ différents; (c) Chacune de ces lignes représente des points dans les paramètres (p, θ), considérer ensemble former une courbe de la forme caractéristique (Bradski et Kaehler, 2008).

En reconnaissant les lignes, le système est capable de découvrir les relations spatiales entre chaque ligne et les contours comme un connecteur dans la section suivante. La classe java qui détecte une ligne "DetectLine" montre les détails pour reconnaître la ligne (vois annexe B).

3.4.9 Distance de Ligne

Parce que les esquires dessinés à la main peuvent avoir des lignes épaisses, plusieurs lignes peuvent être reconnues. Pour résoudre ce problème, nous utilisons la même solution que celle utilisée pour les formes primitives: nous fusionnons des lignes dont la distance mutuelle est inférieure à un certain seuil. En trouvant les lignes les plus proches, nous calculons la coordonnée minimale du point de départ et calculons la coordonnée maximale du point final pour toutes les lignes dont la distance est inférieure au seuil. Ainsi, le système évite de reconnaître une ligne plusieurs fois. En outre, système doit reconnaitre la ligne la plus longue. L'algorithme suivant affiche une partie de la fonction.

Algorithm 6 : Merge lines(L)

Input: A list of lines $L = L_1, ..., L_n$
Output: find the coordinate of longest line by merging the nearest lines

```
initialize Th as a threshold to find the nearest lines
distance ←0
for L = L_1 to L_n do
    initialize distance by calculating the distance of lines
    if distance < Th then
        (x_min, y_min) ←calculate the minimum start point of lines
        (x_max, y_max) ←calculate the maximum end point of lines
return ((x_min, y_min), (x_max, y_max))
```

La classe Java "CalculateDistanceBetweenLines", comme indiqué dans l'annexe B, montre les détails.

3.5 Reconnaissance d'Objets Graphiques Composites

Les formes composites sont composées de formes primitives. Ainsi, nous devons combiner les formes primitives qui ont été reconnues à l'étape précédente, en utilisant leurs relations spatiales. Il y a deux formes composites principales, sentinelle et connecteur, sur lesquelles nous allons nous concentrer dans cette section. Par conséquent, le premier problème est:

Comment définir le diamant dans le carré (défini comme un élément de tâche dans le CMMN) et comment reconnaître l'image de sentinelle comme un élément dans le CMMN?

3.5.1 Zone d'Intersection

Afin de reconnaître les sentinelles, nous devons découvrir si le diamant est dans la tâche ou non. Par conséquent, nous définissons une fonction qui reçoit les coordonnées de toutes les tâches et des diamants, puis commence à calculer la distance de chaque sommet de diamant avec la zone de tâche. Lorsque l'une des coordonnées des sommets du diamant fait partie de la tâche, le diamant, appelé sentinelle dans le CMMN, est considéré comme faisant partie de la forme composite. Le code suivant affiche une partie de la fonction.

Algorithm 7 : Intersection Area

Input: A list of sentries $S = S_1, ..., S_n$, A list of tasks $T = T_1, ..., T_n$
Output: find intersection area

```
initialize the coordinates of sentry x_s ← 0, y_s ← 0, w_s ← 0, h_s ← 0
initialize the coordinates of task x_t ← 0, y_t ← 0, w_t ← 0, h_t ← 0
while S ≠ ϕ and T ≠ ϕ do
    if (x_s > x_t) and (x_s < x_t + w_t) and (y_s > y_t) and (y_s < y_t + h_t) then
        return true
    else
    if (x_s + (w_s/2) > x_t) and (x_s + (w_s/2) < x_t + x_w) and
     (y_s − (h_s/2) > y_t) and (y_s − (h_s/2) > y_t + h_t) then
        return true
    else
    if (x_s + w_s) > x_t) and (x_s + w_s < x_t + w_t) and (y_s > y_t) and
     (y_s < y_t + h_t) then
        return true
    else
    if (x_s + (w_s/2) > x_t) and (x_s + (w_s/2) < x_t + w_t) and
     (y_s + (h_s/2) > y_t) and (y_s + (h_s/2) < y_t + h_t) then
        return true
return false
```

Pour plus de détails voir, la classe Java "CalculateIntersectionArea" qui affiche le calcul en détails dans l'annexe B.

3.5.2 Détection de Connecteur

Le deuxième problème de la reconnaissance d'objets graphiques composites est:

Comment reconnaître la connexion entre les lignes et les autres contours?

Pour résoudre ce problème, nous devons trouver les contours qui sont connectés aux lignes, en découvrant des relations spatiales, si nécessaire. Nous aurions besoin de calculer la distance entre chaque forme et le point de départ et le point final de chaque ligne.

En ce qui concerne le cadre englobant autour de chaque contour, en supposant que le point de départ de la ligne ou du point final de la ligne peut être relié au bord vertical d'un contour ou d'un bord horizontal du contour, Nous pouvons commencer à calculer la distance de chaque point de départ et de chaque extrémité avec l'un des deux bords sélectionnés..

Pour calculer la distance, le système doit d'abord détecter si la ligne à l'esquisse dessinée à la main est verticale ou horizontale. Parce que le modélisateur CMMN ne reconnaît que les lignes verticales ou horizontales, notre algorithme est basé sur la reconnaissance de la ligne droite. En réalisant cette partie, nous sommes en mesure de définir si la coordonnée d'un des deux points de la ligne est dans la plage du bord horizontal ou du bord vertical.

Enfin, la distance entre le point de début et de fin de la ligne et les bords est calculée et la distance minimale sera sélectionnée.

L'algorithme suivant montre comment une connexion est établie. Le seuil est spécifié pour calculer la distance de chaque point de la ligne et chaque forme primitive pour comparer la distance minimale. Si elle est inférieure au seuil, la forme est connectée à la ligne.

Algorithm 8 : Minimum Distance Between Line And Contours

Input: A list of contours $C = C_1, ..., C_n$. Line L

Output: find the minimum distance between contours and line

initialize the start point or end point of line (x_L, y_L)
initialize the start point and end point of edges of contour $(x_{sc}, y_{sc}).(x_{ec}, y_{ec})$
initialize Th as a threshold to find the minimum distance between edge and start point or end point of line
initialize (p_{sx}, p_{sy}) to hold the coordinates of line start
initialize (p_{ex}, p_{ey}) to hold the coordinates of line end
initialize *distance* to hold the distance of contour and line

if $y_{sc} = y_{ec}$ then
    //It is a horizontal line. Make sure that line start has smaller x
    if $x_{sc} > x_{ec}$ then
        $(p_{sx}, p_{sy}) \leftarrow (x_{ec}, y_{ec})$
        $(p_{ex}, p_{ey}) \leftarrow (x_{sc}, y_{sc})$
        else
        $(p_{sx}, p_{sy}) \leftarrow (x_{sc}, y_{sc})$
        $(p_{ex}, p_{ey}) \leftarrow (x_{ec}, y_{ec})$
    //If the point is not between the line start and line end then return infinite value
    if $x_L < (p_{sx} - Th)$ *or* $x_L > (p_{ex} + Th)$ then
        distance $\leftarrow \infty$
        else
        //If the point is between line start and line end then calculate the distance
        distance $\leftarrow$ distance of edge and one of the start point or end _point of line
    return *distance*

```
22 else
23 if (y_sc > y_ec) then
24 |   //It is a vertical line. Make sure that line start has smaller y
25 |   (p_sx, p_sy) ← (x_ec, y_ec)
26 |   (p_ex, p_ey) ← (x_sc, y_sc)
27 |   else
28 |   (p_sx, p_sy) ← (x_sc, y_sc)
29 |   (p_ex, p_ey) ← (x_ec, y_ec)
30 //If the point is not between the line start and line end then return
   infinite value
31 if y_L < (p_sy − Th) or y_L > (p_ey + Th) then
32 |   distance ←∝
33 |   else
34 |   //If the point is between the line start and line end then calculate the
   |   distance
35 |   distance ← distance of edge and one of the start point or end point of
   |   line
36 return distance
```

La classe Java "Connector" dans l'annexe B montre les détails de la détection du connecteur.

3.6 Reconnaissance et Compréhension de la Connexion Sémantique

En reconnaissant les formes primitives ainsi que les relations spatiales entre elle, et le dernier problème à résoudre est:

Comment afficher ces éléments CMMN détectés et les connexions sémantiques entre eux dans le modeleur CMMN?

Pour ce faire, nous créons un fichier XMI qui stocke les détails de toutes les spécifications des éléments CMMN détectés ainsi que leurs relations spatiales. Ce fichier XML doit être dans un format qui peut être importé dans modeleur CMMN. Ainsi, nous devons comprendre le schéma XMI pour CMMN pour générer les balises

appropriées. Dans cette section, nous nous référerons régulièrement aux aspects de la sémantique d'exécution du CMMN.

La structure de fichier XML dans CMMN, comme indiqué dans la figure 3.22, se compose de deux parties principales qui incluent *cas* (OMG) et *CMMN DI* (OMG). *Cas* est un concept de haut niveau qui combine tous les éléments qui constituent un modèle de cas, et qui définit les relations sémantiques entre les éléments (OMG). Chaque élément de modèle possède des attributs tels que les coordonnées, la largeur, la hauteur, l'étiquette, etc. La section CMMN DI est utilisée pour spécifier les attributs visuels des éléments qui incluent une collection de formes et d'arêtes.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cmmn:definitions author="" exporter="CMMN Modeler" id="_65d434e4-7906-4034-b2aa-45959d0a8996" name="Drawing 1" targetNamespace="http://www.
trisotech.com/cmmn/definitions/_65d434e4-7906-4034-b2aa-45959d0a8996" xmlns="http://www.trisotech.com/cmmn/definitions/_65d434e4-7906-4034-
b2aa-45959d0a8996" xmlns:trisob="http://www.trisotech.com/2014/triso/cpmn" xmlns:cmmndi="http://www.omg.org/spec/CMMN/20151109/CMMNDI" xmlns:rss="
http://purl.org/rss/2.0/" xmlns:cmmn="http://www.omg.org/spec/CMMN/20151109/MODEL" xmlns:di="http://www.omg.org/spec/CMMN/20151109/DI" xmlns:
trisofeed="http://trisotech.com/feed" xmlns:trisocmmn="http://www.trisotech.com/2014/triso/cmmn" xmlns:triso="http://www.trisotech.com/2015/triso/
modeling" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:dc="http://www.omg.org/spec/CMMN/20151109/DC">
    <cmmn:caseFileItemDefinition id="DEF_547c"/>
    <cmmn:case name="Page 1" id="CaseCPM_00b">
        <cmmn:caseFileModel>
            <cmmn:caseFileItem definitionRef="DEF_547c" multiplicity="Unspecified" id="_547c"/>
        </cmmn:caseFileModel>
        <cmmn:casePlanModel autoComplete="false" name="Page 1" id="CPM_00b5">
            <cmmn:planItem definitionRef="PID_8ab9" id="_8ab3"/>
            <cmmn:task isBlocking="true" id="PID_8ab9"/>
        </cmmn:casePlanModel>
    </cmmn:case>
    <cmmndi:CMMNDI>
        <cmmndi:CMMNDiagram name="Page 1" id="_00b54" sharedStyle="_75ca">
            <cmmndi:Size height="1050.0" width="1345.5"/>
            <cmmndi:CMMNShape cmmnElementRef="_1036" id="_e8e5">
                <dc:Bounds height="78.0" width="97.0" x="191.5" y="52.0"/>
                <cmmndi:CMMNLabel/>
            </cmmndi:CMMNShape>
            <cmmndi:CMMNShape cmmnElementRef="_0503" id="_5ebd">
                <dc:Bounds height="28.0" width="20.0" x="279.0" y="76.0"/>
                <cmmndi:CMMNLabel/>
            </cmmndi:CMMNShape>
        </cmmndi:CMMNDiagram>
        <cmmndi:CMMNStyle fontFamily="Arial,Helvetica,sans-serif" id="_75ca"/>
    </cmmndi:CMMNDI>
</cmmn:definitions>
```

Figure 3.22 Structure de fichier XML dans CMMN

3.6.1 Le cas

Pour comprendre la structure d'un modèle de cas, nous fournissons un méta-modèle CMMN partiel dans la figure 4.23. Dans la suite, afin d'expliquer les relations

sémantiques entre les éléments CMMN, nous allons nous concentrer sur les associations "*caseFileModel*" et "*casePlanModel*" et les expliquer en détail.

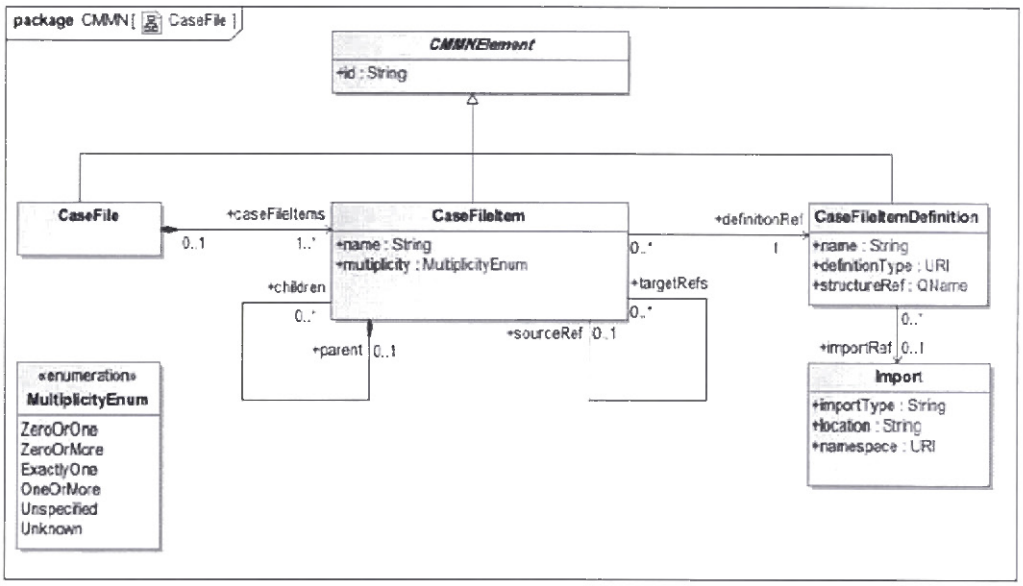


Figure 3.23 Métamodèle du fichier de cas CMMN (OMG)

3.6.1.1 Fichier de Cas

Chaque cas est associé à exactement "CaseFile" (voir la figure 3.24). Les informations de cas sont représentées par "CaseFile". En outre, chaque "CaseFile" doit contenir au moins un "CaseFileItem". Dans ce qui suit, chaque "CaseFileItem" doit être associé à exactement un "CaseFileItemDefinition". Ainsi, afin de spécifier l'association entre chaque "FileItemDefinition" et chaque "CaseFileItem", le *definitionRef* pour chaque "FileItemDefinition" doit être l'*id* de chaque "CaseFileItem". Le fichier XML ci-dessous affiche une partie du fichier XML dans CMMN, qui décrit "CaseFile".

```
<cmmn:caseFileItemDefinition id="DEF__93"/>
<cmmn:case name="Page 1" id="CaseCPM_00b5">
    <cmmn:caseFileModel>
        <cmmn:caseFileItem definitionRef="DEF__93" multiplicity="Unspecified" id="_93"/>
    </cmmn:caseFileModel>
```

Figure 3.24 Structure du fichier de cas au format XML

Les méthodes java "writeFileItemDefinition" et "writeFileItem" dans l'annexe C montrent comment sérialiser un fichier de cas sur un fichier XML.

3.6.1.2 Plan de Cas

Le plan de cas est construit à partir des blocs composés d'éléments PlanItemDefinition (figure 3.25). Chaque "PlanItemDefinition" peut représenter un élément CMMN, qui peut inclure "PlanFragment (et Stage), Task, EventListener ou Milestone". Ainsi, afin de spécifier l'association entre chaque "PlanItemDefinition" et chaque certain élément CMMN, le *DefinitionRef* pour chaque "PlanItemDefinition" doit être l'*id* de chaque élément CMMN.

Le fichier XML ci-dessous affiche une partie du fichier XML dans CMMN, qui décrit un "CasePlanModel" (voir la figure 3.25).

```
<cmmn:casePlanModel autoComplete="false" name="Page 1" id="CPM_00b5">
    <cmmn:planItem definitionRef="PID__1036" id="_1036"/>
    <cmmn:task isBlocking="true" id="PID__1036"/>
</cmmn:casePlanModel>
```

Figure 3.25 Structure du plan de cas au format XML

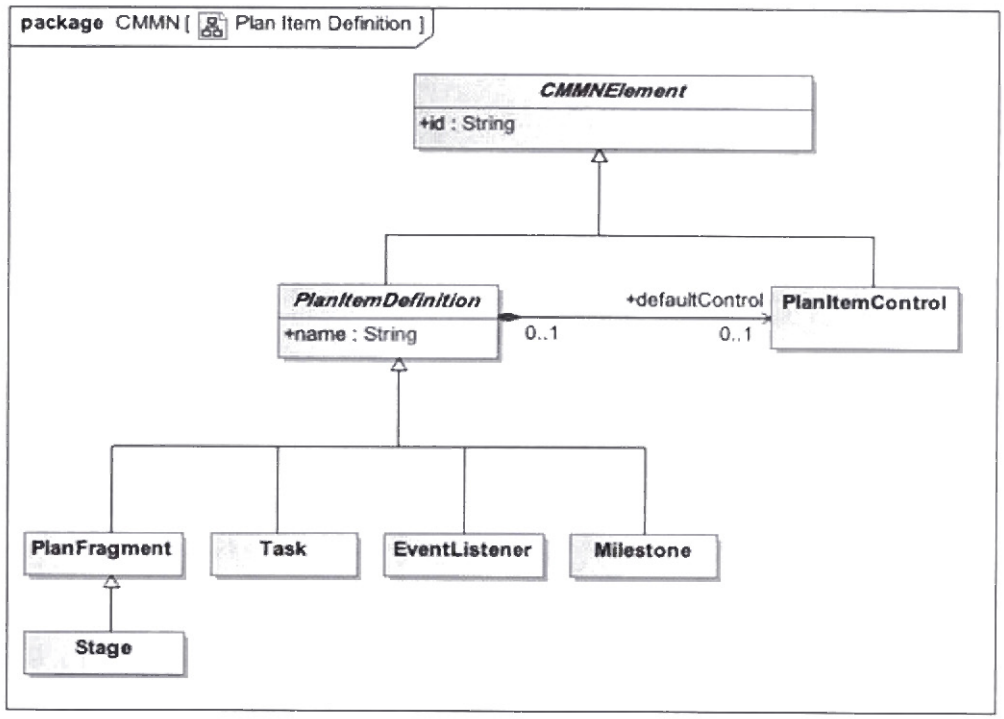


Figure 3.26 Définition de l'élément de plan (OMG)

Les Sentinelles qui sont utilisées comme critère d'entrée, sont un élément CMMN qui n'est pas fourni en tant qu'élément CMMN indépendant, mais qui doit être présent dans le cadre d'autres éléments du CMMN, tels que l'étape ou la tâche (figure 3.27). Ainsi, les relations entre un "PlanItemDefinition" et une Sentinelle sont représentées avec l'association "entryCriterion". Par conséquent, afin de spécifier la connexion entre les éléments CMMN mentionnés et chaque Sentinelle, *sentryRef* pour chaque "entryCriterion" qui doit être l'*id* de chaque Sentinelle. Le fichier XML ci-dessous affiche une partie du fichier XML dans CMMN, qui montre une sentinelle dans le cadre d'une tâche.

```
<cmmn:planItem definitionRef="PID__1036" id="_1036">
    <cmmn:entryCriterion sentryRef="_76fd" id="_0503"/>
</cmmn:planItem>
<cmmn:sentry id="_76fd">
    <cmmn:ifPart id="_1392"/>
</cmmn:sentry>
<cmmn:task isBlocking="true" id="PID__1036"/>
```

Figure 3.27 Structure de sentinelle au format XML

Comme un connecteur dans CMMN est utilisé pour visualiser les dépendances entre les éléments CMMN (voir la figure 3.28), il est nécessaire d'afficher les éléments CMMN auxquels un connecteur appartient. La direction ou l'association du flux de séquence est définie par *critère d'entrée* ou *critère de sortie* (OMG). Ainsi, un côté d'un connecteur sera associé à une sentinelle et présent sous forme de *planItemOnPart*, alors que l'autre côté appartient à l'autre "planItemDefinition" qui est connecté à une sentinelle. Par conséquent, afin de spécifier cette association, nous définissons le *sourceRef* pour chaque "planItemOnPart" qui doit être l'*id* de "planItemDefinition" spécifique qui est connecté à "planItemOnPart". Le fichier XML ci-dessous montre une partie du fichier XML dans CMMN, qui décrit la connexion entre deux tâches.

```
<cmmn:casePlanModel autoComplete="false" name="Page 1" id="CPM_00b5">
    <cmmn:planItem definitionRef="PID__1036" id="_1036">
        <cmmn:entryCriterion sentryRef="_76fd" id="_0503"/>
    </cmmn:planItem>
    <cmmn:planItem definitionRef="PID__8ab9" id="_8ab9"/>
    <cmmn:sentry id="_76fd">
        <cmmn:planItemOnPart sourceRef="_8ab9" id="_8704">
            <cmmn:standardEvent>complete</cmmn:standardEvent>
        </cmmn:planItemOnPart>
        <cmmn:ifPart id="_1392"/>
    </cmmn:sentry>
    <cmmn:task isBlocking="true" id="PID__1036"/>
    <cmmn:task isBlocking="true" id="PID__8ab9"/>
</cmmn:casePlanModel>
```

Figure 3.28 Structure du connecteur au format XML

Les méthodes java "writeplanItem", "writeEvent", "writeTask" et "writeSentry" de l'annexe C montrent comment sérialiser un PlanItemDefinition.

3.6.2 CMMN DI

CMMN DI (OMG) est utilisé pour spécifier les propriétés visuelles des éléments qui incluent une collection de formes et d'arêtes. La figure 3.29 montre le meta-modèle partiel pour le composant CMMNDI. Il montre que CMMNDI est un conteneur pour "CMMNStyle" partagé et tous les "CMMNDiagrams" définis dans Définitions (OMG).

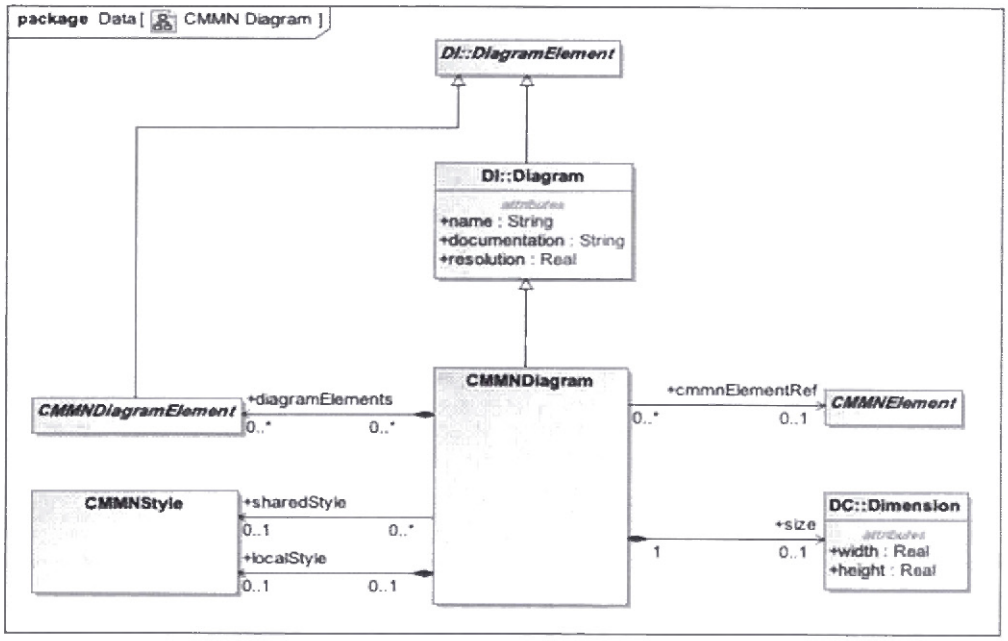


Figure 3.29 Diagramme de classe CMMNDI (OMG)

La classe "CMMNDiagram" (OMG) est une sorte de diagramme qui représente une représentation de la totalité ou d'une partie d'un modèle CMMN (OMG). En d'autres termes, c'est le conteneur de "CMMNDiagramElement" qui est composé de "CMMNShape" et "CMMNEdge" (voir la figure 3.30). Le fichier XML ci-dessous affiche une partie du fichier XML dans CMMN qui décrit un "CMMNDiagram".

```
<cmmndi:CMMNDI>
    <cmmndi:CMMNDiagram name="Page 1" id="_00b54" sharedStyle="_75ca">
        <cmmndi:Size height="1050.0" width="1545.5"/>
        <cmmndi:CMMNShape cmmnElementRef="_0503" id="_bebd">
            <dc:Bounds height="28.0" width="20.0" x="279.0" y="76.0"/>
            <cmmndi:CMMNLabel/>
        </cmmndi:CMMNShape>
        <cmmndi:CMMNEdge cmmnElementRef="_8704" isStandardEventVisible="true" targetCMMNElementRef="_0503" id="
        _dfba5921-5e71-4e86-b3c5-1768277dfee8">
            <di:waypoint x="297.9993999b125814" y="89.99972164102996"/>
            <di:waypoint x="411.5391946702358" y="89.9681169865079"/>
            <cmmndi:CMMNLabel/>
        </cmmndi:CMMNEdge>
    </cmmndi:CMMNDiagram>
    <cmmndi:CMMNStyle fontFamily="Arial,Helvetica,sans-serif" id="_75ca"/>
</cmmndi:CMMNDI>
```

Figure 3.30 Structure CMMNDI au format XML

Comme le montre l'Annexe C, les méthodes "writeFileValues", "writeLineValus" "writeEntryCriterionValus", "writeEventValues" et "writetaskValues" sont utilisées pour sérialiser les entités CMMNDiagramElement, CMMNShape et CMMNEdge.

3.6.2.1 CMMNShape

La forme "CMMNShape" est une sorte de forme qui représente un "CMMNElement" du modèle CMMN (figure 3.31). Par conséquent, afin de l'associer à un élément CMMN, il existe un attribut *cmmnElementRef* qui contient l'*id* de chaque "planItemDefinition". Le fichier XML ci-dessous affiche une partie du fichier XML dans CMMN, qui décrit "CMMNshape".

```
<cmmndi:CMMNShape cmmnElementRef="_0503" id="_bebd">
    <dc:Bounds height="28.0" width="20.0" x="279.0" y="76.0"/>
    <cmmndi:CMMNLabel/>
</cmmndi:CMMNShape>
```

Figure 3.31 Structure CMMNShape au format XML

3.6.2.2 CMMNEdge

La classe "CMMNEdge" représente les relations entre deux éléments de modèle CMMN. Par conséquent, "CMMNEdge" sont utilisés représenter des liens dans le

modèle CMMN (OMG). Afin d'utiliser "CMMNEdge" pour afficher un "PlanItemOnPart", nous définissons *cmmnElementRef* pour chaque "CMMNEdge" qui doit être l'*id* de "PlanItemOnPart". En outre, nous devons définir le *targetCMMNElementRef* pour chaque "CMMNEdge" qui doit être l'*id* de l'un des critères (soit un EntryCriterion ou un ExitCriterion) qui est liée à la Sentinelle tenant le "PlanItemOnPart". Le fichier XML ci-dessous affiche une partie du fichier XML dans CMMN, qui décrit CMMNEdge (figure 3.32).

```
<cmmndi:CMMNEdge cmmnElementRef="_8704" isStandardEventVisible="true" targetCMMNElementRef="_0503"
    id="_dfba5921-5e71-4e86-b3c5-1768277dfee8">
    <di:waypoint x="297.99999996125814" y="89.99972164102996"/>
    <di:waypoint x="411.5391946702358" y="89.9681169865079"/>
    <cmmndi:CMMNLabel/>
</cmmndi:CMMNEdge>
```

Figure 3.32 Structure CMMNEdge au format XML

3.7 Conclusion

Dans ce chapitre, nous avons décrit la mise en œuvre de notre approche qui reconnaît à la fois les éléments CMMN primitives et les formes composites. Nous avons expliqué un ensemble de problèmes à chaque étape de la reconnaissance et trouvé les solutions heuristiques en utilisant la bibliothèque OpenCV utilisée. L'utilisation du fichier XML selon les structures des modèles CMMN permet d'interpréter chaque élément CMMN et leurs relations sémantiques. Par conséquent, les esquisses dessinés à la main seront reconnues et formalisés en important le fichier XML dans les outils de modélisation CMMN. Dans le chapitre suivant, nous évaluerons notre programme en étudiant divers échantillons collectés afin d'évaluer la performance de l'outil.

CHAPITRE IV

LA MISE EN OEUVRE DU TEST ET LES RÉSULTATS

Pour évaluer la précision du système de reconnaissance, nous devons le tester sur différents échantillons. Nous avons recueilli 20 dessins réalisés par des sujets expérimentaux. Ces sujets avaient entre 24 et 33 ans et sont des utilisateurs réguliers d'ordinateurs.

Les sujets ont reçu les informations et instructions suivantes:

1. Ils ont été introduits à la syntaxe CMMN et ses différents éléments;
2. Ils ont été présentés au modélisateur du CMMN et ont été informés de ce que notre système peut faire (reconnaître les dessins à la main et les mettre dans un format qui convient à un outil de modélisation formel);
3. Ils doivent s'assurer que leurs dessins avaient des formes fermées;
4. Ils pouvaient utiliser n'importe quel logiciel de peinture et dispositif de pointage;
5. Ils sont libres de choisir l'épaisseur de la brosse à utiliser. Cependant, ils ne pouvaient pas utiliser l'aérographe, qui laisse des espaces entre les points de peinture;
6. Ils doivent dessiner avec une manière naturelle, sans essayer d'être particulièrement précis.

4.1 Présentation du Paramètre de Test

Après avoir expliqué les exigences, les sujets ont été invités à dessiner 5 de chaque primitives du modèle CMMN (lignes, événements, tâches, fichiers, critères d'entrée (diamant) ainsi que des formes composites, comme le montre la figure 4.1, pour préparer l'expérience. Par exemple, une tâche et un critère d'entrée connectés à une ligne ce qu'on appelle fragments de modèle.

Figure 4.1 Dépendance basée sur sentinelle entre deux tâches

Nous avons commencé par mesurer la performance de la reconnaissance pour les formes primitives. Rappelez-vous que la façon de travailler fonctionne en comparant chaque forme d'entrée du CMMN à la catégorie des formes prédéfinies, et en mesurant la similarité entre la forme d'entrée et les formes CMMN prédéfinies. La forme prédéfinie qui atteint la valeur de similarité la plus élevée est supposée être la forme voulue. Le tableau suivant montre une matrice carrée où chaque ligne correspond à une forme d'entrée, chaque colonne représente une forme prédéfinie et cell *(x, y)* représente le pourcentage de fois que la forme prédéfinie *y* a été identifée la meilleure correspondance pour la forme d'entrée *x*. Ainsi, la forme *(x, x)* représente le pourcentage de reconnaissances précises.

En ce qui concerne les formes composites, la performance de reconnaissance dépend d'une combinaison de:

1. La reconnaissance des formes primitives

2. L'exactitude des calculs des relations spatiales entre les formes primitives et la force des inférences tirées d'une telle relation

Tableau 4.1 Résultats de la reconnaissance des formes primitives: La forme dessinée (attendue) est représentée à gauche et la forme reconnue en haut

	Event	Task	File	Entry Criterion	Line
Event	**98%**	2%			
Task	2%	**92%**	6%		
File	2%	68%	**30%**		
Entry Criterion	8%			**92%**	
Line					**100%**

Tableau 4.2 Résultats de la reconnaissance des fragments de modèles: La forme dessinée (attendue) est montrée sur la gauche et la forme identifiée comme faisant partie des fragments du modèle en haut

	Task	Line	Entry Criterion	Event	File
Task A	92%			2%	6%
Line		100%			
Entry Criterion			80%	20%	
Task B	86%				14%

4.2 Précision de Reconnaissance

Le taux de reconnaissance pour un ensemble de 500 dessins, composé à la fois de formes primitives et de formes composites, est présenté dans les tableaux 4.1 et 4.2. le tableau 4.1 montre des différences significatives entre les taux de reconnaissance pour les différentes formes primitives, allant de 30% pour un élément de la tâche

(File) à 98% pour un élément d'événement (Even). La différence est due, en grande partie, au caractère spécifique à chaque modèle de formes. Par exemple, il existe une grande similitude entre "files" et " Task", qui sont tous deux rectangulaires, mais avec un fichier ayant un angle écrêté (figure 4.3c, figure 4.3d). Par conséquent, le taux de reconnaissance du fichier « file » était le plus faible de toutes les formes (30%). En effet, de nombreux fichiers (68%) ont été reconnus comme des tâches. Le contraire n'est pas vrai. Cela peut être dû à la prévalence des fichiers dans les 500 dessins à la main. Cependant, d'une manière générale, la performance moyenne est acceptable.

Notez également que les lignes (Lines) sont les formes les plus faciles à reconnaître, avec un taux de 100%. En effet, la séquence de points pouvant former une ligne est spécifiée par trois paramètres incluant *threshold*, *minLinLength* et *maxLineGap*. Ces paramètres ont été ajustés afin d'accepter la ligne la plus courte avec le nombre minimum d'intersections pour constituer une ligne.

Les événements (Events) ont également eu un taux de reconnaissance élevé. Les cas où l'esquisse n'a pas été reconnue (figure 4.3a) étaient dans le cas d'un événement confendu en un carré.

Le taux de reconnaissance des critères d'entrée (le diamant) était également élevé. Les cas qui n'étaient pas reconnus (figure 4.3b) étaient souvent dus au fait que le critère d'entrée ressemblait à un événement tourné de 45 degrés.

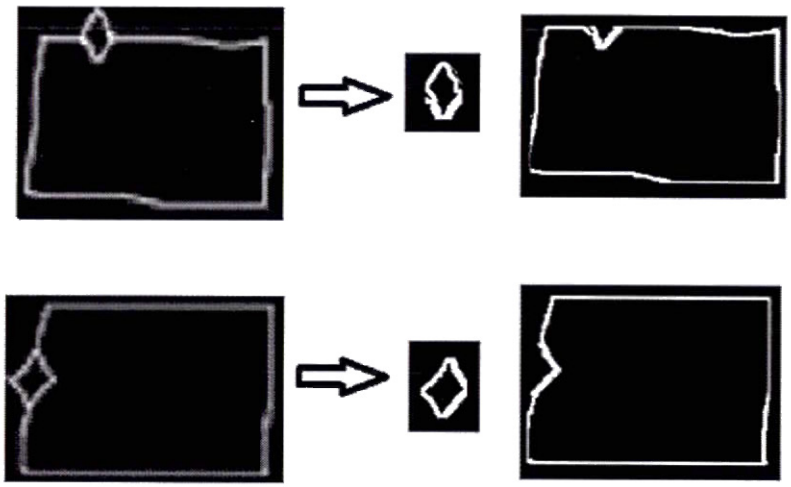
La dépendance entre sentinelles entre deux tâches, comme le montre la figure 4.1, représente des fragments de modèle constitués de formes primitives et d'une forme composite, elle-même constituée d'un critère d'entrée (le diamant) faisant partie d'une tâche.

Le taux de reconnaissance des critères d'entrée (le diamant) en tant que formes primitives (voir le tableau 4.1) est de 92%, tandis que son taux de reconnaissance

dans le deuxième tableau (voir le tableau 4.2) dans une forme composite est sensiblement réduit (80%).

Selon l'algorithme 7 du chapitre III, le critère d'entrée de reconnaissance (le diamant) en tant que partie de la forme composite dépend entièrement du taux de reconnaissance de la tâche B. Par conséquent, nous devons reconnaître une tâche en tant que tâche B, puis passer au deuxième niveau qui est reconnu comme critère d'entrée (le diamant) dans le cadre de la tâche B.

Comme le montre le tableau 4.2, le taux de reconnaissance de la tâche B en tant que tâche est de 86% et celui d'un fichier de 14%. Étant donné que la tâche B, dans la figure 3.6, a été reconnue comme un contour interne (voir la figure 4.2), la similarité du dessin de la tâche B avec la template de tâche a été réduite. Par conséquent, le taux de reconnaissance de la tâche B affecte le taux de reconnaissance du critère d'entrée (le diamant).



(a) Contour extérieur (parent) (b) Contours intérieurs (enfants)

Figure 4.2 Reconnaître les tâches B et les diamants comme contours intérieurs basé sur la figure 3.6

Il existe d'autres raisons pour le faible taux de reconnaissance des critères d'entrée (le diamant). Tout d'abord, il y a la façon dont les humains dessinent des esquisse à la main. Parce que dessiner le diamant en tant que partie de la forme composite est plus difficile que de dessiner comme une forme primitive, la similitude d'un losange (événement) avec le modèle d'événement, comme indiqué dans le tableau 4.2, est augmenté et de nombreux diamants finiront par être reconnus comme des événements.

La deuxième raison pour laquelle les critères d'entrée n'ont pas un bon taux de reconnaissance est liée à la boîte entourant chaque diamant. Comme le montre la figure 4.2, les sommets de diamant sont lissés par des bords de boîte de délimitation. Par conséquent, la similitude du diamant à l'événement sera augmentée.

Alors que le taux de reconnaissance des formes primitives individuelles est élevé (en moyenne), le taux de reconnaissance des agrégats se situe dans la plage [72% - 80%]. C'est à prévoir, compte tenu de la façon dont notre algorithme fonctionne.

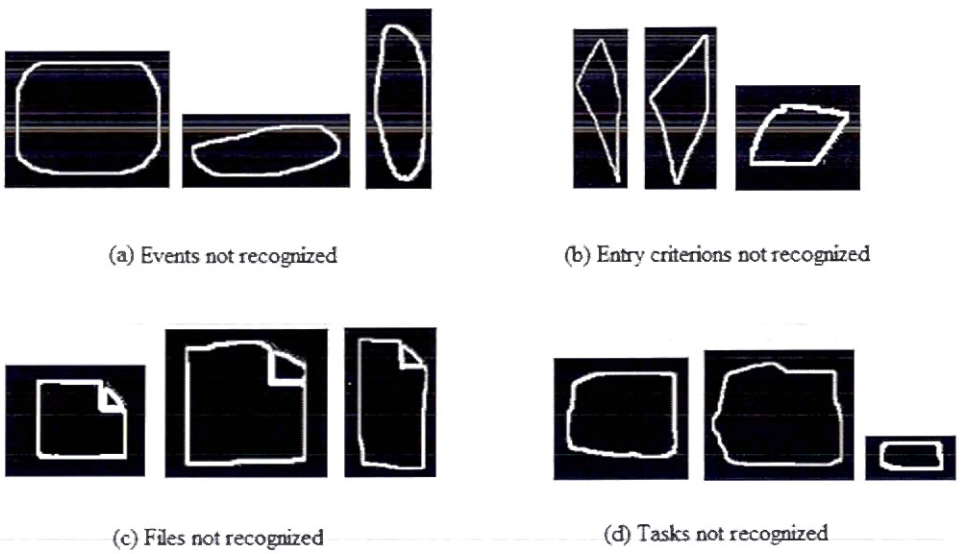


Figure 4.3 Les éléments d'échantillons non reconnus par le système

4.3 Limitations

La solution actuelle a un certain nombre de limitations. D'abord, il est sensible à la rotation. Il existe des moyens de changer les algorithmes pour les rendre résistants à la rotation. Cependant, cela compliquerait la reconnaissance des sentinelles (losanges) qui se base sur leur inclinaison de 45 degrés. Nous devrons alors prendre en compte la taille relative et la position des formes que nous devons identifier, par exemple entre les tâches et les critères d'entrée / sortie, qui nous conduiront à la prochaine amélioration possible.

La deuxième limitation implique la reconnaissance du texte au chapitre II, nous avons parlé de la reconnaissance optique des caractères et de la reconnaissance des caractères hors ligne comme l'une de ses sous-catégories, qui comprend des étapes générales telles que le prétraitement, la segmentation, l'extraction de caractéristiques et la classification.

Reconnaître le texte dans le contexte des dessins de figures géométriques, comme c'est le cas avec le CMMN (ou d'autres types de modèles), est plus compliqué que de reconnaître du texte dans des documents purement textuels. Nous envisageons un processus en plusieurs étapes. La première étape serait identique à celle de la reconnaissance de formes, et consisterait en un prétraitement et une détection d'objets. Une deuxième étape consisterait à séparer les contours qui contiennent du texte des autres formes. Une troisième étape consisterait à extraire le texte des formes. Une quatrième étape consisterait à classifier le texte en fonction de la catégorie la plus simple (les chiffres [0-9] et les lettres de l'alphabet anglais [a-z]). La cinquième étape serait de reconstruire le texte en rassemblant toutes les lettres et les nombres pour construire des chaînes dans les bonnes positions qui peuvent être à l'intérieur ou à l'extérieur des formes. La dernière étape consisterait à attacher ces chaînes en tant qu'étiquettes aux formes dans lesquelles elles apparaissaient au début.

Nous aurions besoin de mettre en œuvre et d'expérimenter un tel système pour voir comment cela fonctionne.

Enfin, notez que nous avons donné aux sujets expérimentaux quelques directives sur la façon de faire leurs esquisses à la main. Par exemple, nous leur avons demandé de s'assurer qu'ils "créent des forment fermées", pour faciliter le calcul des contours et la reconnaissance des formes, bien que, nous ayons utilisé différents seuils pour nous assurer que nos algorithmes peuvent compléter ou «fermer» des formes imparfaitement fermées. Aussi, les sujets expérimentaux ont été informés de ne pas utiliser "le pinceau" ou des stylos épais en dessinant. Ce ne sont pas des limitations sérieuses, mais nous ne pouvons pas dire que nous avons effectué les expériences sur des dessins totalement naturels.

CONCLUSION

Un travailleur de la connaissance qui souhaite détenir une collection de documents d'affaires et d'autres informations pertinentes à leurs processus d'affaires; a besoin d'utiliser des systèmes de gestion de cas comme un premier élément constitutif. Dans le contexte de la gestion de cas, le Modèle et la notation de gestion des cas (CMMN) soutient la représentation d'un large éventail d'activités de gestion du travail social et des applications connexes telles que les processus de réclamation en assurance, les processus de soins, les services sociaux, etc. (Marin *et al.*, 2012).

L'utilisation du CMMN comme outil de modélisation formel lors des premières activités d'identification des exigences pendant le cycle de vie du développement logiciel n'est pas efficace, de même pour d'autres outils flexibles tels que les outils de bureau ou les tableaux blancs. Ces outils ont certaines restrictions comme le manque de gestion des cohérences, la complexité de la publication des modifications et la difficulté de la migration des informations. Par conséquent, une nouvelle approche intermédiaire est nécessaire pour réduire l'écart entre ces deux approches.

Le but de notre recherche est de développer une nouvelle approche intermédiaire afin de réduire l'écart entre ces deux approches. Notre approche permet de lire des esquisses de modèles CMMN faites à la main et de les transformer en un format pouvant être importé dans un outil de modélisation formel de CMMN. Dans ce mémoire, nous avons présenté en détail un ensemble d'algorithmes pour extraire et reconnaître les contours des modèles CMMN dessinés à la main des formes primitives et des formes composites. Différents algorithmes de prétraitement ont été appliqués aux esquisse d'entrée pour les préparer à la reconnaissance. Les étapes à suivre sont les suivantes :

- Récupérer les contours d'une esquisse dessinée à la main;
- Clarifier les contours en utilisant des algorithmes de modélisation de patrons;
- Identifier une relation spatiale entre des formes primitives;
- Et utiliser ces relations pour reconnaître les formes composites.

Pour implémenter notre système, nous avons utilisé la bibliothèque OpenCV, qui fournit un ensemble riche de fonctions de traitement d'images. Étant donnée notre système est basé sur la reconnaissance des esquisses à main CMMN, il pourrait facilement être paramétré pour reconnaître les esquisses à la main sous n'importe outils de dessin graphique, à condition que les icônes graphiques utilisées pour les divers éléments puissent être raisonnablement distinguées. Actuellement, l'algorithme reconnaît d'abord les formes primitives, puis les formes composites. Un meilleure version de l'algorithme pourrait utiliser un algorithme de reconnaissance bidirectionnelle qui affine la reconnaissance des formes primitives individuelles en fonction de la reconnaissance de la forme composite.

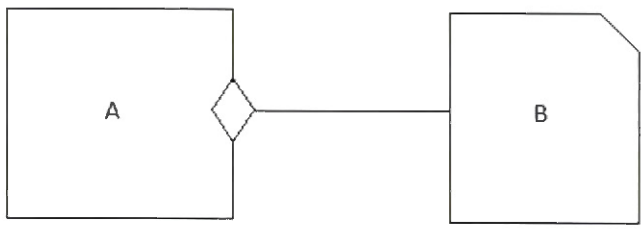


Figure 4.4 La reconnaissance du fichier au lieu de la tâche en forme composite

Comme le montre la figure 4.4, même si notre système trouve qu'un fichier est la meilleure correspondance pour B, car les sentinelles ne sont utilisées que pour lier des tâches, nous pouvons réviser la classification de B comme une tâche. Bien sûr, l'humain pourrait faire une erreur de modélisation, mais nous croyons que l'interaction entre les deux va améliorer le taux de reconnaissance pour les formes composites.

D'un autre côté, l'utilisateur humain a un rôle principal dans l'implémentation de l'esquisse dessinée à la main. Par conséquent, différentes manières dont un utilisateur choisit de dessiner les formes, ou même d'écrire le texte, peuvent être identifiées. Enquête sur le maniérisme de l'utilisateur humain peut affecter la performance d'algorithmes de reconnaissance. Par conséquent, l'évaluation de ces aspects nécessiterait des entrevus les sujets et de surveiller leur comportement, et d'utiliser un cadre de test plus réaliste.

APPENDICE A

CLASSES JAVA DE RECONNAISSANCE DE FORME PRIMITIVE

```
1
2  package CMMNElementsketchRecognitionSystem;
3
4  /**
5   *  This class composed of the main method, start listing all
   files
6   *  from the main directory and its sub directories and then
   calling the class <code>TemplateMatchingDemo</code>
7   *  in order to use the template matching method in the
   OpenCV library and comparing each shape
8   *  of the original image to the template images that is
   specified in the main directory
9   *  and its sub directories. in the following the class of
   WriteXmlFile start writhing the XML file
10  *  according to the structure of CMMN modeler which is able
   to import the XML file inside the CMMN Modeler software
11  *  @author SaraAmirsardari
12  *
13  */
14 public class SketchRecognition {
15
16    public static final int LINE_DETECTION_TRESHOLD =10;
17    public static final int MIN_LINE_LENGTH=7;
18    public static final int MAX_LINE_GAP=30;
19
20    public static void main(String[] args) {
21
22       System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
23
24       // reading the folders and sub folders
25       ReadFolders tc = new ReadFolders();
26       File MainDirectory = new
         File("C:/Users/SARA/Desktop/opencv/sample/template100");
27       ArrayList<String> pathsList = new ArrayList<>();
28       pathsList = tc.readDir(MainDirectory);
29       TemplateMatchingDemo md = new TemplateMatchingDemo();
30       // define for loop in order to read the files with the
         suffix of JPG or PNG
31       ArrayList<String> listOfTemplates=new ArrayList<>();
```

```
32          for (int i = 0; i < pathsList.size(); i++) {
33              if (pathsList.get(i).contains("png") ||
                pathsList.get(i).contains("jpg")) {
34                  listOfTemplates.add(pathsList.get(i));
35              }
36          }
37
38          md.preprocessAllTemplates(listOfTemplates);
39          String sketchFileName =
            "C:/Users/SARA/Desktop/opencv/sample/s.png";
40          md.runMatchingDemo(sketchFileName);
41          DetectLine dl = new DetectLine();
42          dl.setInitialImage(md.getCleanedUpImage());
43          dl.setShapesToRemove(md.getParentContrours());
44          dl.detectLine();
45          WriteXmlFile writeXmlShapes = new WriteXmlFile();
46          writeXmlShapes.setResultOfLines(dl.getResultOfLines());
47          writeXmlShapes.WriteXml(md);
48
49      }
50  }
51
5
```

```
1  package CMMNElementsketchRecognitionSystem;
2
3  import java.io.File;
4  import java.util.ArrayList;
5
6  /**Contains some methods to list files and folders from a
   directory
7   * @author SaraAmirsardari
8   */
9
10 public class ReadFolders{
11
12    /**
13     * get the path of main directory
14     * @param f main directory path to be listed
15     */
16
17    public void readFile(File f){
18       System.out.println(f.getPath());
19    }
20
21    /**
22     * List all files from a directory and its sub directories
23     * @param f sub directory paths to be listed
24     * @return pathList of all directory and its sub directories
25     */
26    public ArrayList<String> readDir(File f){
27
28       File subdir[]=f.listFiles();
29       ArrayList<String> pathsList= new ArrayList<>();
30
31       //verify the sub directory is file or is directory
32       for(File f_arr:subdir){
33
34          if(f_arr.isFile()){
35
36             //if the sub directory is file so read the file path
37             pathsList.add(f_arr.getPath());
38             this.readFile(f_arr);
```

```
39              }
40
41              if (f_arr.isDirectory()) {
42
43                  //if the sub directory is a directory, so list all
                    files path inside the directory
44                  ArrayList<String> dirFiles = this.readDir(f_arr);
45                  pathsList.addAll(dirFiles);
46              }
47          }
48          return pathsList ;
49      }
50
51  }
```

```
1  package CMMNElementsketchRecognitionSystem;

3
4  import java.io.File;
5  import java.util.ArrayList;
6  import java.util.HashMap;
7  import java.util.Iterator;
8  import java.util.List;
9  import java.util.Set;
10 import java.util.Vector;
11
12 import org.opencv.core.Core;
13 import org.opencv.core.Core.MinMaxLocResult;
14 import org.opencv.core.CvType;
15 import org.opencv.core.Mat;
16 import org.opencv.core.MatOfPoint;
17 import org.opencv.core.Point;
18 import org.opencv.core.Range;
19 import org.opencv.core.Rect;
20 import org.opencv.core.Scalar;
21 import org.opencv.core.Size;
22 import org.opencv.highgui.Highgui;
23 import org.opencv.imgproc.Imgproc;
24 import org.opencv.utils.Converters;
25 import org.w3c.dom.css.RGBColor;
26
27 /**
28  * This class does the matching process between input image
    and template's images that include the following steps:
29  * 1) First of all, start doing pre-processing on the input
    image and the template's images
30  * 2) extract features of each contour in the input image
31  * 3) resize the size of each contour according to the width
    of template image as well as keeping the aspect ratio
32  * 4) At the end start finding the best match between each
    contour and template's images
33  * The OpenCV library is used in order to do the
    Pre-processing process as well as doing the template matching
34  * @author SaraAmirsardari
```

```
35   *
36   */
37  class TemplateMatchingDemo {
38
39      private List<MatOfPoint> parentContrours = new ArrayList<MatOfPoint>();
40      private Mat cleanedUpImage;
41
42      public Mat getCleanedUpImage() {
43          return cleanedUpImage;
44      }
45
46      public List<MatOfPoint> getParentContrours() {
47          return parentContrours;
48      }
49
50      public void setParentContrours(List<MatOfPoint> parentContrours) {
51          this.parentContrours = parentContrours;
5..     }
53
54      private int id = -1;
55      public int nextId() {
56          id = id + 1;
57          return id;
58      }
59
60      private int sh = -1;
61
62      public int nextShape() {
63          sh = sh + 1;
64          return sh;
65      }
66
67      /**
68       * this variable will hold the list of templates,organized
         by shape type/name
69       *
70       */
71      private HashMap<String, Mat> templateTable = new HashMap<String,
        Mat>();
```

```
72
73      /**
74       * This variable will contain arrays of coordinates of the
         various shapes
75       * recognized in the input figure, organized by shape
         type/name
76       *
         */
78      private HashMap<String, ArrayList<CoordinatesOfContours>>
        shapeCoordinates = null;
79
80      public ArrayList<CoordinatesOfContours>
        getListOfCoordinatesOfShapesOfType (String shapeName) {
81          return shapeCoordinates.get(shapeName);
82      }
83
84      /**
85       * This class represents a bitmap that was already
         segmented. The actual
86       * bitmap is in <code>segmentedBitMap</code> and the
         contours are
87       * represented in the instance variable
         <code>contours</code>.
88       *
89       * @author SaraAmirsardari
90       *
91       */
92
93      class SegmentedImage {
94          public Mat segmentedBitMap;
95
96          public ArrayList<MatOfPoint> contours;
97
98          public SegmentedImage (Mat bitmap, ArrayList<MatOfPoint>
            listOfContours) {
99              segmentedBitMap = bitmap;
100             contours = listOfContours;
101         }
102     }
```

```
103
104     /**
105      * This function does pre-processing process(gaussian
         blurring, thresholding)
106      * in a picture file (JPEG or PNG) to prepare them for
         matching.
107      * the file name is contained in the string inFile. It
         first loads
108      * the "bitmap" from the file <code>inFile</code> that
         applies filters to it.
109      * @param
110      * @return the processed image matrix
111      */
112     public Mat preprocessImage(String inFile) {
113        // load the image and convert it to gray
114        Mat img = Highgui.imread(inFile,
           Highgui.CV_LOAD_IMAGE_GRAYSCALE);
115        Mat destination = new Mat(img.rows(), img.cols(), img.type());
116        //blur operation reduces noise and smoothing the
           grayscale image
117        Imgproc.GaussianBlur(img, destination, new Size(3, 3), 0);
118        // Threshold operation which converts a grayscale image
           into a binary image
119        Imgproc.threshold(destination, destination, -1, 255,
           Imgproc.THRESH_BINARY_INV + Imgproc.THRESH_OTSU);
120        this.cleanedUpImage = destination.clone();
121        return destination;
122     }
123
124     /**
125      * This function gets the list of
         templates<code>listOfTemplates</code> as a
126      * string and convert them to matrix and then store them
         in the ArrayList
127      * <Mat> of <code>template</code>.
128      * @param listOfTemplates is list of template's images
129      * @return the processed template matrix to the template
         table
130      */
```

```
131
132     public void preprocessAllTemplates (ArrayList<String> listOfTemplates) {
133
134         for (int i = 0; i < listOfTemplates.size(); i++) {
135             String nextTemplateFileName = listOfTemplates.get(i);
136             // find the name of the file, without the .jpg or
                .png extension. That file name will represent the
                name of the
137             // CMMN construct (file, sentry, event, etc. Here is
                an example of what it looks like
138             //
                C:\Users\SARA\Desktop\opencv\sample\template100\task\ta
                sk.png. first, separate file name based on \, and
                remove extension
139
140             String splitter = File.separator.replace("\\", "\\\\");
141             String[] pathElements = nextTemplateFileName.split(splitter);
142             String fileName = pathElements[pathElements.length - 1];
143             String templateName = (fileName.split("\\."))[0];
144             // load the template and convert it to gray
145             Mat img = Highgui.imread(nextTemplateFileName,
                Highgui.CV_LOAD_IMAGE_GRAYSCALE);
146             Mat destination = new Mat(img.rows(), img.cols(), img.type());
147             //blur operation reduces noise and smoothing the
                grayscale template image
148             Imgproc.GaussianBlur(img, destination, new Size(3, 3), 0);
149             // Threshold operation which converts a grayscale
                template image into a binary template image
150             Imgproc.threshold(destination, destination, -1, 255,
                Imgproc.THRESH_BINARY_INV + Imgproc.THRESH_OTSU);
151             // add the processed template matrix to the template
                table
152             templateTable.put(templateName, destination);
153         }
154     }
155
156
157     /**
158      * this function takes as input the name of a graphical
```

```
        file (PNG or JPEG)
159     * and returns a record (an instance of
        <code>SegmentedImage</code>)
160     * consisting of, 1) the bitmap of the segmented image,
        and 2) the list of
161     * contours (each contour being a vector of points).
162     *
163     * @param inFile is image matrix
164     * @return
165     */
166    public SegmentedImage segmentImage(String inFile) {
167        Mat segmentedBitMap = preprocessImage(inFile);
168
169        // find the contours inside input image
170        Mat hierarchy = new Mat();
171        ArrayList<MatOfPoint> contours = new ArrayList<MatOfPoint>();
172        Imgproc.findContours(segmentedBitMap,
           contours,hierarchy,Imgproc.RETR_CCOMP,
           Imgproc.CHAIN_APPROX_NONE);
173        ArrayList<MatOfPoint> contoursToRemove = new
           ArrayList<MatOfPoint>();
174        for ( int idx=0; idx<contours.size(); idx++ ){
175            double[] contourHierarchy = hierarchy.get(0,idx);
176            if(contourHierarchy[3]!=-1){
177                Imgproc.drawContours(segmentedBitMap, contours,idx, new
                   Scalar(255, 255, 255),2);
178                this.parentContrours.add(contours.get(idx));
179            } else {
180                contoursToRemove.add(contours.get(idx));
181            }
182        }
183
184        for(MatOfPoint i: contoursToRemove){
185            contours.remove(i);
186        }
187
188        return new SegmentedImage(segmentedBitMap, contours);
189    }
190
```

```
191     /**
192      * in this function, the feature of each contour inside
         the image extracted
193      * @param enclosingBitmap is the input image
194      * @param shape is the contour inside input image
19       * @return the new contour matrix that include the contour
         feature such as start coordinate (x,y), width and height.
196      */
197     public Mat getShapeSubBitMap(Mat enclosingBitmap, MatOfPoint shape) {
198
199         int w = Imgproc.boundingRect(shape).width;
200         int h = Imgproc.boundingRect(shape).height;
201         int x = Imgproc.boundingRect(shape).x;
202         int y = Imgproc.boundingRect(shape).y;
203         System.out.println(x + " " + y + " " + w + " " + h + " ");
204         Range rowRange = new Range(y, y + h);
20          Range colRange = new Range(x, x + w);
206         return new Mat(enclosingBitmap, rowRange, colRange);
207     }
20
209     /**
210      * This function resize the contours (subShapeBitMap)
         according to the
211      * template size by preserving the scale of the contours.
212      * @param subShapeBitMap is contour matrix
213      * @param templateWidth is width of template image
214      * @return the new size of contour matrix
215      */
21
217     public Size getResizeSize(Mat subShapeBitMap, double templateWidth
        ) {
18
219         double scale = (double) subShapeBitMap.width() / (double)
            subShapeBitMap.height();
220         double newW = templateWidth;
221         double newH = newW / scale;
222         return new Size(newW, newH);
22.     }
224
```

```
225     /**
226      * This function finds the matching between the contour
         that was defined in
227      * <code>currentShapeSubBitMap</code> and the template
         that was defined in
228      * <code>templates</code>
229      *
230      */
231
232     public ArrayList<FindMatching> findMatching (Mat
        currentShapeSubBitMap, Mat segmentedInputBitMap, int shapeId) {
233
234         ArrayList<FindMatching> returnValue = new ArrayList<> ();
235         Set<String> templateNames = templateTable.keySet ();
236         Iterator<String> templateNameIterator = templateNames.iterator ();
237         ArrayList<MinMaxLocResult> results = new
            ArrayList<MinMaxLocResult> ();
238
239         while (templateNameIterator.hasNext ()) {
240             String templateName = templateNameIterator.next ();
241
242             Mat segmentedTemplateBitMap = templateTable.get (templateName);
243             // resize the contour according to the template size
244             Size newSize = getResizeSize (currentShapeSubBitMap,
                segmentedTemplateBitMap.width ());
245             Mat resizedImage = new Mat ();
246             Imgproc.resize (currentShapeSubBitMap, resizedImage, newSize);
247             Mat resizedImage1 = new Mat ();
248             resizedImage1 = resizedImage;
249
250
251             Mat biggerImage, smallerImage;
252             if (resizedImage1.rows () > segmentedTemplateBitMap.rows () ||
253                     resizedImage1.cols () > segmentedTemplateBitMap.cols ()) {
254                 // the image is bigger
255                 biggerImage = resizedImage1;
256                 smallerImage = segmentedTemplateBitMap;
257             } else {
258                 // the template is bigger
```

```
259             biggerImage = segmentedTemplateBitMap;
260             smallerImage = resizedImage1;
61          }
6
6.          int result_cols = biggerImage.cols() - smallerImage.cols() + 1;
264         int result_rows = biggerImage.rows() - smallerImage.rows() + 1;
265         Mat result = new Mat(result_rows, result_cols,
            CvType.CV_32FC1);
2 6         // these two method (Imgproc.TM_SQDIFF and
            Imgproc.TM_SQDIFF_NORMED)give the minimum value
267         Imgproc.matchTemplate(biggerImage, smallerImage, result,
            Imgproc.TM_CCORR_NORMED);
268
269
270
271         ArrayList<Double> listOfMaxVal = new ArrayList<>();
7           //The functions minMaxLoc find the minimum and
            maximum element values and their positions
7           MinMaxLocResult mmr = Core.minMaxLoc(result);
274         results.add(mmr);
            Point matchLoc = mmr.maxLoc;
276         double maxValue = mmr.maxVal;
277         listOfMaxVal.add(maxValue);
278         double globalMaximimum = MaximumValue(listOfMaxVal);
279         if (maxValue > 0) {
280             FindMatching findM = new
                FindMatching(segmentedTemplateBitMap, result, matchLoc,
                maxValue,
281                     templateName);
282             returnValue.add(findM);
283         }
84      }
285     return returnValue;
86
8       }
8
89  private double MaximumValue(ArrayList<Double> listOfMaxVal) {
90
291     double maxValue = listOfMaxVal.get(0);
```

```
292         for (int s = 0; s < listOfMaxVal.size(); s++) {
293             if (listOfMaxVal.get(s) > maxValue)
294                 maxValue = listOfMaxVal.get(s);
295         }
296         return maxValue;
297     }
298
299     /**
300      * runMtchingDemo finds the template in original image
301      * @param inFile is original image
302      * @param templateFile is the template image
303      * @param outFile
304      * @param match_method
305      */
306
307     public void runMatchingDemo(String inFile) {
308
309         System.out.println("\nRunning Template Matching");
310         System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
311
312         // find the contours in the input image
313         SegmentedImage segmentedInputImage = segmentImage(inFile);
314         Mat segmentedInputBitMap = segmentedInputImage.segmentedBitMap;
315         ArrayList<MatOfPoint> contours = segmentedInputImage.contours;
316
317         // let us now iterate over the different shapes/contours
            in the input image, trying to find a match in each case
318
319         shapeCoordinates = new HashMap<String,
            ArrayList<CoordinatesOfContours>>();
320
321         for (int i = 0; i < contours.size(); i++) {
322             MatOfPoint currentShape = contours.get(i);
323             Mat currentShapeSubBitMap =
                getShapeSubBitMap(segmentedInputBitMap, currentShape);
324             int x = Imgproc.boundingRect(currentShape).x;
325             int y = Imgproc.boundingRect(currentShape).y;
326
327
```

```
328             // Doing the template matching and returning an array
                list of Mat with two values:
329             // 1.The template that was compared to
                <code>currentShapeSubBitMap</code>
330             // 2.The result comparison of
                <code>currentShapeSubBitMap</code> and
                <code>segmentedTemplateBitMap</code>
331             ArrayList<FindMatching> results =
                findMatching(currentShapeSubBitMap, segmentedInputBitMap, i);
332             FindMatching bestResult = computeBestResult(results);
333
334
335             // get the shape name
336             String shapeName = bestResult.getTemplateName();
337             // add the current match to the appropriate list of
                shapes
338             ArrayList<CoordinatesOfContours> sameShapeCoordinateArray =
                shapeCoordinates.get(shapeName);
339
340             // if array is empty (this is the first shape of this
                type encountered in the figure) then initialize it
341             if (sameShapeCoordinateArray == null) {
342                 sameShapeCoordinateArray = new
                    ArrayList<CoordinatesOfContours>();
343                 shapeCoordinates.put(shapeName, sameShapeCoordinateArray);
344             }
345
346             CoordinatesOfContours recognizedShapeCoordinates = new
                CoordinatesOfContours(bestResult.getMatchLoc().x + x,
347                     bestResult.getMatchLoc().y + y,
                        bestResult.getSegmentedTemplateBitMap().cols(),
348                     bestResult.getSegmentedTemplateBitMap().rows(),
                        nextId());
349
350             // Just add the recognized square to the list if it
                does not overlap any other
351             CalculateDistanceBetweenContours t = new
                CalculateDistanceBetweenContours();
352             if (!t.isOverlapping(recognizedShapeCoordinates,
```

```
                sameShapeCoordinateArray)) {
353                 recognizedShapeCoordinates.setType(shapeName);
354                 sameShapeCoordinateArray.add(recognizedShapeCoordinates);
355             }
356
357         }
358
359         Set<String> tmpKeySet = shapeCoordinates.keySet();
360         for(String key : tmpKeySet){
 61             ArrayList<CoordinatesOfContours> tmpCordinates =
                shapeCoordinates.get(key);
362             for(CoordinatesOfContours coordinate : tmpCordinates){
363                 System.err.println(coordinate);
364             }
365         }
366     }
367
368     FindMatching computeBestResult(ArrayList<FindMatching> results){
369         FindMatching bestResult = null;
370         for(FindMatching currentResult : results){
371             if(bestResult==null ||
                currentResult.getMaxValue()>bestResult.getMaxValue()){
372                 bestResult = currentResult;
373             }
374         }
375         return bestResult;
376     }
37  }
```

```
1  package CMMNElementsketchRecognitionSystem;
2
3  import org.opencv.core.Mat;
4  import org.opencv.core.Point;
5
6  /** This class represents:
7   * 1. The templates that were already converted from string
    to Mat and stored in <code>segmentedTemplateBitMap</code>
8   * 2. The result that was already received from template
    matching method in order to compare the template and contour
    together
9   * 3. The match location and maximum value for each result
    that were already defined by the <code>matchLoc</code> and
    <code>maxValue</code>
10  * 4. The <code>idTemplate</code> is specified for
    identifying each template
11  * @author SaraAmirsardari
12  *
13  */
14
15 public class FindMatching {
16
17     Mat segmentedTemplateBitMap;
18     Mat result ;
19     Point matchLoc;
20     double maxValue;
21     String templateName;
22
23     public FindMatching(Mat segmentedTemplateBitMap, Mat result,Point
       matchLoc,double maxValue,String templateName ) {
24         this.segmentedTemplateBitMap = segmentedTemplateBitMap;
25         this.result = result;
26         this.matchLoc=matchLoc;
27         this.maxValue=maxValue;
28         this.templateName=templateName;
29     }
30
31     public void setSegmentedTemplateBitMap(Mat
       segmentedTemplateBitMap) {
```

```
32          this.segmentedTemplateBitMap = segmentedTemplateBitMap;
33      }
34      public Mat getSegmentedTemplateBitMap() {
35          return segmentedTemplateBitMap;
36      }
37
38      public void setResult(Mat result) {
39          this.result = result;
40      }
41      public Mat getResult() {
42          return result;
43      }
44
45      public void setMatchLoc(Point matchLoc) {
46          this.matchLoc = matchLoc;
47      }
48      public Point getMatchLoc() {
49          return matchLoc;
50      }
51
52      public void setMaxValue(Double maxValue) {
53          this.maxValue = maxValue;
54      }
55      public Double getMaxValue() {
56          return maxValue;
57      }
58      public void setTemplateName(String templateName) {
59
60          this.templateName=templateName;
61      }
62      public String getTemplateName() {
63          return templateName;
64      }
6
66  }
```

```
1  package CMMNElementsketchRecognitionSystem;
2  import java.util.ArrayList;
3
4  /**
5   * This class calculates the distance between each shape and
    the rest of the shapes in input image
6   * @author SaraAmirsardari
7   */
8  public class CalculateDistanceBetweenContours {
9
10    /**
11     * This method calculate the distance between two shapes
12     * @param s1 is the first shape to calculate.
13     * @param s2 is the second shape to calculate.
14     * @return the distance between shapes.
15     */
16
17    public double calculateDistance(CoordinatesOfContours s1,
      CoordinatesOfContours s2) {
18
19       double distance = 0;
20       double dx = s1.getX() - s2.getX();
21       double dy = s1.getY() - s2.getY();
22       distance = Math.sqrt(dx*dx + dy*dy);
23       return distance;
24    }
25
26    /**
27     * This method Verifies if the shape is overlapping with
      any other shape in the list
28     * @param s is the shape to compare if it is overlapping.
29     * @param list is the list of all other shape that will
      compare to shape.
30     * @return True if the shape overlaps any other shapes,
      false otherwise.
31     */
32
33    public boolean isOverlapping(CoordinatesOfContours s,
      ArrayList<CoordinatesOfContours> list) {
```

```
34
35          boolean overlap = false;
36          for (int i = 0; i < list.size(); i++) {
37              // define a threshold for specifying the overlap
                distance between to shapes
38              double threshold = 7;
39              if(calculateDistance(s, list.get(i)) <= threshold) {
40                  overlap = true;
41                  return overlap;
42              }
43          }
44
45          return overlap;
46      }
47
48  }
49
```

Figure a.4 Represent 'coordinate' java class

```
1   package CMMNElementsketchRecognitionSystem;
2
3   /**
4    * Interface for all the coordinates.
5    * @author SaraAmirsardari
6    */
7   public interface Coordinates {
8
9       public double getX();
10      public double getY();
11      public double getWidth();
12      public double getHeight();
13
14  }
15
```

```
1  package CMMNElementsketchRecognitionSystem;
2
   import org.opencv.core.Point;

   /**
6   * This method gets and sets the start point and end point of
    each contour edge
7   * @author SaraAmirsardari
    *
9   */
10 public class CoordinatesOfContourEdge {
11    private Point startLine;
      private Point endLine;
1
14
1     public Point getStartLine() {
16       return startLine;
17    }
18    public void setStartLine(Point startLine) {
19       this.startLine = startLine;
20    }
21    public Point getEndLine() {
22       return endLine;
23    }
24    public void setEndLine(Point endLine) {
25       this.endLine = endLine;
26    }
   }
28
29
```

```
1  package CMMNElementsketchRecognitionSystem;
2
3  import org.opencv.core.Point;
4
5  public class CoordinatesOfContours implements Coordinates {
6
7     private double x;
8     private double y;
9     private double width;
10    private double height;
11    private int id;
12    private String type = "";
13
14    /**
15     * This class defines the coordinate of  shapes inside the
       input image
16     */
17    public CoordinatesOfContours(double x, double y, double width,
      double height, int id) {
18       this.x = x;
19       this.y = y;
20       this.width = width;
21       this.height = height;
22       this.id = id;
23    }
24
25    public double getX() {
26       return x;
27    }
28
29    public double getY() {
30       return y;
31    }
32
33    public double getWidth() {
34       return width;
35    }
36
37    public double getHeight() {
```

```
38          return height;
39      }
40      public int getid() {
41          return id;
42      }
43
44      public String getType() {
45          return type;
46      }
47
48      public void setType(String type) {
49          this.type = type;
50      }
51
52      public String toString(){
53          return "Coordinate: ("+this.x+","+this.y+"),
            Width:"+this.width+", Height:"+this.height;
54      }
55
56      public CoordinatesOfContourEdge getTopLine(){
57          CoordinatesOfContourEdge result = new CoordinatesOfContourEdge();
58          result.setStartLine(new Point(x,y));
59          result.setEndLine(new Point(x+width, y));
60          return result;
61      }
62
63      public CoordinatesOfContourEdge getBottomLine(){
64          CoordinatesOfContourEdge result = new CoordinatesOfContourEdge();
65          result.setStartLine(new Point(x,y+height));
66          result.setEndLine(new Point(x+width, y+height));
67          return result;
68      }
69
70      public CoordinatesOfContourEdge getLeftLine(){
71          CoordinatesOfContourEdge result = new CoordinatesOfContourEdge();
72          result.setStartLine(new Point(x,y));
73          result.setEndLine(new Point(x, y+height));
74          return result;
75      }
```

```
76
77      public CoordinatesOfContourEdge getRightLine(){
78          CoordinatesOfContourEdge result = new CoordinatesOfContourEdge();
79          result.setStartLine(new Point(x+width,y));
80          result.setEndLine(new Point(x+width, y+height));
81          return result;
82      }
83
84  }
85
```

APPENDICE B

CLASSES JAVA DE RECONNAISSANCE DE FORME COMPOSITE

```
1  package CMMNElementsketchRecognitionSystem;

3  /**
4   * This class gets and sets the coordinates of lines
5   * @author SaraAmirsardari
6   */
   public class CoordinatesOfLines{
8
9
10    private double x1;
11    private double y1;
12    private double x2;
13    private double y2;
14    private int id;
15
16    /**
17     * This class defines the coordinate of each line inside
       the input image
18     */
19
20    public CoordinatesOfLines(double x1, double y1, double x2, double
      y2,int id) {
21
22       this.x1 = x1;
23       this.y1 = y1;
24       this.x2 = x2;
25       this.y2 = y2;
26       this.id = id;
27    }
28
29    public double getX1() {
30       return x1;
31    }

33    public double getY1() {
34       return y1;
35    }
36
3     public double getX2() {
```

```
        return x2;
    }

    public double getY2() {
        return y2;
    }

    public int getid() {

        return id;
    }

    public String toString(){
        return "start point: ("+this.x1+","+this.y1+"), end
        point: ("+this.x2+","+this.y2+")";
    }
}

```

```
package CMMNElementsketchRecognitionSystem;
/**
 * This class get the input image and delete all closed
 contour
 * and start recognizing the contours which include lines.
 * the OpenCV library is used in order to detect lines
 * @author SaraAmirsardari
 *
 */

public class DetectLine {

    private ArrayList<CoordinatesOfLines> ResultOfLines=new ArrayList<>();

    public ArrayList<CoordinatesOfLines> getResultOfLines() {

        return ResultOfLines;
    }

    private Mat initialImage;
    private List<MatOfPoint> shapesToRemove;


    public Mat getInitialImage() {
        return initialImage;
    }

    public void setInitialImage(Mat initialImage) {
        this.initialImage = initialImage;
    }

    public List<MatOfPoint> getShapesToRemove() {
        return shapesToRemove;
    }

    public void setShapesToRemove(List<MatOfPoint> shapesToRemove) {
        this.shapesToRemove = shapesToRemove;
    }
    /**
```

```
39      * This method start defining the bounding box around each
        closed shape
40      * and then using threshold in order to increase the area
        of each closed shape
41      * @param shape
42      */
43     private void removeShape(MatOfPoint shape){
44
45        int x = Imgproc.boundingRect(shape).x;
46        int y = Imgproc.boundingRect(shape).y;
47        int width = Imgproc.boundingRect(shape).width;
48        int height = Imgproc.boundingRect(shape).height;
49        MatOfPoint mpoints = new MatOfPoint();
50        double threshold = 8;
51        List<Point> points = new ArrayList<Point>();
52        points.add(new Point(x-threshold,y-threshold));
53        points.add(new Point(x+width+threshold,y-threshold));
54        points.add(new Point(x+width+threshold,y+height+threshold));
55        points.add(new Point(x-threshold,y+height+threshold));
56
57        mpoints.fromList(points);
58        //paint all closed contours by black color
59        Core.fillConvexPoly(this.initialImage, mpoints,new Scalar(0,0,0));
60     }
61     public  void detectLine() {
62
63        this.ResultOfLines = new ArrayList<>();
64        for(MatOfPoint shape: this.shapesToRemove){
65           removeShape(shape);
66        }
67
68        //      image - 8-bit, single-channel binary source
          image. The image may be modified by the function.
69        //    lines - Output vector of lines. Each line is
          represented by a 4-element vector  (x_1, y_1, x_2, y_2),
70        //      where  (x_1,y_1) and  (x_2, y_2) are the ending
          points of each detected line segment.
71        //    rho : The resolution of the parameter r in pixels.
          We use 1 pixel.
```

```
72          //    theta: The resolution of the parameter theta in
            radians. We use 1 degree (CV_PI/180)
73          //    threshold: The minimum number of intersections to
            "detect" a line
74          //    minLinLength: The minimum number of points that
            can form a line. Lines with less than this number of
            points are disregarded.
            //    maxLineGap: The maximum gap between two points to
            be considered in the same line.
76
77          Mat line = new Mat();
78          int threshold = SketchRecognition.LINE_DETECTION_TRESHOLD;
79          int minLineLength =SketchRecognition.MIN_LINE_LENGTH;
80          int maxLineGap =SketchRecognition.MAX_LINE_GAP;
81          int id=0;
82          Imgproc.Canny(this.initialImage, this.initialImage, 50, 200);
83          Imgproc.HoughLinesP(this.initialImage, line, 1, Math.PI/180,
            threshold, minLineLength, maxLineGap);
84
85          for(int i = 0; i < line.cols(); i++) {
86              double[] val = line.get(0, i);
87              double  x1 = val[0],
88                      y1 = val[1],
89                      x2 = val[2],
90                      y2 = val[3];
91
92              CoordinatesOfLines recognizeLine = new CoordinatesOfLines(x1,
                y1, x2, y2,id);
93              CalculateDistanceBetweenLines linedistance = new
                CalculateDistanceBetweenLines();
94              linedistance.mergingLines(recognizeLine, ResultOfLines);
95
96          }
97
98      }
99
100 }
101
```

```
1  package CMMNElementsketchRecognitionSystem;
2  import java.util.ArrayList;
3  import java.util.Collections;
4
5  /**
6   * This class calculates the distance between each line and
    the rest of the lines in input image
7   * @author SaraAmirsardari
8   */
9
10 public class CalculateDistanceBetweenLines {
11
12    /**
13     * This method calculate the distance between two lines.
14     * @param line1 is the first line to calculate.
15     * @param line2 is the second line to calculate.
16     * @return
17     */
18    public double calculateDistance(CoordinatesOfLines line1,
      CoordinatesOfLines line2) {
19
20       double distance = 0;
21       double dx = line1.getX1() - line2.getX1();
22       double dy = line1.getY1() - line2.getY1();
23       distance = Math.sqrt(dx*dx + dy*dy);
24       return distance;
25    }
26
27    /**
28     * This method merges the lines according to their distance
29     * @param line is the first line to compare its distance
      with the rest of line in list
30     * @param list is the list of all lines in input image
31     * this method verifies:
32     * first: the distance of the two lines that is less that
      threshold or not,
33     * second: if it is less than the threshold, it starts
      merging two lines based on
34     * the minimum start point of lines and maximum end point
```

```
     of lines
     * this method returns the longest line
     */
    public void mergingLines(CoordinatesOfLines line,
    ArrayList<CoordinatesOfLines> list) {

        ArrayList<CoordinatesOfLines> linesToRemove=new ArrayList<>();
        for (int i = 0; i < list.size(); i++) {
            // define a threshold for specifying the standard
            distance between independent lines
            double threshold = 12;
            int id=0;

            if(calculateDistance(line, list.get(i)) <= threshold) {
                id++;
                ArrayList<Double> coordinateX=new ArrayList<>();
                coordinateX.add(line.getX1());
                coordinateX.add(list.get(i).getX1());
                coordinateX.add(line.getX2());
                coordinateX.add(list.get(i).getX2());
                Double linex1 = Collections.min(coordinateX);
                Double linex2 = Collections.max(coordinateX);

                ArrayList<Double> coordinateY=new ArrayList<>();
                coordinateY.add(line.getY1());
                coordinateY.add(list.get(i).getY1());
                coordinateY.add(line.getY2());
                coordinateY.add(list.get(i).getY2());
                Double liney1 = Collections.min(coordinateY);
                Double liney2 = Collections.max(coordinateY);

                CoordinatesOfLines newLine = new CoordinatesOfLines(linex1,
                liney1, linex2, liney2,id);

                line = newLine;
                linesToRemove.add(list.get(i));
            }

        }
```

```
            list.add(line);
            for(CoordinatesOfLines lineToRemove : linesToRemove){
                list.remove(lineToRemove);
            }
        }
    }


package CMMNElementsketchRecognitionSystem;

import org.opencv.core.Point;

/**
 * This class gets and sets the coordinate of shape and
 coordinate of line as well as the distance between them
 * @author SaraAmirsardari
 *
 */
public class DistanceFromContourToLine {
    CoordinatesOfContours shape;
    Point linePoint;
    double distance;

    public double getDistance() {
        return distance;
    }
    public void setDistance(double distance) {
        this.distance = distance;
    }
    public CoordinatesOfContours getShape() {
        return shape;
    }
    public void setShape(CoordinatesOfContours shape) {
        this.shape = shape;
    }
    public Point getLinePoint() {
        return linePoint;
    }
    public void setLinePoint(Point linePoint) {
        this.linePoint = linePoint;
    }
}
```

```
1  package CMMNElementsketchRecognitionSystem;
   /**
3   * This class calculates the distance of start point and end
    point of each line with two specified shapes( task and
    sentry).
4   * Hence, we need to get the coordinates of lines from
    <code>detectLine</code> class as well as
5   * the coordinates of tasks and sentries from
    <code>WriteXmlFile</code> class
6   * @author SaraAmirsardari
7   *
8   */
9  public class Connector {
10
11     private ArrayList<CoordinatesOfLines> resultOfLines= new ArrayList<>();
12     private ArrayList<CoordinatesOfContours> resultOfTasks= new
       ArrayList<>();
13     private ArrayList<CoordinatesOfContours> resultOfSentries= new
       ArrayList<>();
14
15
16     public void setResultOfLines (ArrayList<CoordinatesOfLines>
       resultOfLines) {
17        this.resultOfLines = resultOfLines;
18     }
19     public ArrayList<CoordinatesOfLines> getResultOfLines () {
20        return resultOfLines;
21     }
22
23     public void setResultOfTasks (ArrayList<CoordinatesOfContours>
       resultOfTasks) {
24        this.resultOfTasks = resultOfTasks;
25     }
26     public ArrayList<CoordinatesOfContours> getResultOfTasks () {
27
28        return resultOfTasks;
29     }
30
31     public void setResultOfSentries (ArrayList<CoordinatesOfContours>
```

```
resultOfSentries) {
32          this.resultOfSentries = resultOfSentries;
33      }
34
35      public ArrayList<CoordinatesOfContours> getResultOfSentries () {
36
37          return resultOfSentries;
38      }
39
40      /**
41       * This method calls the
         <code>findConnexionForPoint</code> method in order to
         calculate
42       * the distance of start point and end point of line with
         the specified shapes
43       * @param line is the coordinate of each line
44       * @return the result which includes the list of shapes
         connected to the line
45       */
46
47      public List<CoordinatesOfContours>
        getCloserShapeForLine (CoordinatesOfLines line) {
48
49          List<CoordinatesOfContours> result= new ArrayList<>();
50          result.add( findConnexionForPoint (new Point (line.getX1 (),
            line.getY1 ()) ) );
51          result.add( findConnexionForPoint (new Point (line.getX2 (),
            line.getY2 ()) ) );
52          return result;
53      }
54
55      /**
56       *
57       * @param p is one of the start point or end point of line
58       * this method calculate the distance of start point or
         end point of line with the specified shapes
59       * @return the minimum distance of each start point or end
         point of line with the specified shapes
60       */
```

```
61      public CoordinatesOfContours findConnexionForPoint(Point p){
6           List<DistanceFromContourToLine> distances = new ArrayList<>();
63          for(CoordinatesOfContours task : this.resultOfTasks){
64              DistanceFromContourToLine distance = new
                DistanceFromContourToLine();
65              distance.setShape(task);
66              distance.setDistance(computeDistance(task, p ));
67              distances.add(distance);
68          }
69          for(CoordinatesOfContours sentry : this.resultOfSentries){
70              DistanceFromContourToLine distance = new
                DistanceFromContourToLine();
71              distance.setShape(sentry);
72              distance.setDistance(computeDistance(sentry, p));
73              distances.add(distance);
74          }
75
76          DistanceFromContourToLine minimalDistance = null;
77          for(DistanceFromContourToLine distance : distances){
78              if(minimalDistance == null ||
                distance.getDistance()<minimalDistance.getDistance()){
79                  minimalDistance = distance;
80              }
81          }
82          return minimalDistance.getShape();
83      }
84
85      /**
86       * This class computes the distance of start point and end
         point of line with two
87       * edges of closed counter that can be vertical or
         horizontal
88       */
89      private double computeDistanceToLine(Point pointToCompute,
        CoordinatesOfContourEdge contourEdge){
90
91          double threshold = 5;
92          Point lineStart = contourEdge.getStartLine();
93          Point lineEnd = contourEdge.getEndLine();
```

```
94          boolean isHorizontal = (lineStart.y == lineEnd.y);
95          if(isHorizontal){
96              //It is a horizontal line, Make sure that lineStart
                has smaller x
97              if(lineStart.x>lineEnd.x){
98                  Point p = lineStart;
99                  lineStart = lineEnd;
100                 lineEnd = p ;
101             }
102             //If the point is not between the start point of line
                and end point of line then return infinite value
103             if(pointToCompute.x<lineStart.x-threshold ||
                pointToCompute.x>lineEnd.x+threshold){
104                 return Double.MAX_VALUE;
105             }else{
106                 //If the point is between the start point of line
                    and end point of line  then calculate the distance
107
108                 return Math.abs(lineStart.y-pointToCompute.y);
109             }
110         } else {
111             //It is a vertical line, Make sure that lineStart has
                smaller y
112             if(lineStart.y>lineEnd.y){
113                 Point p = lineStart;
114                 lineStart = lineEnd;
115                 lineEnd = p ;
116             }
117             //If the point is not between the start point of line
                and end point of line  then return infinite value
118             if(pointToCompute.y<lineStart.y-threshold ||
                pointToCompute.y>lineEnd.y+threshold){
119
120                 return Double.MAX_VALUE;
121             }else{
122
123                 //If the point is between the start point of line
                    and end point of line  then calculate the distance
124                 return Math.abs(lineStart.x-pointToCompute.x);
```

```
            }
        }
    }

    public double computeDistance(CoordinatesOfContours shape, Point p){

        double result = Double.MAX_VALUE;
        List<Double> distances = new ArrayList<>();

        distances.add( new Double( computeDistanceToLine( p,
        shape.getTopLine() ) ) );
        distances.add( new Double( computeDistanceToLine( p,
        shape.getBottomLine() ) ) );
        distances.add( new Double( computeDistanceToLine( p,
        shape.getLeftLine() ) ) );
        distances.add( new Double( computeDistanceToLine( p,
        shape.getRightLine() ) ) );

        for(Double distance : distances){
            if(distance.doubleValue() < result )
                result = distance.doubleValue();
        }

        return result;

    }
}
```

```
1  package CMMNElementsketchRecognitionSystem;
2
3  import org.opencv.core.Point;
4
5  /**
6   * This class gets and sets the coordinate of lines and the
    shapes that are connected to lines
7   * @author SaraAmirsardari
8   *
9   */
10 public class ConnectorResult {
11
12     CoordinatesOfContours shape;
13     Point linePoint;
14
15     public ConnectorResult(CoordinatesOfContours shape, Point linePoint) {
16     this.shape=shape;
17     this.linePoint=linePoint;
18
19     }
20     public CoordinatesOfContours getShape() {
21         return shape;
22     }
23     public void setShape(CoordinatesOfContours shape) {
24         this.shape = shape;
25     }
26     public Point getLinePoint() {
27         return linePoint;
28     }
29     public void setLinePoint(Point linePoint) {
30         this.linePoint = linePoint;
31     }
32 }
33
```

APPENDICE C

CLASSES JAVA DE RECONNAISSANCE DE CONNEXION SÉMANTIQUE

```
1  package CMMNElementsketchRecognitionSystem;
   /**
3   * This class start writhing the XML file according to the
    structure of CMMN modeler
4   * which is able to import the XML file inside the CMMN
    Modeler software
5   * @author SaraAmirsardari
6   *
7   */
8  public class WriteXmlFile {
9
10     private Map<CoordinatesOfContours, CoordinatesOfContours>
       connectionsMap = new HashMap<>();
11     private Map<CoordinatesOfContours, CoordinatesOfLines> shapesToLines
       = new HashMap<>();
12
13     private ArrayList<CoordinatesOfContours> resultOfTasks= new
       ArrayList<>();
14     public ArrayList<CoordinatesOfContours> getResultOfTasks (){
15        return resultOfTasks;
16     }
17
18     private ArrayList<CoordinatesOfContours> resultOfSentries= new
       ArrayList<>();
19     public ArrayList<CoordinatesOfContours> getResultOfSentries (){
20        return resultOfSentries;
21     }
22
23     private ArrayList<CoordinatesOfLines> resultOfLines= new ArrayList<>();
24     public ArrayList<CoordinatesOfLines> getResultOfLines () {
25
26        return resultOfLines;
27     }
28
29     public void setResultOfLines (ArrayList<CoordinatesOfLines>
       resultOfLines) {
30        this.resultOfLines = resultOfLines;
31     }
32
```

```
33      public void WriteXml (TemplateMatchingDemo md) {
34
35          try {
36
37              DocumentBuilderFactory docFactory =
                DocumentBuilderFactory.newInstance();
38              DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
39
40              // root elements
41              Document doc = docBuilder.newDocument();
42              doc.setXmlStandalone(true);
43              Element rootElement = doc.createElement("cmmn:definitions");
44              doc.appendChild(rootElement);
45
46              // staff elements
47
48              Element staff1 =
                doc.createElement("cmmn:caseFileItemDefinition");
49              rootElement.appendChild(staff1);
50
51              Element staff = doc.createElement("cmmn:case");
52              rootElement.appendChild(staff);
53
54              Element staff2 = doc.createElement("cmmndi:CMMNDI");
55              rootElement.appendChild(staff2);
56
57              //set attribute for root element
58
59              Attr defv1 = doc.createAttribute("author");
60              defv1.setValue("");
61              rootElement.setAttributeNode(defv1);
62
63              Attr defv2 = doc.createAttribute("exporter");
64              defv2.setValue("CMMN Modeler");
65              rootElement.setAttributeNode(defv2);
66
67              Attr defv3 = doc.createAttribute("id");
68              defv3.setValue("_bcc573eb-adf3-4fb4-abb5-434ae50ac5ce");
69              rootElement.setAttributeNode(defv3);
```

```
70
71          Attr defv4 = doc.createAttribute("name");
72          defv4.setValue("Drawing 1");
73          rootElement.setAttributeNode(defv4);
74
75          Attr defv5 = doc.createAttribute("targetNamespace");
76
            defv5.setValue("http://www.trisotech.com/cmmn/definitions
            /_bcc573eb-adf3-4fb4-abb5-434ae50ac5ce");
77          rootElement.setAttributeNode(defv5);
78
79          Attr defv6 = doc.createAttribute("xmlns");
80
            defv6.setValue("http://www.trisotech.com/cmmn/definitions
            /_bcc573eb-adf3-4fb4-abb5-434ae50ac5ce");
81          rootElement.setAttributeNode(defv6);
82
83          Attr defv7 = doc.createAttribute("xmlns:dc");
84
            defv7.setValue("http://www.omg.org/spec/CMMN/20151109/DC"
            );
85          rootElement.setAttributeNode(defv7);
86
87          Attr defv8 = doc.createAttribute("xmlns:trisofeed");
88          defv8.setValue("http://trisotech.com/feed");
89          rootElement.setAttributeNode(defv8);
90
91          Attr defv9 = doc.createAttribute("xmlns:triso");
92
            defv9.setValue("http://www.trisotech.com/2015/triso/model
            ing");
93          rootElement.setAttributeNode(defv9);
94
95          Attr defv10 = doc.createAttribute("xmlns:di");
96
            defv10.setValue("http://www.omg.org/spec/CMMN/20151109/DI
            ");
97          rootElement.setAttributeNode(defv10);
98
```

```
99          Attr defv11 = doc.createAttribute("xmlns:rss");
100         defv11.setValue("http://purl.org/rss/2.0/");
101         rootElement.setAttributeNode(defv11);
102
103         Attr defv12 = doc.createAttribute("xmlns:cmmndi");
104
            defv12.setValue("http://www.omg.org/spec/CMMN/20151109/CM
            MNDI");
105         rootElement.setAttributeNode(defv12);
106
107         Attr defv13 = doc.createAttribute("xmlns:trisob");
108
            defv13.setValue("http://www.trisotech.com/2014/triso/bpmn
            ");
109         rootElement.setAttributeNode(defv13);
110
111         Attr defv14 = doc.createAttribute("xmlns:cmmn");
112
            defv14.setValue("http://www.omg.org/spec/CMMN/20151109/MO
            DEL");
113         rootElement.setAttributeNode(defv14);
114
115         Attr defv15 = doc.createAttribute("xmlns:xsi");
116
            defv15.setValue("http://www.w3.org/2001/XMLSchema-instanc
            e");
117         rootElement.setAttributeNode(defv15);
118
119         Attr defv16 = doc.createAttribute("xmlns:trisocmmn");
120
            defv16.setValue("http://www.trisotech.com/2014/triso/cmmn
            ");
121         rootElement.setAttributeNode(defv16);
122
123         //finish set attribute for root element
124
125         // get the coordinate of contours which are matched
            with the template's images
126
```

```
127             ArrayList<CoordinatesOfContours> Filelist =
                md.getListOfCoordinatesOfShapesOfType("file");
128             if(Filelist==null) Filelist = new
                ArrayList<CoordinatesOfContours>();
129
130             ArrayList<CoordinatesOfContours> squarelist =
                md.getListOfCoordinatesOfShapesOfType("task");
131             if(squarelist==null) squarelist = new
                ArrayList<CoordinatesOfContours>();
132
133             this.resultOfTasks=squarelist;
134
135             ArrayList<CoordinatesOfContours> Sentrieslist =
                md.getListOfCoordinatesOfShapesOfType("sentry");
136             if(Sentrieslist==null) Sentrieslist = new
                ArrayList<CoordinatesOfContours>();
137
138             ArrayList<CoordinatesOfContours> Eventlist =
                md.getListOfCoordinatesOfShapesOfType("event");
139             if(Eventlist==null) Eventlist = new
                ArrayList<CoordinatesOfContours>();
140
141             // set attribute to staff1
                element(caseFileItemDefinition)
142             for (CoordinatesOfContours recognizefile : Filelist) {
143                 writeFileItemDefinition(doc,staff1, recognizefile);
144             }
145
146             // set attribute to staff element(case)
147             Attr attr = doc.createAttribute("id");
148
                attr.setValue("Case_3b0a4c03-c271-47c3-9e87-30c57c034fdb")
                ;
149             staff.setAttributeNode(attr);
150
151             Attr attr1 = doc.createAttribute("name");
152             attr1.setValue("Page 1");
153             staff.setAttributeNode(attr1);
154
```

```
155             // set attribute to casefilemodel
156             Element casefilemodel =
                doc.createElement("cmmn:caseFileModel");
157             staff.appendChild(casefilemodel);
158
159             for (CoordinatesOfContours recognizefile : Filelist) {
160                 writeFileItem(doc, casefilemodel, recognizefile);
161             }
162
163             Element caseplanmodel =
                doc.createElement("cmmn:casePlanModel");
164             staff.appendChild(caseplanmodel);
165
166             // set attribute to caseplanmodel element
167             Attr caseplan = doc.createAttribute("id");
168
                caseplan.setValue("_3b0a4c03-c271-47c3-9e87-30c57c034fdb")
                ;
169             caseplanmodel.setAttributeNode(caseplan);
170
171             Attr caseplan1 = doc.createAttribute("autoComplete");
172             caseplan1.setValue("false");
173             caseplanmodel.setAttributeNode(caseplan1);
174
175             Attr caseplan2 = doc.createAttribute("name");
176             caseplan2.setValue("Page 1");
177             caseplanmodel.setAttributeNode(caseplan2);
178
179             // calculate if there is an intersection between
                square and diamond or not
180             ArrayList <CoordinatesOfContours> intersectionSentries= new
                ArrayList<>();
181             for (CoordinatesOfContours coordinatesOfSquare : squarelist) {
182                 // define the list of sentries that have
                    intersection with squares
183
184                 for (CoordinatesOfContours coordinatesOfSentries :
                    Sentrieslist) {
185
```

```
            if(CalculateIntersectionArea.recognizeIntersection(coordinates
            OfSentries,coordinatesOfSquare)){
186             intersectionSentries.add(coordinatesOfSentries);
187         }
188     }
189
190     this.resultOfSentries=intersectionSentries;
191     writeplanItem(doc, caseplanmodel, coordinatesOfSquare,
        intersectionSentries, null);
192 }
193
194 //define the connector and the shapes connected to it
195
196 Connector connector=new Connector();
197 connector.setResultOfLines(this.resultOfLines);
198 connector.setResultOfSentries(this.getResultOfSentries());
199 connector.setResultOfTasks(this.getResultOfTasks());
200
201 for(CoordinatesOfLines line : this.resultOfLines){
202
203     List<CoordinatesOfContours> shapes =
        connector.getCloserShapeForLine(line);
204
205     connectionsMap.put(shapes.get(0), shapes.get(1));
206     connectionsMap.put(shapes.get(1), shapes.get(0));
207     shapesToLines.put(shapes.get(0), line);
208     shapesToLines.put(shapes.get(1), line);
209 }
210
211 //writhing  planItem element
21  for (CoordinatesOfContours coordinationEvent : Eventlist) {
213     writeplanItem(doc, caseplanmodel, null, null,
        coordinationEvent);
214 }
215 //writhing Sentry element
216 for (CoordinatesOfContours coordinationSentries :
    this.resultOfSentries) {
217     writeSentry(doc, caseplanmodel,
        coordinationSentries,connectionsMap.get(coordinationSentries),sh
```

```
            apesToLines.get(coordinationSentries));
        }

        //writhing  Event element
        for (CoordinatesOfContours coordinationEvent : Eventlist) {
            writeEvent(doc, caseplanmodel, coordinationEvent);
        }
        //writhing Task element
        for (CoordinatesOfContours coordinationSquare : squarelist) {
            writeTask(doc, caseplanmodel, coordinationSquare);
        }

        //writhing CMMN Diagram

        Element CMMNDiagram =
        doc.createElement("cmmndi:CMMNDiagram");
        staff2.appendChild(CMMNDiagram);

        // set attribute
        Attr Diagramv1 = doc.createAttribute("id");

        Diagramv1.setValue("_180025a0-f126-4805-8689-7ee0a0f3c190
        ");
        CMMNDiagram.setAttributeNode(Diagramv1);

        Attr Diagramv2 = doc.createAttribute("name");
        Diagramv2.setValue("Page 1");
        CMMNDiagram.setAttributeNode(Diagramv2);

        Attr Diagramv3 = doc.createAttribute("sharedStyle");

        Diagramv3.setValue("cb1a46a0-82e9-4c14-8495-8d3f50061e96"
        );
        CMMNDiagram.setAttributeNode(Diagramv3);

        //writhing the size as child of CMMN Diagram

        Element cmmndiSize = doc.createElement("cmmndi:Size");
        CMMNDiagram.appendChild(cmmndiSize);
```

```
251             // set attribute
25              Attr Sizev1 = doc.createAttribute("height");
253             Sizev1.setValue("1050.0");
254             cmmndiSize.setAttributeNode(Sizev1);
255
256             Attr Sizev2 = doc.createAttribute("width");
257             Sizev2.setValue("1485.0");
258             cmmndiSize.setAttributeNode(Sizev2);
259
260             //writhing the shape as child of CMMN Diagram
261
262             Element CMMNShape = doc.createElement("cmmndi:CMMNShape");
263             CMMNDiagram.appendChild(CMMNShape);
264
265             // set attribute
266             Attr Shapev1 = doc.createAttribute("cmmnElementRef");
267
                Shapev1.setValue("_3b0a4c03-c271-47c3-9e87-30c57c034fdb")
                ;
268             CMMNShape.setAttributeNode(Shapev1);
269
270             Attr Shapev2 = doc.createAttribute("id");
271
                Shapev2.setValue("_d8d81e5a-d265-4ba1-9f94-4b0d47037451")
                ;
272             CMMNShape.setAttributeNode(Shapev2);
273
274             Element dcBounds  = doc.createElement("dc:Bounds");
275             CMMNShape.appendChild(dcBounds);
276
277             Attr boundv1 = doc.createAttribute("height");
278             boundv1.setValue("600.0");
279             dcBounds.setAttributeNode(boundv1);
280
281             Attr boundv2 = doc.createAttribute("width");
282             boundv2.setValue("800.0");
283             dcBounds.setAttributeNode(boundv2);
284
285             Attr boundv3 = doc.createAttribute("x");
```

```
286             boundv3.setValue("34.0");
287             dcBounds.setAttributeNode(boundv3);
288
289             Attr boundv4 = doc.createAttribute("y");
290             boundv4.setValue("34.0");
291             dcBounds.setAttributeNode(boundv4);
292
293             Element cmmndiCMMNLabe =
                doc.createElement("cmmndi:CMMNLabel");
294             CMMNShape.appendChild(cmmndiCMMNLabe);
295
296
297             for(CoordinatesOfContours coordinateOfSentry :
                this.resultOfSentries){
298                 CoordinatesOfLines line =
                    shapesToLines.get(coordinateOfSentry);
 99                 writeLineValus(doc, CMMNDiagram, line, coordinateOfSentry);
300             }
301
302             for (CoordinatesOfContours coordinatesOfSquare : squarelist ) {
303                 writetaskValues(doc, CMMNDiagram, coordinatesOfSquare);
304             }
305             for (CoordinatesOfContours coordinatesOfSentries : Sentrieslist ) {
306                 writeEntryCriterionValus(doc, CMMNDiagram,
                    coordinatesOfSentries);
307             }
308             for (CoordinatesOfContours coordinationOfEvent : Eventlist ) {
309                 writeEventValues(doc, CMMNDiagram, coordinationOfEvent);
310             }
311
312             for (CoordinatesOfContours coordinatesOfFile : Filelist) {
313                 writeFileValues(doc, CMMNDiagram, coordinatesOfFile);
314             }
315             //writhing the style as child of CMMN Diagram
316             Element cmmndiStyle = doc.createElement("cmmndi:CMMNStyle");
317             staff2.appendChild(cmmndiStyle);
318
319             // set attribute
320             Attr Stylev1 = doc.createAttribute("fontFamily");
```

```
321             Stylev1.setValue("Arial,Helvetica,sans-serif");
32              cmmndiStyle.setAttributeNode(Stylev1);
323
324             Attr Stylev2 = doc.createAttribute("id");
325             Stylev2.setValue("cb1a46a0-82e9-4c14-8495-8d3f50061e96");
326             cmmndiStyle.setAttributeNode(Stylev2);
327
328             // write the content into xml file
329             TransformerFactory transformerFactory =
                TransformerFactory.newInstance();
330             Transformer transformer = transformerFactory.newTransformer();
331             transformer.setOutputProperty(OutputKeys.STANDALONE, "yes");
332             DOMSource source = new DOMSource(doc);
333             StreamResult result = new StreamResult(new
                File("C:/Users/SARA/Desktop/opencv/result.cmmn"));
334             transformer.transform(source, result);
335             System.out.println("File saved!");
336
337         } catch (ParserConfigurationException pce) {
338             pce.printStackTrace();
339         } catch (TransformerException tfe) {
340             tfe.printStackTrace();
341         }
342     }
343
344     public void writeFileItemDefinition(Document doc,Element staff1,
        CoordinatesOfContours recognizefile){
345
346         Attr planitemv1 = doc.createAttribute("id");
347         planitemv1.setValue("fr"+ recognizefile.getid());
348         staff1.setAttributeNode(planitemv1);
349
350     }
351
352     public void writeFileItem(Document doc,Element casefilemodel,
        CoordinatesOfContours recognizefile){
353
354         Element cmmncaseFileItem =
            doc.createElement("cmmn:caseFileItem");
```

```
355         casefilemodel.appendChild(cmmncaseFileItem);
356
357         //set attribute
358         Attr fileitemv1 = doc.createAttribute("definitionRef");
359         fileitemv1.setValue("fr" + recognizefile.getid());
360         cmmncaseFileItem.setAttributeNode(fileitemv1);
361
362         Attr fileitemv2 = doc.createAttribute("multiplicity");
363         fileitemv2.setValue("Unspecified");
364         cmmncaseFileItem.setAttributeNode(fileitemv2);
365
366         Attr fileitemv3 = doc.createAttribute("id");
367         fileitemv3.setValue("fv" + recognizefile.getid());
368         cmmncaseFileItem.setAttributeNode(fileitemv3);
369
370     }
371
372     public void writeplanItem(Document doc,Element caseplanmodel,
    CoordinatesOfContours
    coordinatesOfSquare,ArrayList<CoordinatesOfContours>
    intersectionSentries,CoordinatesOfContours  coordinationEvent){
373
374         Element cmmnplanItem = doc.createElement("cmmn:planItem");
375         caseplanmodel.appendChild(cmmnplanItem);
376
377         // set attribute
378
379         Attr planitemv1 = doc.createAttribute("definitionRef");
380
381         if(coordinatesOfSquare!= null){
382             planitemv1.setValue("t" + coordinatesOfSquare.getid());
383         }else if(coordinationEvent != null){
384             planitemv1.setValue("e" + coordinationEvent.getid());
385         }
386         cmmnplanItem.setAttributeNode(planitemv1);
387
388         Attr planitemv2 = doc.createAttribute("id");
389         if(coordinatesOfSquare!= null){
390             planitemv2.setValue("pi" + coordinatesOfSquare.getid() );
```

```
391         } else if(coordinationEvent != null){
392             planitemv2.setValue("pi" + coordinationEvent.getid() );
393         }
394         cmmnplanItem.setAttributeNode(planitemv2);
395
396
397         if(intersectionSentries!=null){
398             if(!intersectionSentries.isEmpty()){
399                 for (CoordinatesOfContours coordinationSentries :
                    intersectionSentries) {
400
401                     Element entryCriterion =
                        doc.createElement("cmmn:entryCriterion");
402                     cmmnplanItem.appendChild(entryCriterion);
403
404                     Attr entryCriterionv1 = doc.createAttribute("sentryRef");
405                     entryCriterionv1.setValue("senR" +
                        coordinationSentries.getid());
406                     entryCriterion.setAttributeNode(entryCriterionv1);
407
408                     Attr entryCriterionv2 = doc.createAttribute("id");
409
                        entryCriterionv2.setValue("Rsen"+coordinationSentries.getid()
                        );
410                     entryCriterion.setAttributeNode(entryCriterionv2);
411                 }
412             }
413         }
414     }
415
416     /**
417      * This method write the eventListener
418      * @param doc
419      * @param caseplanmodel
420      * @param coordinationEvent
421      */
422     public void writeEvent(Document doc,Element caseplanmodel,
        CoordinatesOfContours coordinationEvent){
423         Element eventListener = doc.createElement("cmmn:eventListener");
```

```
424         caseplanmodel.appendChild(eventListener);
425
426         // set attribute
427         Attr cmmneventListenerv1 = doc.createAttribute("id");
428         cmmneventListenerv1.setValue("e" + coordinationEvent.getid());
429         eventListener.setAttributeNode(cmmneventListenerv1);
430     }
431
432     /**
433      * This method write the Task
434      * @param doc
435      * @param caseplanmodel
436      * @param recognizesquare
437      */
438
439     public void writeTask(Document doc,Element caseplanmodel,
        CoordinatesOfContours recognizesquare){
440         Element cmmntask = doc.createElement("cmmn:task");
441         caseplanmodel.appendChild(cmmntask);
442         // set attribute
443         Attr cmmntaskv1 = doc.createAttribute("isBlocking");
444         cmmntaskv1.setValue("true");
445         cmmntask.setAttributeNode(cmmntaskv1);
446
447         Attr cmmntaskv2 = doc.createAttribute("id");
448         cmmntaskv2.setValue("t" + recognizesquare.getid());
449         cmmntask.setAttributeNode(cmmntaskv2);
450     }
451
452     /**
453      * This method write the Sentry as well as the connection
        to connector
454      * @param doc
455      * @param caseplanmodel
456      * @param coordinationSentries
457      * @param coordinatesOfSquare
458      * @param line
459      */
460     public void writeSentry(Document doc,Element caseplanmodel,
```

```
CoordinatesOfContours coordinationSentries,CoordinatesOfContours
coordinatesOfSquare,CoordinatesOfLines line){
461
462     System.err.println("taskkkkkk"+ coordinatesOfSquare);
463     System.err.println("sentryyyyyyy"+coordinationSentries);
464     System.err.println("lineeeeeeee"+ line);
465     Element cmmnsentry = doc.createElement("cmmn:sentry");
466     caseplanmodel.appendChild(cmmnsentry);
467
468     if(coordinationSentries!=null){
469
470         Attr sentryva1 = doc.createAttribute("id");
471         sentryva1.setValue("senR" + coordinationSentries.getid());
472         cmmnsentry.setAttributeNode(sentryva1);
473     }
474
475     Element cmmnplanItemOnPart =
    doc.createElement("cmmn:planItemOnPart");
476     cmmnsentry.appendChild(cmmnplanItemOnPart);
477
478     if(coordinatesOfSquare!= null){
479         Attr planItemOnPart = doc.createAttribute("sourceRef");
480         planItemOnPart.setValue("pi" +coordinatesOfSquare.getid());
481         cmmnplanItemOnPart.setAttributeNode(planItemOnPart);
482     }
483
484     if(line!= null){
485         Attr planItemOnPart1 = doc.createAttribute("id");
486         planItemOnPart1.setValue("line"+line.getid());
487         cmmnplanItemOnPart.setAttributeNode(planItemOnPart1);
488     }
489
490     Element cmmnstandardEvent =
    doc.createElement("cmmn:standardEvent");
491     cmmnplanItemOnPart.appendChild(cmmnstandardEvent);
492     // set attribute
493     Attr standardEvent= doc.createAttribute("complete");
494     cmmnstandardEvent.setAttributeNode(standardEvent);
495
```

```
496         Element cmmnifpart = doc.createElement("cmmn:ifPart");
497         cmmnsentry.appendChild(cmmnifpart);
498         // set attribute
499         Attr ifpartv1 = doc.createAttribute("id");
500         ifpartv1.setValue("ifpa"+coordinationSentries.getid());
501         cmmnifpart.setAttributeNode(ifpartv1);
502
503     }
504
505     /**
506      * This method writes Task specifications
507      * @param doc
508      * @param CMMNDiagram
509      * @param coordinatesOfSquare
510      */
511     public void writetaskValues(Document doc, Element CMMNDiagram,
        CoordinatesOfContours coordinatesOfSquare) {
512         Element CMMNShape2 = doc.createElement("cmmndi:CMMNShape");
513         CMMNDiagram.appendChild(CMMNShape2);
514
515         // set attribute
516         Attr Shape2v1 = doc.createAttribute("cmmnElementRef");
51          Shape2v1.setValue("pi"+coordinatesOfSquare.getid() );
518         CMMNShape2.setAttributeNode(Shape2v1);
519
520         Attr Shape2v2 = doc.createAttribute("id");
521         Shape2v2.setValue("sh"+ coordinatesOfSquare.getid() );
522         CMMNShape2.setAttributeNode(Shape2v2);
523
5.4         Element dcBounds1  = doc.createElement("dc:Bounds");
525         CMMNShape2.appendChild(dcBounds1);
526
527         Attr bound2v1 = doc.createAttribute("height");
528         bound2v1.setValue(Double.toString(coordinatesOfSquare.getHeight()));
529         dcBounds1.setAttributeNode(bound2v1);
530
531         Attr bound2v2 = doc.createAttribute("width");
532         bound2v2.setValue(Double.toString(coordinatesOfSquare.getWidth()));
533         dcBounds1.setAttributeNode(bound2v2);
```

```
534
535         Attr bound2v3 = doc.createAttribute("x");
536         bound2v3.setValue(Double.toString(coordinatesOfSquare.getX()));
537         dcBounds1.setAttributeNode(bound2v3);
538
539         Attr bound2v4 = doc.createAttribute("y");
540         bound2v4.setValue(Double.toString(coordinatesOfSquare.getY()));
541         dcBounds1.setAttributeNode(bound2v4);
542
543         Element cmmndiCMMNLabel2 =
            doc.createElement("cmmndi:CMMNLabel");
544         CMMNShape2.appendChild(cmmndiCMMNLabel2);
545
546     }
547
548     /**
549      * This method writes Event specifications
550      * @param doc
551      * @param CMMNDiagram
552      * @param coordinationOfEvent
553      */
554     public void writeEventValues(Document doc, Element CMMNDiagram,
        CoordinatesOfContours coordinationOfEvent) {
555         Element CMMNShape2 = doc.createElement("cmmndi:CMMNShape");
556         CMMNDiagram.appendChild(CMMNShape2);
557
558         // set attribute
559         Attr Shape2v1 = doc.createAttribute("cmmnElementRef");
560         Shape2v1.setValue("pi"+coordinationOfEvent.getid() );
561         CMMNShape2.setAttributeNode(Shape2v1);
562
563         Attr Shape2v2 = doc.createAttribute("id");
564         Shape2v2.setValue("sh"+ coordinationOfEvent.getid() );
565         CMMNShape2.setAttributeNode(Shape2v2);
566
567         Element dcBounds1 = doc.createElement("dc:Bounds");
568         CMMNShape2.appendChild(dcBounds1);
569
570         Attr bound2v1 = doc.createAttribute("height");
```

```
571         bound2v1.setValue(Double.toString(coordinationOfEvent.getHeight()));
572         dcBounds1.setAttributeNode(bound2v1);
573
574         Attr bound2v2 = doc.createAttribute("width");
575         bound2v2.setValue(Double.toString(coordinationOfEvent.getWidth()));
576         dcBounds1.setAttributeNode(bound2v2);
577
578         Attr bound2v3 = doc.createAttribute("x");
579         bound2v3.setValue(Double.toString(coordinationOfEvent.getX()));
580         dcBounds1.setAttributeNode(bound2v3);
581
582         Attr bound2v4 = doc.createAttribute("y");
583         bound2v4.setValue(Double.toString(coordinationOfEvent.getY()));
584         dcBounds1.setAttributeNode(bound2v4);
585
586         Element cmmndiCMMNLabel2 =
            doc.createElement("cmmndi:CMMNLabel");
587         CMMNShape2.appendChild(cmmndiCMMNLabel2);
588
589     }
590
591     /**
592      * This method writes sentry specifications
593      * @param doc
594      * @param CMMNDiagram
595      * @param coordinatesOfSentries
596      */
597     public void writeEntryCriterionValus(Document doc, Element
        CMMNDiagram, CoordinatesOfContours coordinatesOfSentries){
598         Element CMMNShape2 = doc.createElement("cmmndi:CMMNShape");
599         CMMNDiagram.appendChild(CMMNShape2);
600
601         // set attribute
602         Attr Shape2v1 = doc.createAttribute("cmmnElementRef");
603         Shape2v1.setValue("Rsen"+coordinatesOfSentries.getid() );
604         CMMNShape2.setAttributeNode(Shape2v1);
605
606         Attr Shape2v2 = doc.createAttribute("id");
607         Shape2v2.setValue("sh"+ coordinatesOfSentries.getid() );
```

```
608         CMMNShape2.setAttributeNode(Shape2v2);
609
610         Element dcBounds1  = doc.createElement("dc:Bounds");
611         CMMNShape2.appendChild(dcBounds1);
612
613         Attr bound2v1 = doc.createAttribute("height");
614
            bound2v1.setValue(Double.toString(coordinatesOfSentries.getHeight()))
            ;
615         dcBounds1.setAttributeNode(bound2v1);
616
617         Attr bound2v2 = doc.createAttribute("width");
618         bound2v2.setValue(Double.toString(coordinatesOfSentries.getWidth()));
619         dcBounds1.setAttributeNode(bound2v2);
620
621         Attr bound2v3 = doc.createAttribute("x");
622         bound2v3.setValue(Double.toString(coordinatesOfSentries.getX()));
623         dcBounds1.setAttributeNode(bound2v3);
624
625         Attr bound2v4 = doc.createAttribute("y");
626         bound2v4.setValue(Double.toString(coordinatesOfSentries.getY()));
627         dcBounds1.setAttributeNode(bound2v4);
628
629         Element cmmndiCMMNLabel2  =
            doc.createElement("cmmndi:CMMNLabel");
630         CMMNShape2.appendChild(cmmndiCMMNLabel2);
631
632     }
633
634     /**
635      * This method writes Line specifications
636      * @param doc
637      * @param CMMNDiagram
638      * @param line
639      * @param coordinatesOfSentries
640      */
641     public void writeLineValus(Document doc, Element CMMNDiagram,
        CoordinatesOfLines line,CoordinatesOfContours coordinatesOfSentries){
642
```

```
643         System.out.println("show me lines: " + line);
644         Element CMMNShape2 = doc.createElement("cmmndi:CMMNEdge");
645         CMMNDiagram.appendChild(CMMNShape2);
646
647         // set attribute
648         Attr Shape2v1 = doc.createAttribute("cmmnElementRef");
649         if(line!=null){
650             Shape2v1.setValue("line"+line.getid());
651             CMMNShape2.setAttributeNode(Shape2v1);
652
653             Attr Shape2v2 = doc.createAttribute("id");
654             Shape2v2.setValue("li"+ line.getid() );
655             CMMNShape2.setAttributeNode(Shape2v2);
656
657             Attr Shape2v3 = doc.createAttribute("targetCMMNElementRef");
658             Shape2v3.setValue("Rsen"+coordinatesOfSentries.getid() );
659             CMMNShape2.setAttributeNode(Shape2v3);
660
661             Attr Shape2v4 = doc.createAttribute("isStandardEventVisible");
662             Shape2v4.setValue("true" );
663             CMMNShape2.setAttributeNode(Shape2v4);
664
665             Element diwaypoint  = doc.createElement("di:waypoint");
666             CMMNShape2.appendChild(diwaypoint);
667
668             Attr bound2v1 = doc.createAttribute("x");
669             bound2v1.setValue(Double.toString(line.getX1()+8));
670             diwaypoint.setAttributeNode(bound2v1);
671
672             Attr bound2v2 = doc.createAttribute("y");
673             bound2v2.setValue(Double.toString(line.getY1()));
674             diwaypoint.setAttributeNode(bound2v2);
675
676             Element diwaypoint2  = doc.createElement("di:waypoint");
677             CMMNShape2.appendChild(diwaypoint2);
678
679             Attr bound2v3 = doc.createAttribute("x");
680             bound2v3.setValue(Double.toString(line.getX2()+8));
681             diwaypoint2.setAttributeNode(bound2v3);
```

```
682
683             Attr bound2v4 = doc.createAttribute("y");
684             bound2v4.setValue(Double.toString(line.getY2()));
685             diwaypoint2.setAttributeNode(bound2v4);
686
687         }
688         Element cmmndiCMMNLabel2 =
            doc.createElement("cmmndi:CMMNLabel");
689         CMMNShape2.appendChild(cmmndiCMMNLabel2);
690     }
691
692     /**
693      * This method writes File specifications
694      * @param doc
695      * @param CMMNDiagram
696      * @param coordinatesOfFile
697      */
698     public void writeFileValues(Document doc, Element CMMNDiagram,
        CoordinatesOfContours coordinatesOfFile){
699         Element CMMNShape2 = doc.createElement("cmmndi:CMMNShape");
700         CMMNDiagram.appendChild(CMMNShape2);
701
702         // set attribute
703         Attr Shape2v1 = doc.createAttribute("cmmnElementRef");
704         Shape2v1.setValue("fv"+coordinatesOfFile.getid());
705         CMMNShape2.setAttributeNode(Shape2v1);
706
707         Attr Shape2v2 = doc.createAttribute("id");
708         Shape2v2.setValue("sf"+ coordinatesOfFile.getid());
709         CMMNShape2.setAttributeNode(Shape2v2);
710
711         Element dcBounds1 = doc.createElement("dc:Bounds");
712         CMMNShape2.appendChild(dcBounds1);
713
714         Attr bound2v1 = doc.createAttribute("height");
715         bound2v1.setValue(Double.toString(coordinatesOfFile.getHeight()));
716         dcBounds1.setAttributeNode(bound2v1);
717
718         Attr bound2v2 = doc.createAttribute("width");
```

```
719             bound2v2.setValue(Double.toString(coordinatesOfFile.getWidth()));
720             dcBounds1.setAttributeNode(bound2v2);
721
722             Attr bound2v3 = doc.createAttribute("x");
723             bound2v3.setValue(Double.toString(coordinatesOfFile.getX()));
724             dcBounds1.setAttributeNode(bound2v3);
725
726             Attr bound2v4 = doc.createAttribute("y");
727             bound2v4.setValue(Double.toString(coordinatesOfFile.getY()));
728             dcBounds1.setAttributeNode(bound2v4);
729
730             Element cmmndiCMMNLabel2 =
                doc.createElement("cmmndi:CMMNLabel");
731             CMMNShape2.appendChild(cmmndiCMMNLabel2);
732
733         }
734     }
735
736
737
```

BIBLIOGRAPHIE

Arica, N. et Yarman-Vural, F.T. (2001). An overview of character recognition focused on off-line handwriting. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 31*(2), 216-233.

Bailey, B.P. et Konstan, J.A. (2003). Are informal tools better?: comparing DEMAIS, pencil and paper, and authorware for early multimedia design. Proceedings of the SIGCHI conference on human factors in computing systems (p. 313-320). : ACM

Bailey, B.P., Konstan, J.A. et Carlis, J.V. (2001). DEMAIS: designing multimedia applications with interactive storyboards. Proceedings of the ninth ACM international conference on Multimedia (p. 241-250). : ACM

Bishop, C.M. (1995). *Neural networks for pattern recognition.* : Oxford university press.

Bradski, G. et Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library.* : " O'Reilly Media, Inc.".

Calhoun, C., Stahovich, T.F., Kurtoglu, T. et Kara, L.B. (2002). Recognizing multi-stroke symbols. AAAI Spring Symposium on Sketch Understanding (p. 15-23).

Chakraborty, A., Baowaly, M.K., Arefin, A. et Bahar, A.N. (2012). The role of requirement engineering in software development life cycle. *Journal of emerging trends in computing and information sciences, 3*(5), 723-729.

Chen, C.P. et Xie, S. (1996). Freehand drawing system using a fuzzy logic concept. *Computer-Aided Design, 28*(2), 77-89.

Chen, G. et Kégl, B. (2010). Invariant pattern recognition using contourlets and AdaBoost. *pattern recognition, 43*(3), 579-583.

Chen, Q., Grundy, J. et Hosking, J. (2008). SUMLOW: early design-stage sketching of UML diagrams on an E-whiteboard. *Softw., Pract. Exper., 38*(9), 961-994.

Cheriet, M. et Suen, C.Y. (1993). Extraction of key letters for cursive script recognition. *Pattern Recognition Letters, 14*(12), 1009-1017.

Coyette, A., Schimke, S., Vanderdonckt, J. et Vielhauer, C. (2007). Trainable sketch recognizer for graphical user interface design. Proceedings of the 11th IFIP TC 13 international conference on Human-computer interaction (p. 124-135). Rio de Janeiro, Brazil : Springer-Verlag

Coyette, A. et Vanderdonckt, J. (2005). A sketching tool for designing anyuser, anyplatform, anywhere user interfaces. *Human-Computer Interaction-INTERACT 2005*, 550-564.

de Carvalho, R.M., Mili, H., Boubaker, A., Gonzalez-Huerta, J. et Ringuette, S. (2016). On the analysis of CMMN expressiveness: revisiting workflow patterns. Enterprise Distributed Object Computing Workshop (EDOCW), 2016 IEEE 20th International (p. 1-8). : IEEE

Duyne, D.K.V., Landay, J. et Hong, J.I. (2002). *The design of sites: patterns, principles, and processes for crafting a customer-centered Web experience.* : Addison-Wesley Longman Publishing Co., Inc.

Freund, Y. et Mason, L. (1999). The alternating decision tree learning algorithm. icml (p. 124-133).

Garain, U. et Chaudhuri, B.B. (2004). Recognition of online handwritten mathematical expressions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 34*(6), 2366-2376.

George, A. et Gafoor, F. (2014). Contourlet Transform Based Feature Extraction For Handwritten Malayalam Character Recognition Using Neural Network. *International Journal of Industrial Electronics and Electrical Engineering, 2*(4).

Hammond, T. et Davis, R. (2006a). LADDER: A language to describe drawing, display, and editing in sketch recognition. ACM SIGGRAPH 2006 Courses (p. 27). : ACM

Hammond, T. et Davis, R. (2006b). Tahuti: A geometrical sketch recognition system for uml class diagrams. ACM SIGGRAPH 2006 Courses (p. 25). : ACM

Hammond, T.A. (2007). *Ladder: A perceptually-based language to simplify sketch recognition user interface development.* Massachusetts Institute of Technology.

Kpalma, K. et Ronsin, J. (2007). *An overview of advances of pattern recognition systems in computer vision* : Advanced Robotic Systems.

Lam, L., Lee, S.-W. et Suen, C.Y. (1992). Thinning methodologies-a comprehensive survey. *IEEE Transactions on pattern analysis and machine intelligence, 14*(9), 869-885.

Landay, J.A. et Myers, B.A. (2001). Sketching interfaces: Toward more human interface design. *Computer, 34*(3), 56-64.

Larose, D.T. (2005). k-Nearest Neighbor Algorithm. *Discovering Knowledge in Data: An Introduction to Data Mining*, 90-106.

Lin, J., Newman, M.W., Hong, J.I. et Landay, J.A. (2002). Denim: An informal sketch-based tool for early stage web design. Proceedings of the 2002 AAAI Spring Symposium-Sketch Understanding (p. 148-149).

Liu, W. (2003). On-line graphics recognition: State-of-the-art. International Workshop on Graphics Recognition (p. 291-304). : Springer

Lladós, J., Valveny, E., Sánchez, G. et Martí, E. (2001). Symbol recognition: Current advances and perspectives. International Workshop on Graphics Recognition (p. 104-128). : Springer

Majumdar, A. (2007). Bangla basic character recognition using digital curvelet transform. *Journal of Pattern Recognition Research, 2*(1), 17-26.

Mamatha, H., Sucharitha, S. et Murthy, K.S. (2013). Handwritten Kannada Numeral Recognition based on the Curvelets and Standard Deviation. *International Journal of Signal Processing Systems, 1*(1), 74-78.

Marin, M., Hull, R. et Vaculín, R. (2012). Data centric bpm and the emerging case management standard: A short survey. International Conference on Business Process Management (p. 24-30). : Springer

Marquardt, D.W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics, 11*(2), 431-441.

Michalski, R.S., Carbonell, J.G. et Mitchell, T.M. (2013). *Machine learning: An artificial intelligence approach*. : Springer Science & Business Media.

Mori, S., Yamamoto, K. et Yasuda, M. (1984). Research on machine recognition of handprinted characters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*(4), 386-405.

Motwani, M.C., Gadiya, M.C., Motwani, R.C. et Harris, F.C. (2004). Survey of image denoising techniques. Proceedings of GSPX (p. 27-30).

Mup. Récupéré le April 22,2017 de http://www.mup.co.il/OpenCV/

Nemmour, H. et Chibani, Y. (2011). Handwritten Arabic word recognition based on Ridgelet transform and support vector machines. High Performance Computing and Simulation (HPCS), 2011 International Conference on (p. 357-361). : IEEE

Newman, M.W., Lin, J., Hong, J.I. et Landay, J.A. (2003). DENIM: An informal web site design tool inspired by observations of practice. *Human-Computer Interaction, 18*(3), 259-324.

OMG. *Case Management Model and Notation, version 1.0*. May 2014 de http://www.omg.org/spec/CMMN/1.0/PDF

OpenCV. Récupéré le April 22, 2017 de http://opencv.org

OpenCV documentation. (2013). Récupéré le April 22, 2017 de http://docs.opencv.org/2.4/doc/tutorials/tutorials.html

Ossher, H., Andr, #233, Hoek, v.d., Storey, M.-A., Grundy, J., Bellamy, R. et Petre, M. (2011). Workshop on flexible modeling tools (FlexiTools 2011). Proceedings of the 33rd International Conference on Software Engineering (p. 1192-1193). Waikiki, Honolulu, HI, USA : ACM

Ossher, H., Bellamy, R., Simmonds, I., Amid, D., Anaby-Tavor, A., Callery, M., Desmond, M., de Vries, J., Fisher, A. et Krasikov, S. (2010). Flexible modeling tools for pre-requirements analysis: conceptual architecture and research challenges. *ACM Sigplan Notices, 45*(10), 848-864.

Plamondon, R. et Srihari, S.N. (2000). Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on pattern analysis and machine intelligence, 22*(1), 63-84.

Plimmer, B. et Apperley, M. (2003a). Software to sketch interface designs. Ninth International Conference on Human-Computer Interaction (p. 73-80).

Plimmer, B. et Apperley, M. (2003b). *Software to sketch interface designs*. (Ninth International Conference on Human-Computer Interaction).

Qiu, L. (2007). Sketchuml: The design of a sketch-based tool for uml class diagrams. *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications*, 986-994.

Rao, D. et Panduranga, P.P. (2006). A survey on image enhancement techniques: classical spatial filter, neural network, cellular neural network, and fuzzy filter. Industrial Technology, 2006. ICIT 2006. IEEE International Conference on (p. 2821-2826). : IEEE

RS, S.N. et Afseena, S. (2015). Handwritten Character Recognition–A Review. *International Journal of Scientific and Research Publications*.

Rubine, D. (1991). *Specifying gestures by example*. (Vol. 25) : ACM.

Sezgin, M. (2004). Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic imaging, 13*(1), 146-168.

Signavio. Récupéré le 4/17/2017 de https://editor.signavio.com/userguide/en/modeling_and_notations/cmmn/editing_cmmn.html

Soisalon-Soininen, E. (2011). *Online Sketch Recognition: Geometric Shapes*. Aalto University.

Suen, C.Y., Berthod, M. et Mori, S. (1980). Automatic recognition of handprinted characters—the state of the art. *Proceedings of the IEEE, 68*(4), 469-487.

Szeliski, R. (2010). *Computer vision: algorithms and applications*. : Springer Science & Business Media.

Trisotech. Récupéré le 11/7/2016 2016 de http://www.trisotech.com/digital-enterprise-suite

Umbaugh, S.E. (1997). *Computer vision and image processing: A practical approach using CViptools with Cdrom*. : Prentice Hall PTR.

Vector vs. Raster Graphics. de http://www.abetdisc.com/answers/cd-cover-design/vector-vs-bitmap-or-raster-graphics/

Vinciarelli, A. (2002). A survey on off-line cursive word recognition. *Pattern recognition, 35*(7), 1433-1446.

Wenyin, L., Jin, X. et Sun, Z. (2001). Sketch-based user interface for inputting graphic objects on small screen devices. International Workshop on Graphics Recognition (p. 67-80). : Springer

Wüest, D., Seyff, N. et Glinz, M. (2012). Flexible, lightweight requirements modeling with Flexisketch. Requirements Engineering Conference (RE), 2012 20th IEEE International (p. 323-324). : IEEE

Xiangyu, J., Wenyin, L., Jianyong, S. et Sun, Z. (2002). On-line graphics recognition. Computer Graphics and Applications, 2002. Proceedings. 10th Pacific Conference on (p. 256-264). : IEEE

Yu, B. et Cai, S. (2003). A domain-independent system for sketch recognition. Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia (p. 141-146). : ACM