UNIVERSITÉ DU QUÉBEC À MONTRÉAL

CMMN MODELS HAND-DRAWN SKETCHES RECOGNITION SYSTEM

MASTER THESIS

PRESENTED

AS A PARTIAL REQUIREMENT

FOR THE MASTER IN COMPUTER SCIENCE

BY

SARA AMIRSARDARI

OCTOBER 2017

UNIVERSITÉ DU QUÉBEC À MONTRÉAL

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

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

SARA AMIRSARDARI

OCTOBRE 2017

ACKNOWLEDGMENTS

It gives me great pleasure in expressing my gratitude to all those people who have supported me and had their contributions in making this thesis possible.

First and foremost, I would like to thank my advisor, Professor Hafedh Mili, for his constant guidance, support, motivation, inspiration, enthusiasm, and immense knowledge.

I would also like to acknowledge Renata Carvalho, as the second reader of this thesis, and I am gratefully indebted to her for her very valuable comments on this thesis.

I am indebted to my friends and colleagues for providing a stimulating environment in which I could learn and grow, especially I thank my friends, Imen, Anis, Amani and Golrokh.

I would like to thank all the staff members of the Computer Science department at UQAM for their direct and indirect helps during my studies at UQAM.

Last but not least, I would like to express my very profound gratitude to my parents, Mohammad and Akram, and to Marco for providing me with unfailing support for their love, encouragement, advice throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

RÉSUMÉ

La nature des premières activités de spécification des exigences nécessite une approche de modélisation plus flexible par rapport à celle fournie par des 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 pour 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, modèlent CMMN, outils de modélisation CMMN.

ABSTRACT

The nature of early requirements activities requires a more flexible approach to modeling than is provided with traditional modeling tools. A variety of modeling tools exist to capture of business process models in a structured form. Despite their advantages, such tools are unnatural for the human user, and are not used in early stages of modeling and development. In comparison, more flexible approaches such as office tools or whiteboards are more common in the early stage of system modeling, as well as more natural for the human user. However, these informal tools give the user flexibility and freedom, at the expense of consistency management, change management and model interchange.

Because neither flexible approaches nor traditional modeling tools are ideal, in this thesis we propose a new intermediate approach in order to reduce the gap between these two approaches. We propose a tool that can recognize early hand sketches of CMMN models and transform them into a format that can be imported into a formal tool. In our approach, we use shape recognition and pattern matching to recognize freehand drawings of CMMN models and translate them into formal CMMN models. Next, these formal models are serialized into an XML file that is compliant with the CMMN model interchange format and can then be imported into CMMN-compliant tools. The effectiveness of our approach has been tested on more than 500 drawings. The results confirmed the effectiveness of our approach.

Keywords: pre-requirement analysis, formal modeling, free-form approaches, hand-drawn sketch, image processing, computer vision, CMMN models, CMMN modeler.

INTRODUCTION

Prior to the invention of computers, paper and pencil were considered the tools that provided the primary support of different activities. Simple sheets of paper or whiteboards were among early means of documenting ideas. These days, with vast technological improvements in the field of computer science, there is more of a tendency to use digital devices instead of the traditional paper or white boards. The invention of computers and their accessories, such as the mouse, keyboard, and monitor, has introduced a very suitable alternative for primary tools. Paper and pencil-based tools were, and still are, used in the pre-requirements step of the software development life cycle.

Most activities during the software development life cycle involve a process that ensures that good software is built. Requirements engineering is referred to as an essential phase in this process (Chakraborty *et al.*, 2012). Before getting into this critical phase, some form of business analysis that is called pre-requirements is executed in order to determine whether new development is required (Ossher *et al.*, 2010). «In this phase, before requirements are formulated, a business analyst needs to collect information, organizing it to achieve insight, envisioning alternative futures and presenting insights and recommendations to stakeholders» (Ossher *et al.*, 2010).

Hence, business analysts in this stage need to use a simple way to interact with the stakeholders in order to explain the structure, policies, problems, needs, and opportunities for improvement at all levels of a project, while stakeholders are interviewed in order to discover their needs and requirements.

However, currently, with the increasing popularity of "touchscreen technology", all kinds of users, including business analysts, are usually provided with the option of entering the information directly through the screen using their fingers or a pen rather than using a mouse. They can also do things like move things on the screen and scroll them, and make them bigger or smaller. Therefore, by having a tablet, and using office tools such as word processors, and drawing or presentation tools, analysts during the early stages of requirements engineering could interact with stakeholders much more easily. In regards to this interaction, it is not static, because the needs and requests are changeable, and so analysts and stakeholders might edit or delete unnecessary parts, as well as create or extend necessary items.

0.1 Problem Statement

The more flexible approaches such as paper and pencil, whiteboards, or office tools known as "**free-form**" approaches (Ossher *et al.*, 2011) are used in the early stage of system modeling, in order to have freehand drawings or writings (sketches). According to (Coyette *et al.*, 2007) free-form approaches have a variety of advantages, such as:

- The sketcher is free to use as an approach at any stage of design, without the need to follow a certain chain of steps or framework (Newman *et al.*, 2003);
- The sketcher does not need a training course in order to draw or write sketches, and the result can be produced quickly (Duyne *et al.*, 2002);
- This approach allows the sketcher to concentrate on basic structural issues, rather than trivial details (e.g., exact alignment, typography and colors) (Landay et Myers, 2001);
- This approach encourages creativity, and lets the sketcher to bring their ideas on the paper without any limitation (Landay et Myers, 2001), and

- The collaborative execution of sketches between business analyst and stakeholders, allow them to evolve designs while discussing and taking turns between sketching and annotating designs (Plimmer et Apperley, 2003b).

Despite these strengths, free-form approaches have weaknesses as well. Using these approaches, the structured form of the documented information would not be available and the opportunity for changes and post-processing could be limited. Thus, time wasting and overpriced remodeling of early sketches is necessary to make further modifications possible (Wüest *et al.*, 2012).

Opposing free-form approaches, it is the "**formal modeling tool**" which uses business process model in a structured form (Ossher *et al.*, 2011). Formal modeling is more unnatural for humans and more understandable for devices. They have a variety of advantages, such as:

- «Support multiple view on the same model for visualization and convenience of manipulation» (Ossher *et al.*, 2010);
- «Facilitate consistency management of the model» (Ossher *et al.*, 2010);
- «Provide domain-specific assistance (e.g., "content assist") based on model structure» (Ossher *et al.*, 2010);
- Prepare documentation of the modeling decisions (e.g., rationales) (Ossher *et al.*, 2011);
- «Provide syntax, semantic model and semantic mapping» (Ossher *et al.*, 2010), and
- «Integration with other formal tools and processes, such as model-driven engineering (MDE) and model checking» (Ossher *et al.*, 2010).

Despite these advantages, formal modeling tools are not efficient at early stages of modeling and development. In comparison, it is more common to use informal mechanisms such as free-form approaches (Ossher *et al.*, 2011).

(Ossher *et al.*, 2010) argued that the input they received from many practitioners clearly indicated that neither informal modeling nor formal modeling is ideal. In this context, we believe that a new class of tools is required to reduce the gap between the two approaches. Such intermediate approaches should be able to handle sketches that are produced at early stages of modeling, and transfer such informal sketches into formal models. This would allow business analysts to migrate easily to a later stage with a formal tool.

0.2 Objective

The purpose of this thesis is to develop a tool that migrates hand-drawn sketches of CMMN (OMG) models to formal models that can be imported into a CMMN modeling tool. CMMN is an OMG standard for representing case models, i.e., models of business processes that involve a lot of knowledge intensive tasks, such as a medical diagnosis, or a law case. For this project, we will generate models for Trisotech's CMMN Modeler (Trisotech).

To this end, we will need to:

- Define a method for accepting the totally unstructured and unclear input, including freehand drawing of CMMN models;
- Develop a technique for matching the inputs with the default patterns of CMMN models;
- Build the recognized model fragments into a CMMN model that can be serialized into the XML interchange format for CMMN, to support the exchange of the model, and its importation into CMMN-compliant tools.

0.3 Methodology

The research methodology relies on the literature on image processing, pattern recognition, template matching, and a semantic description of hand-drawn shapes.

After evaluating several architectures, we decided to base this study on the freehand sketches that can be recognized and converted into the user's regular intended shapes. Hence, the approach evaluates the raw sequence of points as an input according to default patterns of CMMN models and determines whether the points signify some more organized input. Moreover, sketch recognition as part of the image processing and computer vision is covering a lot of detail calculations. Thus, extending, developing and proposing a general approach based on OpenCV library and Template Matching rules for converting user's input to CMMN models into a CMMN modeling tool is defined. At the end, simulation tests to assess the maximum usability of the application are done.

0.4 Thesis Plan

The second chapter of this thesis introduces the essential concepts of online and offline shape recognition, and the methods that each approach should follow for investigating the data input. In addition, the advantages and disadvantages of the low fidelity prototyping tools and high fidelity prototyping tools are compared. The third chapter describes the CMMN models and their structures. The approach which comprises the foundation of this study is recognizing and converting hand-drawn sketches into CMMN modeling tool that is described in the fourth chapter. The fifth chapter evaluates the application and finalizes the thesis with the conclusions and the implications for the future research.

CHAPTER I

STATE OF THE ART ON SKETCH RECOGNITION SYSTEMS

Pattern recognition is a branch of machine learning (Michalski *et al.*, 2013) that emphasizes the recognition of data patterns and data regularities in order to classify them into a number of categories or classes (Kpalma et Ronsin, 2007). It is composed of a collection of mathematical, statistical, heuristic, and inductive techniques of the fundamental role in order to find the actual problems through mathematical methods (Liu *et al.*, 2006). The development of pattern recognition is increasing very fast. Nowadays, pattern recognition is a wide research area that can impact a wide range of disciplines such as engineering, mathematics, art, and medicine. Optical character recognition systems represent one of the most successful applications of technology in the field of pattern recognition and artificial intelligence. The main purpose of these applications is classifying the input pattern in a specific class (Kpalma et Ronsin, 2007). Hence, this chapter focuses on the essential concepts of representing image data that is composed of vector graphics and raster graphics. In the following, the pattern recognition approaches that can be separated into two sections, online recognition and offline recognition, are explained. To complete the discussion, the methods and similarities and differences between the two are described.

1.1 Vector Graphics and Raster Graphics

Computer graphics are pictures and movies created using computers. In this field, there are two substantial ways of representing image data (see figure 1.1): raster (bitmap) graphics and vector graphics (Umbaugh, 1997). Raster graphics (Umbaugh, 1997) are defined by pixels. These pixels are non-scalable, and each of these tiny square dots represents a color. Thus, to make an image, these dots combine into pattern and raster graphics programs define which pixel will be which color, and what the dimensions of the image should be. In raster graphics, the definition of resolution is the number of pixels contained within a file that is often referred to as DPI (dots per inch). Therefore, raster graphics are dependent on resolution. By contrast, vector graphics (Umbaugh, 1997) are defined by a series of mathematical equations for specifying lines, curves, and shapes, as well as the editable attributes such as line's direction, thickness, and color. Vector files do not need to account for each pixel. Hence vectors can be scalable to any arbitrary size and they are independent of the resolution. Also, they require much less memory compared to raster graphics.

Consequently, the vector graphics are more advantageous than raster graphics. The attempt is to vectorize input data for more efficient storage and easier handling to analyse and process geometric shapes.

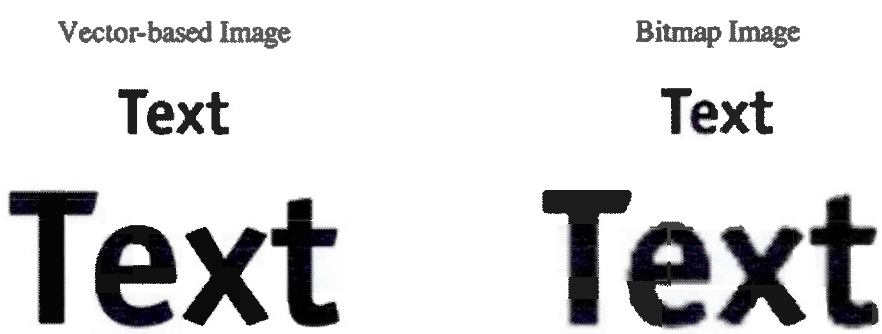


Figure 1.1 Represents vector and raster graphics (Vector vs. Raster Graphics)

1.2 Optical Character Recognition

With the development of digital computers, a wide range of applications such as banking, security, postal processing, and language identification with the methodologies of OCR (Optical Character Recognition) systems are exposed.

OCR methodology is based on the recognition of printed documents or handwritten by using computers (RS et Afseena, 2015). Hence, this technology uses a digital camera or a scanner to record and store different types of documents such as paper documents or character images, and then translates all of these documents into machine editable formats like ASCII code (RS et Afseena, 2015). Therefore, the storage space required for documents is reduced, and the speed of data recovery is increased. For instance, OCR can be used in various fields like banking, where they must deal with vast amounts of paper. With OCR, it could be processed without human intervention.

«OCR can be classified into two categories based on text type and acquisition of documents» (RS et Afseena, 2015) (see figure 1.2). OCR is composed of two types on the basis of text type: HCR (Handwritten Character Recognition) and PCR (Printed Character Recognition) (RS et Afseena, 2015). HCR recognizes handwritten input such as paper documents, and PCR recognizes printed documents (RS et Afseena, 2015). In the second layer, HCR as a subdivision of OCR is divided into offline and online recognition systems based on acquisition of documents, which can include overall stages such as pre-processing, segmentation, feature extraction, and classification (RS et Afseena, 2015).

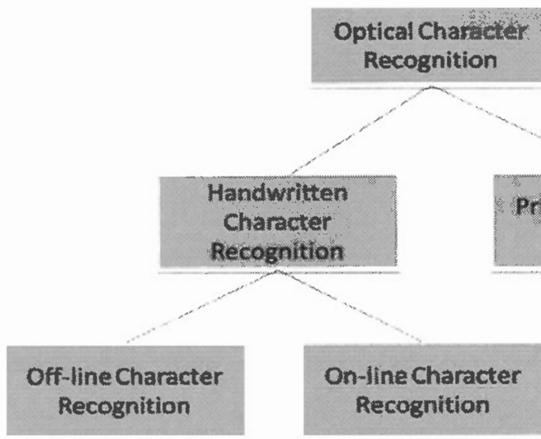


Figure 1.2 Classification of optical character recognition (RS et Afseena, 2015)

1.2.1 Offline Recognition

There has been a lot of research in the field of offline character recognition (Arica et Yarman-Vural, 2001; Mori *et al.*, 1984; Plamondon et Srihari, 2000; RS et Afseena, 2015; Suen *et al.*, 1980; Vinciarelli, 2002). In this approach, a digital image that is usually obtained by scanning or by photographing is taken as an input for recognition (figure 1.3). Hence, in offline recognition, before getting into the character recognition phase, several imperfect and costly pre-processing steps have to be executed. The purpose of pre-processing steps is to exclude irrelevant information, and include relevant information in the input (RS et Afseena, 2015).

The first step of pre-processing is thresholding, which is composed of several techniques (Sezgin, 2004). Thresholding is used to distinguish objects from the background of the image, and to convert a grayscale image into a binary black and white image (RS et Afseena, 2015; Sezgin, 2004).

In the second step, in order to improve the recognition performance, some sort of noise removal must be used to extract the foreground textual matter from, for instance, textured background by removing interfering strokes, impulse noise, Gaussian noise, speckle noise, and photon noise (Cheriet et Suen, 1993; Plamondon et Srihari, 2000; RS et Afseena, 2015). Image denoising has various other applications and has been discussed in these papers (Motwani *et al.*, 2004; Rao et Panduranga, 2006).

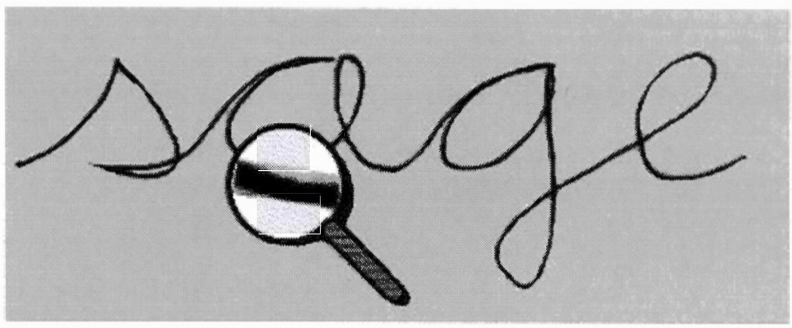


Figure 1.3 Offline recognition.The image of the word is converted into gray-level pixels using a scanner (Plamondon et Srihari, 2000).

In the second step, in order to improve the recognition performance, some sort of noise removal must be used to extract the foreground textual matter from, for instance, textured background by removing interfering strokes, impulse noise, Gaussian noise, speckle noise, and photon noise (Cheriet et Suen, 1993; Plamondon et Srihari, 2000; RS et Afseena, 2015). Image denoising has various other applications and has been discussed in these papers (Motwani *et al.*, 2004; Rao et Panduranga, 2006).

The last step of pre-processing is using the black and white image as the input in order to utilize the thinning process on it. This process reduces patterns to thin-line representations. The aim of the thin-line technique is keeping the geometrical and topological properties of the image intact, and this makes it appropriate for analysis in the next phases (Lam *et al.*, 1992). As shown in figure 1.4, the steps involved in preprocessing are displayed.

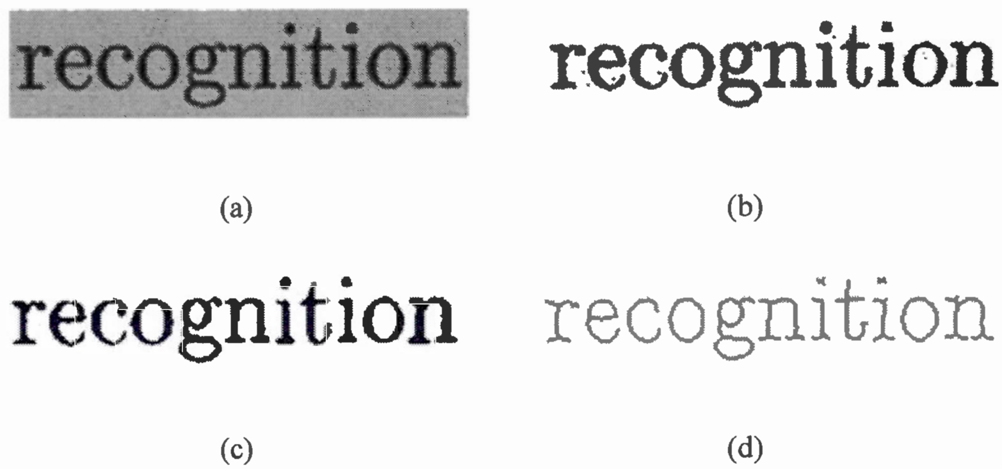


Figure 1.4 Pre-processing steps of offline recognition.: (a) Scanned raw input image; (b) Thresholded black and white with noise; (c) Denoised image; (d) Thinned image (Soisalon-Soininen, 2011)

The preprocessing steps explained above are common to all offline recognition problems, and can be used in characters or graphics issues. The second phase of offline recognition, which includes segmentation, is the process of converting input images into text. It does so in three steps: line segmentation, word segmentation, and character segmentation (RS et Afseena, 2015). Hence, this step separates sentences from text and then divides words and letters of sentences subsequently (Suen *et al.*, 1980).

The last phase, the feature extraction, which extracts most relevant features, mainly depends on the application (RS et Afseena, 2015). Different feature extraction methods are explained in the literature (Chen et Kégl, 2010; George et Gafoor, 2014; Majumdar, 2007; Mamatha *et al.*, 2013; Nemmour et Chibani, 2011). These methods describe how the features of the segmented character are extracted and, according to these features, each character is assigned to one of the specified classes such as the upper and lower case letters, the ten digits, and special symbols (Plamondon et Srihari, 2000).

Because there is a strong relation between character and shape recognition (Arica et Yarman-Vural, 2001; Lladós *et al.*, 2001), it is useful to study the basic approaches, methods, and applications related to offline recognition. For example, the recognition task of mathematical formula involves two tasks: symbol recognition, and two-dimensional structure interpretation (Garain et Chaudhuri, 2004). Although the approaches in online and offline recognition methods are different, understanding the challenges in offline recognition leads us to discover and use the benefits of online methodologies (Arica et Yarman-Vural, 2001).

1.2.2 Online Recognition

In online systems, the character or shape recognition process is executed while the user is writing or drawing (Plamondon et Srihari, 2000). Hence, a freehand stroke is captured by computer mouse, or a finger on a touchscreen device, based on the mouse or finger's movement (figure 1.5). Furthermore, a freehand stroke drawn by human user is usually very cursive, inaccurate, and contains imperfections. For example, supposedly straight lines will be drawn as arcs, and circles will be drawn as ellipses with irregular shapes and significant noise.

(Liu, 2003) proposed an approach for online recognition that presents a general overview to the user in order to specify the general problems of online graphics recognitions, and find the solutions for converting the sequence of coordinate points into the user intended input. This approach (see figure 1.7) consists of three steps: primitive shape recognition, composite graphic object recognition and document recognition and understanding. To describe the *primitive shape recognition* (see figure 1.6), four sub steps were defined that include (a) stroke curve pre-processing, (b) shape classification (c) shape fitting and (d) shape regularization.

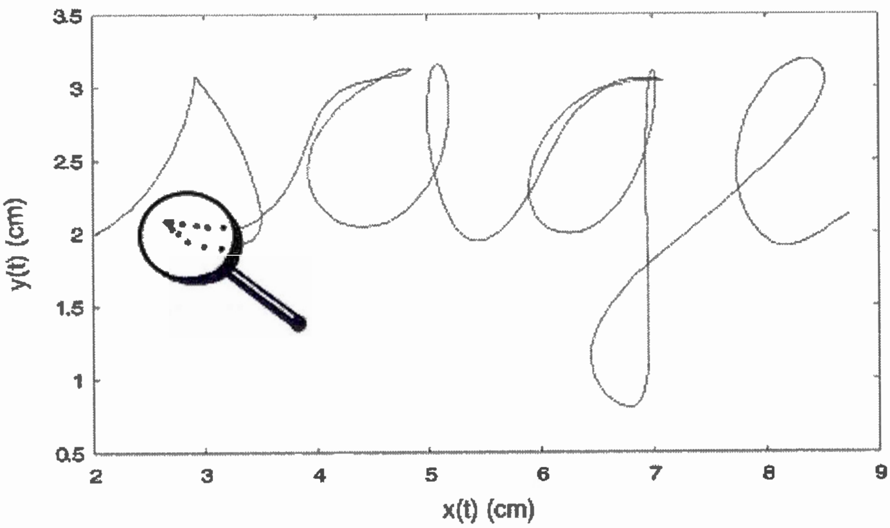


Figure 1.5 Online recognition. Similar data as (figure 1.3) presents as point trajectory data. The x, y coordinate is recorded as a function of time with a digitizer (Plamondon et Srihari, 2000).

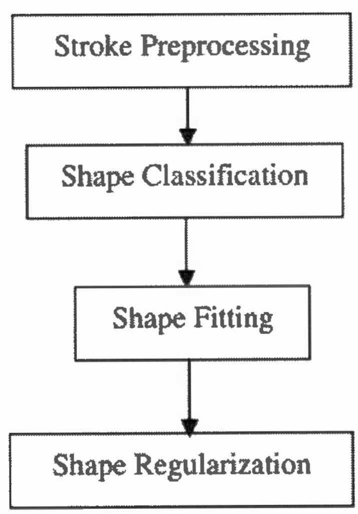


Figure 1.6 Primitive shape recognition steps (Xiangyu *et al.*, 2002)

While the user is drawing a freehand stroke, the concern of *primitive shape recognition* is to determine the type and parameters of the primitive shape, which can be a line, a triangle, a rectangle, an ellipse, etc. (Liu, 2003). After recognizing and converting the current stroke, it is possible to combine the current stroke (recognized primitive shape) together with previously recognized primitive shapes, based on their spatial relationships. This is the *composite graphic object recognition* step (Liu, 2003). *Document recognition and understanding* is the last step of online recognition. After recognizing and converting the graphical elements (primitive shapes and composite graphic objects), we need to understand the connections and relationship among the elements, as well as their semantics (Liu, 2003). In the following section we explain more about the subset of primitive shape recognition.

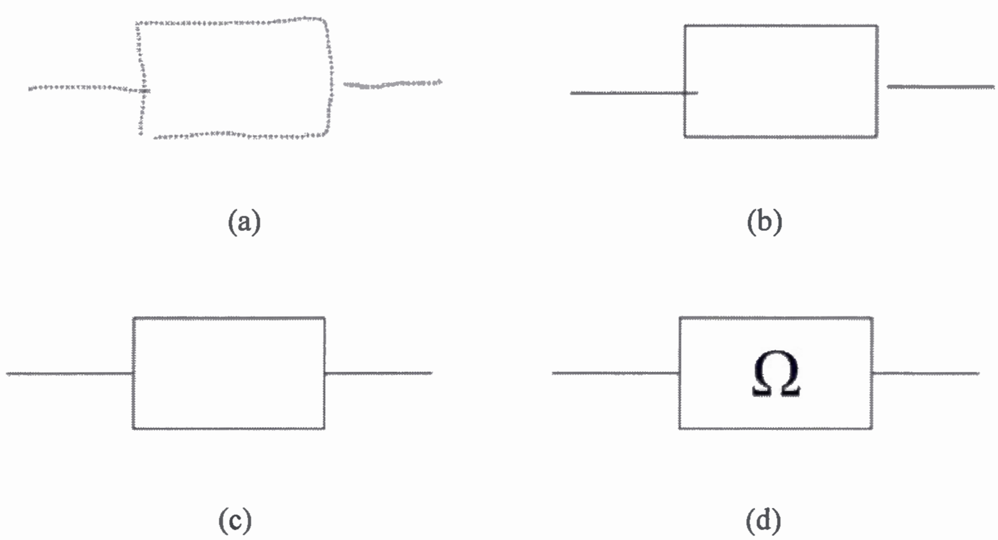


Figure 1.7 Subprocesses of online graphics recognition: (a) Raw input strokes; (b) Strokes recognized as primitive shapes; (c) Primitive shapes combined into a composite object according to their spatial relationship; (d) The semantic of the composite object interpreted using context information (Soisalon-Soininen, 2011).

1.2.2.1 Pre-Processing

Pre-processing in online recognition has the same role as in offline recognition. It aims to remove the noise and minor mistakes in order to make the strokes more similar to the user's intention. Preprocessing can be divided into four steps: *polygonal approximation*, *agglomerate points filtering*, *end points refinement*, and *convex hull calculation* (Liu, 2003; Wenyin *et al.*, 2001). The *convex hull calculation* is used individually for recognizing closed shapes and thus will not be discussed here.

(Xiangyu *et al.*, 2002) explain that, generally, the input hardware produces a lot more points than are necessary to define the shape of the stroke. By removing these points from the chain, the sketchy line will be approximately displayed by a polyline with much fewer critical vertices. Therefore, by determining the extent to which this is controlled by parameter ε, the polyline can maintain the original shape (figure 1.8). «If the distance of a point to the straight-line segment formed by connecting its neighbors is smaller than ε, this point is non-critical and should be removed from the polyline» (Xiangyu *et al.*, 2002). Hence, on the one hand more vertices are left by getting smaller the ε value, on the other hand the edge accuracy is improved. This process is referred to as *polygonal approximation.*

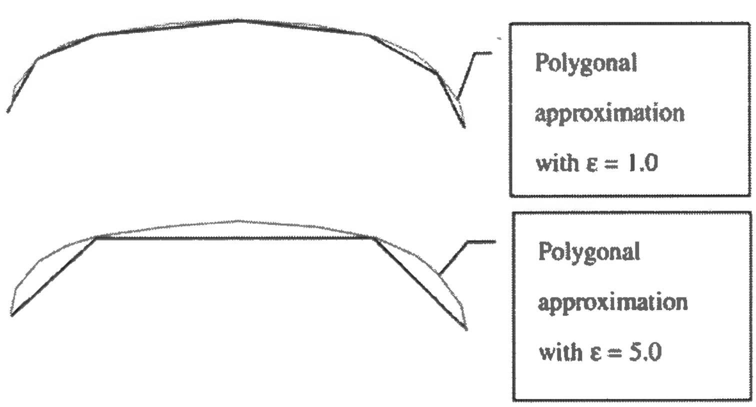


Figure 1.8 Illustrations of the polygonal approximation process(Xiangyu *et al.*, 2002).

Using a pen or digitizer might produce a hooklet or circlet (see figure 1.9) at the end of the stroke which usually has much higher point agglomerations than the average value of the whole polyline.

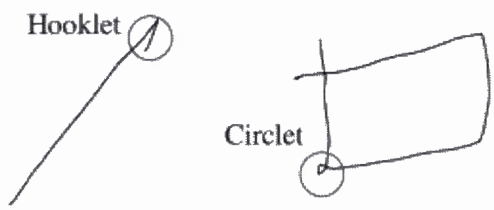
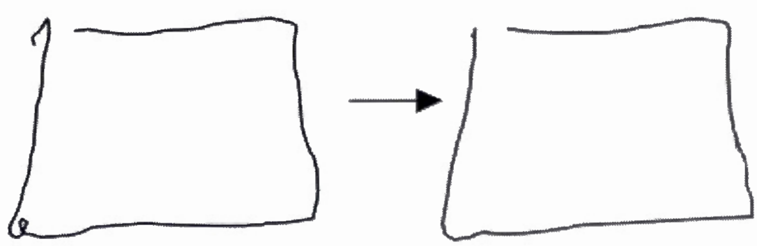


Figure 1.9 Agglomerate pointsas a hooklet and a circlet (Xiangyu *et al.*, 2002).

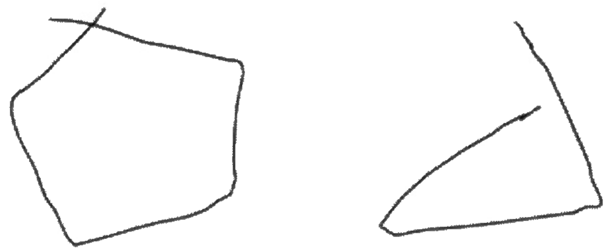
The task of *agglomerate points filtering* (see figure 1.10) examines the point agglomerations of the input polyline. By finding these segments, it starts removing these noises and uses fewer points to represent the segment.



(a) (b)

Figure 1.10 Agglomerate points filtering: (a) the sketchy line before processing; (b) the sketchy line after processing (Xiangyu *et al.*, 2002).

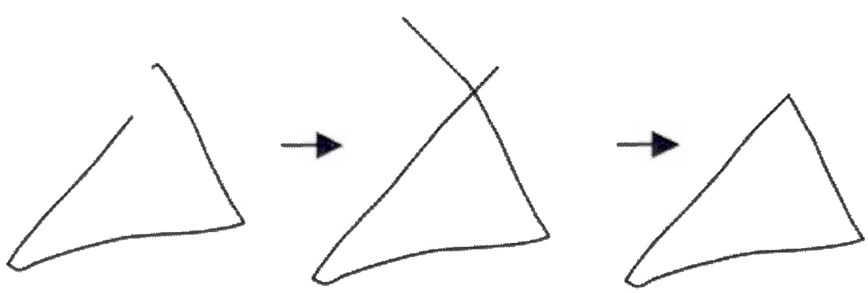
Another "noise" introduced by hand drawing is the case where the stroke that is painted by the user is intended to be a polygon, but ends up as a non-closed form, or a form with a cross near its end points (see figure 1.11).



(a) (b)

Figure 1.11 Shapes with improper endpoints(a) A pentagon with a cross; (b) an un-closed triangle (Xiangyu *et al.*, 2002).

Hence, these improper endpoints bring enough barriers for both shape classification and regularization. As a result, end point refinement can be used to close an open stroke by extending its endpoints along its end directions (see figure 1.12).



(a) (b) (c)

Figure 1.12 Examples of endpoint refinement: (a) The sketchy line before processing; (b) After pulling the end points; (c) After deleting extra points (Xiangyu *et al.*, 2002).

1.2.2.2 Shape Classification

After pre-processing, shape classification is the most essential part of shape recognition. It is used to decide whether an unclear stroke drawn by a user can represent a predefined shape such as a square, a circle, and arrow, etc. Moreover, in the field of shape recognition, the main focus of the academic papers is shape classification (Liu, 2003).

In general, the purpose of shape classification is to extract beneficial information from the input data to facilitate the classification. The extent of classification methods due to the amount of detail in the process and the ability of combining different parts from different approaches is abundant.

(Lladós *et al.*, 2001) proposed the traditional categorization of pattern recognition in the context of symbol recognition methods. Such methods can be divided into *statistical* and *structural* methods. «In statistical pattern recognition, each pattern is represented as an n-dimensional feature vector extracted from the image» (Lladós *et al.*, 2001). Therefore, this classification is conducted by partitioning the feature space into different classes. Thus, we need to focus on the selection of the features, as well as the selection of the method in order to partition the feature space.

The properties of the patterns that need to be classified can affect the selection of the features space. «The main criteria must be to minimize the distance among patterns belonging to the same class and to maximize the distance among patterns belonging to different classes» (Lladós *et al.*, 2001). Once a set of features has been chosen, we use classification to partition the feature space and assign each feature vector to one of the predefined classes. Three of the most common selection methods are: *k-nearest neighbours*, *decision tree* and *neural networks* (Lladós *et al.*, 2001).

The *k-nearest neighbors* method (Larose, 2005; Lladós *et al.*, 2001) is based on a similarity measure. We need to define a distance function among feature vectors in order to assign each input image to the class with the closest representative (Lladós *et al.*, 2001). Once all representatives are defined for each input pattern and each class, the set of the k-closest representatives is built, and the pattern will be assigned to the class that has the most representatives in this set (Lladós *et al.*, 2001).

The *decision tree* (Freund et Mason, 1999; Lladós *et al.*, 2001) method constructs using simple decision rules. In this method, specific conditions about the value of a particular feature are tested in each node (Freund et Mason, 1999; Lladós *et al.*, 2001). Classification will be executed by tracing the branches in the tree based on the result of condition testing (Freund et Mason, 1999; Lladós *et al.*, 2001). As a result, one of the tree leaves corresponds to the recognized symbols is reached.

The *neural networks* (Bishop, 1995; Lladós *et al.*, 2001) solves problems in the same function of the human brain. Hence, the learning ability is one of the advantages of this approach. Based on this ability, neural networks have obtained good classification rates in many different domains. This advantage enables them to adapt themselves according to the properties of the training set (Bishop, 1995; Lladós *et al.*, 2001). By learning automatically, the neural network optimizes its parameters in order to recognize the symbols in the training set (Bishop, 1995; Lladós *et al.*, 2001).

In the *structural* approach (Lladós *et al.*, 2001), a description of the shapes is used as a reference. Hence, this method uses a set of geometric primitives, and their relationships to represent symbols (Lladós *et al.*, 2001). Straight lines and arcs are the primitives used to describe the shape of the symbols, as well other geometric primitives such as loops, contours, or simple shapes (circles, rectangles, etc.) (Lladós *et al.*, 2001). For example, a diamond can be represented as a set of four lines with certain constraints (Lladós *et al.*, 2001). Therefore, a model of an ideal shape is built for each symbol by using these primitives (Lladós *et al.*, 2001). As a result, an input image is classified based on the best match between the representation of the image and the model of the symbol (Lladós *et al.*, 2001).

1.2.2.3 Shape Fitting and Regularization

Shape fitting and regularization is less complicated than shape classification. After classifying the sketch, the fitting process is employed to investigate whether a shape from a class is similar to the sketch. Hence, the aim is to use different methods to find the parameters of the fitted shape that has the lowest average distance to the sketch (Xiangyu *et al.*, 2002). Several approaches have been proposed to do this, are explained below.

Chen and Xie, proposed an approach which finds the fitting shape based on an analysis of the drawn curve with reference models (Chen et Xie, 1996). Their technique uses fuzzy filtering rules in order to recognize the correct model and to eliminate the undesirable points. There are different fitting methods for lines, circles, circular arcs, ellipses and elliptical arcs.

For generating a straight line from a sequence of points, they attempt to find a line with the best approximates of the scattered data. Thus, the method of least-squares (Marquardt, 1963) is used to generate it. Furthermore, three non-collinear points can be used to determine whether an arc is a circle or a circular arc. Thus, for representing the circle, fuzzy information is needed to obtain a weighted average of centers and radius by choosing all possible triplets of points in the freehand drawing. For ellipse fitting, the liming multiplier technique is proposed. More methods for ellipse and polygonal fitting are suggested by Liu (Liu, 2003; Wenyin *et al.*, 2001; Xiangyu *et al.*, 2002).

Liu discussed a set of shape regularization rules that are described in more detail in his work (Xiangyu *et al.*, 2002). These rules attempt to correct the defects of the drawing of a sketcher. They suggest a process that includes of two sub-processes: *inner-shape regularization* and *inter-shape regularization*.

The *inner-shape regularization* (figure 1.13) consists of making modifications to the fitted shapes according to several rectifications (Xiangyu *et al.*, 2002). The first rectification adjusts the edges or the A-axis and B-axis of polygons in order to equalize their lengths. The second rectification adjusts the edges of polygons to make them parallel. The third rectification connects the inner angles of polygons to lean towards regular figures such as rectangles. Furthermore, horizontal/vertical rectification as the last rectification in inner shape regularization is able to rectify the edges of a polygon, or the axes of an ellipses or diagonals of a diamond to horizontal or vertical.

(a) (b)

Figure 1.13 Inner shape regularization (a) rectifying a triangle into an isosceles triangle; (b) rectifying two rectangles into the same size (Xiangyu *et al.*, 2002).

The *inter-shape regularization* (figure 1.14) introduces a group of rectifications including size, position, and critical points (Xiangyu *et al.*, 2002). Size rectification adjusts a group of adjacent primitive shapes that have the same type or approximately of the same size. The edges of two adjacent polygons that are nearly on the same horizontal/vertical line are adjusted by position rectification. At the end, the critical points rectification is applied to rectify the centers of ellipses, the vertexes of polygons, and the mid points of edges.

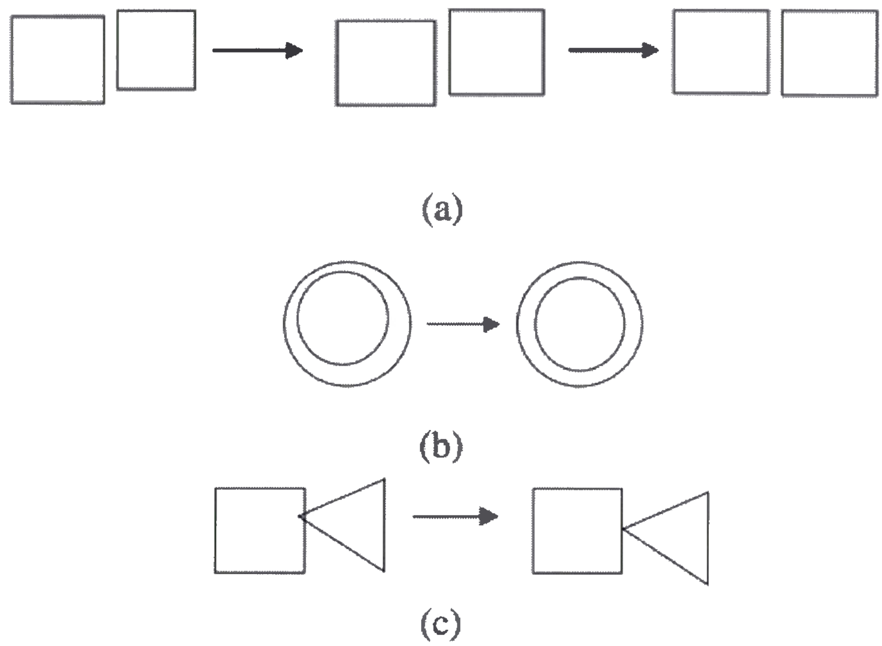


Figure 1.14 Inter-shape regularization: (a) Scale the two adjacent squares to have the same size; (b) Shift the two adjacent circles to have the same position; (c) Shift the triangle and the square, the leftmost vertex of the triangle is at the center of the rightmost edge of the square (Xiangyu *et al.*, 2002).

1.3 Sketch Recognition Systems

Coyette and Vanderdonckt proposed three categories for UI prototypes that were defined according to their degree of fidelity, which refers the accuracy that allows them to present the reality of the UI (Coyette et Vanderdonckt, 2005). The first UI prototype tool is the high-fidelity (Hi-Fi) that shows the final result inapplicable. It can be considered as a high fidelity tool which supports building a UI that almost complete and usable (Coyette et Vanderdonckt, 2005). Hence, this kind of UI prototype includes editing functions such as undo, erase, and move can build a complete GUI (Graphical User Interface) for designers (Coyette et Vanderdonckt, 2005).

The medium-fidelity (Me-Fi) approach falls between high-fidelity and low-fidelity prototypes in order to present the major functions and details (Coyette et Vanderdonckt, 2005). This approach maintains the information relevant to color schemes, typography or other minor details (Coyette et Vanderdonckt, 2005). The low-fidelity (Lo-Fi) approach is used to keep the general information to obtain a general understanding of what is desired (Coyette et Vanderdonckt, 2005). Hand sketch on paper is well known as one of the most effective ways to represent the first drafts of a future UI (Bailey et Konstan, 2003; Coyette et Vanderdonckt, 2005; Landay et Myers, 2001; Newman *et al.*, 2003). This unlimited approach known as a low-fidelity UI prototype has many advantages. For example, during any design step, sketches can be drawn without any prerequisites (Newman *et al.*, 2003). This method is fast and quick to produce (Duyne *et al.*, 2002). Hence, instead of confusing the user with unessential details, it lets the sketcher focus on essential structural issues (Landay et Myers, 2001). As another advantage, not only can it encourage creativity (Landay et Myers, 2001) but it also increases the level of collaboration between designers and end-users (Plimmer et Apperley, 2003a). Furthermore, «creating a low-fidelity UI prototype (such as UI sketches) is at least 10 to 20 times easier and faster than its equivalent with a high-fidelity prototype such as produced in UI builders» (Coyette et Vanderdonckt, 2005; Duyne *et al.*, 2002).

By comparing these UI prototypes, a Lo-Fi prototype has a set of advantages to compare the other prototypes (see a summary of these advantages in table 1.1). By having several screens that have a lot in common, using copy and paste instead of rewriting the whole screen is more profitable. despite, lack of assistance in this approach is palpable (Coyette et Vanderdonckt, 2005). Consequently, by considering the Lo-Fi advantages and combining these approaches, two families of software tools which contain UI sketching with or without code generation will be developed (Coyette et Vanderdonckt, 2005).

This current section is divided in two subsections. Section 1.3.1 describes about sketch-based tools for UML class diagrams, and section 1.3.2 explains the sketch based tools in other domains.

Table 1.1Comparison of software fidelity UI prototyping tools (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 Sketch-Based Tools for UML Class Diagrams

Hammond and Davis proposed Tahuti as a sketch-based tool for UML class diagrams (Hammond et Davis, 2006b). Tahuti uses a multi-layer recognition framework to recognize sketches by their geometrical properties (Hammond et Davis, 2006b). It allows users to sketch object freely in many different ways, instead of requiring the user to draw the object in a pre-defined manner (Hammond et Davis, 2006b). Moreover, Tahuti uses two different views, draw view and interpreted view, in order to display user sketches and the result of the recognition process. Furthermore, the user can switch between the two views.

The multi-layer recognition framework of Tahuti uses a formal language called LADDER, which was created by Hammond and Davis (Hammond et Davis, 2006a; Hammond, 2007). «LADDER is the first sketch description language for user interface developers to describe how sketched diagrams in a domain are drawn, displayed and edited» (Hammond et Davis, 2006a). Hence, the language employed in this framework implements the first prototype system that has the ability to automatically generate a sketch interface for a domain only by considering domain description (Hammond et Davis, 2006a).

This framework is divided into three parts: *Domain Description, Translation, and Domain Independent Sketch Recognition System* (Hammond, 2007). Figure 1.15 shows an overview of such a framework.

According to the figure 1.15, a number of predefined shapes, constraints, display methods, and editing behaviors are supplied in domain description. Moreover this domain consists of a display section and an editing section. «A display section specifies what should be displayed on the screen when the shape is recognized. Editing section specifies how the shape can be edited. Common editing commands involve movement and deletion of the shape» (Hammond et Davis, 2006a). Following the domain description, the translation process is started to parse the shape's definitions and generate code that is needed to recognize shapes, edit triggers, and display the shapes once they are recognized (Hammond et Davis, 2006a). As the last part of this framework, the domain independent sketch recognition System is used. When the stroke is drawn, first of all, the system searches the drawn shapes database in order to check whether the gesture drawn is an editing trigger for any shape or not (Hammond et Davis, 2006a). If the stroke is found, not to be an editing gesture, it must be a drawing gesture.

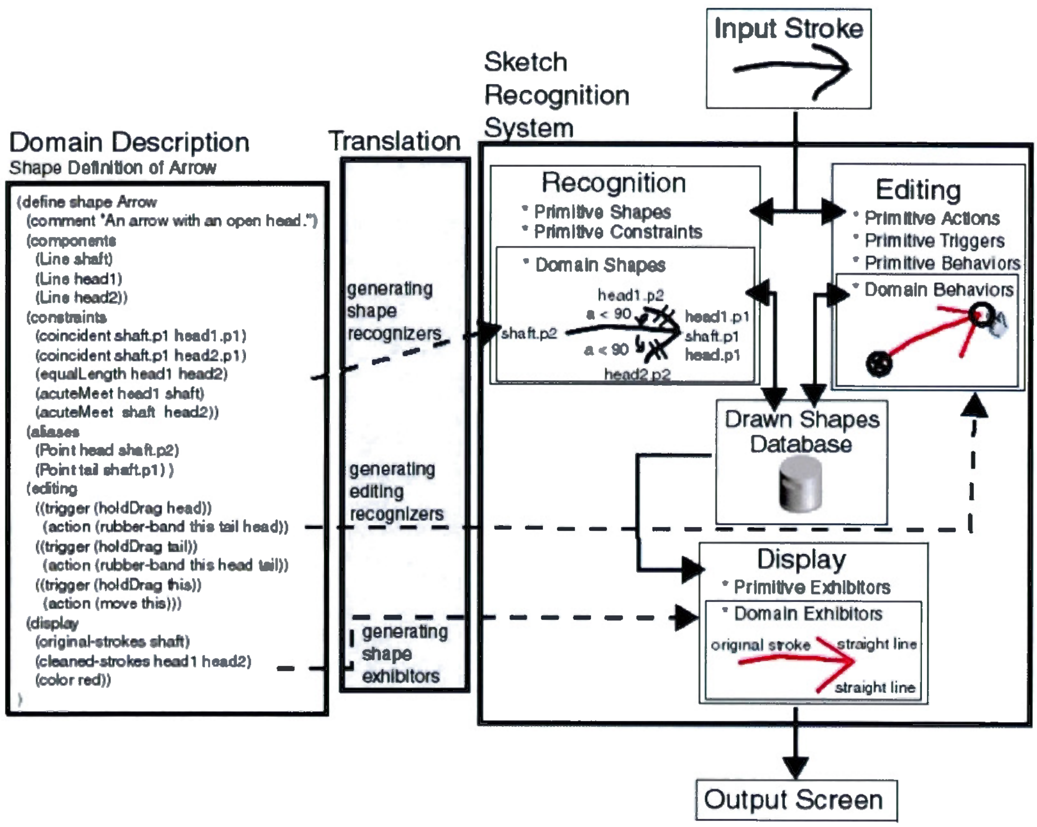


Figure 1.15 LADDER Framework (Hammond, 2007)

Thus, the system preprocesses the stroke into a collection of primitives in order to add them to the drawn shapes database (Hammond et Davis, 2006a). The recognition module tries to build a higher order shape by examining the drawn shapes database (Hammond et Davis, 2006a). At the end, the display module displays the viewable shapes which are defined by the domain description.

Qiu presents SketchUML as a sketch based tool that allows users to sketch UML class diagrams on a computer with editing capabilities (Qiu, 2007). Therefore, it can recognize sketches immediately, and the user does not need to switch to different view to see the recognition results. The tool also supports text recognition and enables users to write text directly inside class objects (Qiu, 2007). SketchUML needs to define sketching flexibility and recognition accuracy. Hence, a geometry-based approach is used to recognize common element in the diagram (Qiu, 2007). Furthermore, it uses a graffiti-based approach to identify special gestures (Qiu, 2007).

Chen and Grundy present SUMLOW as a unified modeling language (UML) diagram tool that uses an E-whiteboard and sketching-based user interface (Chen *et al.*, 2008). This experimental tool proposes sketching-based techniques for early-phase requirements and design modeling. Thus, it allows designers to sketch UML constructs mixing different UML diagram elements, diagram annotations and hand-drawn text (Chen *et al.*, 2008). Their approach has a number of advantages, including keeping a copy of the hand sketches, whereas other sketching based UML design tools convert hand sketches to formal models, and lose the originals. Further, users can use various pen strokes to manipulate sketched diagrammatic elements. Finally, SUMLOW can recognize various UML constructs in sketch view that are hand-drawn, and formalized elements in diagram view.

1.3.2 Sketch-Based Tools in Other Domains

DENIM (Lin *et al.*, 2002; Newman *et al.*, 2003) is one example of a Lo-Fi prototyping tool that allows designers to quickly sketch web pages and present them in different levels of details including a site map, a storyboard, and an individual page. DENIM unified these levels through zooming views. DEMAIS (Bailey *et al.*, 2001) is another Lo-Fi prototyping that bridges the gap in multimedia design and enables a designer to quickly sketch temporal and interactive design ideas. Hence, it is made up of an interactive multimedia storyboard tool that is able to capture about 90 percent of a designer's behavioral design ideas (Bailey et Konstan, 2003). Neither DENIM nor DEMAIS produces any final code or other output. By contrast, the Hi-Fi prototyping tools which are discussed next, support code generation (Coyette et Vanderdonckt, 2005).

Jin et al. propose an approach for pen-based user interfaces that follow a novel and fast shape classification and regularization algorithm for online sketchy graphics recognition (Xiangyu *et al.*, 2002). This recognition process is divided into four stages, including preprocessing, shape classification, shape fitting, and regularization, which explained the previous chapter. The *attraction force model* is used to combine the vertices on the input sketch stroke based on certain threshold (Xiangyu *et al.*, 2002). Thus, the total number of vertices is reduced before the type of shape can be determined. Following this process, the shape is classified as a primitive shape according to a rule-based approach (Xiangyu *et al.*, 2002). Finally, using shape fitting and regularization gradually rectifies the primitive shape into a regular one that fits the user-intended shape (Xiangyu *et al.*, 2002).

Yu and Cai focus on low-level geometric features, instead of working on domain-specific knowledge in order to achieve a domain-independent system for sketch recognition (Yu et Cai, 2003). Their approach follows two steps. First, the stroke is approximated according to one primitive shape or a combination of primitive shapes that can be further used in domain-specific applications. Accordingly, a recursive algorithm segments the stroke using *direction* and *curvature information*. Furthermore, the algorithm uses *geometrical features* and *stroke direction data* to distinguish between graphical primitives (line, arc, circle, and helix).

The second step takes all the recognized primitive shapes as input and analyzes connectivity between them according to following these segmentations (Yu et Cai, 2003): 1) remove false and redundant elements, 2) adjust the size and position of the elements, and 3) adjust their layout to make a match predefined domain independent objects.

Their hierarchical output is presented in three steps, including (a) *the lowest level,* which maintains the original information of stroke (b) *the mid-level,* which stores the vertexes and primitive shapes and (c) *the highest level* , the semantic level composed of the relation tables which consist of recognized primitive shapes and basic objects (Yu et Cai, 2003).

Landay and Myers proposed a system called SILK (Sketching Interfaces Like Krazy) (Landay et Myers, 2001). SILK tries to recognize primitive components that are basic shapes such as rectangles, squiggly lines (representing text), straight lines and ellipses. This approach looks for spatial relationships between new components and other components in the sketch in order to combine the primitive components to make up a UI component such as a button, text field, menu bar, or scrollbar. In addition, SILK recognizes editing gestures such as delete, and grouping or ungrouping objects.

The underlying recognition engine of SILK uses Rubine's gesture recognition algorithm to identify the primitive components (Rubine, 1991). According to this algorithm, each of the primitive components is trained with 15 to 20 examples, and they vary in size, and drawing direction (Landay et Myers, 2001). However, this algorithm has some limitations. For example, the gestures used in Rubine's algorithm are all single strokes, to avoid the segmentation problem of multi-stroke character recognition. Furthermore, it does not handle variations in size and rotation (Landay et Myers, 2001).

Calhoun et al. proposed a recognition system that can recognize symbols composed of multiple strokes (Calhoun *et al.*, 2002). Their approach applies a trainable recognizer in order to describe the definition of the geometric primitives, as well as the geometric relationships between them. Geometric primitives are characterized by intrinsic properties such as line or arc, length, relative length, slope (for lines only), and radius (for arcs only) (Calhoun *et al.*, 2002). Geometric relationships between primitives are built based on certain parameters such as the existence of intersections between primitives, the relative location of intersections, the angle between intersecting lines, and the existence of parallel lines (Calhoun *et al.*, 2002). The approach supports two types of recognition methods. In the first method, each primitive symbol needs to follow the specified definition, which is learned by examining a few examples of the symbol and its geometric relationships. However, this method requires some attention from the drawer (Calhoun *et al.*, 2002). The second method uses a form of best-first search based on a speculative quality metric and pruning (Calhoun *et al.*, 2002).

1.4 Conclusion

We investigated sketch-based tools in different domains, as well as their approaches to recognize hand-drawn sketches. However, these approaches were inspirational for understanding the concept of sketch recognition, despite, the context of this thesis is based on recognizing CMMN hand-drawn sketches as primitive shapes and composite shapes. Hence, we need to implement a new approach to recognize a freehand drawing of CMMN models. Moreover, we verify an approach to transfer these recognized elements into a format that can be imported into a formal modeling tool. Thus, the second issue is to serialize these formal models into an XML file that is compliant with the CMMN model interchange format and can then be imported into CMMN compliant tools. In the next chapters, the meaning of CMMN, as well as the definition of each CMMN model was explained.

CHAPITRE II

CASE MANAGEMENT MODEL AND NOTATION (CMMN)

In regards to case management systems, in order to define, create, and manage business processes, it's necessary to provide a case folder as the primary building, which holds a collection of business documents and other information (Marin *et al.*, 2012). Case management was developed to manage social work and related application areas such as insurance claim processes, healthcare processes, lawsuit services processes, social services process, etc. (Marin *et al.*, 2012).

Case Management Model and Notation (CMMN) is an industry-wide standard by the Object Management Group (OMG) to represent and manage cases within the context of case management. CMMN supports the representation of a wide range of knowledge worker activities, including planning (e.g., business strategic planning initiatives), follow-up (e.g., maintenance and repairs), collaboration (e.g., specification development), record-keeping (e.g., audits), decision-making (e.g., court cases), and problem resolution (e.g., customer service) (Trisotech). Hence, CMMN is used to « document case-oriented business processes to drive process improvement efforts for knowledge work » (Trisotech).

2.1 Notations

In this section, we provide an overview of the CMMN notation in order to model the core constructs of a case.

2.1.1 Case

Knowledge workers, who are experts in their specific field, are able to execute different instances of the model, which is called a case in the case management domain (de Carvalho *et al.*, 2016). For instance, a doctor in the healthcare domain can specify a case that involves caring for a patient, in terms of a medical history and current medical problems (de Carvalho *et al.*, 2016; OMG). In another example, a judge, for the law domain, can define a case involving the application of the law to a subject in a particular situation (de Carvalho *et al.*, 2016; OMG). Generally speaking a case is a top-level concept that combines all the elements that constitute a case model. Hence, « A case is a proceeding that involves actions taken regarding a subject in a particular situation to achieve a desired outcome» (OMG). In the following subsections, we will describe the components of the case modeling.

2.1.2 Case Plan Model

The case plan model (OMG) contains all of the activities for the case. It is composed of all of the elements that represent the initial plan of the case, as well as all elements that support the further evolution of the plan (OMG). A "Folder" shape, as shown in figure 2.1, is considered to display a case plan model. Therefore, the name of the case can be enclosed in the upper left rectangle.

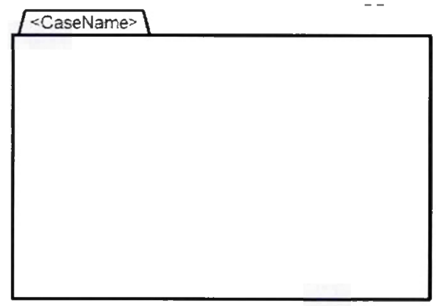


Figure 2.1 Represents Case Plan Model shape

2.1.3 Task

A task, as shown in figure 2.2, is a unit of work, as well as a base class for representing all the work that is done in a case (OMG). Task as a central element in CMMN composed of five different types of task that include: the *non-blocking human task,* the *blocking human task,* the *process task,* the *decision task and* the *case task.* In addition, there is a *discretionary task* that is executed depending on the case manager discretion.

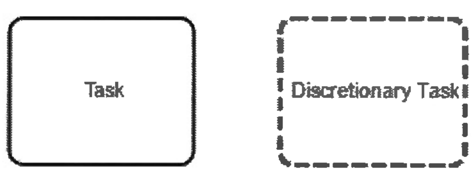


Figure 2.2 Ordinary Task shape and Discretionary Task shape (left to right)

The *non-blocking human* task (OMG), as shown in figure 2.3, is completed at the same moment that it is started. Hence, in the case model, when a *non-blocking human task* starts, we do not wait for it to complete: the sequence flow continues with the next task in the sequence. By contrast, the *blocking human task* (OMG), as shown in figure 2.4, stops the sequence flow until it is completed. All other tasks are by default "blocking".

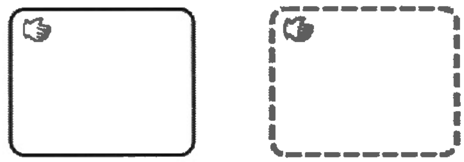


Figure 2.3 Non-Blocking Human Task shapes

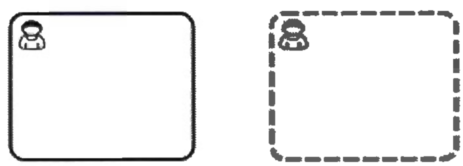


Figure 2.4 Blocking Human Task shapes

The *process task* (OMG), as shown in figure 2.5, is used in order to links to a BPMN diagram. Hence, by clicking the symbol in the upper left corner of the element, a link to a BPMN diagram will be created (Signavio).

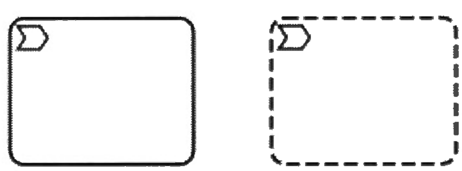


Figure 2.5 Process Task shapes

A *decision task* (OMG), as shown in figure 2.6, is used to repeat a task that consists of a decision represented by a DMN diagram. Hence, by clicking the symbol in the upper left corner of the element, a link to a DMN diagram will be created (Signavio).

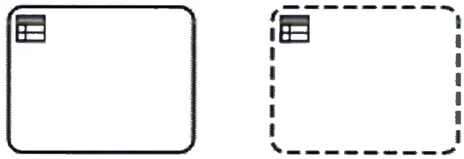


Figure 2.6 Decision Task shape

Finally, the *case Task* (OMG), as shown in figure 2.7, is used to embed an existing case model in another. Therefore, by clicking the symbol in the upper left corner of the case task element, a link to a CMMN diagram will be created (Signavio).

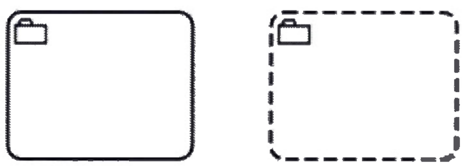
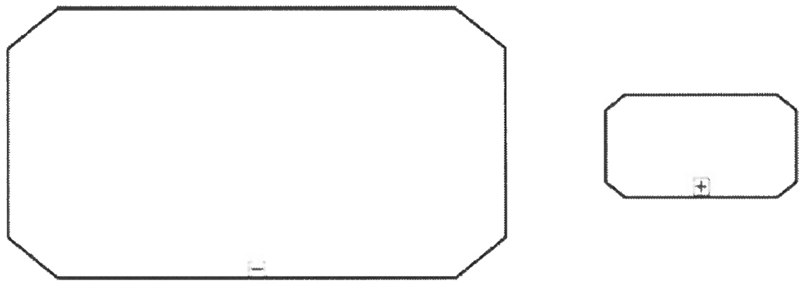


Figure 2.7 Case Task shapes

2.1.4 Stage

A stage (OMG) is defined as a container to visually organize tasks and other CMMN elements. Hence, a certain number of sequence flows, tasks, and sub-stages can be represented by a stage (Signavio). As shown in figure 2.8, stages can be expanded or collapsed and have a - or + on the bottom center. In an *expanded stage*, the elements that constitute it become visible and a *collapsed stage* is linked to another CMMN diagram (Signavio).



(a) (b)

Figure 2.8 Expanded Stage shape and Collapsed Stage shape (left to right)

2.1.5 Event

An event (OMG) waits for specific things to happen in the case. Events typically mark the enabling, activation, and termination of stages and tasks, or the achievement of milestones (OMG). CMMN supports the representation of event listeners, which wait for a particular event to occur. Different types of listener exist (OMG), as shown in figure 2.9. There are *timer event listeners* that wait for a certain amount of time to elapse for a certain predefined point in time. There are also *generic event listeners* that wait for an event to occur. Finally there are *user event listeners* that wait for user input.

Figure 2.9 Event shapes: "Event Listener", "Timer Event Listener" and "User Event Listener" elements (left to right).

2.1.6 Case File

A case file (OMG), as shown in figure 2.10, represents documents that contain information that is used, or produced, by the case. Such documents could include pieces of unstructured or structured information that can be tended from simple to complex sources. The contents of a case file can be defined using any information modeling language (de Carvalho *et al.*, 2016). Case files can be attached to another element using a connector (Signavio).

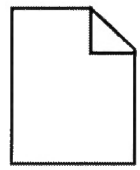


Figure 2.10 Case File shape

2.1.7 Milestone Item

A milestone (OMG), as shown in figure 2.11, represents the state of the case. Therefore, milestones as sub-goals within the case process indicate whether a certain point or stage has been reached or completed (Signavio). A milestone is depicted by a rectangle shape with half-rounded ends. Furthermore, a milestone may have zero or more entry criteria when it is reached.

Figure 2.11 Milestone shape

2.1.8 Sentry

The diamond shaped *Entry Criterion* (OMG) and *Exit Criterion* (OMG) called "sentries," as shown in figure 2.12, specify the main conditions that need to occur to influence the further proceedings of a case (OMG) .These sentries can be attached to tasks, stages, milestones, and case files. Additionally, they don’t even need to be attached to other elements; sentries can stand alone within a sequence flow (Signavio).

A sentry used as an entry criterion is depicted by a shallow "diamond." this indicates that the incoming sequence flow directly attached to the sentry has to be finished before the sequence flow can continue (Signavio). A sentry used as an exit criterion is depicted by a solid "diamond" that presents when a plan item is complete. It implies that the sequence can continue in what direction.

Figure 2.12 Entry Criterion shape and Exit Criterion shape (left to right)

2.1.9 Connector

Connectors (OMG) define relations between CMMN elements, as shown in figure 2.13. A connector object is represented by a dotted line that does not have arrowheads. However, the direction of the flow or association is determined by the presence of a sentry (*entry criterion or exit criterion*) (OMG).

Figure 2.13 Connector Shape

For example, the diagram shown in figure 2.14 illustrates a situation where the entry criterion of Task B depends on the completion of Task A (OMG).

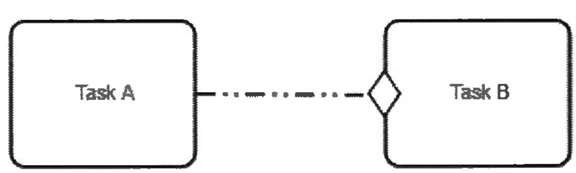


Figure 2.14 Sentry based dependency between two tasks

2.1.9.1 Connector usage

Connectors can be used to visualize dependencies between plan items. For example, the following figure 2.15 shows a situation where Task C can be activated only if Task A and Task B complete (OMG).

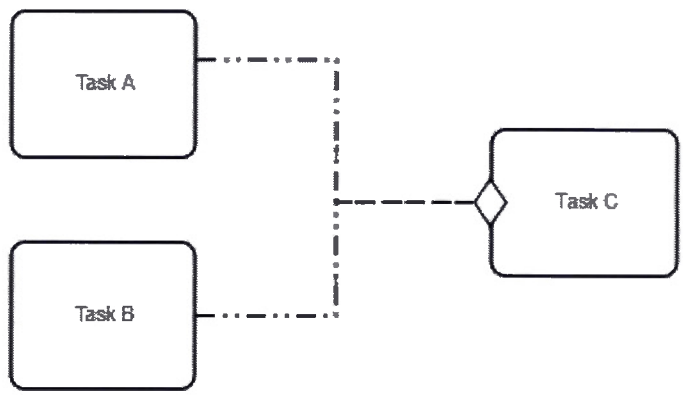


Figure 2.15 Using sentry-based connectors to visualize "AND"

Figure 2.16 illustrates a situation where Task C can be activated if Task A *or* Task B completes (OMG).

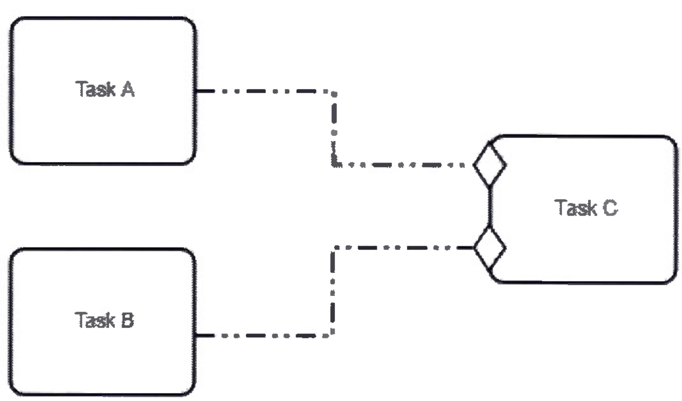


Figure 2.16 Using sentry-based connectors to visualize "OR"

Figure 2.17 shows a situation where Stage B depends on the *exit criterion* of Stage A (OMG).

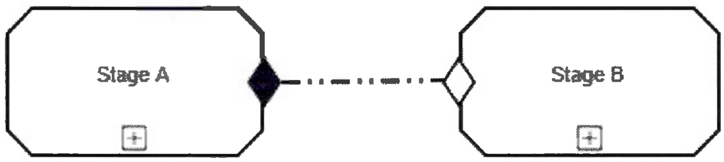


Figure 2.17 Visualize dependency between stages using sentry-based *connector*

Figure 2.18 shows a situation where Task A depends on the achievement of Milestone A (OMG).

Figure 2.18 visualize dependency between a task and a milestone using the sentry-based connector

Figure 2.19 shows a situation where Task A depends on a *timer event listener* receiving a *timer event* (OMG).

Figure 2.19 Visualize dependency between a Task and a Timer Event Listenerusing the Sentry-based connector

Figure 2.20 shows a situation where Task A depends on a Case File Item (OMG).

Figure 2.20 Visualize dependency between a task and a case file itemusing the sentry-based connector

2.2 Example of Case Plan Model

In this section, as shown in figure 2.21, we illustrate the use of the various elements by representing a claim processing example used in the standard.

The case plan model below contains all the activities for representing the case of a CMMN training certificate. This case is organized with two separate stages that include Secretary and CMMN-Trainer. At the beginning, the task of the *user event listener* waits for the new employees who want to start the training. On the one hand, the sequence flow is stopped until the list employees without training will be ready. On the other hand, according to the case manager discretion, the discretionary task checks when potential trainees are free. These two tasks should be complete in order to inform potential trainees and trainer what course is planned. In the following, *milestone* defines the state of the stage of Secretary represented that indicates CMMN training is planned.

In the CMMN-Trainer stage, before starting the training, the *timer event listener* defines a certain amount of time for the duration of the course. In the following, the trainer explains the topics of the training to employees. At the end of the training, the *case file element* that is produced by the case contains a CMMN training certificate. A sentry used as an exit criterion presents when a plan item is complete.

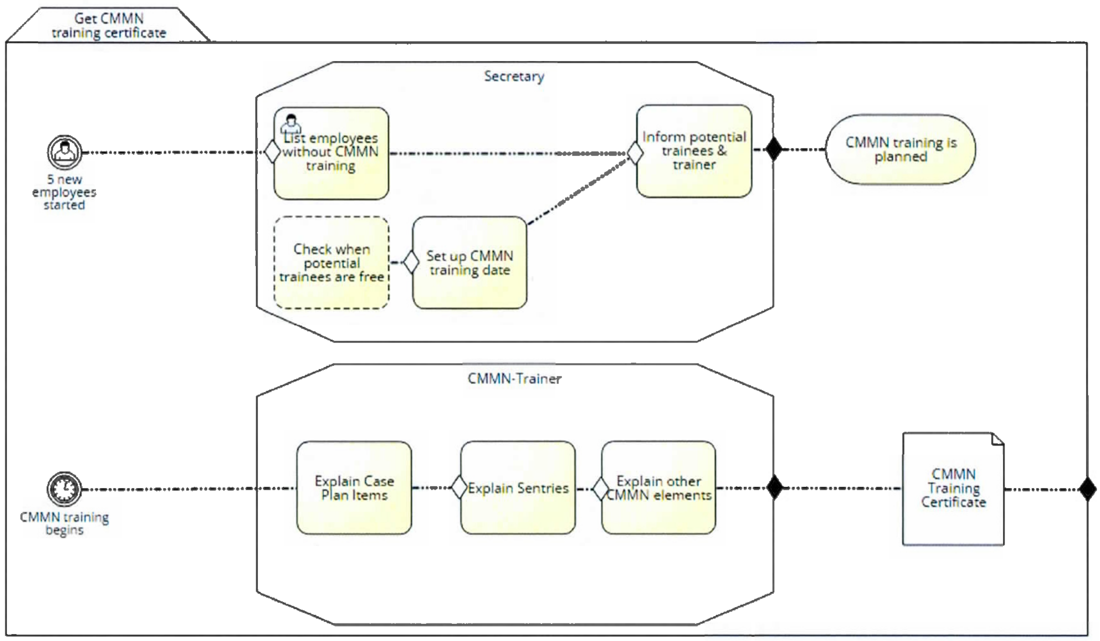


Figure 2.21 combinations of various elements in CMMN (Signavio)

2.3 Conclusion

In this chapter, the concept of CMMN was described. Moreover, the CMMN notations and their meaning in the context of CMMN in order to model the core constructs of a case were explained. In the following, we employed an example of claim processing to illustrate the use of the various CMMN elements. In the next chapter, hand-drawn sketches of CMMN models are recognized using feature extractions and heuristics. We will describe our approach in detail and explain the certain issues in each step of the implementation.

CHAPITRE III

EXPERIMENTAL RESULTS

Sketch recognition domain is an approach that automatically recognizes hand-drawn diagrams with the use of a computer. Hence, research in this domain during recent years has expanded due to the advancements in artificial intelligence and human-computer interaction. Similarly, hand-drawn recognition can be introduced as a subset of the sketch recognition domain. Thus, it can be part of the ability of a computer to receive and interpret handwritten input from sources which can include a paper document, touch-screens, picture, and several other devices. Moreover, different types of recognition algorithms are used, such as gesture-based, appearance-based, geometry-based, or a combination thereof.

In this chapter, we explain the implementation of the tool that investigates different stages of the CMMN elements hand-drawn sketch recognition, and analyzes a set of issues at each stage and describing their solutions in details.

3.1 Case Study

This thesis is planning to deal with recognizing the CMMN elements hand-drawn sketches. Therefore, we need to address two issues: 1) recognizing elementary CMMN constructs within user hand-drawn sketches, and 2) recognizing semantic relationships between them in a way that is consistent with the semantics of CMMN. Figure 3.1 shows on the left hand side what an input might look like. On the right hand side, we show the corresponding CMMN model, as visualized by a CMMN modeling tool.

In other words, we need to create a prototype of a case model from a hand-drawn sketch in order to export it into a business process management or case management system for case automation.

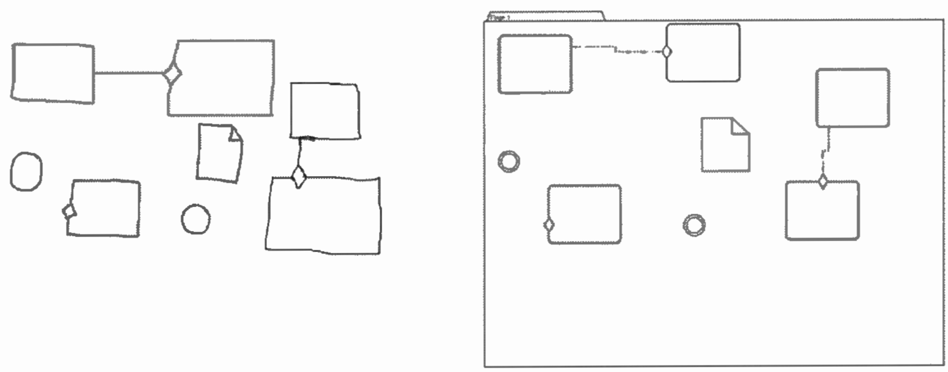


Figure 3.1 Formalizes case model from a CMMN hand-drawn sketch

Indeed, in order to recognize CMMN elements from hand-drawn sketches, we specified three steps that include:

- Primitive shape recognition;
- Composite graphic object recognition, and
- Semantic connections recognition and understanding.

In the beginning, the user draws a sketch that includes CMMN elements. Therefore, the first step consists of reading a raw input image and recognizing the contours as primitive shapes. The second step, consisting of recognizing composite shapes that correspond to CMMN elements, combines the primitive shapes based on their spatial relationships. The third step of recognition consists of understanding the semantic connections between the primitive shapes, according to the CMMN elements spatial relationships. Once this is done, we can export the result model into an XML file using CMMN's model interchange format.

In the next part of this chapter, each step is explained in detail. In particular, we will discover the problems inherent in each step, and describe algorithms that will attempt to solve them.

3.2 Technology Used

Our prototype was produced in Java on the eclipse IDE, as well as using the OpenCV library (OpenCV). This library is an open source C++ library, as shown in Figure 3.2, which is optimized for real-time image processing and computer vision applications. OpenCV has a modular structure, meaning it has several hundreds of image processing and computer vision algorithms which make developing advanced computer vision applications easy and efficient (OpenCV)..

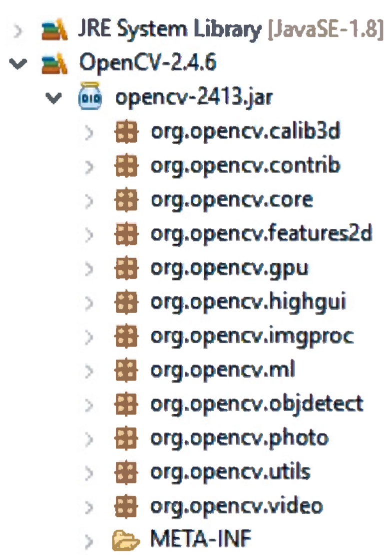


Figure 3.2 OpenCV library on the Eclipse IDE

In OpenCV, digital images are represented using numerical variables for each point. Thus, images are represented using matrices as well as metadata about the image. To give an illustration (see figure 3.3), the mirror of the car is nothing more than a matrix containing all the intensity values of the pixel points (OpenCV documentation, 2013).

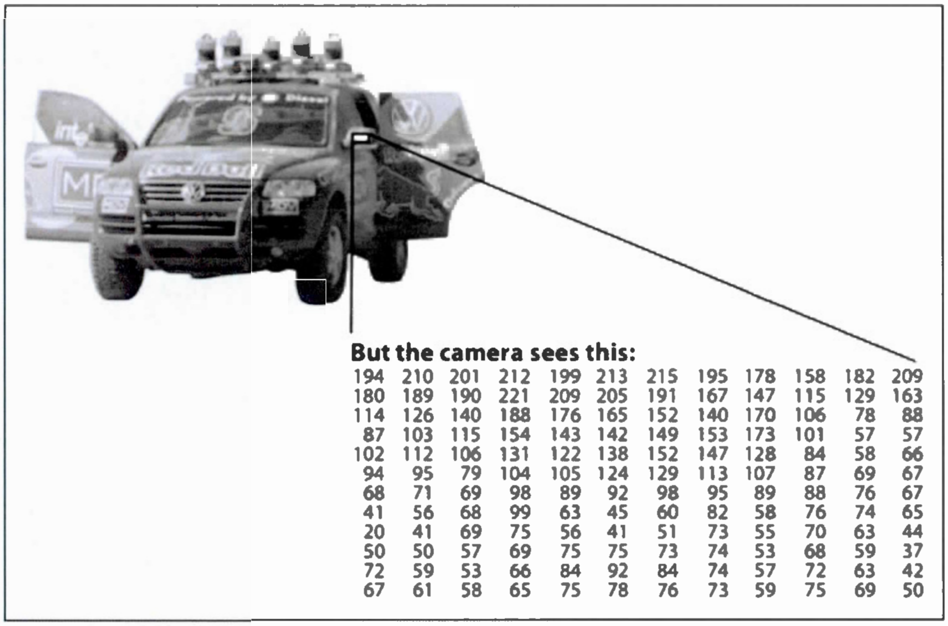


Figure 3.3 Represents the data structure in OpenCV (Szeliski, 2010)

3.3 Sketch Recognition Design and Implementation

Our example, despite its simplicity, offers an overview of the recognition system and the connection of classes in order to implement the three recognition steps mentioned earlier. In the following, each step is described in detail. Figure 3.4 gives an overview of the design model of the prototype. Subsequent sections will give more details about the relevant parts of the design model.

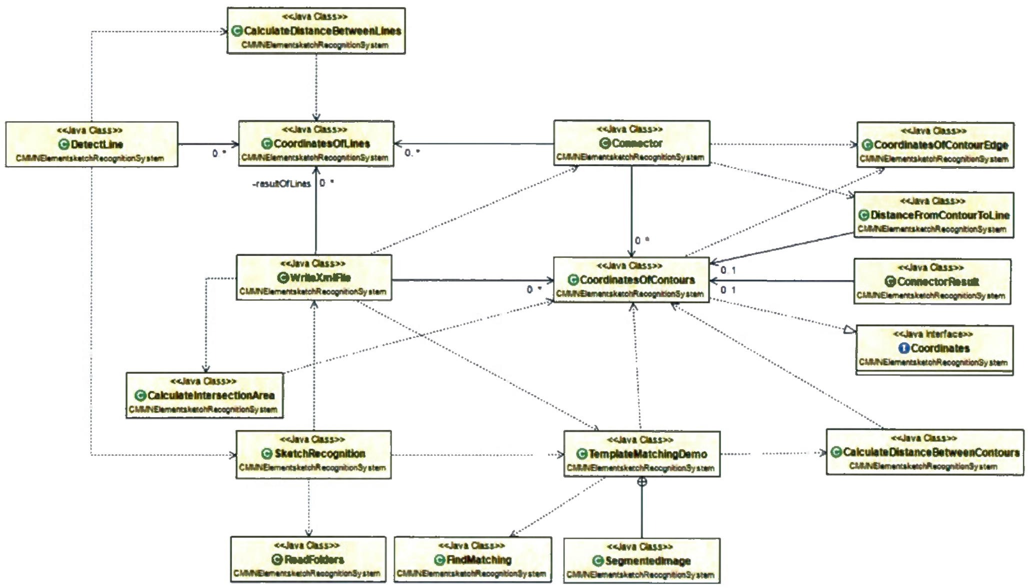


Figure 3.4 Design model compatible with the case study

3.4 Primitive Shape Recognition

The definition of primitive shape includes recognition of elementary CMMN constructs within user hand sketches. In this section, we investigate how to extract the contours from inside the sketch and recognize them as the primitive shapes that compose the CMMN notation.

3.4.1 Reading Hand-Drawn Sketch

In order to recognize the primitive shapes within a hand sketch, the program needs to find and read the hand-drawn sketch as an input. Figure 3.5 shows a sample of CMMN primitive shapes. The definition of each shape was explained in Chapter II.

Thus, we need to find the paths that both input and predefined templates are stored. Hence, "ReadFolders" Java class was defined, as shown in Appendix A, in order to read the path of the main directory and also listing all files from the directory and its subdirectories. In addition, we defined a condition for reading only the files with the suffix of JPG or PNG to avoid reading the files with difference suffix and format.

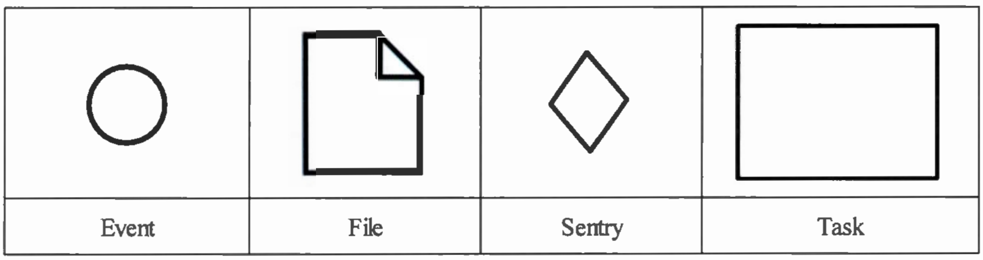


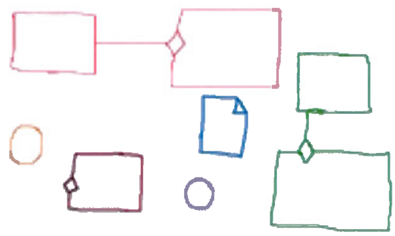
Figure 3.5 Templates images

3.4.2 Pre-Processing

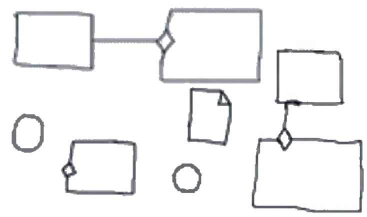
In order to perform any object detection, as we mentioned in chapter II, pre-processing steps are necessary. Hence, we used the OpenCV library to implement some pre-processing functions on the input image to improve the final result (Figure 3.6). These functions are used to accomplish various linear or non-linear filtering operations on 2D images that were represented by a two-dimensional matrix (OpenCV).

The first step of the pre-processing process converts an image from RGB to grayscale. It means that an input image will be converted from one color space to another (OpenCV documentation, 2013). The default color format in OpenCV is BGR, in other words, the bytes are reversed. Thus, a standard (24-bit) color image is composed of an 8-bit Blue, 8-bit Green and 8-bit Red. This sequence is repeated for each pixel (Blue, Green, Red), and so on (OpenCV documentation, 2013). «When grayscale images are converted to color images, all components of the resulting image are taken to be equal; but for the reverse transformation, the gray value is computed using the perceptually weighted equation» (Bradski et Kaehler, 2008):

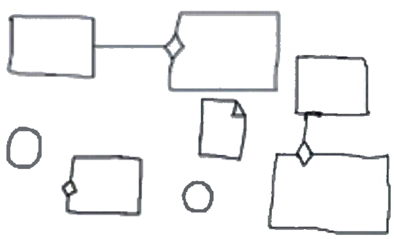
$$\text{RGB TO GRAY:} \quad Y \leftarrow 0.299\,R + 0.587\,G + 0.114\,B \qquad (3.1)$$



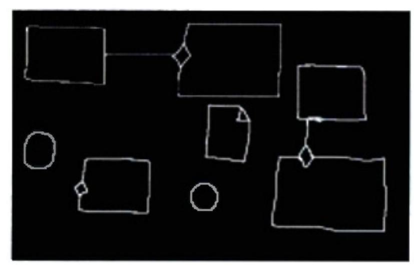
(a) Original input image (colorful)



(b) Convert RGB to grayscale image



(c) Smoothing grayscale image



(d) Convert grayscale to binary image

Figure 3.6 Pre-processing steps of the recognition system: (a)→(b) Grayscale operation converts RGB image to grayscale image; (b) →(c) Blur operation reduces noise and smoothing the grayscale image; (c)→(d) Threshold operation assigns one value (black) to the pixel value greater than the threshold value, else assign another value (white).

The second step of the pre-processing process is called image smoothing, as well as image blurring, which reduces noise and smoothies the image. In order to perform a smoothing operation, we apply a filter. The overall performance of the filter is that «an output pixel's value *(i.e.* $g(i, j)$*)* is determined as a weighted sum of input neighborly pixel values *(i.e.* $f(i+k, j+l)$*)* » (OpenCV documentation, 2013), typically $-1 \leq k \leq 1$ and $-1 \leq l \leq 1$ (figure 3.7). In addition, the coefficient of the filter that is called the kernel is defined by h (k, l) (OpenCV documentation, 2013). The equation below shows the smoothing operation (OpenCV documentation, 2013):

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

The OpenCV library comes with several filters. We used the Gaussian filter (OpenCV documentation, 2013; Szeliski, 2010) , which is done by «convolving each point in the input array with a *Gaussian kernel* and then summing all the points contribute to produce the output array» (OpenCV documentation, 2013; Szeliski, 2010). In order to make the context clearer, a 1D Gaussian kernel is presented (figure 3.8). Hence, the pixel located in the middle contains the biggest weight. Therefore, « the weight of its neighbors decreases as the spatial distance between them and the center pixel increases » (OpenCV documentation, 2013; Szeliski, 2010).

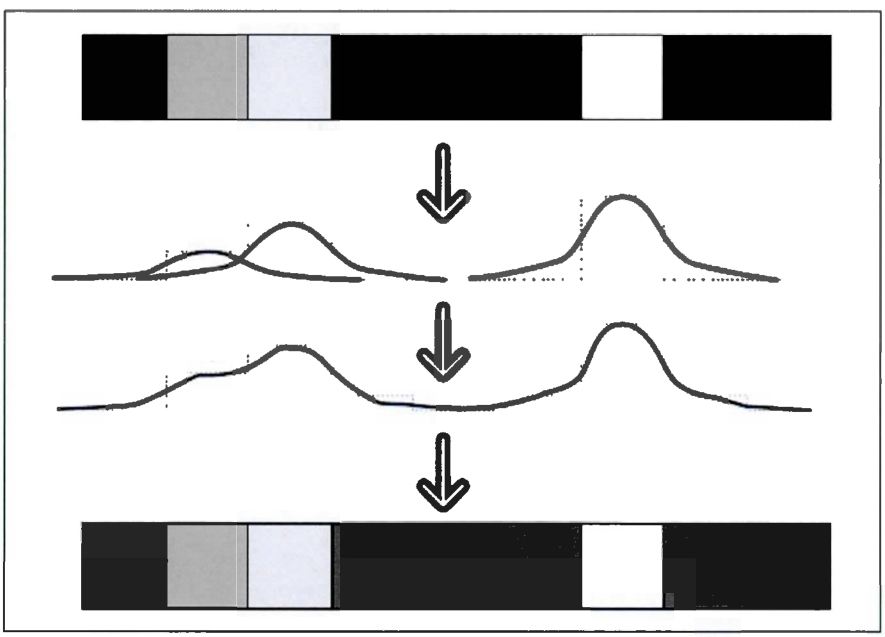


Figure 3.7 Gaussian blur on 1D pixel array (Szeliski, 2010)

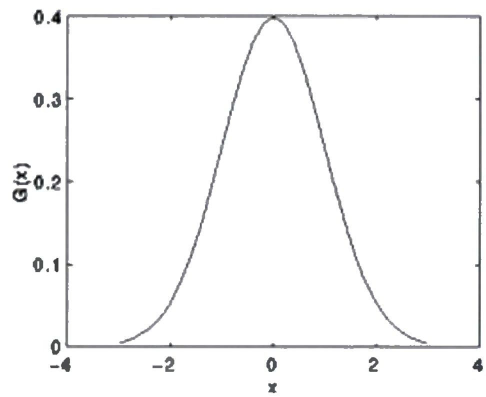


Figure 3.8 1D Gaussian kernel (OpenCV documentation, 2013)

As a result, a 2D Gaussian can be represented using the following equation (OpenCV documentation, 2013; Szeliski, 2010):

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

Where μ is the peak and σ represents the variance (for each of the variables x and y) (OpenCV documentation, 2013; Szeliski, 2010).

Thresholding (OpenCV documentation, 2013), as the last step of the image pre-processing, is a non-linear operation that segments grayscale images in order to convert them into binary images. Thresholding gives threshold values, pixels that have a value above the threshold are kept white, and the others are changed to black.

There are several threshold operations in OpenCV, but we chose the binary inverted function (Figure 3.9) (OpenCV documentation, 2013), to perform a threshold operation. Hence, if the pixel value is greater than a threshold value, it is assigned one value (black), otherwise, it is assigned to another value (white). The threshold operation can be expressed as the following equation (OpenCV documentation, 2013):

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

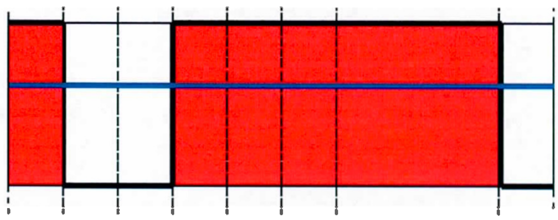


Figure 3.9 Threshold operation: If the intensity of the pixel *src(x,y)* is higher than *thresh*, then the new pixel intensity is set to a *0*. Otherwise, it is set to *MaxVal* .The horizontal blue line represents the threshold *thresh* (OpenCV documentation, 2013)

The "preprocessImage" java method, as shown in Appendix A, shows the details of pre-processing image operations.

For our purposes, we need to compare each primitive shape contained in the hand-drawn sketch to the predefined template's images. Hence, a table will be specified to classify the template's images according to their name, and storing their features based on their name. Note that the same preprocessing is applied to the predefined CMMN shapes to allow for a fair comparison. Appendix A shows the methods preprocess all templates with details.

3.4.3 Contour Finding

Before we compare hand sketches to CMMN templates, we need to convert both the input image and the templates to contours. A contour is a list of points that represent a binary image (Bradski et Kaehler, 2008), A hand-drawn sketch is composed of the sequence of points, which represent the shapes. Thus, the first step of finding shapes in the hand-drawn sketch and identifies their features is to detect contours.

The function "findcontours" (Bradski et Kaehler, 2008) in OpenCV library, finds contours within a binary image. Hence, each contour is a sequence of points, each represented by four values that consist of four important elements as pointers to other points in the sequence. OpenCV represents contour as an array of quadruples (Bradski et Kaehler, 2008):

(h_prev, h_next, v_prev, and v_next)

As shown in figure 3.10, where **h_prev** is the horizontal coordinate (x) of the previous point in the contour, **h_next** is the horizontal coordinate of the next point in the sequence, and **v_prev** and **v_next** are the vertical (y) coordinates of the previous and next point in the sequence, respectively.

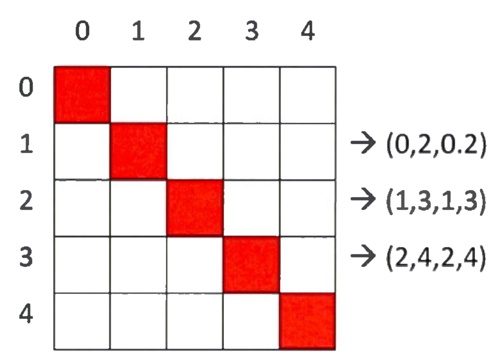


Figure 3.10 Show graphically a pixel image and the corresponding contour

Note however that, in some cases, some contours are inside other contours, like the sentry in CMMN, which as a diamond that is part of a task (see figure 3.11). The first problem that can occur is trying to figure out how to extract the inner contours from outer contours.

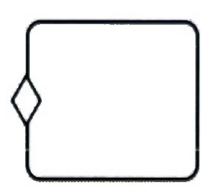


Figure 3.11 Represents a sentry model

In this case, we can call outer contour as a *parent* and inner contour as a *child*. By defining this structure (Bradski et Kaehler, 2008), we can specify how the contours are connected to each other. In other words, the contour can be specified as a child of some other contour or it can be defined as a parent. This type of relationship is called the *hierarchy* (Bradski et Kaehler, 2008).

In order to retrieve the hierarchy, the contour retrieval mode that is an argument of function "findcontours" is used. There are four different types of Retrieval Mode that include (Bradski et Kaehler, 2008):

RETR_EXTERNAL, RETR_LIST, RETR_TREE and RETR_CCOMP

RETR_EXTERNAL (Bradski et Kaehler, 2008) returns only extreme outer contours. As shown in figure 3.12, there is only one outer contour. Therefore, figure 3.13-(a), represents the first contour points as an outermost sequence and there are no inner contours.

RETR_LIST (Bradski et Kaehler, 2008) is the simplest retrieval mode that retrieves all the contours without creating any parent-child relationship and puts them on the list. In other words, all contours are on the same level. Figure 3.13-(c), illustrates the list from the image in figure 3.12. Therefore, all contours are connected to one another by **h_prev** and **h_next**.

RETR_TREE (Bradski et Kaehler, 2008) retrieves all the contours in order to create a full hierarchy list. As shown in figures 3.12 and figures 3.13-(d), the root node of the tree is the outermost contour. Below the root node, each hole is connected to the other hole at the same level. In addition, «each of those holes, in turn, has children, which are connected to their parents by vertical links. This continues down to the most interior contours in the image, which become the leaf nodes in the tree.» (Bradski et Kaehler, 2008).

Finally, *RETR_CCOMP* (Bradski et Kaehler, 2008) retrieves all the contours and arranges them into a two-level hierarchy. Hence, the top-level boundaries are placed in hierarchy-1 which is the first level; the boundaries of the holes are placed in hierarchy-2 which is second level (Bradski et Kaehler, 2008). As shown in figure 3.13-(b), « the boundaries of the holes are connected to their corresponding exterior boundaries by v_next and v_prev » (Bradski et Kaehler, 2008). In addition, all of the holes are connected to one another by the h_prev and h_next pointers (Bradski et Kaehler, 2008).

Recall that the RETR_CCOMP mode retrieves a two level hierarchy where the first level represents outer contours which act as parents, and the second level contains inner contours which act as children. Hence, we need to write an algorithm that only retrieves the children of parents. In the algorithm below, according to the figure 3.12, we represent how to retrieve all the hierarchy levels as a child and parent and only extract the hierarchy level relevant to the inner contours.

Algorithm 1: Finding Second Level Of Hierarchy(h)

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

Output: retrieve all the inner contours

for $C = C_1$ *to* C_k **do**

  find the first level of hierarchy as parents $c = c_0, c_{000}, c_{010}$

  find the second level of hierarchy as children $h = h_{00}, h_{01}, h_{0000}, h_{0100}$

  remove $c = c_0, c_{000}, c_{010}$

return $h = h_{00}, h_{01}, h_{0000}, h_{0100}$

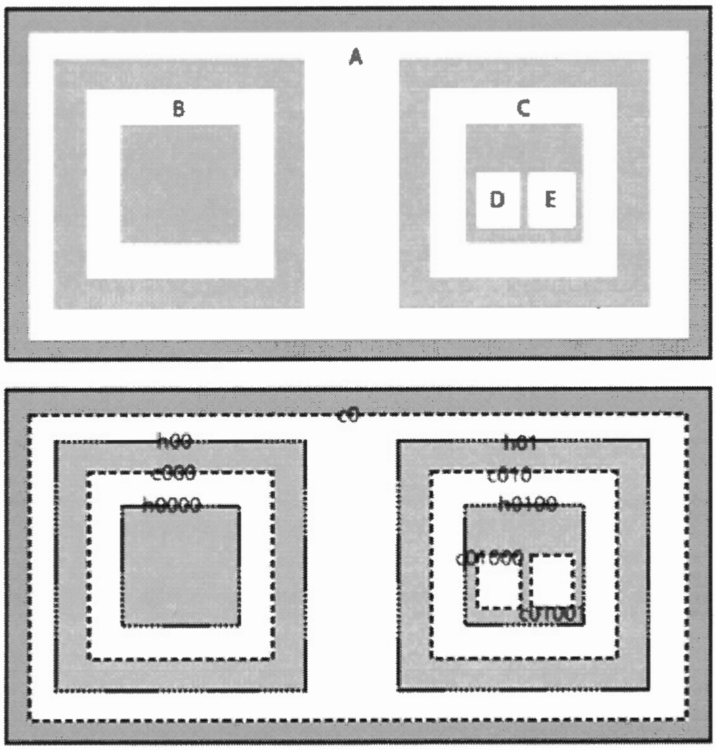


Figure 3.12 A test image presents the contours that could be exterior contours (dashed lines) or interior contours (dotted lines) (Bradski et Kaehler, 2008)

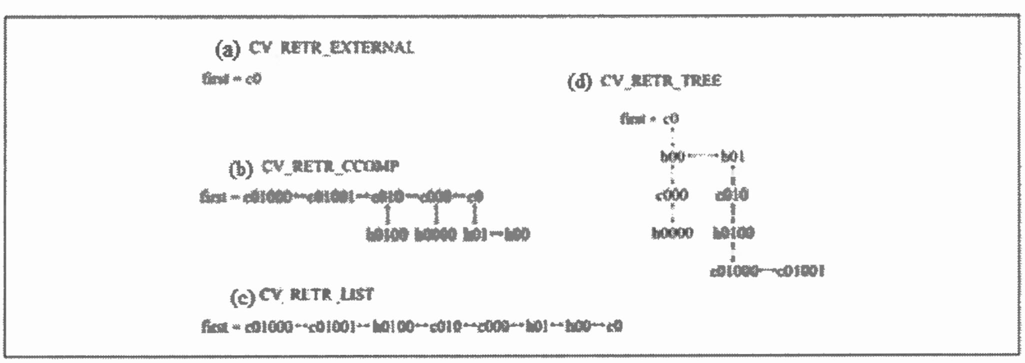


Figure 3.13 Different types of Retrieval Modesin order to find contours in OpenCV (Bradski et Kaehler, 2008)

The "segmentImage" Java class, shown in Appendix A, is responsible for finding contours inside the hand-drawn sketch.

3.4.4 Feature Extraction

In this stage, after finding all children contours, we need to extract the features of each contour. By finding the contour features such as length, area, and bounding box, we specify the contours as independent shapes. By extracting these features, we can start looking for matches between the independent shapes and CMMN's template images.

Using the "boundingRect" function in OpenCV, we can find the bounding box around each contour (see figure 3.14) and extract its features such as the top-left coordinate of the rectangle as the starting point of the contour, as well as its width and height as the width and height of the contour (Bradski et Kaehler, 2008). Thus, each contour will be presented based on these features.

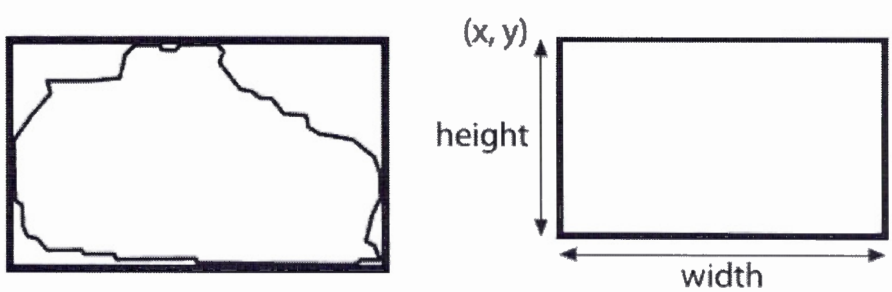


Figure 3.14 Bounding box around the contour (Bradski et Kaehler, 2008)

The "getShapeSubBitMap" Java Class, as shown in Appendix A, is responsible for specifying the bounding box around each contour. The following algorithm shows the main function.

Algorithm 2: 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**
  $C_w \leftarrow$(y, y+h)
  $C_h \leftarrow$(x, x+w)
return (C_w, C_h)

3.4.5 Contour Resizing

When drawing hand sketches, people rarely worry about the size of their sketches on the proportionality of their figures, i.e., the relative size of height versus width for example. In order to compare the contours recognized in hand drawing to those in CMMN's templates, we need to resize the hand drawn contours to the same dimension as the template we are trying to match it to. In other words, we need to rescale each contour according to the area of each template image by keeping aspect ratio. The following code, shown in Appendix A as the "getResizeSize" Java class shows the way that we rescaled the contour by keeping its aspect ratio.

Algorithm 3: 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

C_w: the width of contour
C_h: the height of contour
T_w: the width of template
R_w: the re-size width of template
R_h: the re-size height of template
$R_w \leftarrow 0$
$R_h \leftarrow 0$
for $C = C_1$ *to* C_k **do**
    scale $\leftarrow C_w \ / \ C_h$
    $R_w \leftarrow T_w$
    $R_h \leftarrow R_w$ / *scale*
return (R_w, R_h)

By rescaling the contours, we can enter into the next step, which is finding the match between each contour and each template image.

3.4.6 Template Matching

Now that all closed contours in the input image were retrieved and resized according to each template's image area, we can start focusing on the main part of sketch recognition, which is finding a match between each contour extracted and a collection of template's images. Template Matching is a method for searching and finding the location of a template image in a contour. Therefore, OpenCV provides a function "matchTemplate" (Bradski et Kaehler, 2008) for this purpose. This method (figure 3.15, figure 3.17) starts to slide the template image over the contour in two dimensions, and compares the template and contour under the template image, looking for a strong match (Bradski et Kaehler, 2008).

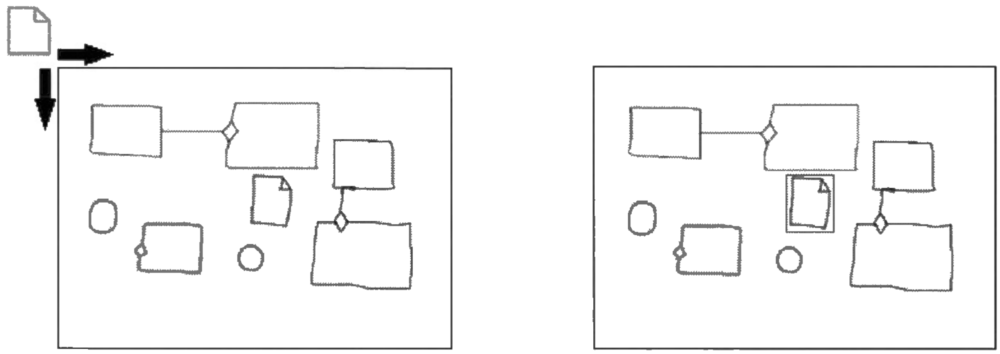


Figure 3.15 Represents MatchTemplate function: starts to slide the template image over the input image in order to find matches (Mup).

The template matching function can implement one of three matching methods that include:

(a) Square difference matching method (method = TM_SQDIFF)
(b) Correlation matching methods (method = TM_CCORR)
(c) Correlation coefficient matching methods (method = TM_CCOEFF)

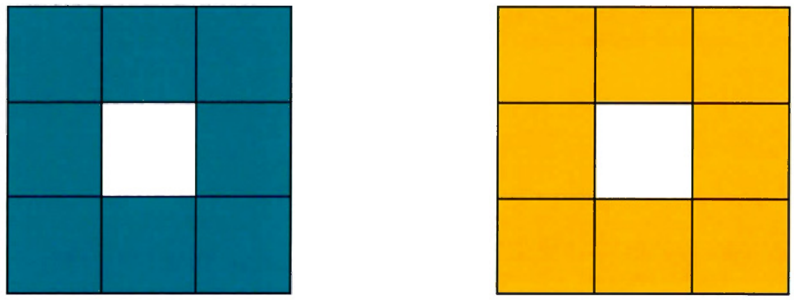
In the following, we will explain each method along with its mathematical formula.

(a) Square difference matching method (method = TM_SQDIFF)

«The results of these methods (figure 3.17) match the squared difference, so a perfect match will be 0 and bad matches will be large » (Bradski et Kaehler, 2008).

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

In order to clarify the meaning of the method, as shown in figure 3.16, we defined two matrices that have the same dimensions. When the two matrices have the same size, we will have a single value to compute. So, normally, the point with the highest score is always going to be (Ø, Ø).



(a) Input image by 3-by-3 matrix (b) Template image by 3-by-3 matrix

Figure 3.16 Represents the input image and template image as matrix

(b) Correlation matching methods (method = TM_CCORR)

« The result of these methods (Figure 3.17) multiplicatively match the template against the image, so a perfect match will be large and bad matches will be small or 0 » (Bradski et Kaehler, 2008).

$$R_{ccorr}(x,y)=\sum_{x',y'}[T(x',y')\cdot I(x+x',y+y')]^2 \tag{3.6}$$

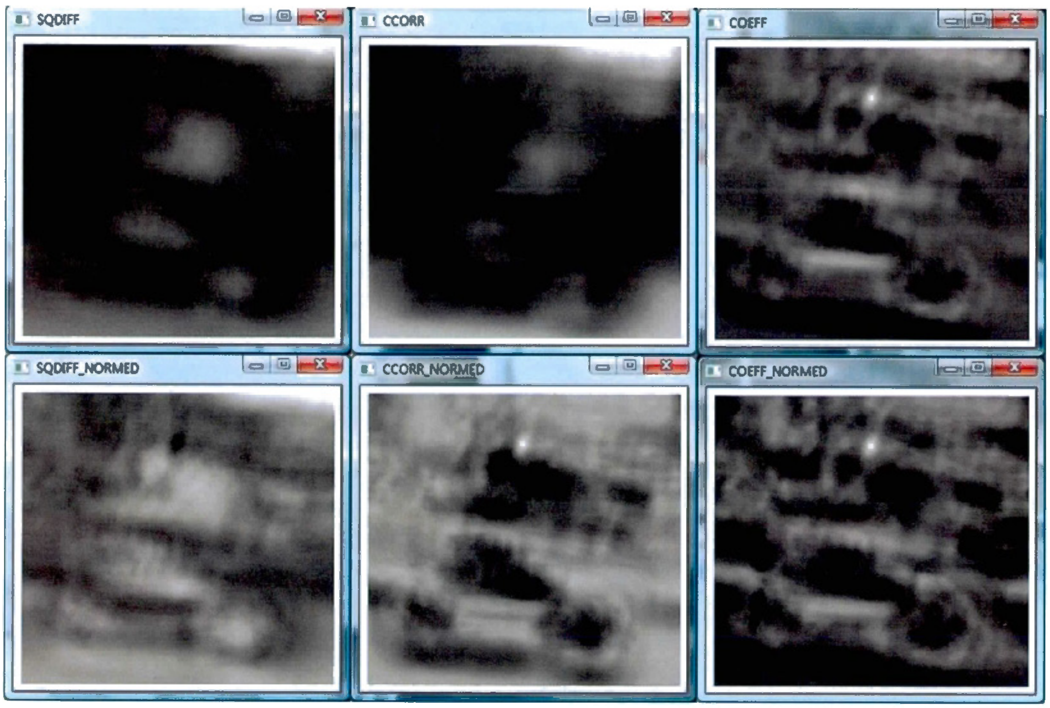
(c) Correlation coefficient matching methods (method = TM_CCOEFF)

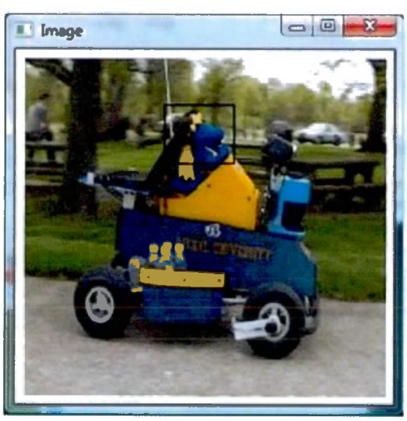
« The result of these methods (figure 3.17) match a template relative to its mean against the image relative to its mean, so a perfect match will be 1 and a perfect mismatch will be -1; a value of 0 simply means that there is no correlation (random alignments) » (Bradski et Kaehler, 2008).

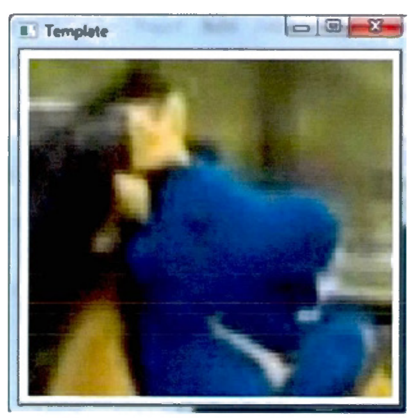
$$R_{ccoeff}(x,y)=\sum_{x',y'}[T'(x',y')\cdot I'(x+x',y+y')]^2 \tag{3.7}$$

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

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







a) Input image

b) Template image

Figure 3.17 Represents match results of six matching methods according to the illustrations a) input image and b) template image (Mup)

The best match for square difference is 0 and for the other methods it is the maximum point; thus, matches are indicated by dark areas in the left column and by bright spots in the other two columns » (Bradski et Kaehler, 2008).

By implementing these three matching methods, based on the highest experimental matching result, we selected the *TM_CCORR_NORMED* method that represents the following formula:

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

The result of this comparison returns a grayscale image, where each pixel defines how much the neighborhood of that pixel matches with each template image (OpenCV).

An empty matrix needs to be defined in order to store the result. On one hand, by default in OpenCV, the input area image is bigger than template area. On the other hand, the area of each contour that is already extracted and resized by keeping aspect ratio can be bigger than the size of the template. Hence, before defining the empty matrix for storing the match result, we need to specify which area (contour area or template area) is bigger. In the following, the size of the bigger matrix is defined as *(WxH)* and the size of the smaller matrix is defined as *(wxh)*.The size of the result that contains the output image will be *(W-w+1, H-h+1)*. The following algorithm displays the function.

Algorithm 4: 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**
    $(W, H) \leftarrow (\mathrm{R}_w, R_h)$ and $(w, h) \leftarrow (\mathrm{T}_w, T_h)$
    else
    $(W, H) \leftarrow (\mathrm{T}_w, T_h)$ and $(w, h) \leftarrow (\mathrm{R}_w, R_h)$
return $(W - w + 1, H - h + 1)$

By having the results comparison of each contour and each template image, we need to find the best match for each contour. Therefore, the "minMaxLoc" method in OpenCV returns four outputs that include minimum value, maximum value, minimum point location (in two dimensions), and maximum point location (in two dimensions). Hence, for each contour, an array list of results is obtained by comparing each contour with all template images is retrieved. Consequently, the best match is composed of the result with the maximum value. The "MaximumValue" java class, implements this function, whose algorithm is shown next.

Algorithm 5: Best Matching Result(R)

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

find a list of maximum value of matching results
find a list of the location of maximum values
$Max_v \leftarrow \mathrm{R}_1$
for $R_i = R_2$ *to* R_n **do**
  if $R_i > Max_v$ **then**
    $Max_v \leftarrow \mathrm{R}_i$
return Max_v

As shown in Appendix A, the "FindMatching" Java class shows the details of the template matching process.

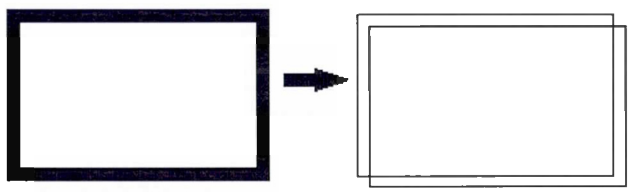
We should note that even though the matching method compares contours that are one pixel wide (see below the issue with thick drawings), the algorithm performed well, for the following reasons:

- The preprocessing steps that we perform prior to computing the contours clean the image and remove the "noise" that can result from wobbly drawings, by smoothing lines prior to detecting contours;
- By reducing both contours to the same bounding box, we eliminate errors due to differences in dimension ratios (e.g. ratio of height to width in hand sketch versus in template);
- The matching algorithm that we chose (TM_CCORR_NORMED), which was validated with experimental results, looks at means as opposed to individual points, and finally
- The matching algorithm returns the best match among the available templates.

This is what enabled us to get good results, despite the fact that we are matching predefined templates against one-pixel wide hand-drawn shapes.

3.4.7 Contour Distance

When the user draws a hand sketch using a thick pen or brush, the contour recognition step can return manifold contours (see figure 3.18). We have no easy way of detecting manifold contours. Thus, we simply rely on the proximity of contours to disregard some of them.



(a) (b)

Figure 3.18 Represents doubles of contour: (a) hand-drawn contour; (b) recognized primitive shapes

Hence, for solving this issue, we need to calculate the distance from the starting point of each contour with the rest of the recognized contours inside the hand-drawn sketch and define the threshold for comparing the distance between them. Therefore, if the distance of the considered contour from the rest of the detected contours in the list is less than the threshold, we dismiss the contour. We will repeat the same calculation for the remaining contours on the list. The "CalculateDistanceBetweenContour" Java class, as shown in Appendix A, shows the details.

3.4.8 Line Detection

The template matching method explained earlier only works for *contours*, which are *closed* geometric figures that only used to recognize the closed contours. Then, how do we recognize lines? Given a hand-drawn sketch, if we remove all of the closed contours, as shown in the algorithm below, we should be left with lines. However, we should point out that in case the lines are drawn with a thick pen or brush, as shown on the left hand-side of figure 3.19, then the OpenCV function for determining contours will return a closed contour that corresponds to the outer edges of the thick lines (see right hand side of figure 3.19). To this end, we used a threshold for the ratio between the dimensions of the contours to filter out contours that correspond to edges of thick lines.

Figure 3.19 Represents detected edges of line

Algorithm 6: 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
$w \leftarrow 0$, $h \leftarrow 0$, $x \leftarrow 0$, $y \leftarrow 0$
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
  $p_1 \leftarrow (x - Th\ ,\ y - Th)$
  $p_2 \leftarrow (x + w + Th\ ,\ y - Th)$
  $p_3 \leftarrow (x + w + Th\ ,\ y + h + Th)$
  $p_4 \leftarrow (x - Th\ ,\ y + h + Th)$
  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

There are two reasons for specifying a threshold:

1. To avoid recognizing the edges of each closed contour as a line;
2. A hand drawn line can be in a different position against the closed contour, as shown in figure 3.20. Thus, a proper line for the rest of the process is created.

To detect lines, we first apply edge detection to the lines. In the following, using "Hough Line Transform" method in OpenCV that search a binary image for straight lines is required. This method investigates that whether any point in a binary image could be part of some set of possible lines (Bradski et Kaehler, 2008) .

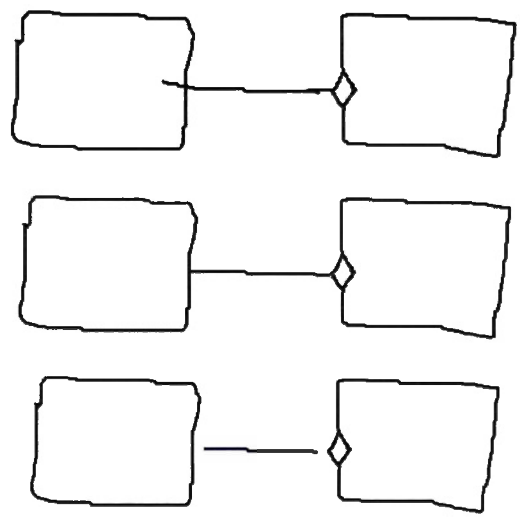


Figure 3.20 Different styles for drawing line

This method (see figure 3.21) expresses the line in the polar coordinate system. Hence, the equation for a line can be written as (Bradski et Kaehler, 2008; OpenCV documentation, 2013):

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

In general, for each point (x_0, y_0), we can define a family of lines that goes through that point as (Bradski et Kaehler, 2008; OpenCV documentation, 2013):

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

In other words, each pair (P_0, Θ) represents each line that passes by (x_0, y_0) (see figure 3.21-a) (Bradski et Kaehler, 2008; OpenCV documentation, 2013). For each point (x_0, y_0), the family of lines that go through it is defined (see figure 3.21-b)

(Bradski et Kaehler, 2008; OpenCV documentation, 2013). Thus, this operation will be done for all the points in an image. Hence, if the curves of two different points intersect, it means that both points belong to the same line (see figure 3.21-c) (Bradski et Kaehler, 2008; OpenCV documentation, 2013).

In order to detect line, a *threshold* as the minimum number of intersections is defined. Thus, if the number of intersection between curves of every point in the image is above the threshold, then a line with the parameters (Θ, p) of the intersection point is recognized (OpenCV documentation, 2013).

OpenCV implements two kind of Hough Line Transforms that include "*HoughLines*" (Bradski et Kaehler, 2008) and "*HoughLinesP*" (Bradski et Kaehler, 2008).The first function returns a vector of couples (Θ, P_0) as a result and the second function, rather than collecting every possible point, it collects only a fraction of them. Hence, if the peak is going to be high enough, then it returns the start point and end point of the detected lines (x0,y0,x1,y1) (Bradski et Kaehler, 2008; OpenCV documentation, 2013).

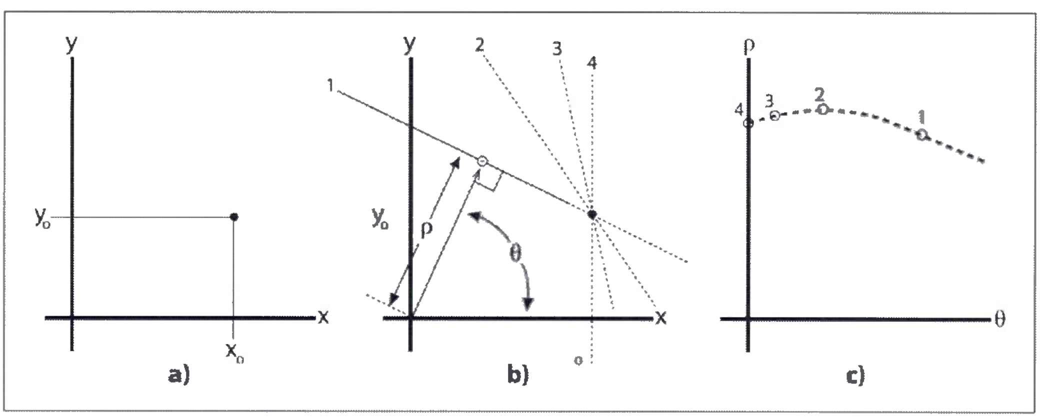


Figure 3.21 Represents point in the image (a) Represent a point (x_0, y_0) in the image: (b) represents lines that parameterized by a different p and θ; (c) Any of these lines represents points in the parameters (p, θ), consider together form a curve of characteristic shape (Bradski et Kaehler, 2008).

By recognizing lines, the system is able to discover the spatial relationships between each line and contours as a connector in the next section. The java class detects line, as shown in Appendix B; "DetectLine" shows the details for recognizing line.

3.4.9 Line Distance

Because hand drawn sketches can have thick lines, multiple lines may end up being recognized. For solving this issue, we use the same solution that is used for primitive shapes: we merge lines whose mutual distance is below a certain threshold. By finding the nearest lines, we calculate the minimum coordinate of the start point, as well as calculate the maximum coordinate of the end point for all lines whose distance is less than the threshold. Thus, the system avoids recognizing one line several times. Further, we want the system to recognize the longest line. The following algorithm displays some part of the function.

Algorithm 7: 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
$distance \leftarrow 0$
for $L = L_1$ *to* L_n **do**
  initialize $distance$ by calculating the distance of lines
  if $distance < Th$ **then**
    $(x_{min}, y_{min}) \leftarrow$calculate the minimum start point of lines
    $(x_{max}, y_{max}) \leftarrow$calculate the maximum end point of lines
return $((x_{min}, y_{min}), (x_{max}, y_{max}))$

"CalculateDistanceBetweenLines" Java class, as shown in Appendix B shows the details.

3.5 Composite Graphic Object Recognition

Composite shapes are composed of primitive shapes. Thus, we need to combine the primitive shapes that were recognized in the previous step, using their spatial relationships. There are two main composite shapes, sentry and connector, that we will focus on them in this section. Hence, the first issue is:

How do we define the diamond in the square (defined as a task element in CMMN) and recognize the sentry image as an element in CMMN?

3.5.1 Intersection Area

In order to recognize sentries, we need to discover whether the diamond is inside the task or not. Hence, we define a function that receives the coordinates of all tasks and diamonds and then starts calculating the distance of each vertex of diamond with the task area. Therefore, whenever one of the coordinates of the vertices of the diamond is part of the task, the diamond, which is called sentry in CMMN, is considered to be part of the composite shape. The following code displays some part of the function.

Algorithm 8: 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 $x_s \leftarrow 0$, $y_s \leftarrow 0$, $w_s \leftarrow 0$, $h_s \leftarrow 0$
initialize the coordinates of task $x_t \leftarrow 0$, $y_t \leftarrow 0$, $w_t \leftarrow 0$, $h_t \leftarrow 0$
while $S \neq \phi$ *and* $T \neq \phi$ **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*

As shown in Appendix B, "CalculateIntersectionArea" Java class shows the Details calculation.

3.5.2 Connector Detection

The second issue in composite graphic object recognition is:

How do we recognize the connection between lines and other contours?

To solve this issue, we need to find the contours that are connected to lines, by discovering spatial relationships, if necessary. We would need to calculate the distance between each shape and the start point and end point of each line.

Regarding the bounding box around each contour, by assuming that the start point of the line or end point of the line can be connected to the vertical edge of a contour or horizontal edge of the contour, we can start calculating the distance of each start point and end point of line with one of two selected edges.

To calculate the distance, first of all, the system needs to detect whether the line at hand-drawn sketch is vertical or horizontal. Because the CMMN modeler recognizes only vertical or horizontal lines, our algorithm is based on recognizing straight line. By clarifying this part, we are able to define whether the coordinate of one of the two points of line is in the range of the horizontal edge or vertical edge.

Finally, the distance between the start and end point of the line and the edges is calculated and the minimum distance will be selected.

The following algorithm shows how a connection is established. The threshold is specified for calculating the distance of each point of the line and each primitive shape to compare the minimum gained distance. If it is less than the threshold, the shape is connected to the line.

Algorithm 9: 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 $\leftarrow \propto$
        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*

else
if $(y_{sc} > y_{ec})$ **then**
 //It is a vertical line, Make sure that line start has smaller y
 $(p_{sx}.p_{sy}) \leftarrow (x_{ec}, y_{ec})$
 $(p_{ex}, p_{ey}) \leftarrow (x_{sc}, y_{sc})$
 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 $y_L < (p_{sy} - Th)$ *or* $y_L > (p_{ey} + Th)$ **then**
 distance $\leftarrow \propto$
 else
 //If the point is between the 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*

As shown in Appendix B, the "Connector" Java class shows the details of detect connector.

3.6 Semantic Connection Recognition and Understanding

By recognizing the primitive shapes as well as the spatial relationships between them, the next and final issue is:

How do we display these detected CMMN elements and semantic connections between them in CMMN modeler?

To do this, we create an XMI file that stores details of all the detected CMMN element specifications as well as their spatial relationships. This XML file should be in a format that can be imported in CMMN modeler. Thus, we need to understand the XMI schema for CMMN to generate the appropriate tags. In this section, we will regularly refer to aspects of CMMN execution semantics.

XML file structure in CMMN, as shown in figure 3.22, consists of two major parts that includes the *case* (OMG) and *CMMN DI* (OMG). Case is a top-level concept that combines all the elements that constitute a case model, and that defines the semantic relationships between the elements (OMG). Each model element has attributes such as coordinates, width, height, label and so on. The CMMN DI section is used to specify the visual attributes of elements that include a collection of shapes and edges.


```xml
<?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/bpmn" 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="_8ab9"/>
            <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="1545.5"/>
            <cmmndi:CMMNShape cmmnElementRef="_1036" id="_e8e5">
                <dc:Bounds height="76.0" width="97.0" x="191.5" y="52.0"/>
                <cmmndi:CMMNLabel/>
            </cmmndi: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>
        </cmmndi:CMMNDiagram>
        <cmmndi:CMMNStyle fontFamily="Arial,Helvetica,sans-serif" id="_75ca"/>
    </cmmndi:CMMNDI>
</cmmn:definitions>
```


Figure 3.22 XML file structure in CMMN

3.6.1 Case

To understand the structure of a case model, we provide a partial CMMN meta-model in figure 4.23. In the following, in order to explain the semantic relationships between CMMN elements, we will focus on the association *caseFileModel* and *casePlanModel* and explain them in detail.

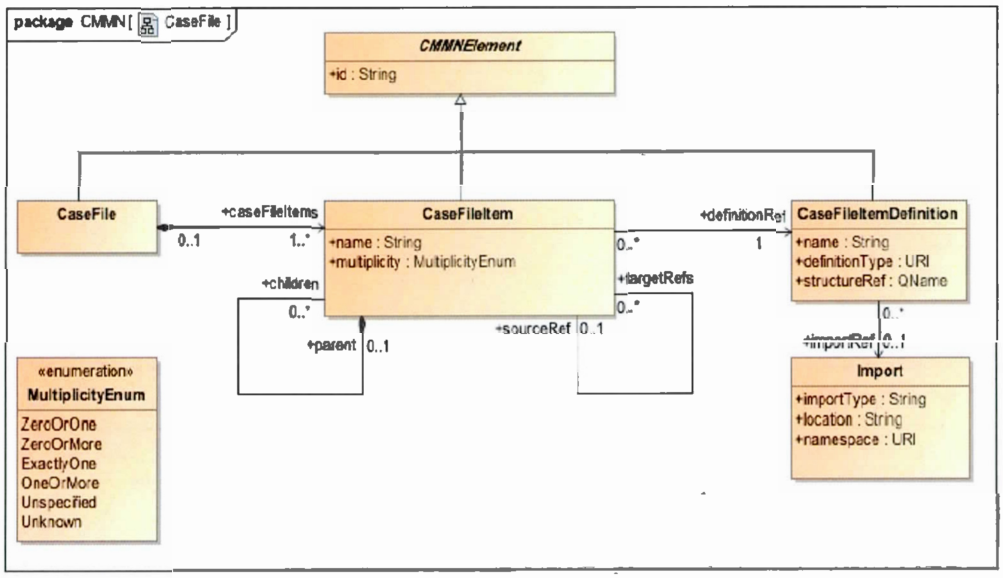


Figure 3.23 CMMN case file meta-model (OMG)

3.6.1.1 CaseFile

Every case is associated with exactly one CaseFile (see figure 3.24). The case information is represented by the CaseFile. In addition, every CaseFile must contain at least one CaseFileItem. In the following, every CaseFileItem must be associated with exactly one CaseFileItemDefinition. Thus, in order to specify the association between each FileItemDefinition and each CaseFileItem, the *definitionRef* for each FileItemDefinition must be the *id* of each CaseFileItem. The XML file below displays some part of the XML file in CMMN, which describes 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 Case file structure in XML format

As shown in Appendix C, the “writeFileItemDefinition” and “writeFileItem” java methods show how to serialize a case file on an XML file.

3.6.1.2 Case Plan

CasePlan is constructed from the building blocks that are composed of PlanItemDefinition elements (figure 3.25). Each PlanItemDefinition can represent one CMMN element, which can include PlanFragment (and Stage), Task, EventListener or Milestone. Thus, in order to specify the association between each PlanItemDefinition, and each certain CMMN element, the *definitionRef* for each PlanItemDefinition must be the *id* of each CMMN element.

The XML file below displays some part of the XML file in CMMN, which describes a CasePlanModel (see 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 Case Plan structure in XML format

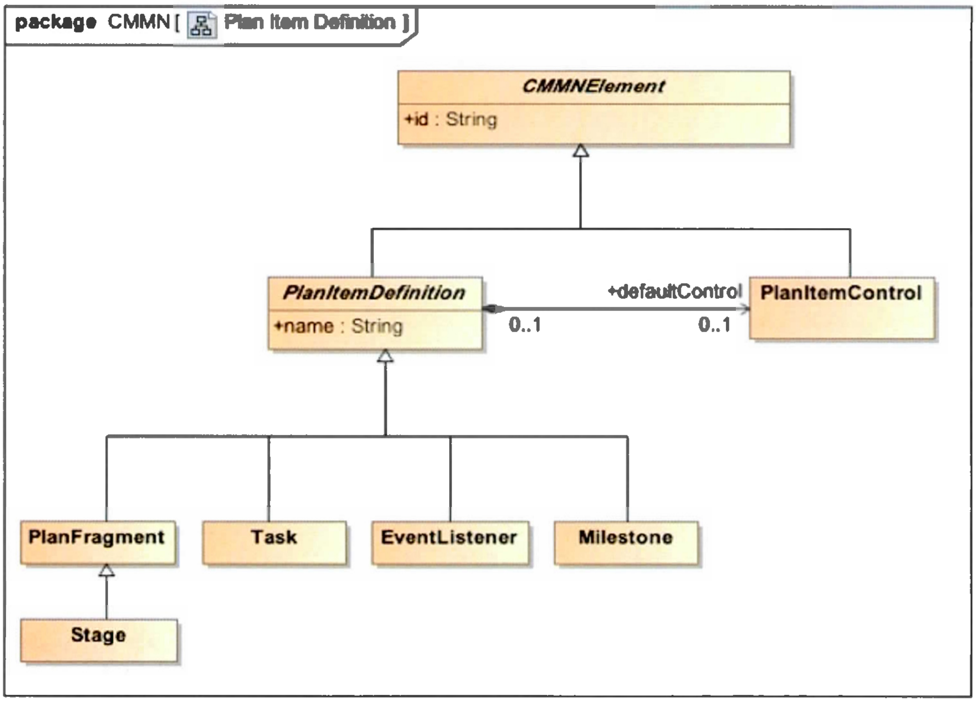


Figure 3.26 Plan Item Definition (OMG)

Sentries, which are used as entry criteria, are a CMMN element that not provided as an independent CMMN element, but need to be present as part of other CMMN elements, such as stage or task (figure 3.27). Hence, the relations between a PlanItemDefinition and sentry are represented with the association "entryCriterion". Therefore, in order to specify the connection between mentioned CMMN elements and each Sentry, *sentryRef* for each entryCriterion that must be the *id* of each sentry. The XML file below displays some part of the xml file in CMMN, which shows a sentry as part of a task.

```xml
<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 Sentry structure in XML format

Because a connector in CMMN is used to visualize dependencies between CMMN elements (see figure 3.28), it is necessary to show which CMMN elements a connector belongs to. The sequence flow direction or association is defined by an *entry criterion* or *exit criterion* (OMG).Thus, one side of a connector will be associated with a sentry and present as *planItemOnPart,* while the other side belongs to the other planItemDefinition that is connected to sentry. Therefore, in order to specify this association, we define the *sourceRef* for each planItemOnPart that must be the *id* of specific planItemDefinition that is connected to planItemOnPart. The XML file below displays some part of the XML file in CMMN, which describes the connection between two tasks.

```xml
<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 Connector structure in XML format

As shown in Appendix C, "writeplanItem", "writeEvent", "writeTask" and "writeSentry" java methods show how to serialize a PlanItemDefinition.

3.6.2 CMMN DI

CMMN DI (OMG) is used to specify the visual properties of elements that include a collection of shapes and edges. Figure 3.29 shows the partial Meta model for the CMMNDI component. It shows that CMMNDI is a container for the shared CMMNStyle and all the CMMNDiagrams defined in Definitions (OMG).

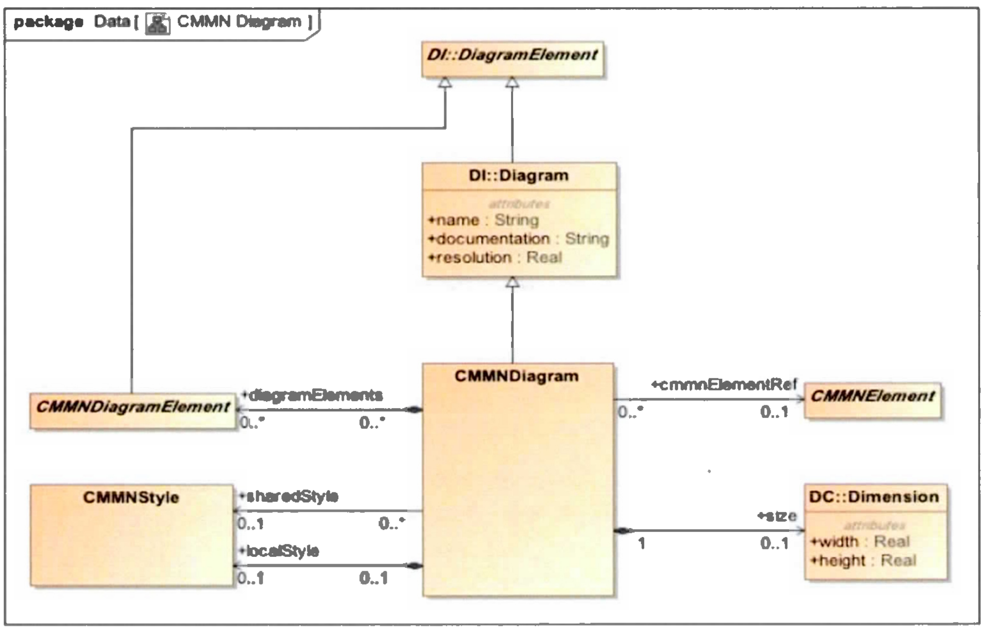


Figure 3.29 CMMNDI class diagram (OMG)

The class CMMNDiagram (OMG) is a kind of Diagram that represents a depiction of all or part of a CMMN model (OMG). In other words, it is the container of CMMNDiagramElement that is composed of CMMNShape and CMMNEdge (figure 3.30). The XML file below displays some part of the XML file in CMMN that describes a CMMNDiagram.

```xml
<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="297.99999996125814" 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 CMMNDI structure in XML format

As shown in Appendix C, the methods "writeFileValues", "writeLineValus" "writeEntryCriterionValus", "writeEventValues" and "writetaskValues" are used to serialize the entities CMMNDiagramElement, CMMNShape and CMMNEdge.

3.6.2.1 CMMNShape

The CMMNShape is a kind of Shape that depicts a CMMNElement from the CMMN model (figure 3.31). Hence, in order to associate it with a CMMN element, there is a *cmmnElementRef* attribute contains the *id* of each planItemDefinition. The XML file below displays some part of the XML file in CMMN, which describes CMMNshape.

```xml
<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 CMMNShape structure in XML format

3.6.2.2 CMMNEdge

The CMMNEdge class represents relationships between two CMMN model elements. Hence, CMMNEdge are used to depict links in the CMMN model (OMG). In order to use CMMNEdge to show a PlanItemOnPart, we define *cmmnElementRef* for each

CMMNEdge that must be the *id* of PlanItemOnPart. Further, we need to define the *targetCMMNElementRef* for each CMMNEdge that must be the *id* of one of the criterion (either an EntryCriterion or an ExitCriterion) that is linked to the Sentry holding the PlanItemOnPart. The XML file below displays some part of the XML file in CMMN, which describes 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 CMMNEdge structure in XML format

3.7 Conclusion

In this chapter, we described the implementation of our approach that recognizes both primitive shapes and composite shapes. We explained a set of issues at each stage of recognition and found the heuristic solutions, as well as an employed OpenCV library. In the following, using XML file according to the CMMN models' structures makes it possible to interpret each CMMN element and their semantic relationships. Therefore, the hand-drawn sketches will be recognized and formalized by importing the XML file in CMMN modeling tools. In the next chapter, we will evaluate our program by investigating various samples are collected and specifying the performance of the tool.

CHAPITRE IV

TEST SETTING AND RESULTS

To evaluate the accuracy of the recognition system, we need to test it on various samples. We collected 20 drawings made by experimental subjects. These subjects were aged between 24 -33 and are regular computer users.

The subjects received the following information and instructions:

1. They were introduced to CMMN syntax and its different elements;
2. They were introduced to the CMMN modeler and were told about what our system can do (recognize hand-drawings and put them into a format that is appropriate for a formal modeling tool);
3. They were told to make sure that their drawings had closed shapes;
4. They were told that they could use any painting software and pointing device;
5. They were told not to worry about scale or paint brush thickness. However, they could not use the air brush, which leaves gaps between paint points;
6. Finally, they were told to "draw normally," without trying to be particularly precise.

4.1 Test Setting Overview

After explaining the requirements, the subjects were asked to draw 5 of each CMMN model primitives (lines, events, tasks, files, entry criteria (diamond)) as well as composite shapes, as shown in figure 4.1, to prepare for the experiment. For example, one task and one entry criterion connected to a line that is called model fragments.

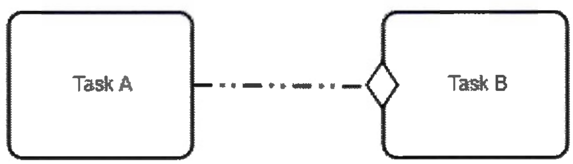


Figure 4.1 Sentry based dependency between two tasks

We began by measuring the recognition performance for primitive shapes. Recall that the way matching works is by comparing each CMMN input shape to the category of predefined shapes, and measuring the similarity between the input shape and the predefined CMMN shapes. The predefined shape that achieves the highest similarity value is assumed to be the intended shape. The next table shows a square matrix where each row corresponds to an input shape, each column represents a predefined shape, and cell *(x, y)* represents the percentage of times that predefined shape *y* was found to be the best match for input shape *x*. Thus, shape *(x, x)* represents the percentage of accurate recognitions.

With regards to composite shapes, the recognition performance depends on a combination of:

1. The recognition of the primitive shapes

2. The accuracy of the calculations of the spatial relationships between the primitive shapes, and the strength of the inferences drawn from such relationship

Table 4.1 Results of primitive shapes recognition:The drawn (expected) shape is shown on the left and the recognized shape at the top

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

Table 4.2 Results of model fragments recognition: The drawn (expected) shape is shown on the left and the recognized shape as part of model fragments at the top

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

4.2 Recognition Accuracy

The recognition rate for a test set of 500 drawings, composed of both primitive shapes and composite shapes, is presented in tables 4.1 and 4.2. table 4.1 shows significant differences between recognition rates for the different primitive shapes, ranging from 30% for a file, to a 98% for an event. The difference is due, in good part, to the distinctiveness of the shapes. For example, there is a a big similarity between files and tasks, which are both rectangular, but with a file having a clipped angle (Figure 4.3c, Figure 4.3d). Hence, the recognition rate of the file was the lowest of all shapes (30%). Indeed, many files (68%) were actually recognized as tasks. The opposite is not true. This may be due to the prevalence of files in the 500 handrawings. However, generally speaking, the average performance is acceptable.

Note also that lines are the easiest shapes to recognize, with a 100% rate. Indeed, the sequence of points that can form a line is specified by three parameters that include *threshold*, *minLinLength* and *maxLineGap*. These parameters were adjusted in order to accept the shortest line with the minimum number of intersections to constitute a line.

Events also had a high recognition rate. The cases where the sketch was not recognized (figure 4.3a) were mostly due to the event being drawn similar to a square.

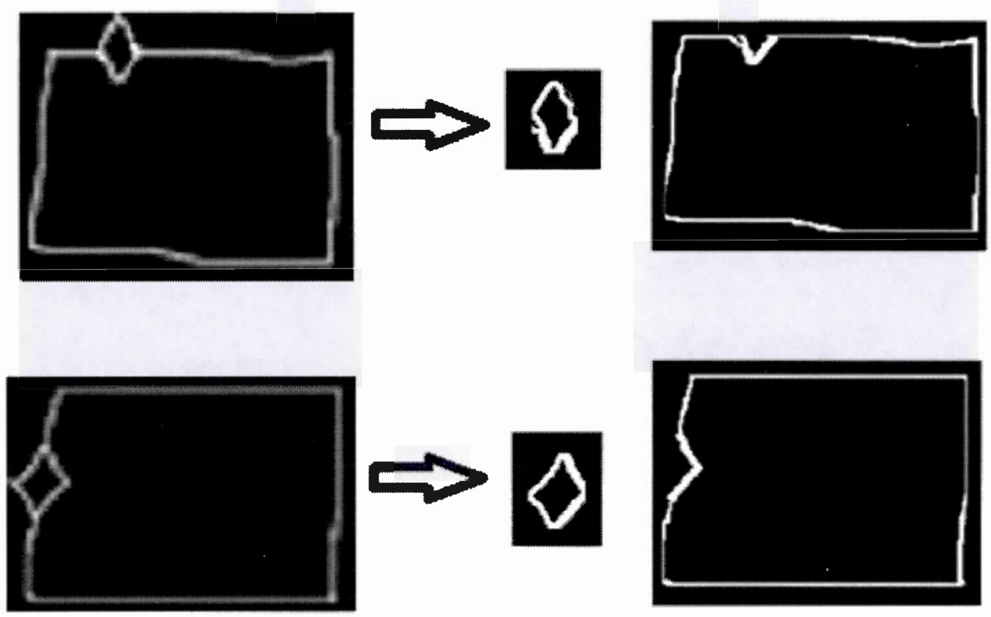
The recognition rate for entry criteria (diamond) was also high the cases that were not recognized (Figure 4.3b) were often due to the fact that the entry criterion resembled an event that was rotated 45 degrees.

Sentry based dependency between two tasks, as shown in figure 4.1, represents as model fragments that consist of primitive shapes and one composite shape, itself consisting of an entry criterion (diamond) that is part of a task.

The recognition rate for entry criteria (diamond) as primitive shapes (see table 4.1) is 92%, while its recognition rate in the second table (see table 4.2) as part of a composite shape, is noticeably reduced (80%).

According to the definition structure of algorithm 8 in chapter III, recognition entry criterion (diamond) as part of the composite shape is completely dependent on the recognition rate of Task B. Hence, we need to recognize a task as Task B and then go into the second level that is recognized entry criterion (diamond) as part of Task B.

As shown in table 4.2, recognition rate of Task B as a task is 86% and as a file is 14%. Due to the fact that the Task B, As shown in figure 3.6, was recognized as an inner contour (see figure 4.2), the similarity of the Task B drawing to the Task template was decreased. By extension, the recognition rate of Task B affects the recognition rate of the entry criterion (diamond).

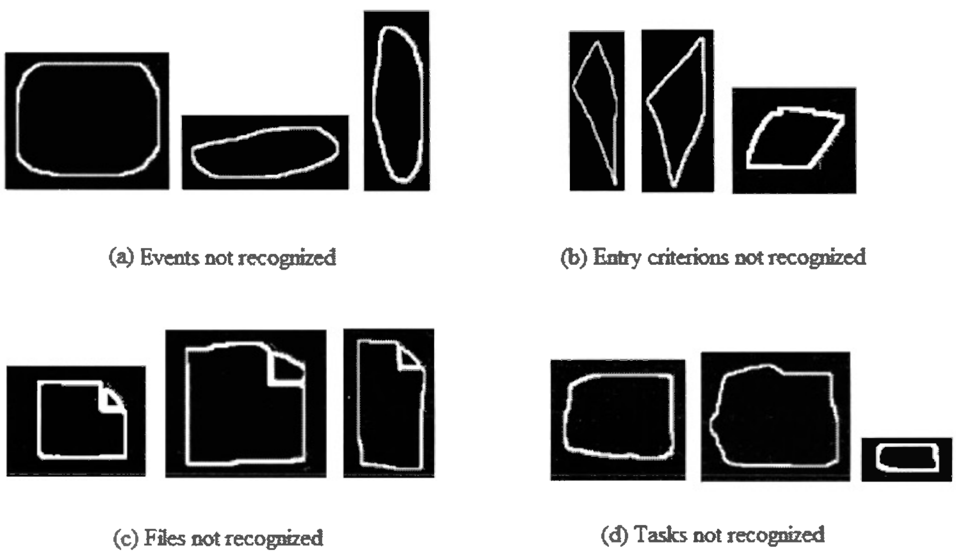


a) Exterior contour (parent) b) Interior contours (children)

Figure 4.2 Recognize Tasks B and Diamonds as inner contours based on figure 3.6

There are other reasons for the low recognition rate of entry criteria (diamond). First, there is the way that humans draw sketches by hand. Because drawing the diamond as part of the composite shape is more difficult than drawing it as a primitive shape, the similarity of a diamond to the event template, as shown in table 4.2, is increased and many diamonds will end up being recognized as events. The second reason why entry criteria do not have good recognition rate is related to the bounding box around each diamond. As shown in figure 4.2, the diamond vertices are smoothed by bounding box edges. Hence, the similarity of the diamond to the event will be increased.

While the recognition rate of the individual primitive shapes is high (on average), the recognition rate for the aggregates is in the [72% - 80%] range. This is to be expected, considering the way our algorithm works.



(a) Events not recognized

(b) Entry criterions not recognized

(c) Files not recognized

(d) Tasks not recognized

Figure 4.3 Samples elements not recognized by the system

4.3 Limitations

The current implementation has a number of limitations. First, it is sensitive to rotation. There are ways to change the algorithms to make them rotation proof. However, this would complicate the recognition of sentries (diamonds) which relies on their 45 degree tilting. We would then need to take into account the relative size and position of the shapes we need to distinguish, for example between tasks and entry/exit criteria, which lead us to the next possible improvement.

The second limitation involves recognizing the text. In chapter II, we talked about optical character recognition and offline character recognition as one of its subcategories, which includes overall stages such as pre-processing, segmentation, feature extraction, and classification.

Recognizing text within the context of geometric figure drawings, as is the case with CMMN (or other types) of models, is more complicated than recognizing text within purely textual documents. We envision a multi-step process, as explained next. The first step would be common with shape recognition, and would consist of pre-processing and object detection. A second step would consist of separating contours that contain text from other shapes. A third step would extract the text from shapes. A fourth step would classify the text based on the simplest category (the digits [0-9] and letters of the English alphabet [a-z]). The fifth step would reconstruct the text by putting together all the letters and numbers to construct strings in the right positions that can be within or outside the shapes. The last step would attach those strings as labels to the shapes within which they appear. We would need to implement and experiment with such a system to see how well it works.

Finally, note that we gave the experimental subjects some directives about how to do their handsketches. For example, we asked them to make sure they "close their shapes", to ease contour computation and shape recognition, although we used various thresholds to make sure that our algorithms can complete or "close" imperfectly closed shapes. Also, the experiemental subjects were instructed not to use "paint brush" or thick pens when drawing. These are not serious limitations, but we cannot say that we performed the experiments on totally natural drawings.

CONCLUSION

Knowledge worker in order to hold a collection of business documents and other information relevant to their business processes; need to use case management systems as the primary building. Within the context of case management, Case Management Model and Notation (CMMN) support the representation of a wide range of knowledge worker activities to manage social work and related application areas such as insurance claim processes, healthcare processes, lawsuit services processes, social services process, etc. (Marin *et al.*, 2012).

Using CMMN as a formal modeling tool at early requirements activities in the software development life cycle is not efficient as well as flexible approaches such as office tools or whiteboards composed of some restrictions that include lack of consistency management, published changes, or information migration. Hence a new intermediate approach in order to reduce the gap between these two approaches is required.

The purpose of our research is to develop a new intermediate approach in order to reduce the gap between these two approaches. Our approach reads early hand sketches of CMMN models and transforms them into a format that can be imported into a formal CMMN modeling tool. In this thesis, we presented in detail a set of algorithms to extract and recognize the contours of CMMN models hand-drawn sketches of primitive shapes and composite shapes. Different pre-processing algorithms were applied to the input sketches to prepare them for recognition. We can present the list of steps as below:

- Retrieve contours from a hand-drawn sketch;
- Clarify contours using pattern modeling algorithms;
- Identify a spatial relationship between primitive shapes;
- And use their relationships to recognize composite shapes.

To implement our system, we used the OpenCV library, which provides a rich set of image processing functions. While our system was geared for recognition of CMMN hand sketches, it could easily be parameterized to recognize hand sketches in any graphical modeling language, provided that the graphical icons used for the various elements are reasonably distinguishable.

At present, the algorithm first recognizes primitive shapes and then composite shapes. The better algorithm could use a two-way recognition algorithm that refines the recognition of the individual primitive shapes based on following of the composite shape.

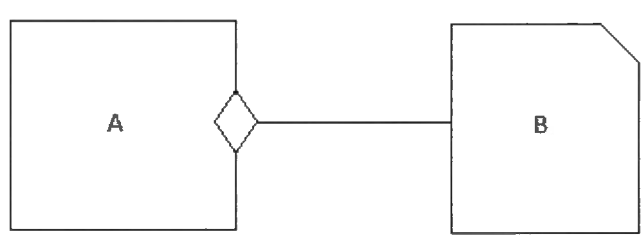


Figure 4.4 Recognizes file instead of task in composite shape

As shown in figure 4.1, even if our system finds that a file is the best match for B, because sentries are only used to link tasks, we can revise the classification of B as a task. Of course, the human could be making a modeling mistake, but we believe that the interplay between the two will enhance the recognition rate for composite shapes.

On the other hand, the human user has a main role in implementing the hand-drawn sketch. Hence, different ways that a user chooses to draw the shapes, or even write the text, can be identified. Investigation on human user mannerisms can affect the performance of recognition algorithms. Consequently, assessing these aspects would require interviewing the subjects and monitoring their behavior, as well as utilizing a more realistic test setting.

APPENDICE A

PRIMITIVE SHAPE RECOGNITION JAVA CLASSES

```java
package CMMNElementsketchRecognitionSystem;

/**
 *  This class composed of the main method, start listing all files
 *  from the main directory and its sub directories and then calling the class <code>TemplateMatchingDemo</code>
 *  in order to use the template matching method in the OpenCV library and comparing each shape
 *  of the original image to the template images that is specified in the main directory
 *  and its sub directories. in the following the class of WriteXmlFile start writhing the XML file
 *  according to the structure of CMMN modeler which is able to import the XML file inside the CMMN Modeler software
 *  @author SaraAmirsardari
 *
 */
public class SketchRecognition {

    public static final int LINE_DETECTION_TRESHOLD =10;
    public static final int MIN_LINE_LENGTH=7;
    public static final int MAX_LINE_GAP=30;

    public static void main(String[] args) {

        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);

        // reading the folders and sub folders
        ReadFolders tc = new ReadFolders();
        File MainDirectory = new File("C:/Users/SARA/Desktop/opencv/sample/template100");
        ArrayList<String> pathsList = new ArrayList<>();
        pathsList = tc.readDir(MainDirectory);
        TemplateMatchingDemo md = new TemplateMatchingDemo();
        // define for loop in order to read the files with the suffix of JPG or PNG
        ArrayList<String> listOfTemplates=new ArrayList<>();
```

```
            for (int i = 0; i < pathsList.size(); i++) {
                if (pathsList.get(i).contains("png") ||
                pathsList.get(i).contains("jpg")) {
                    listOfTemplates.add(pathsList.get(i));
                }
            }

            md.preprocessAllTemplates(listOfTemplates);
            String sketchFileName =
            "C:/Users/SARA/Desktop/opencv/sample/s.png";
            md.runMatchingDemo(sketchFileName);
            DetectLine dl = new DetectLine();
            dl.setInitialImage(md.getCleanedUpImage());
            dl.setShapesToRemove(md.getParentContrours());
            dl.detectLine();
            WriteXmlFile writeXmlShapes = new WriteXmlFile();
            writeXmlShapes.setResultOfLines(dl.getResultOfLines());
            writeXmlShapes.WriteXml(md);

        }
    }
```

```java
package CMMNElementsketchRecognitionSystem;

import java.io.File;
import java.util.ArrayList;

/**Contains some methods to list files and folders from a
directory
 * @author SaraAmirsardari
 */

public class ReadFolders{

    /**
     * get the path of main directory
     * @param f main directory path to be listed
     */

    public void readFile(File f) {
        System.out.println(f.getPath());
    }

    /**
     * List all files from a directory and its sub directories
     * @param f sub directory paths to be listed
     * @return pathList of all directory and its sub directories
     */
    public ArrayList<String> readDir(File f) {

        File subdir[]=f.listFiles();
        ArrayList<String> pathsList= new ArrayList<>();

        //verify the sub directory is file or is directory
        for(File f_arr:subdir) {

            if(f_arr.isFile()) {

                //if the sub directory is file so read the file path
                pathsList.add(f_arr.getPath());
                this.readFile(f_arr);
```

```
            }

            if (f_arr.isDirectory()){

               //if the sub directory is a directory, so list all
               files path inside the directory
               ArrayList<String> dirFiles = this.readDir(f_arr);
               pathsList.addAll(dirFiles);
            }
         }
         return pathsList ;
      }

   }
```

```java
package CMMNElementsketchRecognitionSystem;

import java.io.File;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import java.util.Vector;

import org.opencv.core.Core;
import org.opencv.core.Core.MinMaxLocResult;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.MatOfPoint;
import org.opencv.core.Point;
import org.opencv.core.Range;
import org.opencv.core.Rect;
import org.opencv.core.Scalar;
import org.opencv.core.Size;
import org.opencv.highgui.Highgui;
import org.opencv.imgproc.Imgproc;
import org.opencv.utils.Converters;
import org.w3c.dom.css.RGBColor;

/**
 * This class does the matching process between input image
 and template's images that include the following steps:
 * 1) First of all, start doing pre-processing on the input
 image and the template's images
 * 2) extract features of each contour in the input image
 * 3) resize the size of each contour according to the width
 of template image as well as keeping the aspect ratio
 * 4) At the end start finding the best match between each
 contour and template's images
 * The OpenCV library is used in order to do the
 Pre-processing process as well as doing the template matching
 * @author SaraAmirsardari
```

```
 *
 */
class TemplateMatchingDemo {

    private List<MatOfPoint> parentContrours = new ArrayList<MatOfPoint>();
    private Mat cleanedUpImage;

    public Mat getCleanedUpImage() {
        return cleanedUpImage;
    }

    public List<MatOfPoint> getParentContrours() {
        return parentContrours;
    }

    public void setParentContrours(List<MatOfPoint> parentContrours) {
        this.parentContrours = parentContrours;
    }

    private int id = -1;
    public int nextId() {
        id = id + 1;
        return id;
    }

    private int sh = -1;

    public int nextShape() {
        sh = sh + 1;
        return sh;
    }

    /**
     * this variable will hold the list of templates,organized
     by shape type/name
     *
     */
    private HashMap<String, Mat> templateTable = new HashMap<String,
    Mat>();
```

```

/**
 * This variable will contain arrays of coordinates of the various shapes
 * recognized in the input figure, organized by shape type/name
 *
 */
private HashMap<String, ArrayList<CoordinatesOfContours>> shapeCoordinates = null;

public ArrayList<CoordinatesOfContours> getListOfCoordinatesOfShapesOfType(String shapeName) {
    return shapeCoordinates.get(shapeName);
}

/**
 * This class represents a bitmap that was already segmented. The actual
 * bitmap is in <code>segmentedBitMap</code> and the contours are
 * represented in the instance variable <code>contours</code>.
 *
 * @author SaraAmirsardari
 *
 */

class SegmentedImage {
    public Mat segmentedBitMap;

    public ArrayList<MatOfPoint> contours;

    public SegmentedImage(Mat bitmap, ArrayList<MatOfPoint> listOfContours) {
        segmentedBitMap = bitmap;
        contours = listOfContours;
    }
}
```

```
/**
 * This function does pre-processing process(gaussian
 blurring, thresholding)
 * in a picture file (JPEG or PNG) to prepare them for
 matching.
 * the file name is contained in the string inFile. It
 first loads
 * the "bitmap" from the file <code>inFile</code> that
 applies filters to it.
 * @param
 * @return the processed image matrix
 */
public Mat preprocessImage(String inFile) {
   // load the image and convert it to gray
   Mat img = Highgui.imread(inFile,
   Highgui.CV_LOAD_IMAGE_GRAYSCALE);
   Mat destination = new Mat(img.rows(), img.cols(), img.type());
   //blur operation reduces noise and smoothing the
   grayscale image
   Imgproc.GaussianBlur(img, destination, new Size(3, 3), 0);
   // Threshold operation which converts a grayscale image
   into a binary image
   Imgproc.threshold(destination, destination, -1, 255,
   Imgproc.THRESH_BINARY_INV + Imgproc.THRESH_OTSU);
   this.cleanedUpImage = destination.clone();
   return destination;
}

/**
 * This function gets the list of
 templates<code>listOfTemplates</code> as a
 * string and convert them to matrix and then store them
 in the ArrayList
 * <Mat> of <code>template</code>.
 * @param listOfTemplates is list of template's images
 * @return the processed template matrix to the template
 table
 */
```

```
public void preprocessAllTemplates (ArrayList<String> listOfTemplates) {

    for (int i = 0; i < listOfTemplates.size(); i++) {
        String nextTemplateFileName = listOfTemplates.get(i);
        // find the name of the file, without the .jpg or
        .png extension. That file name will represent the
        name of the
        // CMMN construct (file, sentry, event, etc. Here is
        an example of what it looks like
        //
        C:\Users\SARA\Desktop\opencv\sample\template100\task\ta
        sk.png. first, separate file name based on \, and
        remove extension

        String splitter = File.separator.replace("\\", "\\\\");
        String[] pathElements = nextTemplateFileName.split(splitter);
        String fileName = pathElements[pathElements.length - 1];
        String templateName = (fileName.split("\\."))[0];
        // load the template and convert it to gray
        Mat img = Highgui.imread(nextTemplateFileName,
        Highgui.CV_LOAD_IMAGE_GRAYSCALE);
        Mat destination = new Mat(img.rows(), img.cols(), img.type());
        //blur operation reduces noise and smoothing the
        grayscale template image
        Imgproc.GaussianBlur(img, destination, new Size(3, 3), 0);
        // Threshold operation which converts a grayscale
        template image into a binary template image
        Imgproc.threshold(destination, destination, -1, 255,
        Imgproc.THRESH_BINARY_INV + Imgproc.THRESH_OTSU);
        // add the processed template matrix to the template
        table
        templateTable.put(templateName, destination);
    }
}


/**
 * this function takes as input the name of a graphical
```

```
file (PNG or JPEG)
* and returns a record (an instance of
<code>SegmentedImage</code>)
* consisting of, 1) the bitmap of the segmented image,
and 2) the list of
* contours (each contour being a vector of points).
*
* @param inFile is image matrix
* @return
*/
public SegmentedImage segmentImage(String inFile) {
    Mat segmentedBitMap = preprocessImage(inFile);

    // find the contours inside input image
    Mat hierarchy = new Mat();
    ArrayList<MatOfPoint> contours = new ArrayList<MatOfPoint>();
    Imgproc.findContours(segmentedBitMap,
    contours,hierarchy,Imgproc.RETR_CCOMP,
    Imgproc.CHAIN_APPROX_NONE);
    ArrayList<MatOfPoint> contoursToRemove = new
    ArrayList<MatOfPoint>();
    for ( int idx=0; idx<contours.size(); idx++ ){
        double[] contourHierarchy = hierarchy.get(0,idx);
        if(contourHierarchy[3]!=-1){
            Imgproc.drawContours(segmentedBitMap, contours,idx, new
            Scalar(255, 255, 255),2);
            this.parentContrours.add(contours.get(idx));
        } else {
            contoursToRemove.add(contours.get(idx));
        }
    }

    for(MatOfPoint i: contoursToRemove){
        contours.remove(i);
    }

    return new SegmentedImage(segmentedBitMap, contours);
}
```

```java
/**
 * in this function, the feature of each contour inside
 the image extracted
 * @param enclosingBitmap is the input image
 * @param shape is the contour inside input image
 * @return the new contour matrix that include the contour
 feature such as start coordinate (x,y), width and height.
 */
public Mat getShapeSubBitMap(Mat enclosingBitmap, MatOfPoint shape) {

    int w = Imgproc.boundingRect(shape).width;
    int h = Imgproc.boundingRect(shape).height;
    int x = Imgproc.boundingRect(shape).x;
    int y = Imgproc.boundingRect(shape).y;
    System.out.println(x + " " + y + " " + w + " " + h + " ");
    Range rowRange = new Range(y, y + h);
    Range colRange = new Range(x, x + w);
    return new Mat(enclosingBitmap, rowRange, colRange);
}

/**
 * This function resize the contours (subShapeBitMap)
 according to the
 * template size by preserving the scale of the contours.
 * @param subShapeBitMap is contour matrix
 * @param templateWidth is width of template image
 * @return the new size of contour matrix
 */

public Size getResizeSize(Mat subShapeBitMap, double templateWidth
) {

    double scale = (double) subShapeBitMap.width() / (double)
    subShapeBitMap.height();
    double newW = templateWidth;
    double newH = newW / scale;
    return new Size(newW, newH);
}
```

```
/**
 * This function finds the matching between the contour that was defined in
 * <code>currentShapeSubBitMap</code> and the template that was defined in
 * <code>templates</code>
 *
 */

public ArrayList<FindMatching> findMatching(Mat currentShapeSubBitMap, Mat segmentedInputBitMap, int shapeId) {

    ArrayList<FindMatching> returnValue = new ArrayList<>();
    Set<String> templateNames = templateTable.keySet();
    Iterator<String> templateNameIterator = templateNames.iterator();
    ArrayList<MinMaxLocResult> results = new ArrayList<MinMaxLocResult>();

    while (templateNameIterator.hasNext()) {
        String templateName = templateNameIterator.next();

        Mat segmentedTemplateBitMap = templateTable.get(templateName);
        // resize the contour according to the template size
        Size newSize = getResizeSize(currentShapeSubBitMap, segmentedTemplateBitMap.width());
        Mat resizedImage = new Mat();
        Imgproc.resize(currentShapeSubBitMap, resizedImage, newSize);
        Mat resizedImage1 = new Mat();
        resizedImage1 = resizedImage;


        Mat biggerImage, smallerImage;
        if (resizedImage1.rows() > segmentedTemplateBitMap.rows() ||
                resizedImage1.cols() > segmentedTemplateBitMap.cols()) {
            // the image is bigger
            biggerImage = resizedImage1;
            smallerImage = segmentedTemplateBitMap;
        } else {
            // the template is bigger
```

```
            biggerImage = segmentedTemplateBitMap;
            smallerImage = resizedImage1;
        }

        int result_cols = biggerImage.cols() - smallerImage.cols() + 1;
        int result_rows = biggerImage.rows() - smallerImage.rows() + 1;
        Mat result = new Mat(result_rows, result_cols,
        CvType.CV_32FC1);
        // these two method (Imgproc.TM_SQDIFF and
        Imgproc.TM_SQDIFF_NORMED)give the minimum value
        Imgproc.matchTemplate(biggerImage, smallerImage, result,
        Imgproc.TM_CCORR_NORMED);



        ArrayList<Double> listOfMaxVal = new ArrayList<>();
        //The functions minMaxLoc find the minimum and
        maximum element values and their positions
        MinMaxLocResult mmr = Core.minMaxLoc(result);
        results.add(mmr);
        Point matchLoc = mmr.maxLoc;
        double maxValue = mmr.maxVal;
        listOfMaxVal.add(maxValue);
        double globalMaximimum = MaximumValue(listOfMaxVal);
        if (maxValue > 0) {
            FindMatching findM = new
            FindMatching(segmentedTemplateBitMap, result, matchLoc,
            maxValue,
                    templateName);
            returnValue.add(findM);
        }
    }
    return returnValue;

}

private double MaximumValue(ArrayList<Double> listOfMaxVal) {

    double maxValue = listOfMaxVal.get(0);
```

```
        for (int s = 0; s < listOfMaxVal.size(); s++) {
            if (listOfMaxVal.get(s) > maxValue)
                maxValue = listOfMaxVal.get(s);
        }
        return maxValue;
    }

    /**
     * runMtchingDemo finds the template in original image
     * @param inFile is original image
     * @param templateFile is the template image
     * @param outFile
     * @param match_method
     */

    public void runMatchingDemo(String inFile) {

        System.out.println("\nRunning Template Matching");
        System.loadLibrary(Core.NATIVE__LIBRARY__NAME);

        // find the contours in the input image
        SegmentedImage segmentedInputImage = segmentImage(inFile);
        Mat segmentedInputBitMap = segmentedInputImage.segmentedBitMap;
        ArrayList<MatOfPoint> contours = segmentedInputImage.contours;

        // let us now iterate over the different shapes/contours
        in the input image, trying to find a match in each case

        shapeCoordinates = new HashMap<String,
        ArrayList<CoordinatesOfContours>>();

        for (int i = 0; i < contours.size(); i++) {
            MatOfPoint currentShape = contours.get(i);
            Mat currentShapeSubBitMap =
            getShapeSubBitMap(segmentedInputBitMap, currentShape);
            int x = Imgproc.boundingRect(currentShape).x;
            int y = Imgproc.boundingRect(currentShape).y;
```

```
        // Doing the template matching and returning an array list of Mat with two values:
        // 1.The template that was compared to <code>currentShapeSubBitMap</code>
        // 2.The result comparison of <code>currentShapeSubBitMap</code> and <code>segmentedTemplateBitMap</code>
        ArrayList<FindMatching> results = findMatching(currentShapeSubBitMap, segmentedInputBitMap, i);
        FindMatching bestResult = computeBestResult(results);


        // get the shape name
        String shapeName = bestResult.getTemplateName();
        // add the current match to the appropriate list of shapes
        ArrayList<CoordinatesOfContours> sameShapeCoordinateArray = shapeCoordinates.get(shapeName);

        // if array is empty (this is the first shape of this type encountered in the figure) then initialize it
        if (sameShapeCoordinateArray == null) {
            sameShapeCoordinateArray = new ArrayList<CoordinatesOfContours>();
            shapeCoordinates.put(shapeName, sameShapeCoordinateArray);
        }

        CoordinatesOfContours recognizedShapeCoordinates = new CoordinatesOfContours(bestResult.getMatchLoc().x + x,
                bestResult.getMatchLoc().y + y,
                bestResult.getSegmentedTemplateBitMap().cols(),
                bestResult.getSegmentedTemplateBitMap().rows(),
                nextId());

        // Just add the recognized square to the list if it does not overlap any other
        CalculateDistanceBetweenContours t = new CalculateDistanceBetweenContours();
        if (!t.isOverlapping(recognizedShapeCoordinates,
```

```
                sameShapeCoordinateArray)) {
                    recognizedShapeCoordinates.setType(shapeName);
                    sameShapeCoordinateArray.add(recognizedShapeCoordinates);
                }

            }

            Set<String> tmpKeySet = shapeCoordinates.keySet();
            for(String key : tmpKeySet){
                ArrayList<CoordinatesOfContours> tmpCordinates =
                shapeCoordinates.get(key);
                for(CoordinatesOfContours coordinate : tmpCordinates){
                    System.err.println(coordinate);
                }
            }
        }

        FindMatching computeBestResult(ArrayList<FindMatching> results){
            FindMatching bestResult = null;
            for(FindMatching currentResult : results){
                if(bestResult==null ||
                currentResult.getMaxValue()>bestResult.getMaxValue()){
                    bestResult = currentResult;
                }
            }
            return bestResult;
        }
    }
```

```java
package CMMNElementsketchRecognitionSystem;

import org.opencv.core.Mat;
import org.opencv.core.Point;

/** This class represents:
 * 1. The templates that were already converted from string
 to Mat and stored in <code>segmentedTemplateBitMap</code>
 * 2. The result that was already received from template
 matching method in order to compare the template and contour
 together
 * 3. The match location and maximum value for each result
 that were already defined by the <code>matchLoc</code> and
 <code>maxValue</code>
 * 4. The <code>idTemplate</code> is specified for
 identifying each template
 * @author SaraAmirsardari
 *
 */

public class FindMatching {

    Mat segmentedTemplateBitMap;
    Mat result ;
    Point matchLoc;
    double maxValue;
    String templateName;

    public FindMatching(Mat segmentedTemplateBitMap, Mat result,Point
    matchLoc,double maxValue,String templateName ) {
        this.segmentedTemplateBitMap = segmentedTemplateBitMap;
        this.result = result;
        this.matchLoc=matchLoc;
        this.maxValue=maxValue;
        this.templateName=templateName;
    }

    public void setSegmentedTemplateBitMap(Mat
    segmentedTemplateBitMap) {
```

```java
        this.segmentedTemplateBitMap = segmentedTemplateBitMap;
    }
    public Mat getSegmentedTemplateBitMap() {
        return segmentedTemplateBitMap;
    }

    public void setResult(Mat result) {
        this.result = result;
    }
    public Mat getResult() {
        return result;
    }

    public void setMatchLoc(Point matchLoc) {
        this.matchLoc = matchLoc;
    }
    public Point getMatchLoc() {
        return matchLoc;
    }

    public void setMaxValue(Double maxValue) {
        this.maxValue = maxValue;
    }
    public Double getMaxValue() {
        return maxValue;
    }
    public void setTemplateName(String templateName) {

        this.templateName=templateName;
    }
    public String getTemplateName() {
        return templateName;
    }

}
```

```java
package CMMNElementsketchRecognitionSystem ;
import java.util.ArrayList;

/**
 * This class calculates the distance between each shape and
 the rest of the shapes in input image
 * @author SaraAmirsardari
 */
public class CalculateDistanceBetweenContours {

    /**
     * This method calculate the distance between two shapes
     * @param s1 is the first shape to calculate.
     * @param s2 is the second shape to calculate.
     * @return the distance between shapes.
     */

    public double calculateDistance (CoordinatesOfContours s1,
    CoordinatesOfContours s2) {

        double distance = 0;
        double dx = s1.getX() - s2.getX();
        double dy = s1.getY() - s2.getY();
        distance = Math.sqrt(dx*dx + dy*dy);
        return distance;
    }

    /**
     * This method Verifies if the shape is overlapping with
     any other shape in the list
     * @param s is the shape to compare if it is overlapping.
     * @param list is the list of all other shape that will
     compare to shape.
     * @return True if the shape overlaps any other shapes,
     false otherwise.
     */

    public boolean isOverlapping (CoordinatesOfContours s,
    ArrayList<CoordinatesOfContours> list) {
```

```
        boolean overlap = false;
        for (int i = 0; i < list.size(); i++) {
            // define a threshold for specifying the overlap
            distance between to shapes
            double threshold = 7;
            if(calculateDistance(s, list.get(i)) <= threshold) {
                overlap = true;
                return overlap;
            }
        }

        return overlap;
    }

}
```

Figure a.4 Represent 'coordinate' java class

```
package CMMNElementsketchRecognitionSystem;

/**
 * Interface for all the coordinates.
 * @author SaraAmirsardari
 */
public interface Coordinates {

    public double getX();
    public double getY();
    public double getWidth();
    public double getHeight();

}
```

```java
package CMMNElementsketchRecognitionSystem;

import org.opencv.core.Point;

/**
 * This method gets and sets the start point and end point of
 each contour edge
 * @author SaraAmirsardari
 *
 */
public class CoordinatesOfContourEdge {
   private Point startLine;
   private Point endLine;


   public Point getStartLine() {
      return startLine;
   }
   public void setStartLine(Point startLine) {
      this.startLine = startLine;
   }
   public Point getEndLine() {
      return endLine;
   }
   public void setEndLine(Point endLine) {
      this.endLine = endLine;
   }
}
```

```java
package CMMNElementsketchRecognitionSystem;

import org.opencv.core.Point;

public class CoordinatesOfContours implements Coordinates {

    private double x;
    private double y;
    private double width;
    private double height;
    private int id;
    private String type = "";

    /**
     * This class defines the coordinate of  shapes inside the
     input image
     */
    public CoordinatesOfContours(double x, double y, double width,
    double height, int id) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.id = id;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public double getWidth() {
        return width;
    }

    public double getHeight() {
```

```
        return height;
    }
    public int getid() {
        return id;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public String toString(){
        return "Coordinate: ("+this.x+","+this.y+"),
        Width:"+this.width+", Height:"+this.height;
    }

    public CoordinatesOfContourEdge getTopLine(){
        CoordinatesOfContourEdge result = new CoordinatesOfContourEdge();
        result.setStartLine(new Point(x,y));
        result.setEndLine(new Point(x+width, y));
        return result;
    }

    public CoordinatesOfContourEdge getBottomLine(){
        CoordinatesOfContourEdge result = new CoordinatesOfContourEdge();
        result.setStartLine(new Point(x,y+height));
        result.setEndLine(new Point(x+width, y+height));
        return result;
    }

    public CoordinatesOfContourEdge getLeftLine(){
        CoordinatesOfContourEdge result = new CoordinatesOfContourEdge();
        result.setStartLine(new Point(x,y));
        result.setEndLine(new Point(x, y+height));
        return result;
    }
```

```
    public CoordinatesOfContourEdge getRightLine() {
        CoordinatesOfContourEdge result = new CoordinatesOfContourEdge();
        result.setStartLine(new Point(x+width,y));
        result.setEndLine(new Point(x+width, y+height));
        return result;
    }

}
```

APPENDICE B

COMPOSITE SHAPE RECOGNITION JAVA CLASSES

```
package CMMNElementsketchRecognitionSystem;

/**
 * This class gets and sets the coordinates of lines
 * @author SaraAmirsardari
 */
public class CoordinatesOfLines{


    private double x1;
    private double y1;
    private double x2;
    private double y2;
    private int id;

    /**
     * This class defines the coordinate of each line inside
     the input image
     */

    public CoordinatesOfLines(double x1, double y1, double x2, double
    y2,int id) {

        this.x1 = x1;
        this.y1 = y1;
        this.x2 = x2;
        this.y2 = y2;
        this.id = id;
    }

    public double getX1() {
        return x1;
    }

    public double getY1() {
        return y1;
    }

    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+")";
    }
}
```

```java
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;
    }
    /**
```

```
     * This method start defining the bounding box around each
     closed shape
     * and then using threshold in order to increase the area
     of each closed shape
     * @param shape
     */
    private void removeShape(MatOfPoint shape){

        int x = Imgproc.boundingRect(shape).x;
        int y = Imgproc.boundingRect(shape).y;
        int width = Imgproc.boundingRect(shape).width;
        int height = Imgproc.boundingRect(shape).height;
        MatOfPoint mpoints = new MatOfPoint();
        double threshold = 8;
        List<Point> points = new ArrayList<Point>();
        points.add(new Point(x-threshold,y-threshold));
        points.add(new Point(x+width+threshold,y-threshold));
        points.add(new Point(x+width+threshold,y+height+threshold));
        points.add(new Point(x-threshold,y+height+threshold));

        mpoints.fromList(points);
        //paint all closed contours by black color
        Core.fillConvexPoly(this.initialImage, mpoints,new Scalar(0,0,0));
    }
    public void detectLine() {

        this.ResultOfLines = new ArrayList<>();
        for(MatOfPoint shape: this.shapesToRemove){
            removeShape(shape);
        }

        //      image - 8-bit, single-channel binary source
        image. The image may be modified by the function.
        //    lines - Output vector of lines. Each line is
        represented by a 4-element vector  (x_1, y_1, x_2, y_2),
        //      where  (x_1,y_1) and  (x_2, y_2) are the ending
        points of each detected line segment.
        //    rho : The resolution of the parameter r in pixels.
        We use 1 pixel.
```

```
        //    theta: The resolution of the parameter theta in radians. We use 1 degree (CV_PI/180)
        //    threshold: The minimum number of intersections to "detect" a line
        //    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.

        Mat line = new Mat();
        int threshold = SketchRecognition.LINE_DETECTION_TRESHOLD;
        int minLineLength =SketchRecognition.MIN_LINE_LENGTH;
        int maxLineGap =SketchRecognition.MAX_LINE_GAP;
        int id=0;
        Imgproc.Canny(this.initialImage, this.initialImage, 50, 200);
        Imgproc.HoughLinesP(this.initialImage, line, 1, Math.PI/180, threshold, minLineLength, maxLineGap);

        for(int i = 0; i < line.cols(); i++) {
            double[] val = line.get(0, i);
            double  x1 = val[0],
                    y1 = val[1],
                    x2 = val[2],
                    y2 = val[3];

            CoordinatesOfLines recognizeLine = new CoordinatesOfLines(x1, y1, x2, y2,id);
            CalculateDistanceBetweenLines linedistance = new CalculateDistanceBetweenLines();
            linedistance.mergingLines(recognizeLine, ResultOfLines);

        }

    }

}
```

```java
package CMMNElementsketchRecognitionSystem;
import java.util.ArrayList;
import java.util.Collections;

/**
 * This class calculates the distance between each line and
 the rest of the lines in input image
 * @author SaraAmirsardari
 */

public class CalculateDistanceBetweenLines {

    /**
     * This method calculate the distance between two lines.
     * @param line1 is the first line to calculate.
     * @param line2 is the second line to calculate.
     * @return
     */
    public double calculateDistance(CoordinatesOfLines line1,
    CoordinatesOfLines line2) {

        double distance = 0;
        double dx = line1.getX1() - line2.getX1();
        double dy = line1.getY1() - line2.getY1();
        distance = Math.sqrt(dx*dx + dy*dy);
        return distance;
    }

    /**
     * This method merges the lines according to their distance
     * @param line is the first line to compare its distance
     with the rest of line in list
     * @param list is the list of all lines in input image
     * this method verifies:
     * first: the distance of the two lines that is less that
     threshold or not,
     * second: if it is less than the threshold, it starts
     merging two lines based on
     * 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;
    }
}
```

```
package CMMNElementsketchRecognitionSystem;
/**
 * This class calculates the distance of start point and end point of each line with two specified shapes( task and sentry).
 * Hence, we need to get the coordinates of lines from <code>detectLine</code> class as well as
 * the coordinates of tasks and sentries from <code>WriteXmlFile</code> class
 * @author SaraAmirsardari
 *
 */
public class Connector {

    private ArrayList<CoordinatesOfLines> resultOfLines= new ArrayList<>();
    private ArrayList<CoordinatesOfContours> resultOfTasks= new ArrayList<>();
    private ArrayList<CoordinatesOfContours> resultOfSentries= new ArrayList<>();


    public void setResultOfLines (ArrayList<CoordinatesOfLines> resultOfLines) {
        this.resultOfLines = resultOfLines;
    }
    public ArrayList<CoordinatesOfLines> getResultOfLines () {
        return resultOfLines;
    }

    public void setResultOfTasks (ArrayList<CoordinatesOfContours> resultOfTasks) {
        this.resultOfTasks = resultOfTasks;
    }
    public ArrayList<CoordinatesOfContours> getResultOfTasks () {

        return resultOfTasks;
    }

    public void setResultOfSentries (ArrayList<CoordinatesOfContours>
```

```
resultOfSentries) {
    this.resultOfSentries = resultOfSentries;
}

public ArrayList<CoordinatesOfContours> getResultOfSentries() {

    return resultOfSentries;
}

/**
 * This method calls the
 <code>findConnexionForPoint</code> method in order to
 calculate
 * the distance of start point and end point of line with
 the specified shapes
 * @param line is the coordinate of each line
 * @return the result which includes the list of shapes
 connected to the line
 */

public List<CoordinatesOfContours>
getCloserShapeForLine(CoordinatesOfLines line) {

    List<CoordinatesOfContours> result= new ArrayList<>();
    result.add( findConnexionForPoint(new Point(line.getX1(),
    line.getY1()) ) );
    result.add( findConnexionForPoint(new Point(line.getX2(),
    line.getY2()) ) );
    return result;
}

/**
 *
 * @param p is one of the start point or end point of line
 * this method calculate the distance of start point or
 end point of line with the specified shapes
 * @return the minimum distance of each start point or end
 point of line with the specified shapes
 */
```

```
public CoordinatesOfContours findConnexionForPoint(Point p){
    List<DistanceFromContourToLine> distances = new ArrayList<>();
    for(CoordinatesOfContours task : this.resultOfTasks){
        DistanceFromContourToLine distance = new
        DistanceFromContourToLine();
        distance.setShape(task);
        distance.setDistance(computeDistance(task, p ));
        distances.add(distance);
    }
    for(CoordinatesOfContours sentry : this.resultOfSentries){
        DistanceFromContourToLine distance = new
        DistanceFromContourToLine();
        distance.setShape(sentry);
        distance.setDistance(computeDistance(sentry, p));
        distances.add(distance);
    }

    DistanceFromContourToLine minimalDistance = null;
    for(DistanceFromContourToLine distance : distances){
        if(minimalDistance == null ||
        distance.getDistance()<minimalDistance.getDistance()){
            minimalDistance = distance;
        }
    }
    return minimalDistance.getShape();
}

/**
 * This class computes the distance of start point and end
 point of line with two
 * edges of closed counter that can be vertical or
 horizontal
 */
private double computeDistanceToLine(Point pointToCompute,
CoordinatesOfContourEdge contourEdge){

    double threshold = 5;
    Point lineStart = contourEdge.getStartLine();
    Point lineEnd = contourEdge.getEndLine();
```

```
        boolean isHorizontal = (lineStart.y == lineEnd.y);
        if(isHorizontal){
           //It is a horizontal line, Make sure that lineStart
           has smaller x
           if(lineStart.x>lineEnd.x){
              Point p = lineStart;
              lineStart = lineEnd;
              lineEnd = p ;
           }
           //If the point is not between the start point of line
           and end point of line then return infinite value
           if(pointToCompute.x<lineStart.x-threshold ||
           pointToCompute.x>lineEnd.x+threshold){
              return Double.MAX__VALUE;
           }else{
              //If the point is between the start point of line
              and end point of line  then calculate the distance

              return Math.abs(lineStart.y-pointToCompute.y);
           }
        } else {
           //It is a vertical line, Make sure that lineStart has
           smaller y
           if(lineStart.y>lineEnd.y){
              Point p = lineStart;
              lineStart = lineEnd;
              lineEnd = p ;
           }
           //If the point is not between the start point of line
           and end point of line  then return infinite value
           if(pointToCompute.y<lineStart.y-threshold ||
           pointToCompute.y>lineEnd.y+threshold){

              return Double.MAX__VALUE;
           }else{

              //If the point is between the start point of line
              and end point of line  then calculate the distance
              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;

    }
}
```

```java
package CMMNElementsketchRecognitionSystem;

import org.opencv.core.Point;

/**
 * This class gets and sets the coordinate of lines and the
 shapes that are connected to lines
 * @author SaraAmirsardari
 *
 */
public class ConnectorResult {

    CoordinatesOfContours shape;
    Point linePoint;

    public ConnectorResult(CoordinatesOfContours shape, Point linePoint){
    this.shape=shape;
    this.linePoint=linePoint;

    }
    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;
    }
}
```

APPENDICE C

SEMANTIC CONNECTION RECOGNITION JAVA CLASSES

```java
package CMMNElementsketchRecognitionSystem;
/**
 * This class start writhing the XML file according to the
 structure of CMMN modeler
 * which is able to import the XML file inside the CMMN
 Modeler software
 * @author SaraAmirsardari
 *
 */
public class WriteXmlFile {

    private Map<CoordinatesOfContours, CoordinatesOfContours>
    connectionsMap = new HashMap<>();
    private Map<CoordinatesOfContours, CoordinatesOfLines> shapesToLines
    = new HashMap<>();

    private ArrayList<CoordinatesOfContours> resultOfTasks= new
    ArrayList<>();
    public ArrayList<CoordinatesOfContours> getResultOfTasks(){
        return resultOfTasks;
    }

    private ArrayList<CoordinatesOfContours> resultOfSentries= new
    ArrayList<>();
    public ArrayList<CoordinatesOfContours> getResultOfSentries(){
        return resultOfSentries;
    }

    private ArrayList<CoordinatesOfLines> resultOfLines= new ArrayList<>();
    public ArrayList<CoordinatesOfLines> getResultOfLines() {

        return resultOfLines;
    }

    public void setResultOfLines(ArrayList<CoordinatesOfLines>
    resultOfLines) {
        this.resultOfLines = resultOfLines;
    }

```

```
public void WriteXml(TemplateMatchingDemo md) {

    try {

        DocumentBuilderFactory docFactory =
        DocumentBuilderFactory.newInstance();
        DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

        // root elements
        Document doc = docBuilder.newDocument();
        doc.setXmlStandalone(true);
        Element rootElement = doc.createElement("cmmn:definitions");
        doc.appendChild(rootElement);

        // staff elements

        Element staff1 =
        doc.createElement("cmmn:caseFileItemDefinition");
        rootElement.appendChild(staff1);

        Element staff = doc.createElement("cmmn:case");
        rootElement.appendChild(staff);

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

        //set attribute for root element

        Attr defv1 = doc.createAttribute("author");
        defv1.setValue("");
        rootElement.setAttributeNode(defv1);

        Attr defv2 = doc.createAttribute("exporter");
        defv2.setValue("CMMN Modeler");
        rootElement.setAttributeNode(defv2);

        Attr defv3 = doc.createAttribute("id");
        defv3.setValue("_bcc573eb-adf3-4fb4-abb5-434ae50ac5ce");
        rootElement.setAttributeNode(defv3);
```

```
Attr defv4 = doc.createAttribute("name");
defv4.setValue("Drawing 1");
rootElement.setAttributeNode(defv4);

Attr defv5 = doc.createAttribute("targetNamespace");

defv5.setValue("http://www.trisotech.com/cmmn/definitions/_bcc573eb-adf3-4fb4-abb5-434ae50ac5ce");
rootElement.setAttributeNode(defv5);

Attr defv6 = doc.createAttribute("xmlns");

defv6.setValue("http://www.trisotech.com/cmmn/definitions/_bcc573eb-adf3-4fb4-abb5-434ae50ac5ce");
rootElement.setAttributeNode(defv6);

Attr defv7 = doc.createAttribute("xmlns:dc");

defv7.setValue("http://www.omg.org/spec/CMMN/20151109/DC");
rootElement.setAttributeNode(defv7);

Attr defv8 = doc.createAttribute("xmlns:trisofeed");
defv8.setValue("http://trisotech.com/feed");
rootElement.setAttributeNode(defv8);

Attr defv9 = doc.createAttribute("xmlns:triso");

defv9.setValue("http://www.trisotech.com/2015/triso/modeling");
rootElement.setAttributeNode(defv9);

Attr defv10 = doc.createAttribute("xmlns:di");

defv10.setValue("http://www.omg.org/spec/CMMN/20151109/DI");
rootElement.setAttributeNode(defv10);
```

```
Attr defv11 = doc.createAttribute("xmlns:rss");
defv11.setValue("http://purl.org/rss/2.0/");
rootElement.setAttributeNode(defv11);

Attr defv12 = doc.createAttribute("xmlns:cmmndi");

defv12.setValue("http://www.omg.org/spec/CMMN/20151109/CM
MNDI");
rootElement.setAttributeNode(defv12);

Attr defv13 = doc.createAttribute("xmlns:trisob");

defv13.setValue("http://www.trisotech.com/2014/triso/bpmn
");
rootElement.setAttributeNode(defv13);

Attr defv14 = doc.createAttribute("xmlns:cmmn");

defv14.setValue("http://www.omg.org/spec/CMMN/20151109/MO
DEL");
rootElement.setAttributeNode(defv14);

Attr defv15 = doc.createAttribute("xmlns:xsi");

defv15.setValue("http://www.w3.org/2001/XMLSchema-instanc
e");
rootElement.setAttributeNode(defv15);

Attr defv16 = doc.createAttribute("xmlns:trisocmmn");

defv16.setValue("http://www.trisotech.com/2014/triso/cmmn
");
rootElement.setAttributeNode(defv16);

//finish set attribute for root element

// get the coordinate of contours which are matched
with the template's images

```

```
ArrayList<CoordinatesOfContours> Filelist =
md.getListOfCoordinatesOfShapesOfType("file");
if(Filelist==null) Filelist = new
ArrayList<CoordinatesOfContours>();

ArrayList<CoordinatesOfContours> squarelist =
md.getListOfCoordinatesOfShapesOfType("task");
if(squarelist==null) squarelist = new
ArrayList<CoordinatesOfContours>();

this.resultOfTasks=squarelist;

ArrayList<CoordinatesOfContours> Sentrieslist =
md.getListOfCoordinatesOfShapesOfType("sentry");
if(Sentrieslist==null) Sentrieslist = new
ArrayList<CoordinatesOfContours>();

ArrayList<CoordinatesOfContours> Eventlist =
md.getListOfCoordinatesOfShapesOfType("event");
if(Eventlist==null) Eventlist = new
ArrayList<CoordinatesOfContours>();

// set attribute to staff1
element(caseFileItemDefinition)
for (CoordinatesOfContours recognizefile : Filelist) {
    writeFileItemDefinition(doc,staff1, recognizefile);
}

// set attribute to staff element(case)
Attr attr = doc.createAttribute("id");

attr.setValue("Case_3b0a4c03-c271-47c3-9e87-30c57c034fdb")
;
staff.setAttributeNode(attr);

Attr attr1 = doc.createAttribute("name");
attr1.setValue("Page 1");
staff.setAttributeNode(attr1);
```

```
// set attribute to casefilemodel
Element casefilemodel =
doc.createElement("cmmn:caseFileModel");
staff.appendChild(casefilemodel);

for (CoordinatesOfContours recognizefile : Filelist) {
    writeFileItem(doc, casefilemodel, recognizefile);
}

Element caseplanmodel =
doc.createElement("cmmn:casePlanModel");
staff.appendChild(caseplanmodel);

// set attribute to caseplanmodel element
Attr caseplan = doc.createAttribute("id");

caseplan.setValue("_3b0a4c03-c271-47c3-9e87-30c57c034fdb")
;
caseplanmodel.setAttributeNode(caseplan);

Attr caseplan1 = doc.createAttribute("autoComplete");
caseplan1.setValue("false");
caseplanmodel.setAttributeNode(caseplan1);

Attr caseplan2 = doc.createAttribute("name");
caseplan2.setValue("Page 1");
caseplanmodel.setAttributeNode(caseplan2);

// calculate if there is an intersection between
square and diamond or not
ArrayList <CoordinatesOfContours> intersectionSentries= new
ArrayList<>();
for (CoordinatesOfContours coordinatesOfSquare : squarelist) {
    // define the list of sentries that have
    intersection with squares

    for (CoordinatesOfContours coordinatesOfSentries :
    Sentrieslist) {

```

```
            if(CalculateIntersectionArea.recognizeIntersection(coordinates
            OfSentries,coordinatesOfSquare)){
                intersectionSentries.add(coordinatesOfSentries);
            }
        }

        this.resultOfSentries=intersectionSentries;
        writeplanItem(doc, caseplanmodel, coordinatesOfSquare,
        intersectionSentries, null);
    }

    //define the connector and the shapes connected to it

    Connector connector=new Connector();
    connector.setResultOfLines(this.resultOfLines);
    connector.setResultOfSentries(this.getResultOfSentries());
    connector.setResultOfTasks(this.getResultOfTasks());

    for(CoordinatesOfLines line : this.resultOfLines){

        List<CoordinatesOfContours> shapes =
        connector.getCloserShapeForLine(line);

        connectionsMap.put(shapes.get(0), shapes.get(1));
        connectionsMap.put(shapes.get(1), shapes.get(0));
        shapesToLines.put(shapes.get(0), line);
        shapesToLines.put(shapes.get(1), line);
    }

    //writhing  planItem element
    for (CoordinatesOfContours coordinationEvent : Eventlist) {
        writeplanItem(doc, caseplanmodel, null, null,
        coordinationEvent);
    }
    //writhing Sentry element
    for (CoordinatesOfContours coordinationSentries :
    this.resultOfSentries) {
        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);
```

```
// set attribute
Attr Sizev1 = doc.createAttribute("height");
Sizev1.setValue("1050.0");
cmmndiSize.setAttributeNode(Sizev1);

Attr Sizev2 = doc.createAttribute("width");
Sizev2.setValue("1485.0");
cmmndiSize.setAttributeNode(Sizev2);

//writhing the shape as child of CMMN Diagram

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

// set attribute
Attr Shapev1 = doc.createAttribute("cmmnElementRef");

Shapev1.setValue("_3b0a4c03-c271-47c3-9e87-30c57c034fdb")
;
CMMNShape.setAttributeNode(Shapev1);

Attr Shapev2 = doc.createAttribute("id");

Shapev2.setValue("_d8d81e5a-d265-4ba1-9f94-4b0d47037451")
;
CMMNShape.setAttributeNode(Shapev2);

Element dcBounds  = doc.createElement("dc:Bounds");
CMMNShape.appendChild(dcBounds);

Attr boundv1 = doc.createAttribute("height");
boundv1.setValue("600.0");
dcBounds.setAttributeNode(boundv1);

Attr boundv2 = doc.createAttribute("width");
boundv2.setValue("800.0");
dcBounds.setAttributeNode(boundv2);

Attr boundv3 = doc.createAttribute("x");
```

```
        boundv3.setValue("34.0");
        dcBounds.setAttributeNode(boundv3);

        Attr boundv4 = doc.createAttribute("y");
        boundv4.setValue("34.0");
        dcBounds.setAttributeNode(boundv4);

        Element cmmndiCMMNLabe  = doc.createElement("cmmndi:CMMNLabel");
        CMMNShape.appendChild(cmmndiCMMNLabe);


        for(CoordinatesOfContours coordinateOfSentry : this.resultOfSentries){
            CoordinatesOfLines line = shapesToLines.get(coordinateOfSentry);
            writeLineValus(doc, CMMNDiagram, line, coordinateOfSentry);
        }

        for (CoordinatesOfContours coordinatesOfSquare : squarelist ) {
            writetaskValues(doc, CMMNDiagram, coordinatesOfSquare);
        }
        for (CoordinatesOfContours coordinatesOfSentries : Sentrieslist ) {
            writeEntryCriterionValus(doc, CMMNDiagram, coordinatesOfSentries);
        }
        for (CoordinationOfContours coordinationOfEvent : Eventlist ) {
            writeEventValues(doc, CMMNDiagram, coordinationOfEvent);
        }

        for (CoordinatesOfContours coordinatesOfFile : Filelist) {
            writeFileValues(doc, CMMNDiagram, coordinatesOfFile);
        }
        //writhing the style as child of CMMN Diagram
        Element cmmndiStyle = doc.createElement("cmmndi:CMMNStyle");
        staff2.appendChild(cmmndiStyle);

        // set attribute
        Attr Stylev1 = doc.createAttribute("fontFamily");
```

```java
            Stylev1.setValue("Arial,Helvetica,sans-serif");
            cmmndiStyle.setAttributeNode(Stylev1);

            Attr Stylev2 = doc.createAttribute("id");
            Stylev2.setValue("cb1a46a0-82e9-4c14-8495-8d3f50061e96");
            cmmndiStyle.setAttributeNode(Stylev2);

            // write the content into xml file
            TransformerFactory transformerFactory =
            TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransformer();
            transformer.setOutputProperty(OutputKeys.STANDALONE, "yes");
            DOMSource source = new DOMSource(doc);
            StreamResult result = new StreamResult(new
            File("C:/Users/SARA/Desktop/opencv/result.cmmn"));
            transformer.transform(source, result);
            System.out.println("File saved!");

        } catch (ParserConfigurationException pce) {
            pce.printStackTrace();
        } catch (TransformerException tfe) {
            tfe.printStackTrace();
        }
    }

    public void writeFileItemDefinition(Document doc,Element staff1,
    CoordinatesOfContours recognizefile){

        Attr planitemv1 = doc.createAttribute("id");
        planitemv1.setValue("fr"+ recognizefile.getid());
        staff1.setAttributeNode(planitemv1);

    }

    public void writeFileItem(Document doc,Element casefilemodel,
    CoordinatesOfContours recognizefile){

        Element cmmncaseFileItem =
        doc.createElement("cmmn:caseFileItem");
```

```
        casefilemodel.appendChild(cmmncaseFileItem);

        //set attribute
        Attr fileitemv1 = doc.createAttribute("definitionRef");
        fileitemv1.setValue("fr" + recognizefile.getid());
        cmmncaseFileItem.setAttributeNode(fileitemv1);

        Attr fileitemv2 = doc.createAttribute("multiplicity");
        fileitemv2.setValue("Unspecified");
        cmmncaseFileItem.setAttributeNode(fileitemv2);

        Attr fileitemv3 = doc.createAttribute("id");
        fileitemv3.setValue("fv" + recognizefile.getid());
        cmmncaseFileItem.setAttributeNode(fileitemv3);

    }

    public void writeplanItem(Document doc,Element caseplanmodel,
    CoordinatesOfContours
    coordinatesOfSquare,ArrayList<CoordinatesOfContours>
    intersectionSentries,CoordinatesOfContours  coordinationEvent){

        Element cmmnplanItem = doc.createElement("cmmn:planItem");
        caseplanmodel.appendChild(cmmnplanItem);

        // set attribute

        Attr planitemv1 = doc.createAttribute("definitionRef");

        if(coordinatesOfSquare!= null){
            planitemv1.setValue("t" + coordinatesOfSquare.getid());
        }else if(coordinationEvent != null){
            planitemv1.setValue("e" + coordinationEvent.getid());
        }
        cmmnplanItem.setAttributeNode(planitemv1);

        Attr planitemv2 = doc.createAttribute("id");
        if(coordinatesOfSquare!= null){
            planitemv2.setValue("pi" + coordinatesOfSquare.getid() );
```

```
        } else if(coordinationEvent != null){
            planitemv2.setValue("pi" + coordinationEvent.getid() );
        }
        cmmnplanItem.setAttributeNode(planitemv2);


        if(intersectionSentries!=null){
            if(!intersectionSentries.isEmpty()){
                for (CoordinatesOfContours coordinationSentries :
                intersectionSentries) {

                    Element entryCriterion =
                    doc.createElement("cmmn:entryCriterion");
                    cmmnplanItem.appendChild(entryCriterion);

                    Attr entryCriterionv1 = doc.createAttribute("sentryRef");
                    entryCriterionv1.setValue("senR" +
                    coordinationSentries.getid());
                    entryCriterion.setAttributeNode(entryCriterionv1);

                    Attr entryCriterionv2 = doc.createAttribute("id");

                    entryCriterionv2.setValue("Rsen"+coordinationSentries.getid()
                    );
                    entryCriterion.setAttributeNode(entryCriterionv2);
                }
            }
        }
    }

    /**
     * This method write the eventListener
     * @param doc
     * @param caseplanmodel
     * @param coordinationEvent
     */
    public void writeEvent(Document doc,Element caseplanmodel,
    CoordinatesOfContours coordinationEvent){
        Element eventListener = doc.createElement("cmmn:eventListener");
```

```
        caseplanmodel.appendChild(eventListener);

        // set attribute
        Attr cmmneventListenerv1 = doc.createAttribute("id");
        cmmneventListenerv1.setValue("e" + coordinationEvent.getid());
        eventListener.setAttributeNode(cmmneventListenerv1);
    }

    /**
     * This method write the Task
     * @param doc
     * @param caseplanmodel
     * @param recognizesquare
     */

    public void writeTask(Document doc,Element caseplanmodel,
    CoordinatesOfContours recognizesquare){
        Element cmmntask = doc.createElement("cmmn:task");
        caseplanmodel.appendChild(cmmntask);
        // set attribute
        Attr cmmntaskv1 = doc.createAttribute("isBlocking");
        cmmntaskv1.setValue("true");
        cmmntask.setAttributeNode(cmmntaskv1);

        Attr cmmntaskv2 = doc.createAttribute("id");
        cmmntaskv2.setValue("t" + recognizesquare.getid());
        cmmntask.setAttributeNode(cmmntaskv2);
    }

    /**
     * This method write the Sentry as well as the connection
     to connector
     * @param doc
     * @param caseplanmodel
     * @param coordinationSentries
     * @param coordinatesOfSquare
     * @param line
     */
    public void writeSentry(Document doc,Element caseplanmodel,
```

```
CoordinatesOfContours coordinationSentries,CoordinatesOfContours
coordinatesOfSquare,CoordinatesOfLines line){

    System.err.println("taskkkkkkk"+ coordinatesOfSquare);
    System.err.println("sentryyyyyyy"+coordinationSentries);
    System.err.println("lineeeeeeee"+ line);
    Element cmmnsentry = doc.createElement("cmmn:sentry");
    caseplanmodel.appendChild(cmmnsentry);

    if(coordinationSentries!=null){

        Attr sentryva1 = doc.createAttribute("id");
        sentryva1.setValue("senR" + coordinationSentries.getid());
        cmmnsentry.setAttributeNode(sentryva1);
    }

    Element cmmnplanItemOnPart =
    doc.createElement("cmmn:planItemOnPart");
    cmmnsentry.appendChild(cmmnplanItemOnPart);

    if(coordinatesOfSquare!= null){
        Attr planItemOnPart = doc.createAttribute("sourceRef");
        planItemOnPart.setValue("pi" +coordinatesOfSquare.getid());
        cmmnplanItemOnPart.setAttributeNode(planItemOnPart);
    }

    if(line!= null){
        Attr planItemOnPart1 = doc.createAttribute("id");
        planItemOnPart1.setValue("line"+line.getid());
        cmmnplanItemOnPart.setAttributeNode(planItemOnPart1);
    }

    Element cmmnstandardEvent =
    doc.createElement("cmmn:standardEvent");
    cmmnplanItemOnPart.appendChild(cmmnstandardEvent);
    // set attribute
    Attr standardEvent= doc.createAttribute("complete");
    cmmnstandardEvent.setAttributeNode(standardEvent);
```

```
        Element cmmnifpart = doc.createElement("cmmn:ifPart");
        cmmnsentry.appendChild(cmmnifpart);
        // set attribute
        Attr ifpartv1 = doc.createAttribute("id");
        ifpartv1.setValue("ifpa"+coordinationSentries.getid());
        cmmnifpart.setAttributeNode(ifpartv1);

    }

    /**
     * This method writes Task specifications
     * @param doc
     * @param CMMNDiagram
     * @param coordinatesOfSquare
     */
    public void writetaskValues(Document doc, Element CMMNDiagram,
    CoordinatesOfContours coordinatesOfSquare){
        Element CMMNShape2 = doc.createElement("cmmndi:CMMNShape");
        CMMNDiagram.appendChild(CMMNShape2);

        // set attribute
        Attr Shape2v1 = doc.createAttribute("cmmnElementRef");
        Shape2v1.setValue("pi"+coordinatesOfSquare.getid() );
        CMMNShape2.setAttributeNode(Shape2v1);

        Attr Shape2v2 = doc.createAttribute("id");
        Shape2v2.setValue("sh"+ coordinatesOfSquare.getid() );
        CMMNShape2.setAttributeNode(Shape2v2);

        Element dcBounds1  = doc.createElement("dc:Bounds");
        CMMNShape2.appendChild(dcBounds1);

        Attr bound2v1 = doc.createAttribute("height");
        bound2v1.setValue(Double.toString(coordinatesOfSquare.getHeight()));
        dcBounds1.setAttributeNode(bound2v1);

        Attr bound2v2 = doc.createAttribute("width");
        bound2v2.setValue(Double.toString(coordinatesOfSquare.getWidth()));
        dcBounds1.setAttributeNode(bound2v2);
```

```
        Attr bound2v3 = doc.createAttribute("x");
        bound2v3.setValue(Double.toString(coordinatesOfSquare.getX()));
        dcBounds1.setAttributeNode(bound2v3);

        Attr bound2v4 = doc.createAttribute("y");
        bound2v4.setValue(Double.toString(coordinatesOfSquare.getY()));
        dcBounds1.setAttributeNode(bound2v4);

        Element cmmndiCMMNLabel2  =
        doc.createElement("cmmndi:CMMNLabel");
        CMMNShape2.appendChild(cmmndiCMMNLabel2);

    }

    /**
     * This method writes Event specifications
     * @param doc
     * @param CMMNDiagram
     * @param coordinationOfEvent
     */
    public void writeEventValues(Document doc, Element CMMNDiagram,
    CoordinatesOfContours coordinationOfEvent){
        Element CMMNShape2 = doc.createElement("cmmndi:CMMNShape");
        CMMNDiagram.appendChild(CMMNShape2);

        // set attribute
        Attr Shape2v1 = doc.createAttribute("cmmnElementRef");
        Shape2v1.setValue("pi"+coordinationOfEvent.getid() );
        CMMNShape2.setAttributeNode(Shape2v1);

        Attr Shape2v2 = doc.createAttribute("id");
        Shape2v2.setValue("sh"+ coordinationOfEvent.getid() );
        CMMNShape2.setAttributeNode(Shape2v2);

        Element dcBounds1  = doc.createElement("dc:Bounds");
        CMMNShape2.appendChild(dcBounds1);

        Attr bound2v1 = doc.createAttribute("height");
```

```java
        bound2v1.setValue(Double.toString(coordinationOfEvent.getHeight()));
        dcBounds1.setAttributeNode(bound2v1);

        Attr bound2v2 = doc.createAttribute("width");
        bound2v2.setValue(Double.toString(coordinationOfEvent.getWidth()));
        dcBounds1.setAttributeNode(bound2v2);

        Attr bound2v3 = doc.createAttribute("x");
        bound2v3.setValue(Double.toString(coordinationOfEvent.getX()));
        dcBounds1.setAttributeNode(bound2v3);

        Attr bound2v4 = doc.createAttribute("y");
        bound2v4.setValue(Double.toString(coordinationOfEvent.getY()));
        dcBounds1.setAttributeNode(bound2v4);

        Element cmmndiCMMNLabel2 = doc.createElement("cmmndi:CMMNLabel");
        CMMNShape2.appendChild(cmmndiCMMNLabel2);

    }

    /**
     * This method writes sentry specifications
     * @param doc
     * @param CMMNDiagram
     * @param coordinatesOfSentries
     */
    public void writeEntryCriterionValus(Document doc, Element CMMNDiagram, CoordinatesOfContours coordinatesOfSentries){
        Element CMMNShape2 = doc.createElement("cmmndi:CMMNShape");
        CMMNDiagram.appendChild(CMMNShape2);

        // set attribute
        Attr Shape2v1 = doc.createAttribute("cmmnElementRef");
        Shape2v1.setValue("Rsen"+coordinatesOfSentries.getid() );
        CMMNShape2.setAttributeNode(Shape2v1);

        Attr Shape2v2 = doc.createAttribute("id");
        Shape2v2.setValue("sh"+ coordinatesOfSentries.getid() );
```

```
        CMMNShape2.setAttributeNode(Shape2v2);

        Element dcBounds1  = doc.createElement("dc:Bounds");
        CMMNShape2.appendChild(dcBounds1);

        Attr bound2v1 = doc.createAttribute("height");

        bound2v1.setValue(Double.toString(coordinatesOfSentries.getHeight()))
        ;
        dcBounds1.setAttributeNode(bound2v1);

        Attr bound2v2 = doc.createAttribute("width");
        bound2v2.setValue(Double.toString(coordinatesOfSentries.getWidth()));
        dcBounds1.setAttributeNode(bound2v2);

        Attr bound2v3 = doc.createAttribute("x");
        bound2v3.setValue(Double.toString(coordinatesOfSentries.getX()));
        dcBounds1.setAttributeNode(bound2v3);

        Attr bound2v4 = doc.createAttribute("y");
        bound2v4.setValue(Double.toString(coordinatesOfSentries.getY()));
        dcBounds1.setAttributeNode(bound2v4);

        Element cmmndiCMMNLabel2  =
        doc.createElement("cmmndi:CMMNLabel");
        CMMNShape2.appendChild(cmmndiCMMNLabel2);

    }

    /**
     * This method writes Line specifications
     * @param doc
     * @param CMMNDiagram
     * @param line
     * @param coordinatesOfSentries
     */
    public void writeLineValus(Document doc, Element CMMNDiagram,
    CoordinatesOfLines line,CoordinatesOfContours coordinatesOfSentries){

```

```java
System.out.println("show me lines: " + line);
Element CMMNShape2 = doc.createElement("cmmndi:CMMNEdge");
CMMNDiagram.appendChild(CMMNShape2);

// set attribute
Attr Shape2v1 = doc.createAttribute("cmmnElementRef");
if(line!=null){
    Shape2v1.setValue("line"+line.getid());
    CMMNShape2.setAttributeNode(Shape2v1);

    Attr Shape2v2 = doc.createAttribute("id");
    Shape2v2.setValue("li"+ line.getid() );
    CMMNShape2.setAttributeNode(Shape2v2);

    Attr Shape2v3 = doc.createAttribute("targetCMMNElementRef");
    Shape2v3.setValue("Rsen"+coordinatesOfSentries.getid() );
    CMMNShape2.setAttributeNode(Shape2v3);

    Attr Shape2v4 = doc.createAttribute("isStandardEventVisible");
    Shape2v4.setValue("true" );
    CMMNShape2.setAttributeNode(Shape2v4);

    Element diwaypoint  = doc.createElement("di:waypoint");
    CMMNShape2.appendChild(diwaypoint);

    Attr bound2v1 = doc.createAttribute("x");
    bound2v1.setValue(Double.toString(line.getX1()+8));
    diwaypoint.setAttributeNode(bound2v1);

    Attr bound2v2 = doc.createAttribute("y");
    bound2v2.setValue(Double.toString(line.getY1()));
    diwaypoint.setAttributeNode(bound2v2);

    Element diwaypoint2  = doc.createElement("di:waypoint");
    CMMNShape2.appendChild(diwaypoint2);

    Attr bound2v3 = doc.createAttribute("x");
    bound2v3.setValue(Double.toString(line.getX2()+8));
    diwaypoint2.setAttributeNode(bound2v3);
```

```
            Attr bound2v4 = doc.createAttribute("y");
            bound2v4.setValue(Double.toString(line.getY2()));
            diwaypoint2.setAttributeNode(bound2v4);

        }
        Element cmmndiCMMNLabel2 =
        doc.createElement("cmmndi:CMMNLabel");
        CMMNShape2.appendChild(cmmndiCMMNLabel2);
    }

    /**
     * This method writes File specifications
     * @param doc
     * @param CMMNDiagram
     * @param coordinatesOfFile
     */
    public void writeFileValues(Document doc, Element CMMNDiagram,
    CoordinatesOfContours coordinatesOfFile){
        Element CMMNShape2 = doc.createElement("cmmndi:CMMNShape");
        CMMNDiagram.appendChild(CMMNShape2);

        // set attribute
        Attr Shape2v1 = doc.createAttribute("cmmnElementRef");
        Shape2v1.setValue("fv"+coordinatesOfFile.getid());
        CMMNShape2.setAttributeNode(Shape2v1);

        Attr Shape2v2 = doc.createAttribute("id");
        Shape2v2.setValue("sf"+ coordinatesOfFile.getid());
        CMMNShape2.setAttributeNode(Shape2v2);

        Element dcBounds1  = doc.createElement("dc:Bounds");
        CMMNShape2.appendChild(dcBounds1);

        Attr bound2v1 = doc.createAttribute("height");
        bound2v1.setValue(Double.toString(coordinatesOfFile.getHeight()));
        dcBounds1.setAttributeNode(bound2v1);

        Attr bound2v2 = doc.createAttribute("width");
```

```
        bound2v2.setValue(Double.toString(coordinatesOfFile.getWidth()));
        dcBounds1.setAttributeNode(bound2v2);

        Attr bound2v3 = doc.createAttribute("x");
        bound2v3.setValue(Double.toString(coordinatesOfFile.getX()));
        dcBounds1.setAttributeNode(bound2v3);

        Attr bound2v4 = doc.createAttribute("y");
        bound2v4.setValue(Double.toString(coordinatesOfFile.getY()));
        dcBounds1.setAttributeNode(bound2v4);

        Element cmmndiCMMNLabel2 =
        doc.createElement("cmmndi:CMMNLabel");
        CMMNShape2.appendChild(cmmndiCMMNLabel2);

    }
}
```

BIBLIOGRAPHY

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