



## Deliverable Report

Client : European Commission

Project : FRESH

Project N°: FP6-516059

---

Project Number: FP6-516059  
Document number: DR\_FRESH\_WP2\_2.4.1  
Document Title: Intermediate report with first interactive prototype  
Document status: Final version

---

Abstract **This report summarizes the work done on first prototype for symbol recognition.**

---

Keyword List WP2, prototype, symbol, recognition



## Deliverable Report

Client : European Commission

Project : FRESH

Project N°: FP6-516059

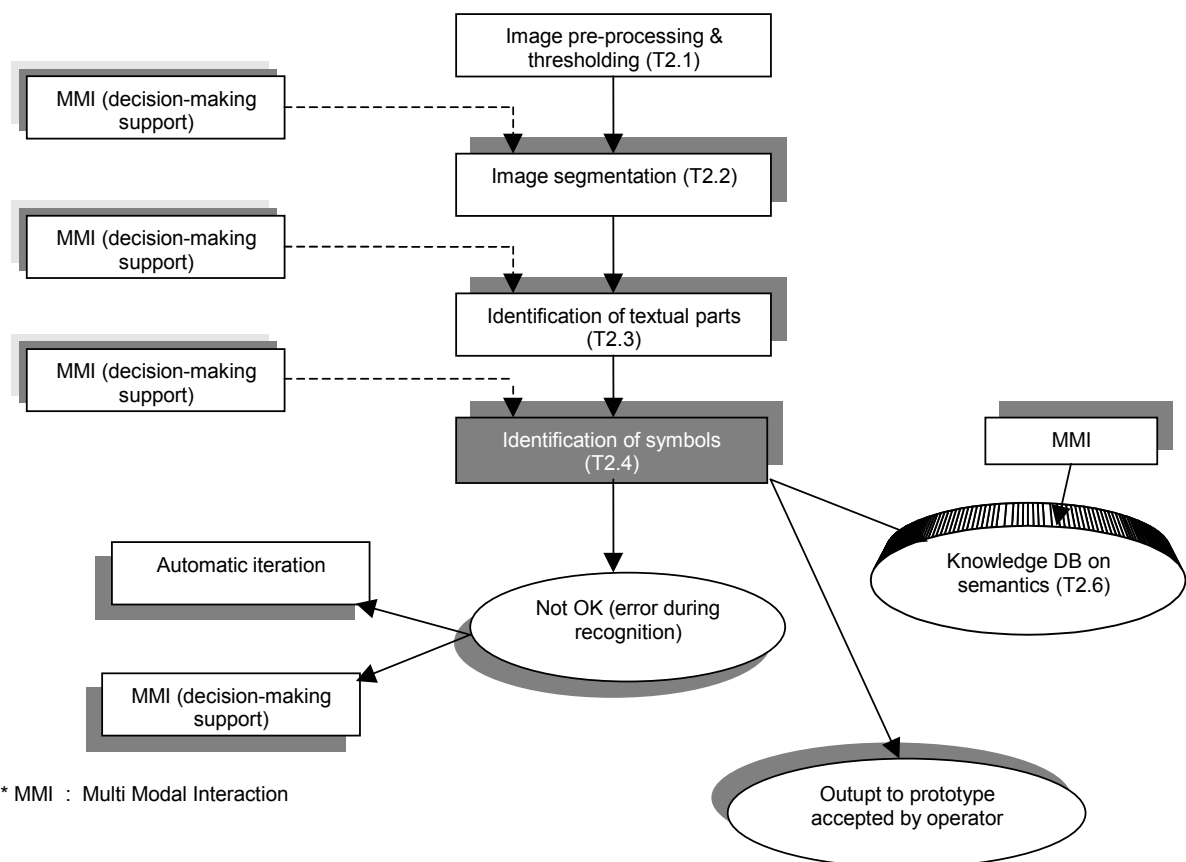
## CONTENTS

<b>1. PURPOSE, SCOPE AND OVERVIEW OF THIS DOCUMENT .....</b>	<b>3</b>
<b>2. SOFTWARE REQUIREMENTS .....</b>	<b>4</b>
2.1. CALCULATIVE FUNCTIONS .....	4
2.1.1. DESCRIPTOR OF CONNECTED COMPONENT NODES OF LEVEL 1 .....	4
2.1.2. DESCRIPTOR OF CONNECTED COMPONENT LEAVES OF LEVEL 1 .....	5
2.1.3. DESCRIPTOR OF CONNECTED COMPONENT LEAVES OF LEVEL 2 .....	5
2.1.4. DESCRIPTOR OF BLACK AND THICK CONNECTED COMPONENTS .....	6
2.1.5. DESCRIPTOR OF EXTREMITIES .....	6
2.1.6. DESCRIPTOR OF OCCLUSIONS .....	8
2.1.7. DESCRIPTOR OF CIRCLE .....	8
2.1.8. DESCRIPTOR OF CORNERS .....	8
2.1.9. DESCRIPTOR OF CLOSED RECTANGLES .....	9
2.1.10. DESCRIPTOR OF OPEN RECTANGLES .....	9
2.1.11. IMAGE SIGNATURE COMPOSITION OF GEOMETRIC DESCRIPTORS .....	10
2.1.12. IMAGE DISTANCE CALCULATION FROM GEOMETRIC DESCRIPTORS .....	10
2.2. UTILITY FUNCTIONS .....	10
2.3. INTEGRATION FUNCTIONS: .....	10
<b>3. SOFTWARE DESIGN .....</b>	<b>11</b>
3.1. SYSTEM ENVIRONMENT, DESIGN APPROACH AND STRATEGIES .....	11
3.2. SYSTEM STRUCTURE .....	12
3.3. INTERFACE DESIGN WITH AN EXTERNAL SOFTWARE .....	12
3.4. PROCESS DESIGN .....	13
3.5. DESCRIPTOR INTEGRATION DESIGN .....	13
3.6. CODING INTERMEDIATE RESULTS OF A DESCRIPTOR .....	14

## 1. PURPOSE, SCOPE AND OVERVIEW OF THIS DOCUMENT

This document is the software documentation of the prototype for symbol recognition. This software is part of Task 2.4 of the Work Package 2 of the Fresh European Project. The WP2 intends to develop a recognition tool that transfers paper plans in reconstructed wiring information. This WP is based on the generic recognition framework as described in the associated sub-activities: the overall process is depicted in the following chart, on which we can identify Task 2.4 of WP2 :

**Figure 1 : Task 2.4 in WP2**



So the Identification of symbols (or Symbol Recognition) Task has for main inputs:

- Small parts of scanned wiring plans, resulting from Image Segmentation Task (T2.2).
- A family of models, which can be attached to some parts of Semantic Knowledge DB (T2.6).

We refer to the Symbol Recognition Computer Tool, in charge of the recognition of Scanned Drawing Images among Database Images, as “software” or “Rec software”.

In next sections of this document we will give the following descriptions:

- The requirements of the Symbol Recognition Software.
- The design of this software.
- The test plan of this software.
- The using guide of this software.
- And perspective of this software.

## **2. SOFTWARE REQUIREMENTS**

In this section we will describe more detailed functionalities of the Symbol Recognition Software. The method chosen for Symbol Recognition has for main process:

- Computation of database images signatures.
- Computation of scanned drawing images signatures.
- For each scanned drawing image, computation of distance between this image and each database image.
- For each scanned drawing image, sort database images in ascending order of distance from the scanned drawing image.

In fact the more complex process is the computation of image signatures. This process involves the use of several geometric descriptors, each given a number of some geometric elements which are present in the image.

Beside these specific calculative functionalities, we have some utility functions like:

- Configuration of input parameters.
- Scan of database images and of scanned drawing images.
- Conversion of image format.
- Generation of intermediate results for study.
- Generation of final results.

Finally, according to the previous diagram, we need some functions allowing integration with external software.

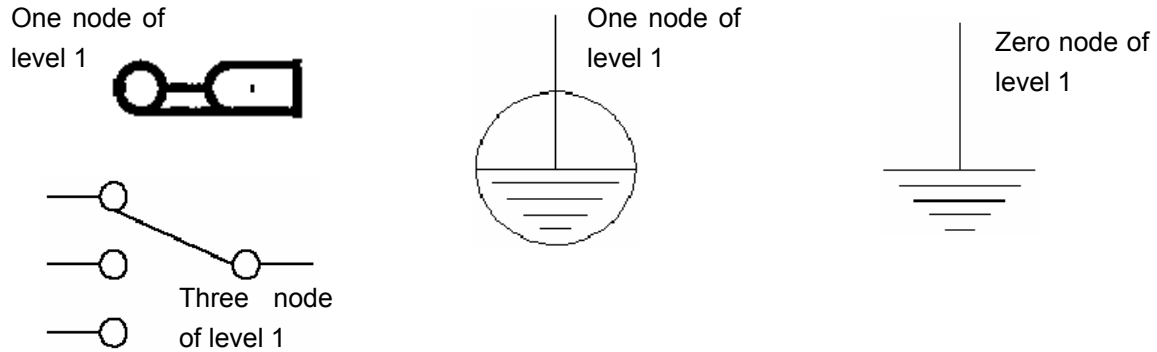
We can now list the functions we will describe below:

- Calculative functions:
  - Geometric descriptors:
  - Image signature composition of geometric descriptors.
  - Images distance calculation from geometric descriptors.
- Utility functions:
  - Configuration of input parameters.
  - Scan of database images and of scanned drawing images.
  - Conversion of image format.
  - Generation of intermediate results for study.
  - Generation of final results.
- Integration functions:
  - Rapid integration of new descriptors.
  - Integration with external software.

### **2.1. CALCULATIVE FUNCTIONS**

#### **2.1.1. DESCRIPTOR OF CONNECTED COMPONENT NODES OF LEVEL 1**

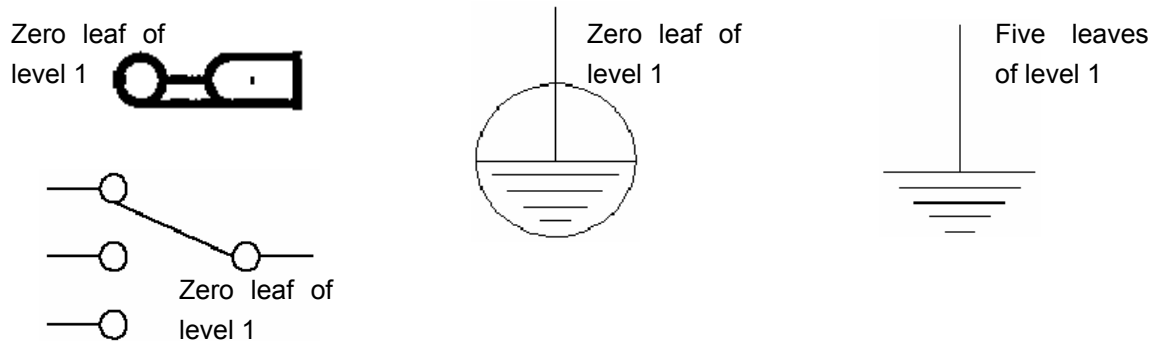
This descriptor gives the number of nodes of level 1 in the connected components tree of the image. Here the term node is taken in its restricted sense i.e. a connected component which has at least one child. It is also pointed out that the level 0 is the background of the image, which is white by convention. So nodes of level 1 are black, are not inside another outline and have some hollows. The user can define a minimum area that must verify these components.



**Figure 2 : Examples of results for descriptor of connected component nodes of level 1**

### 2.1.2. DESCRIPTOR OF CONNECTED COMPONENT LEAVES OF LEVEL 1

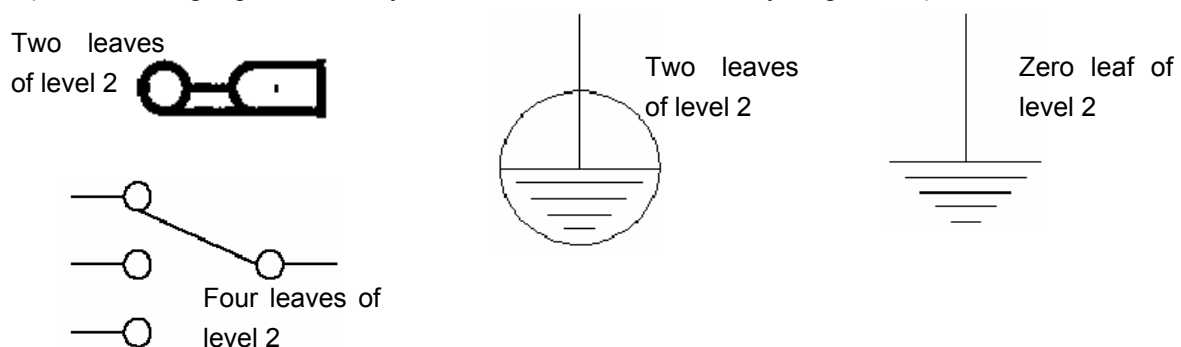
Unlike nodes of level 1 descriptor, we consider here only leaves of level 1 i.e. nodes (in large sense) of level 1 which have no children. So leaves of level 1 are black, are not inside another outline and are full. Like before the user can define a minimum area that must verify these components (a value of 0 deactivates this option).



**Figure 3 : Examples of results for descriptor of connected component leaves of level 1**

### 2.1.3. DESCRIPTOR OF CONNECTED COMPONENT LEAVES OF LEVEL 2

This descriptor gives the number of leaves of level 2 in the components tree of the image (i.e. nodes without children and which are children of nodes of level 1). So these components are white, are of level 2 and have no children (in natural language we can say "hollow of first level without anything inside").



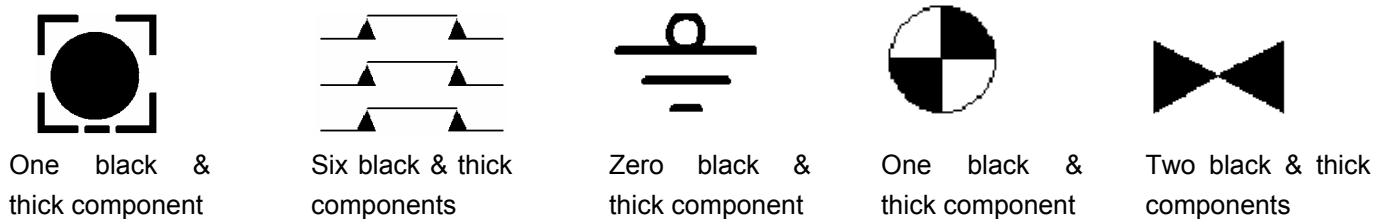
**Figure 4 : Examples of results for descriptor of connected component leaves of level 2**

### 2.1.4. DESCRIPTOR OF BLACK AND THICK CONNECTED COMPONENTS

This descriptor counts all the black connected components in an image where we keep only the thick parts. So to process this descriptor we need two steps:

- First generate an image with only the thick parts of the source image.
- Secondly recursively count the number of black components of the connected components tree of this image.

**Figure 5 : Examples of black and thick components descriptor results**



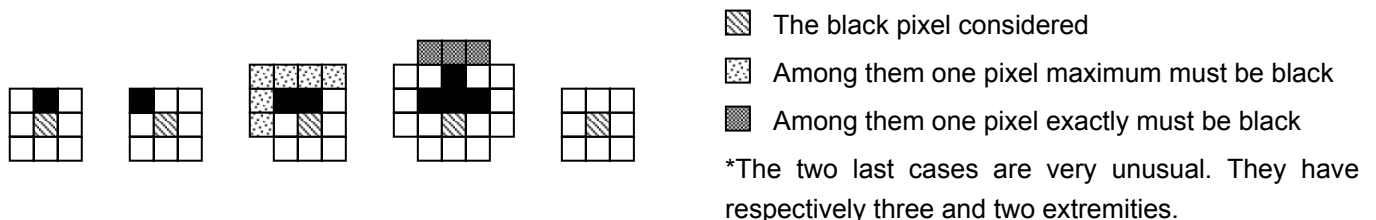
Remark: in the last example, we can see the influence of the structural element (a square) used for the morphologic opening operation. With this structural element the vertices of triangle do not belong to the thick layer, this explains the two black and thick components detected.

### 2.1.5. DESCRIPTOR OF EXTREMITIES

This descriptor counts the number of extremities. An extremity is a natural concept but is not easy to define mathematically. It depends on the thickness of the primitive. It depends on the angle or on the radius of curvature.

We define an extremity as a particular pixel from a specific skeleton. An extremity pixel of this skeleton is a pixel from where during at least two pixels only one return way is possible.

We will have following configurations, up to symmetry:



**Figure 6 : Extremities cases**

So we need to calculate a skeleton in order to implement this method. However, contour noise may affect skeleton by originating spurious skeleton branches, as show in examples below:

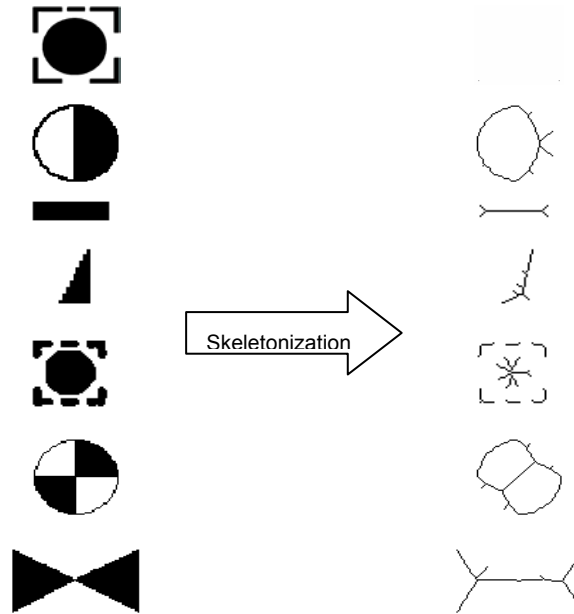


Figure 7 : Examples of skeletonization without pruning

So skeletonization algorithm must include a pruning phase. But which value for pruning do we have to choose? A solution is to use a central tendency for half-of-thickness. The method for this is explained previously in black and thick descriptor. But this pruning is not enough as seen below for the same images as over:

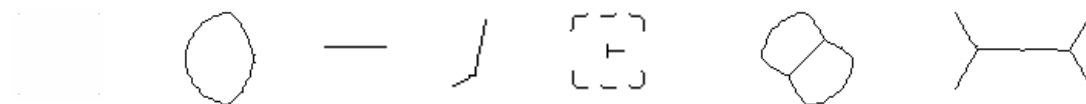


Figure 8 : Examples of skeletonization with intelligent pruning

The second solution is to take only outlines of thick part of image before skeletonization with the same pruning. And for defining thick image we use the same method as for black and thick descriptor with the same central tendency for half-of-thickness. Results obtained with taking outlines of thick parts and pruning is given below, always with the same images as over:

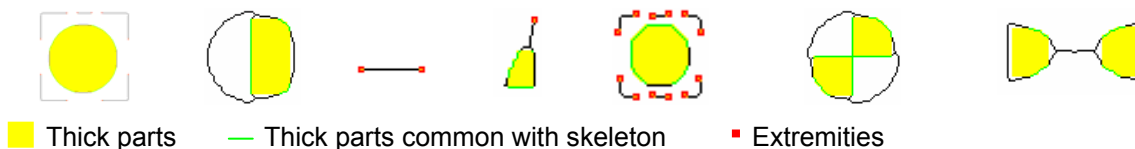


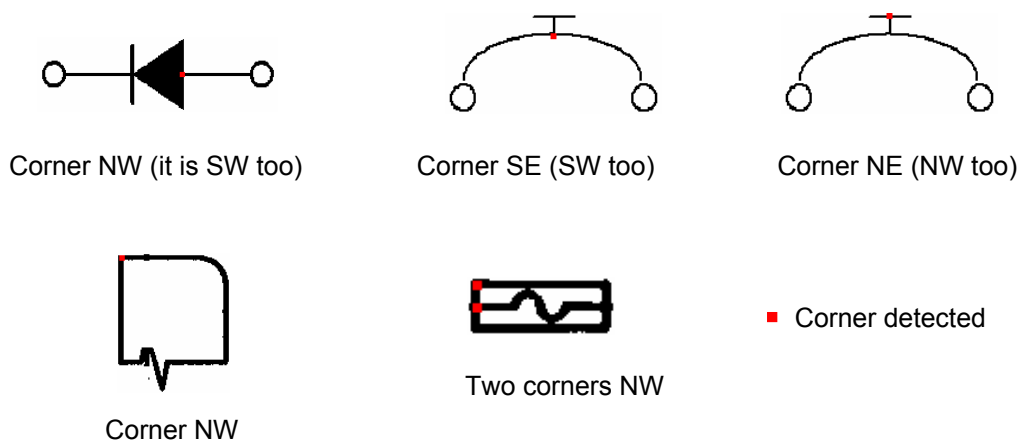
Figure 9 : Examples of skeletonization with thick parts outlines and intelligent pruning



**Figure 11 : Templates used for descriptors of corners**



**Figure 12 : Examples of some corner descriptor results**

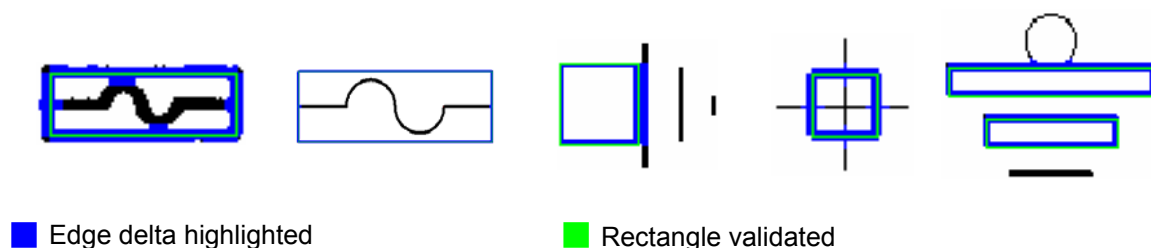


### 2.1.9. DESCRIPTOR OF CLOSED RECTANGLES

This descriptor counts the number of rectangles of the source image. It uses the same function as the corner descriptors for finding corners but it computes them in a particular order. So this descriptor uses the same options defined by user for corners and adds two more parameters:

- An edge threshold which corresponds to the ratio of black pixels in an edge of the rectangle to the length of the edge.
- An edge delta to tolerate some variations around the direction of the edge (horizontal or vertical).

**Figure 13 : Examples of rectangles descriptor results**



### 2.1.10. DESCRIPTOR OF OPEN RECTANGLES

This descriptor simply uses a result of the closed rectangle descriptor. If four corners corrected aligned have been found but if some edges are not covered correctly then the rectangle is labeled as open.

### 2.1.11. IMAGE SIGNATURE COMPOSITION OF GEOMETRIC DESCRIPTORS

An image signature is a vector where each coordinate is the result of a descriptor. Since we use geometric descriptors for which the result is a number of geometric components, geometric image signatures are vectors of non-negative integers. The order of descriptors used inside the vector is naturally the same for all image signatures.

Since the descriptor name is given each time inside the signature file, the order of them is free: the software will order itself in a same way for all images.

### 2.1.12. IMAGE DISTANCE CALCULATION FROM GEOMETRIC DESCRIPTORS

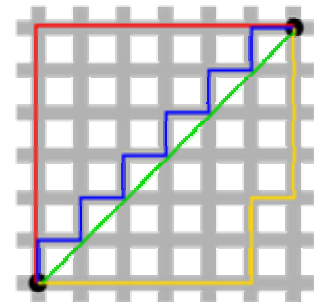
In order to classify database images from nearest to most far from the scanned drawings image, we use a distance calculation between image signatures. The most famous distance method between two vectors is the Euclidean distance.

Since we are not on a Euclidean space we propose a more simple distance, also used in vector space: the L1-distance resulting from 1-Norm and used in Taxicab geometry considered by Minkowski. L1-distance is also named Manhattan distance or taxicab distance or city-block distance:

$$d(x, y) = \sum_{i=1}^n |x_i - y_i|$$

**Figure 14: Manhattan versus Euclidean distance**

The red, blue, and yellow lines representing the Manhattan distance all have the same length (12), whereas the green line representing the Euclidean distance has length  $6\sqrt{2}$ .



But both previous distance values depend on the number of descriptors and especially on the scale of each descriptor. Finally we propose to distinguish distances used on coordinates (descriptor distance named  $d_i$ ) from their composition to form a vector distance.

## 2.2. UTILITY FUNCTIONS

These functions will be detailed in the next document release:

- Configuration of input parameters.
- Scan of database images and of scanned drawing images.
- Conversion of image format.
- Generation of intermediate results for study.
- Generation of final results.

## 2.3. INTEGRATION FUNCTIONS:

These functions will be detailed in the next document release:

- Rapid integration of new descriptors.
- Integration with external software.



## Deliverable Report

Client : European Commission

Project : FRESH

Project N°: FP6-516059

### 3. SOFTWARE DESIGN

#### 3.1. SYSTEM ENVIRONMENT, DESIGN APPROACH AND STRATEGIES

Two important software contexts have to be taken into account to design this software:

- The use of the free Qgar library ([www.qgar.org](http://www.qgar.org)) which is a very important tool, because it capitalizes a lot of achievements of the team in Graphics Recognition algorithms and structures. As seen in the function specifications, we often use the Qgar library functions to process complex or common algorithms (skeletonization, image opening or closure, image dilatation or erosion, connected components calculation ...).
- The interface with external software in order to allow the whole WP2 Fresh process.

Some difficulties were coming from these two contexts:

- The Qgar library runs on Linux in a C++ development environment.
- The external software (see task T2.2 of WP2) is running on MS Windows and is written with another programming language (Delphi).

To find the best compromise for software design and development, the following choices have been made:

- Development on current release of MS Windows, to facilitate integration with external software.
- Programming in C++ language to facilitate the use of functions and complex structures needed for Graphics Recognition present in the Qgar library.

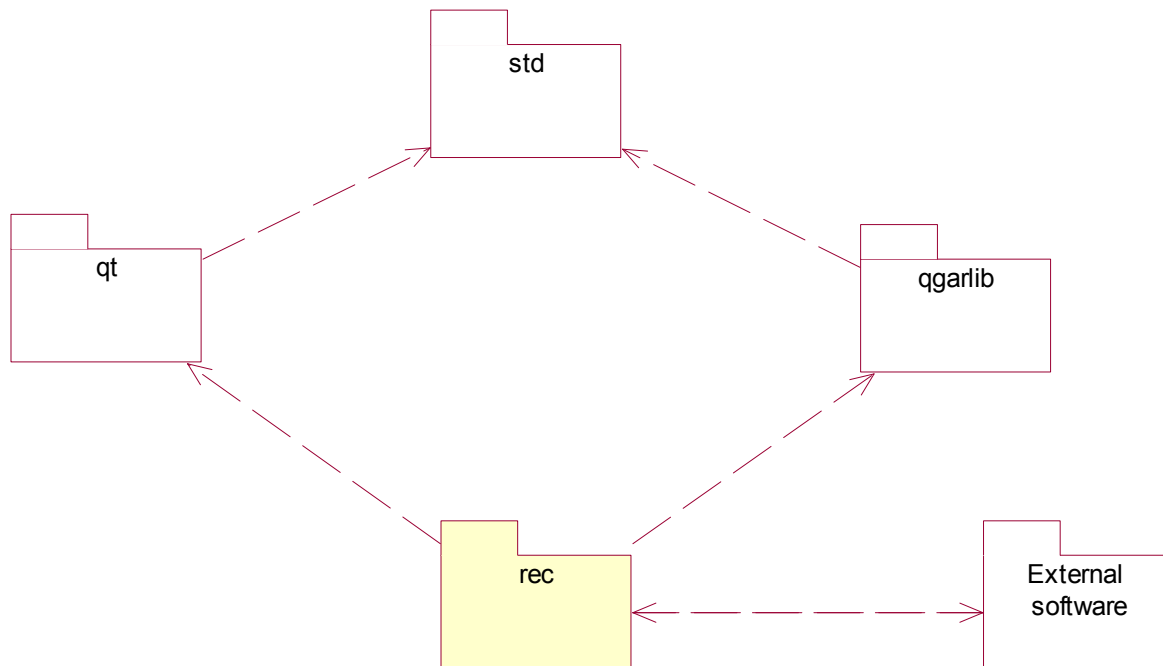
The software can be developed on the free C++ IDE or on the commercial IDE.

Some secondary or dependant choices from previous have been made:

- The use of the free Qt 3.3.5 library for image format conversion (BMP format used frequently in MS Windows and in Web browsers, PBM format used by Qgar library) and for generation of intermediate images for test only. So the Qt library is not used in calculative functions but only in utility functions. Qt is a very portable and performing graphics library.
- The use of the standard library STL for C++ development which offers a lot of containers (list, vector, map ...) and very performing iterators and data manipulation algorithms.
- The possibility to run the software in a standalone manner to facilitate test and development and the possibility to interface the software with another. Therefore two main software components exist: an executable file named rec.exe for standalone execution and a DLL (Dynamic Link Library) file named rec.dll for interfacing. The DLL file is used by the standalone executable too.
- The input and output data file formats, which are:
  - INI file format to save configurations data of the software and to initialize it.
  - Semicolon-separated CSV file format for reading and saving image signatures, and for saving image matching results. Semicolon as field separator has the advantage from comma field separator to allow automatic opening of the file with MS Excel or OpenOffice spreadsheet.

### **3.2. SYSTEM STRUCTURE**

The global diagram illustrating what has been described previously is given below:



**Figure 15: High-level component structure**

We will give in the following chapter an explanation for dependencies between the Rec component and an external software component.

### **3.3. INTERFACE DESIGN WITH AN EXTERNAL SOFTWARE**

The interface for using the Rec software is formed by a DLL file (called below Rec DLL) in order to allow external software programmed in another language to configure and launch the recognition process.

The interface allows only standard data types to be exchanged in order to allow interfacing with different programming language.

The functions of the managed interface (used by external software) of the Rec DLL are given below in the same order they must be called:

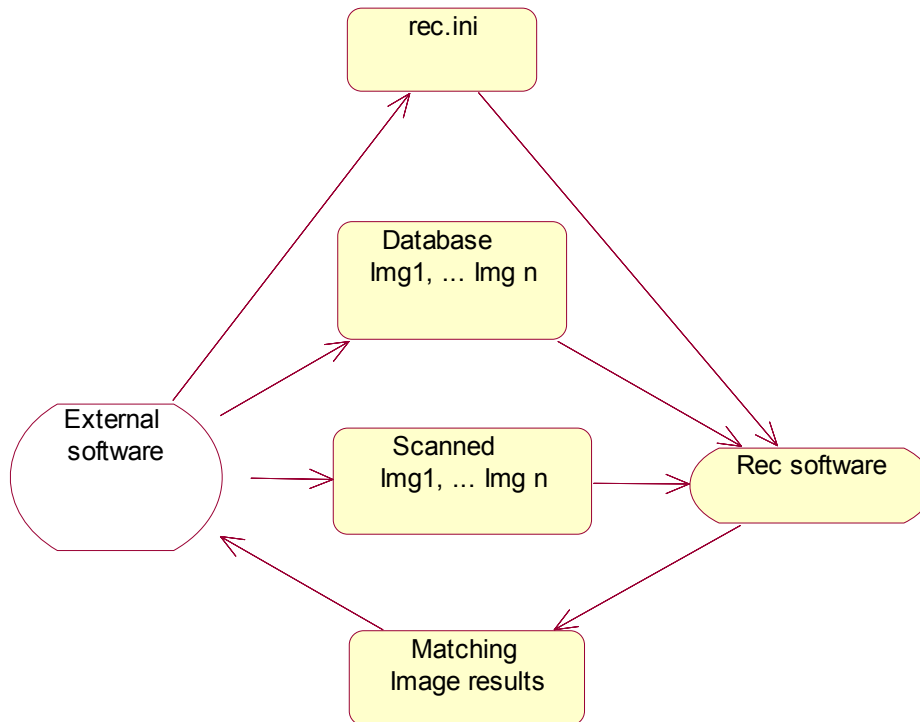
1. procedure `recInfoIniFile(iniPath as string)`
2. procedure `recSignatures(dbSignPath as string, genDbSign as boolean)`
3. procedure `recInfoImage(id as integer, w as integer, h as integer, data as pointer)`
4. procedure `recInfoConnexions(id as integer, size as integer; points as pointer)`
5. function `recDistances(id as integer) returning a pointer`
6. procedure `recClear(id as integer)`

The functions of the standalone interface of the Rec DLL are given below:

- function recExec(iniPath as string) returning an integer.

The general interactions are given in the following diagram:

**Figure 16: General interactions**



### **3.4. PROCESS DESIGN**

The software source files and contents include:

- the main program for standalone execution or for testing the managed interface.
- the application class
- a generalized class of all descriptors, the image signature class and the image distance class.

All the files which concern calculative functions are optional according to user choice to integrate or not some descriptors.

### **3.5. DESCRIPTOR INTEGRATION DESIGN**

Because several descriptors have to be developed and integrated into Rec software, their integration must be fast and easy.



## Deliverable Report

Client : European Commission

Project : FRESH

Project N°: FP6-516059

### **3.6. CODING INTERMEDIATE RESULTS OF A DESCRIPTOR**

Two types of intermediate results can be generated in the descriptor source code:

- Log messages which are drawn in a Qt Scrolled Text Window (if WithConsole is set in the INI file), MS Terminal Window (if executable is launched from MS Terminal Window) and always in the rec.log file localized in the same directory as the INI file rec.ini. Use the verbose boolean (already read in class Descriptor) to verify if user has activated or not this option.
- Intermediate images can be generated in the output directory (based on the directory specified in the INI file rec.ini as an absolute or relative directory, the software adding a prefix .out to this directory).