# Universiteit Leiden
# Opleiding Informatica

Automation of use case diagram recognition from images


Name: Pieter van den Bosch
Studentnr: 0935956


Date: 30/01/2014


1st supervisor: D.R. Stikkolorum
2nd supervisor: M.R.V. Chaudron


BACHELOR THESIS


Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

# Automation of use case diagram recognition from images

Pieter van den Bosch (0935956)

**Abstract**

In the field of software design a lot of research is done with and on UML diagrams. One of the popular UML diagrams is the use case diagram. A lot of these diagrams are only available in a binary image format. In this thesis we describe a way of converting binary use case diagrams to a more generic format. The cross-platform software application we developed can convert a simple use case diagram to an XMI. The XMI files that are generated by the software can be imported and edited in Visual Paradigm. This thesis shows that the software gives a good result converting simple use case diagrams by running the software on various sets of use case diagrams but it also shows that the software still has some shortcomings.

# Contents


**1 Introduction** **4**
- 1.1 Research question . . . 4
- 1.2 Related work . . . 4

**2 Software Design** **5**
- 2.1 Requirements . . . 5
- 2.2 Main program flow . . . 5
- 2.3 Input formats . . . 5
- 2.4 Output format . . . 6
- 2.5 User interface . . . 6

**3 Method and techniques** **6**
- 3.1 Recognizing use cases . . . 7
- 3.2 Recognizing actors . . . 7
- 3.3 Recognizing relations . . . 8
- 3.4 Recognizing system borders . . . 8
- 3.5 Recognizing characters . . . 8
- 3.6 Generating XMI . . . 9
- 3.7 Cross-platform . . . 10

**4 Validation and verification** **10**
- 4.1 Tests . . . 10
- 4.2 Validation criteria . . . 11
- 4.3 Testset 1 . . . 11
  - 4.3.1 Use cases . . . 11
  - 4.3.2 Actors . . . 11
  - 4.3.3 Relations . . . 12
  - 4.3.4 System borders . . . 12
  - 4.3.5 Character recognition . . . 13
- 4.4 Testset 2 . . . 16
  - 4.4.1 Use cases . . . 16
  - 4.4.2 Actors . . . 16
  - 4.4.3 Relations . . . 16
  - 4.4.4 System borders . . . 17
  - 4.4.5 Character recognition . . . 17
- 4.5 Testset 3 . . . 19
  - 4.5.1 Use cases . . . 19
  - 4.5.2 Actors . . . 19
  - 4.5.3 Relations . . . 20
  - 4.5.4 System borders . . . 20
  - 4.5.5 Character recognition . . . 20

**5 Conclusions** **23**
- 5.1 Recognition of use cases . . . 23
- 5.2 Recognition of actors . . . 23
- 5.3 Recognition of relations . . . 23
- 5.4 Recognition of system borders . . . 23
- 5.5 Recognition of characters . . . 24
- 5.6 Generated XMI files . . . 24
- 5.7 User interface . . . 24

**6 Future work** **24**

# 1 Introduction

In the field of software design research is done with and on UML diagrams [1]. One of the popular UML diagrams is the use case diagram [2] introduced in UML 1.1 but the semantic integration improved in UML 2.0. The use case diagram is used to portray various ways users can interact with a system. We experienced that a lot of the use case diagrams are only available in a binary image format. This format is impossible to import and edit in CASE tools[1] such as Visual Paradigm[2] or Enterprise Architect[3] or in analysis tools such as SD Metrics[4]. Converting these diagrams by hand to a generic format such as XMI can be quite time consuming and error prone. Therefore our aim was to develop an application that could do automatic conversion from use case diagrams to a generic format, in our case the XMI format (explained further in Chapter 2.4).

This thesis describes our approach to develop an application converting binary use case diagrams to a more generic form. The first chapter is a brief introduction about our research. Chapter 2 and 3 describe our approach on the software design and implementation. Chapter 4 discusses the validation and verification of our software. We conclude in Chapter 5.

## 1.1 Research question

The main research question is: How to convert a binary UML use case diagram (image file) to a more generic form in an automated way? This raises another question: If it is possible to convert a use case diagram to a more generic format in an automated way, is this method also a reliable method?

We think the software should be able to convert simple use case diagrams to an XMI file that can be imported in a CASE tool. By simple use case diagrams we think of diagrams that have actors, system borders, use cases and relations, an example of such a simple diagram can be found in Figure 1. By relations we only think of plain relations (no directed relations, include relations or extends relations). The software should also be cross-platform (available for GNU/Linux and Microsoft Windows), because we like to target a broad audience that may be using different operating systems.

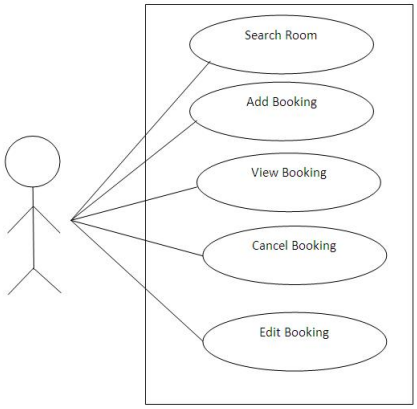


Figure 1: Example of a simple use case diagram

## 1.2 Related work

Image 2 UML: Extracting UML Models from Images [4], describes a way of converting class diagrams to the XMI format. The paper describes software that is developed using the .NET environment. The disadvantage of developing software with the .NET environment is that the software will work other platforms with the open source .NET framework Mono [5] but it

[1] http://nl.wikipedia.org/wiki/Computer-aided_software_engineering
[2] http://www.visual-paradigm.com
[3] http://www.sparxsystems.eu/EnterpriseArchitect
[4] http://www.sdmetrics.com/
[5] http://www.mono-project.com/Main_Page

requires some effort to get it working properly. This paper only focusses on class diagrams, while our aim is to convert use case diagrams.

Utox is a tool that is also focussed on converting class diagrams to the XMI format. It is a proof of concept from a software engineering course at LIACS [6]. We thought it was a good base to start our own application with, because of the seperation of generating the XMI file and the processing of images. It should be quite easy to add use case diagram recognition and use case diagram XMI file generation. Benefits of using Utox as a base are that Utox is developed using C++ (which we are familiar with) and uses libraries that are cross-platform.

Both quite old papers [5] and [6] describe interchanging XMI files among various CASE tools, they show that the result is bad. In [5] is described that no tool supports a two-way interchange between tools, and only a few support a one-way interchange. [6] describes that exporting XMI files from one tool and importing it in another tool required editing of the XMI file. A more recent paper [7] studies the compliance of the XMI standard on 68 tools. They found out that only 4 of the 68 tools only reached an acceptable compliance level, and more than 39% of the tools did not implement the UML specification sufficiently.

# 2 Software Design

In this chapter we will describe the overall design and requirements of our software. The next chapter describes the software in more detail with a focus on the algorithms.

## 2.1 Requirements

Before we began developing the software we set up a list of requirements that the software should be able to do:

1. The software should be able to recognize the following objects in a use case diagram:
   (a) Actors; actors that have a body-like shape, with a circular head and a vertical line as a body, including the actor name.
   (b) Use cases; ellipse shaped use cases should be recognized, including characters inside them.
   (c) System borders; the square shaped system border, including the system border name.
   (d) Relations; relations between actors and use cases, and between use cases and use cases (with exception of the directed relations, include and extends relations).
2. Cross-platform: The software should be cross-platform (Chapter 1.1).
3. The software should be able to generate an XMI file that can be imported in a CASE tool to edit the diagram; The user has to be able to import the XMI file without any errors in Visual Paradigm, and the XMI file has to include all objects that are recognized by the software.

A sidenote on relations; directed, include and extend relations will be recognized as normal relations for now. For character recognition (use cases, actor names and system borderd names) the software only has to recognize english words.

## 2.2 Main program flow

Figure 2 shows the main flow of the software we created. An image loads into the software, after loading the image, image manipulation (further: imaging) is done on the image; the shapes (use cases, actors, system borders) are recognized. When the image processing phase is finished, the character recognition phase is started to recognize the names of the use cases, actors and system borders. When both shapes and characters are recognized, the recognized characters and information about the shapes (shape category, position in the diagram) is converted to an XMI file (XMI is further described in Chapter 2.4).

## 2.3 Input formats

The image formats that can be imported are the formats that are widely used on the web [3], not only for UML models but also for a lot of other web content. The input image can be a

[6] http://www.liacs.nl

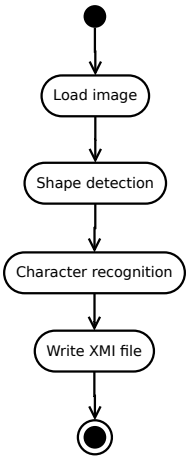


Figure 2: Execution of the software

JPG or PNG file. GIF files are not supported since the image processing library we used does not support GIF files. GIF files are quite easy to convert to JPG or PNG so this should not be too big of a problem, GIF is also the least popular of the three formats.

The use case diagrams vary from UML version 1.1 up to the latest UML version 2.4.1.

## 2.4 Output format

The best generic format that can be used for use case diagrams is XMI (XML Metadata Interchange). XMI is a standard created by the Object Management Group(OMG)®[7]. OMG is also responsible for the UML standard. XMI can be used in various CASE tools. One (major) issue applying the XMI standard is the difference in how XMI files are generated and handled in CASE tools. Quite some CASE tools have their own way of defining XMI files, and those files are not interchangeable among these CASE tools (see Chapter 1.2). The avoid the problem generating a lot of different XMI files for the various applications, we chose to generate XMI files that should be compatible with Visual Paradigm. We chose Visual Paradigm because it is free, maintained but also cross-platform software and supports UML up to version 2.x. The current XMI version of the file that gets generated by our software is XMI version 2.1, the latest version Visual Paradigm supports.

## 2.5 User interface

At first the software was created with a command line interface only, but since the command line is not very user friendly we thought it was evident to create a graphical user interface. We created the graphical user interface with Qt; Qt is a cross-platform UI framework and is quite easy to use in combination with C++. The command line interface is still available. The graphical user interface has some benefits over the command line interface, for example; if an image is imported and the object recognition has finished there is an overview of every object that has been recognized. There are five tabs in the graphical user interface, the first four tabs give overviews with the recognized objects (relations, use cases, actors and system borders). In the fifth tab the source of the generated XMI file can be found. These overviews are not available using the command line interface.

# 3 Method and techniques

This chapter describes the software in more detail; the tools we used to create the software and the algorithms that we developed are explained.

[7] `http://www.omg.org`

The software is developed using C++. We chose to use C++ because Utox (which is the base to software, see Chapter 1.2) was developed using this language. The libraries that we used are listed below. We chose for most of these libraries because Utox was already using them and some of them are well known. The advantage of these libraries is that they are cross-platform so we did not have to search for any alternative libraries while making our software cross-platform.

- OpenCV [8]: Recognition of objects
- Tesseract-ocr [9]: Recognition of characters
- TinyXML [10]: Generating XMI files, XMI files are build using XML
- Qt [11]: Graphical user interface

As described in Chapter 2.1 the software has to recognize various objects in a use case diagram:

- Use cases (ellipses)
- Relations (lines)
- System borders (rectangles)
- Actors (-)
- Characters (-)

The recognition of each of these objects is described further in this chapter.

## 3.1 Recognizing use cases

To recognize use cases we designed our own function and combined it with the OpenCV function `fitEllipse()`[9]. OpenCV has a function `HoughCircles()`[10] that can recognize circles on a given image, it applies the Hough Circle Transformation. Since we need to recognize ellipses instead of circles we vertically stretch the image (if an ellipse is resized vertically it will become a circle at the right resize value, see Figure 3). The software resizes the image several times and after each resize it runs the `HoughCircles()` function and check if there are any circles found in the image. The locations of the found circles are stored. After resizing to a maximum of 13 times the image size (value found by trial and error), the software runs `findContours()`[11] and `fitEllipse()` on the (original) image. `findContours()` finds all contours in an image, and `fitEllipse()` tries to fit ellipses on the found contours. The ellipses found with `fitEllipse()` are compared with recognized circles found by our algorithm. When this is done the circles and ellipses are checked on overlapping locations, if the locations of the found circle and ellipse overlap the location is stored as a use case. The algorithm in psuedo-code:

**Algorithm 1** Use case recognition

```
while resize != MAXRESIZE do
  resizeImage(scale=resize);
  circles := HoughCircles();
  foundcircles := add(circles);
  resize++;
end while
findContours();
ellipses := fitEllipse();
while foundcircles, ellipses do
  finalellipses := overlap(foundcircles, ellipses);
end while
```

## 3.2 Recognizing actors

For actor recognition we developed an algorithm; actors can be in various shapes (examples in Figure 4), the most straightforward way was to create an algorithm that searches for vertical lines (body-shape of an actor) and circles (head-shape of an actor). Since the combination

[8] `http://opencv.willowgarage.com`
[9] `http://code.google.com/p/tesseract-ocr`
[10] `http://www.grinninglizard.com/tinyxml/index.html`
[11] `https://qt-project.org/`

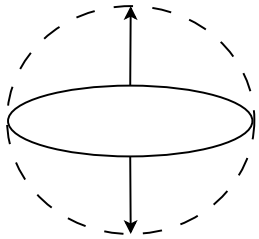

Figure 3: Vertically stretching the image turns an ellipse into a circle

of a vertical line and a circle do not match any other shape in a use case diagram it is also a very minimalistic way to search for an actor, it also covers all of the shapes in Figure 4. The algorithm searches for circles with the `HoughCircles()` function from OpenCV, for the vertical lines the `HoughLinesP()` function is used. If the top line-end (highest point of the line in the image) of a found line is in the area of a found circle, the combination of the line and circle will be marked as an actor. If an image contains a (partly) rotated actor, the actor will not be recognized because of the non-vertical body.

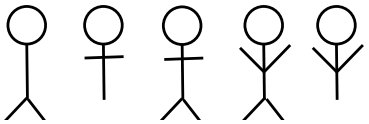

Figure 4: Various representations of an actor

## 3.3 Recognizing relations

Relation recognition is done after use case and actor recognition. Relations are found using the `HoughLinesP()`[8] function from OpenCV. The `HoughLinesP` function can find lines in an image. Not every line that is found by `HoughLinesP()` is a relation (for example the horizontal and vertical lines of a system border, or the 'body' of an actor). Only the lines with line-ends close to an actor or a use case are marked as a relation.

## 3.4 Recognizing system borders

To find system borders we used the `HoughLinesP()` function to find horizontal and vertical lines. Lines that are parallel, have the same length and have the same x or y value are added to a horizontal respectively vertical linepair. Figure 5 shows green and red lines, the green lines are marked as a vertical linepair, and the red lines are marked as a horizontal linepair. The horizontal line-ends are compared with the vertical line-ends, if they are close together (seen in as the blue ellipses in Figure 5) the lines are marked as a system border. Moreover lines that have a length of less than 50 pixels are discarded because we think system borders should be at least 50 by 50 pixels big; we assume smaller system borders do not exist, if such small system borders exist they probably represent a too small diagram to fit our valdiation criteria (further explained in Chapter 4.2). The length constraint ensures that a smaller rectangular shape is not marked as a system border. Rotated system borders will not be found, since we only recognize vertical and horizontal lines.

## 3.5 Recognizing characters

Character recognition is done with a library called tesseract-ocr. To improve the results from tesseract-ocr we had to do some imaging. Because small characters are recognized poorly, we found out that upscaling the image by a scaling factor of 20 improved the character recognition. An issue we had was that tesseract-ocr can only scan in a rectangular box for characters and not inside an ellipse shape. To prevent tesseract-ocr from recognizing non-existing characters, the software draws a white line over the use case border. An example can be found in Figure

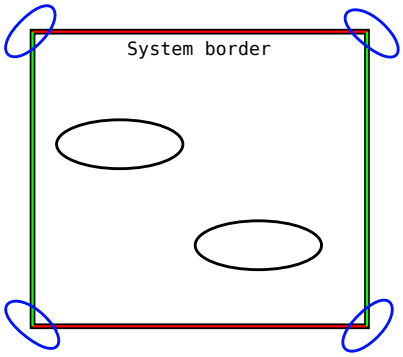


Figure 5: Found linepairs

6, where the red rectangle is the area in which tesseract-ocr recognizes the characters. An alternative to our method is to do character recognition inside the use case, but in that case characters could be missed (shown in Figure 7). Removing the use case border is a more effective way of recognizing characters from a use case.

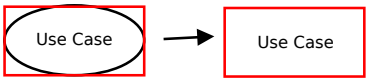


Figure 6: Erasing the use case border to improve character recognition

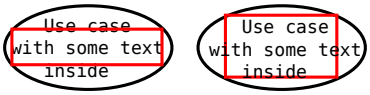


Figure 7: Missing characters with character recognition only inside the use case

Character recognition for system border names and actor names is done in a hard-coded area under or around the shape (Figure 8 shows an example of the hard-coded areas). For system borders names the hard-coded area is 30 pixels above, and 70 pixels under the top horizontal line of the system border, and 15 pixels from each vertical line (this is to avoid recognition of the line as a character). The hard-coded area for actors names is an area of 70 by 120 pixels under the actor. These values are just a guess of where the characters might be. We chose these areas because we found out that in most images the system border names are just under or above the top horizontal line of the system borders and actor names just below the actors. The guessing of the area is because we have not found any better method (yet).

## 3.6 Generating XMI

To get to know the structure of the XMI files of Visual Paradigm, we created an XMI file via the export function of Visual Paradigm. We tried to strip out as much as possible of the XMI file we exported to get a small XMI file and to make it easier to generate the XMI file and to make it easier to generate an XMI file for other tools (future work). The exported XMI file also contained a lot of information that is probably used for the operation of Visual Paradigm, but we found out it was not a problem removing it. By removing it the XMI file became a lot smaller and easier to read and understand. With the stripped XMI file from Visual Paradigm

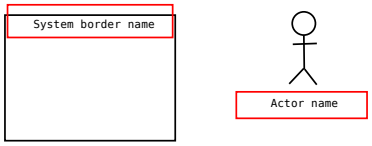


Figure 8: Hard-coded areas (red rectangles) on character recognition for system border names and actor names (not on a 1:1 scale).

we analysed how the structure of the XMI file was constructed and we tried to generate our own XMI files using TinyXML.

### 3.7 Cross-platform

At first we developed and tested (testing is described in Chapter 4.1) the software on Linux only. In a later stage we built the software for Microsoft Windows because we aimed to make the software cross-platform (Chapter 1.1). Because we had chosen to use C++ and several cross-platform libraries early on in the software development process, we were able to get the software running on Microsoft Windows quite easily. The software also runs within Wine [12]. We did not build the software for the Apple Mac operating system for the sake of time, but we don not expect difficulties here because all libraries are also available for the Apple Mac operating system.

## 4 Validation and verification

This chapter describes how the software was tested; the first paragraph describes our test construction briefly, the second chapter discusses the criteria we used for validation and the third chapter presents the test results.

### 4.1 Tests

In order to state our software meets the requirements and is usable we conducted two types of tests: a usability test and a system test.

We made the software available for Linux and Microsoft Windows. We asked a small group of users to test out our software. The users gave us feedback and features for the user interface, some of these functions (start auto generating the XMI file, automatic filenaming) were implemented. The usertest was also useful to find bugs in the software; several bugs were found by the users that we did not find.

For validating the image conversion requirements and identify requirements for future extensions we used three image sets:

- testset 1 contains images that meet our validation criteria (Chapter 4.2)
- testset 2 contains images that meet our criteria but also contains one or two elements we did not include in our validation criteria (Chapter 4.2)
- testset 3 contains images that have two or more elements that do not match our validation criteria (Chapter 4.2)

Testset 1 is the most important testset, this testset will prove if our software meets the requirements we set up (Chapter 2.1). Testset 2 and 3 are used to analyse the software dealing with objects that we did not have listed in our requirements.

Table 1 is a small table with results of the accuracy of the software; in each image we counted the number objects (found in Table 2, 4 and 6) and the number of missing objects (found in Table 3, 5 and 7), the percentage of the recognized objects that have been correctly recognized is shown in Table 1.

[12] `http://www.winehq.org`

| | Testset 1 | | Testset 2 | | Testset 3 | |
|---|---|---|---|---|---|---|
| | Mean (%) | Stdev (%) | Mean (%) | Stdev (%) | Mean (%) | Stdev (%) |
| Rec. Use cases | 92 | 18 | 81 | 31 | 74 | 35 |
| Rec. Actors | 89 | 29 | 74 | 35 | 38 | 44 |
| Rec. Relations actor-usecase | 69 | 36 | 48 | 39 | 21 | 32 |
| Rec. Relations usecase-usecase | 85 | 17 | 47 | 37 | 57 | 31 |
| Rec. System border | 92 | 26 | 88 | 33 | 81 | 46 |

Table 1: Mean and standard deviation of the objects recognized by the software

## 4.2 Validation criteria

To get the best result from the conversion of an image to an XMI file we created some validation criteria the image has to meet. The validation criteria were found while developing and testing the software and by closely looking at the requirements we set up for the software (Chapter 2.1). The validation criteria are listed below:

- For character recognition the font has to be about 80 dpi and size 10 or higher to get a proper result, also italic fonts are not recognized very well.
- The actors need to have a vertical 'body' and a circular 'head' since we search for a vertical line and a circle that are close together to mark them as an actor (see Chapter 3.2).
- The radius of the head of the actor should be bigger than 5 pixels.
- Use cases should be ellipse shaped.
- A use case cannot have a too wide shape since our algorithm will not recognize them because of the maximum stretch value of the use case recognition algorithm (see Chapter 3.1), a ratio of 1:6 (height:width) is a maximum.
- Line-ends of relations cannot be too far from a use case or an actor because they will not be marked as a relation if they are too far away (see Chapter 3.3).
- Relations should be straight lines; curvy relations will not be recognized.
- The image cannot contain other rectangular objects that are bigger than 50 pixels wide and 50 pixels high because they can be recognized as system borders (Chapter 3.4).

## 4.3 Testset 1

The first testset contained 36 images (both JPG and PNG) we gathered from the internet. The images match the criteria described above. The recognition errors are described per category. The results of the object recognition are listed in Table 2 and Table 3 The table is divided in columns for the various shapes. For each shape there are 2 columns, the number of recognized shapes by the software and the number of shapes in the input image in Table 2 and in Table 3 there is a column for objects the software missed, and a column for ghost objects[13].

As seen in Figure 9 the imported image in Visual Paradigm is almost the same as the original image, the objects and the characters are recognized very well except for the actor name and system border name because the location of the actor names and system border names is guessed (see Chapter 3.5).

### 4.3.1 Use cases

The set images contained a total of 149 use cases (an average of 4.2 use cases per image); 137 use cases were recognized. In a total of 8 images the algorithm missed use cases, 7 images were only missing 1 use case and 1 image was missing 3 use cases.

### 4.3.2 Actors

A total of 77 actors where spread among the 36 images (all images contained actors). 9 images contained recognition errors, the actor recognition missed actors on 6 images (all missing 1 actor) and 3 the of images contained 'ghost' actors, actors that were not present but the algorithm had recognized an actor.

[13] An object that is found by the software but does not appear in the image.

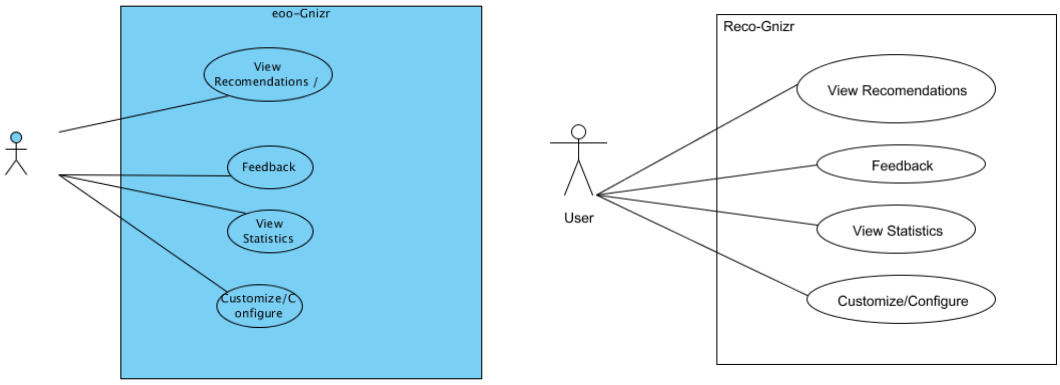


Figure 9: 20.jpg; left a screenshot of the result of the imported XMI file in Visual Paradigm, on the right the original image

### 4.3.3 Relations

We split up the relations in 2 kinds of relations; actor-use case relations and use case-use case relations. The actor relations contained no errors in 10 of the 35 images that contained actor-use case relations. The errors in the 25 images consisted of missing relations in 21 images and 4 images with recognized ghost relations. Use case-use case relations were present in 7 images, only in 2 of 7 the images the relations were properly recognized; 5 images were missing relations and in another 6 images ghost relations were found. Figure 10 shows a converted image that is missing one relation.

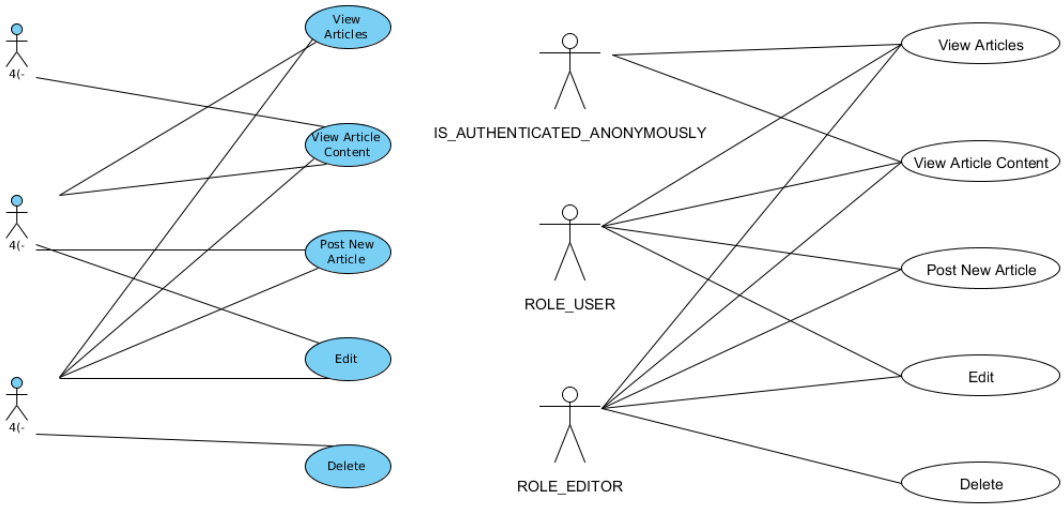


Figure 10: 27.png; left a screenshot of the result of the imported XMI file in Visual Paradigm, on the right the original image

### 4.3.4 System borders

In the set of 36 images, 29 of them contained system borders. Recognition on three images contained errors involving system borders so there is a success rate of 89.7% of sytem border recognition. The recognition missed the system borders on 2 images, and on one image the software recognized 4 system borders where there were only 3 system borders. On the 2 images where the software missed the system borders, the system borders seem to be thicker than in other images.

### 4.3.5 Character recognition

In most cases the characters of the use cases are recognized very well. In some images the recognized characters slightly differ from the original characters in the image, for example image '33.png' has a use case 'fill orders', the recognized characters are 'till orders'. Also in some cases the recognized characters have one or two forward slashes ('/') appended after the recognized characters (Figure 11), or there are some characters recognized that are not in the image. We think this happens because the software scales up the image to get a better character recognition result (Chapter 3.5); this up scaling creates some artefacts in the image. These artefacts could be recognized as characters. The result of the recognized system border names and actor names is poor. The result is poor because the character location of where the names should be is 'guessed' (see Chapter 3.5 for more information about the recognition technique).

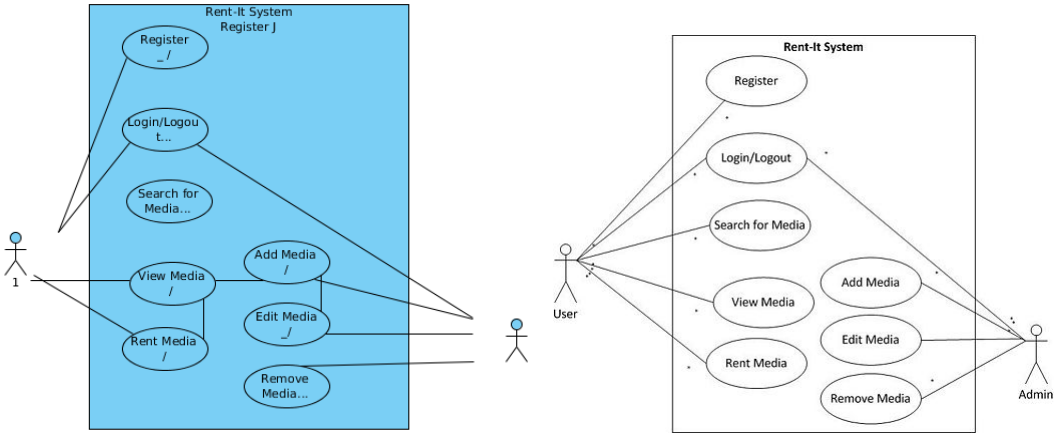


Figure 11: 8.jpg; left a screenshot of the result of the imported XMI file in Visual Paradigm, on the right is the original image

| | Actors | | Relations actor-use case | | Relations use case-use case | | System borders | | Use cases | |
|---|---|---|---|---|---|---|---|---|---|---|
| Input image | Software | Original | Software | Original | Software | Original | Software | Original | Software | Original |
| 1.jpg | 2 | 2 | 0 | 3 | 0 | 0 | 1 | 1 | 3 | 3 |
| 2.jpg | 6 | 6 | 15 | 13 | 2 | 2 | 1 | 1 | 6 | 6 |
| 3.jpg | 1 | 1 | 3 | 4 | 2 | 3 | 1 | 1 | 6 | 7 |
| 4.jpg | 3 | 3 | 5 | 4 | 0 | 0 | 1 | 1 | 4 | 4 |
| 5.jpg | 3 | 3 | 6 | 8 | 0 | 0 | 1 | 1 | 8 | 8 |
| 6.jpg | 1 | 2 | 1 | 2 | 0 | 0 | 1 | 1 | 1 | 1 |
| 7.jpg | 3 | 2 | 3 | 5 | 1 | 0 | 0 | 1 | 3 | 3 |
| 8.jpg | 2 | 2 | 8 | 9 | 3 | 0 | 1 | 1 | 8 | 8 |
| 9.jpg | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 4 | 5 |
| 10.jpg | 1 | 1 | 5 | 5 | 0 | 0 | 1 | 1 | 5 | 5 |
| 11.jpg | 2 | 2 | 1 | 2 | 0 | 0 | 1 | 1 | 1 | 1 |
| 12.jpg | 3 | 3 | 5 | 5 | 0 | 0 | 1 | 1 | 5 | 5 |
| 13.jpg | 1 | 1 | 3 | 3 | 0 | 0 | 1 | 1 | 3 | 3 |
| 14.jpg | 2 | 2 | 4 | 5 | 4 | 0 | 1 | 1 | 4 | 4 |
| 15.jpg | 2 | 2 | 0 | 4 | 0 | 0 | 0 | 0 | 2 | 2 |
| 16.jpg | 1 | 1 | 2 | 2 | 0 | 0 | 1 | 1 | 2 | 2 |
| 17.jpg | 2 | 2 | 2 | 4 | 0 | 0 | 1 | 1 | 1 | 2 |
| 18.jpg | 2 | 1 | 3 | 4 | 5 | 0 | 0 | 0 | 4 | 4 |
| 19.jpg | 1 | 1 | 4 | 4 | 0 | 0 | 0 | 0 | 4 | 4 |
| 20.jpg | 1 | 1 | 4 | 4 | 0 | 0 | 1 | 1 | 4 | 4 |
| 21.jpg | 2 | 2 | 5 | 6 | 3 | 0 | 1 | 1 | 5 | 6 |
| 22.jpg | 3 | 2 | 4 | 3 | 8 | 4 | 0 | 0 | 6 | 7 |
| 23.jpg | 3 | 3 | 10 | 11 | 0 | 0 | 0 | 0 | 5 | 5 |
| 24.jpg | 2 | 3 | 3 | 5 | 0 | 0 | 0 | 0 | 4 | 4 |
| 25.jpg | 4 | 4 | 7 | 6 | 0 | 0 | 1 | 1 | 5 | 5 |
| 26.jpg | 0 | 1 | 0 | 5 | 0 | 0 | 4 | 3 | 2 | 5 |
| 27.png | 3 | 3 | 10 | 11 | 0 | 0 | 0 | 0 | 5 | 5 |
| 28.png | 2 | 2 | 4 | 4 | 0 | 0 | 1 | 1 | 3 | 3 |
| 29.png | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 30.png | 3 | 3 | 6 | 6 | 1 | 0 | 1 | 1 | 5 | 5 |
| 31.png | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 32.png | 1 | 1 | 1 | 3 | 0 | 0 | 0 | 1 | 3 | 3 |
| 33.png | 4 | 4 | 2 | 7 | 0 | 0 | 1 | 1 | 1 | 4 |
| 34.png | 3 | 4 | 8 | 8 | 1 | 0 | 1 | 1 | 4 | 5 |
| 35.png | 2 | 2 | 5 | 6 | 1 | 0 | 1 | 1 | 6 | 6 |
| 36.png | 1 | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 3 | 3 |

Table 2: Results of testset 1

| | Actors | | Relations actor-use case | | Relations use case-use case | | System borders | | Use cases | |
|---|---|---|---|---|---|---|---|---|---|---|
| Input image | Missed | Ghost | Missed | Ghost | Missed | Ghost | Missed | Ghost | Missed | Ghost |
| 1.jpg | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2.jpg | 0 | 0 | 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3.jpg | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 4.jpg | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5.jpg | 0 | 0 | 5 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6.jpg | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7.jpg | 0 | 1 | 2 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 8.jpg | 0 | 0 | 1 | 0 | 0 | 3 | 0 | 0 | 0 | 0 |
| 9.jpg | 1 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 10.jpg | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11.jpg | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12.jpg | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13.jpg | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14.jpg | 0 | 0 | 1 | 0 | 0 | 4 | 0 | 0 | 0 | 0 |
| 15.jpg | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16.jpg | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17.jpg | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 18.jpg | 0 | 1 | 1 | 0 | 0 | 5 | 0 | 0 | 0 | 0 |
| 19.jpg | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20.jpg | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21.jpg | 0 | 0 | 1 | 0 | 0 | 3 | 0 | 0 | 1 | 0 |
| 22.jpg | 0 | 1 | 0 | 1 | 1 | 5 | 0 | 0 | 1 | 0 |
| 23.jpg | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 24.jpg | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 25.jpg | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 26.jpg | 1 | 0 | 5 | 0 | 0 | 0 | 1 | 0 | 3 | 0 |
| 27.png | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 28.png | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 29.png | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 30.png | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 31.png | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 32.png | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 33.png | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 3 | 0 |
| 34.png | 1 | 0 | 2 | 2 | 0 | 1 | 0 | 0 | 1 | 0 |
| 35.png | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 36.png | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 3: Missed relations and ghost relations in testset 1

## 4.4 Testset 2

Testset 2 consists of 23 images where each image does have one or two properties that do not match our validation criteria (Chapter 4.2). Table 4 and Table 5 show the results of running the 23 images through our tool. Figure 12 shows image 10.jpg that has include and extend relations. Both relations are recognized as normal straight relations. There is also a use case with a horizontal line in it (an extension of a use case), this use case has been recognized as a normal use case.

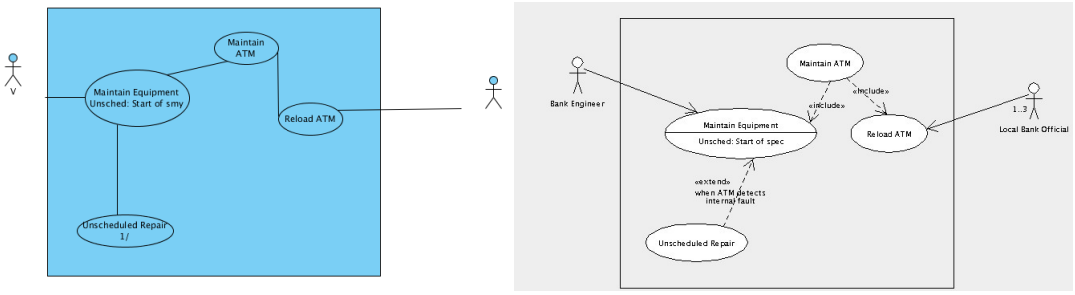


Figure 12: 10.jpg; left a screenshot of the result of the imported XMI file in Visual Paradigm, on the right is the original image

### 4.4.1 Use cases

Most of the images in which use cases are not recognized in testset 2 are images that have use cases that do not match our validation criteria (Chapter 4.2), an example: a too low height:width ratio. Three images also have use cases that contain a line inside the use case (an example in Figure 13), these use cases seem to be recognized worse than use case without the line. This was rather surprising to us because the use cases have a well formed ellipse shape; so we did not expect difficulties with recognizing these use cases.


Figure 13: A use case of an image of testset2

### 4.4.2 Actors

The software missed actors on 10 of the 23 images in testset 2. Three images contained actors that are similar as the actor shown in Figure 14, on all three images the software missed some actors. We assume it is the shape of the head of the actors, although it is circular as stated in the validation criteria, inside the circular head a line can be found. These three images also contain the use cases with the lines described in Chapter 4.4.1.

Figure 14: Actors in 3 images of testset 2

### 4.4.3 Relations

Include and extend relations seem to be recognized as good as non-special (non-directed) relations. 15 images in the set contain at least one include or extend relation, there does not seem to be much of a difference between recognizing these relations and the non-special relations.

| | Actors | | Relations actor-use case | | Relations use case-use case | | System borders | | Use cases | |
|---|---|---|---|---|---|---|---|---|---|---|
| Input image | Software | Original | Software | Original | Software | Original | Software | Original | Software | Original |
| 1.png | 1 | 1 | 2 | 4 | 0 | 1 | 1 | 1 | 4 | 5 |
| 2.jpg | 1 | 2 | 1 | 2 | 0 | 0 | 1 | 1 | 2 | 2 |
| 3.jpg | 2 | 1 | 50 | 28 | 16 | 1 | 0 | 0 | 23 | 28 |
| 4.png | 5 | 4 | 0 | 0 | 0 | 6 | 1 | 1 | 3 | 7 |
| 5.jpg | 3 | 7 | 0 | 16 | 4 | 10 | 1 | 0 | 15 | 18 |
| 6.jpg | 0 | 0 | 0 | 9 | 2 | 4 | 1 | 1 | 6 | 6 |
| 7.jpg | 5 | 5 | 10 | 13 | 2 | 2 | 1 | 1 | 8 | 8 |
| 8.jpg | 1 | 3 | 1 | 4 | 2 | 22 | 0 | 1 | 9 | 26 |
| 9.jpg | 3 | 4 | 3 | 4 | 0 | 0 | 1 | 1 | 3 | 3 |
| 10.jpg | 2 | 2 | 2 | 2 | 3 | 3 | 1 | 1 | 4 | 4 |
| 11.jpg | 2 | 2 | 3 | 2 | 3 | 5 | 0 | 0 | 5 | 5 |
| 12.jpg | 3 | 3 | 10 | 10 | 5 | 0 | 1 | 1 | 7 | 7 |
| 13.png | 1 | 2 | 0 | 3 | 0 | 2 | 1 | 1 | 1 | 4 |
| 14.jpg | 2 | 2 | 3 | 6 | 0 | 0 | 1 | 1 | 3 | 9 |
| 15.png | 1 | 1 | 1 | 2 | 3 | 5 | 1 | 1 | 7 | 7 |
| 16.jpg | 1 | 2 | 2 | 3 | 1 | 3 | 0 | 0 | 3 | 4 |
| 17.jpg | 1 | 3 | 5 | 14 | 1 | 4 | 0 | 0 | 15 | 15 |
| 18.png | 1 | 1 | 3 | 3 | 3 | 3 | 0 | 0 | 5 | 5 |
| 19.png | 2 | 2 | 3 | 4 | 3 | 6 | 0 | 1 | 7 | 7 |
| 20.png | 1 | 1 | 1 | 1 | 1 | 3 | 1 | 1 | 4 | 4 |
| 21.png | 0 | 4 | 0 | 8 | 0 | 0 | 1 | 1 | 0 | 4 |
| 22.png | 0 | 1 | 0 | 1 | 2 | 4 | 1 | 1 | 5 | 5 |
| 23.png | 1 | 1 | 0 | 2 | 0 | 0 | 1 | 1 | 2 | 2 |

Table 4: Results of testset 2

### 4.4.4 System borders

System border recognition failed on three images, on two images no system borders were found. The third image does not have a system border but has a comment box that has been recognized as a system border. (Figure 15 shows the commentbox from 5.jpg).

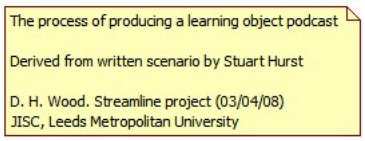


Figure 15: Commentbox from image 5.jpg

### 4.4.5 Character recognition

Recognition of the characters inside the use cases gave a good result, most characters were properly recognized if the fontsize and dpi were high enough (Chapter 4.2). 12.jpg has a smaller font but most of the characters still got recognized. Again recognized characters contained slashes, and both actor names and system border names were recognized poorly as seen in Chapter 4.3.5,

| | Actors | | Relations actor-use case | | Relations use case-use case | | System borders | | Use cases | |
|---|---|---|---|---|---|---|---|---|---|---|
| Input image | Missed | Ghost | Missed | Ghost | Missed | Ghost | Missed | Ghost | Missed | Ghost |
| 1.png | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 2.jpg | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3.jpg | 0 | 1 | 9 | 14 | 1 | 17 | 0 | 0 | 5 | 0 |
| 4.png | 0 | 1 | 3 | 0 | 6 | 0 | 0 | 0 | 4 | 0 |
| 5.jpg | 4 | 0 | 16 | 0 | 7 | 1 | 0 | 1 | 3 | 0 |
| 6.jpg | 2 | 0 | 9 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| 7.jpg | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8.jpg | 2 | 0 | 3 | 0 | 20 | 9 | 1 | 0 | 18 | 0 |
| 9.jpg | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10.jpg | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11.jpg | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 0 |
| 12.jpg | 0 | 0 | 1 | 1 | 0 | 5 | 0 | 0 | 0 | 0 |
| 13.png | 1 | 0 | 3 | 0 | 2 | 0 | 0 | 0 | 3 | 0 |
| 14.jpg | 0 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 6 | 0 |
| 15.png | 0 | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| 16.jpg | 1 | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 1 | 0 |
| 17.jpg | 2 | 0 | 9 | 0 | 3 | 0 | 0 | 0 | 0 | 0 |
| 18.png | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19.png | 0 | 0 | 1 | 0 | 3 | 0 | 1 | 0 | 0 | 0 |
| 20.png | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| 21.png | 4 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 4 | 0 |
| 22.png | 1 | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| 23.png | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 5: Missed relations and ghost relations in testset 2

## 4.5 Testset 3

Testset 3 also consists of 23 images, these images contain 2 or more properties that do not match our criteria. Table 6 and Table 7 show the results. Figure 16 shows a screenshot of our software with the recognized use cases having a red ellipse on top of them.

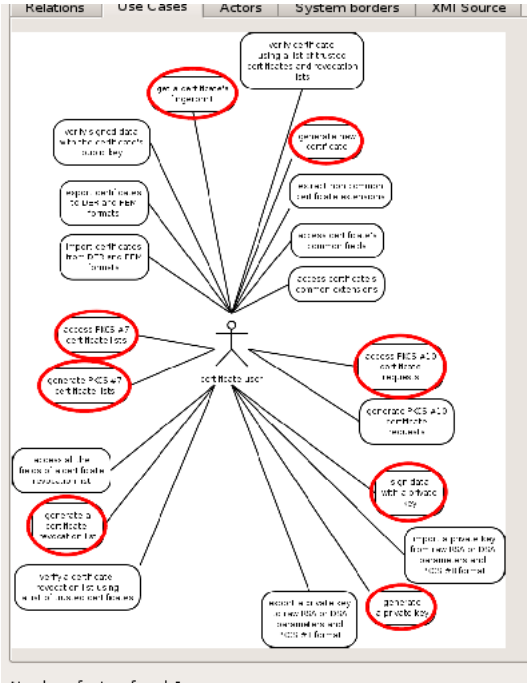


Figure 16: Recognized use cases on 8.png

### 4.5.1 Use cases

Figure 16 shows the use cases that are recognized in image 8.png, the use cases are more rectangle shaped rather than ellipse shaped. Only some use cases are recognized but most of them are not. The same issue seems to appear in 11.jpg, this image also contains more rectangle shaped use cases. Only 2 of the 17 use cases are recognized.

### 4.5.2 Actors

There are quite some ghost actors[14] recognized in this set, 10 images contained ghost images with 7.jpg even containg 7 ghost images. On four images 2 ghost were actors recognized, on 3 images 1 ghost actor was recognized and on 2 images 4 ghost actors were recognized.

[14] Actors recognized by the software that are non-existing in the input image.

### 4.5.3 Relations

A lot of double relations[15] are recognized in some images, this is something that should not happen and was a fault in our software.

### 4.5.4 System borders

Testset 3 contains some objects that have a rectangular shape with a height and width of more than 50 pixels (Chapter 4.2). These rectangular shaped boxes have been recognized as a systemborder in some figures; for an example Figure 17.

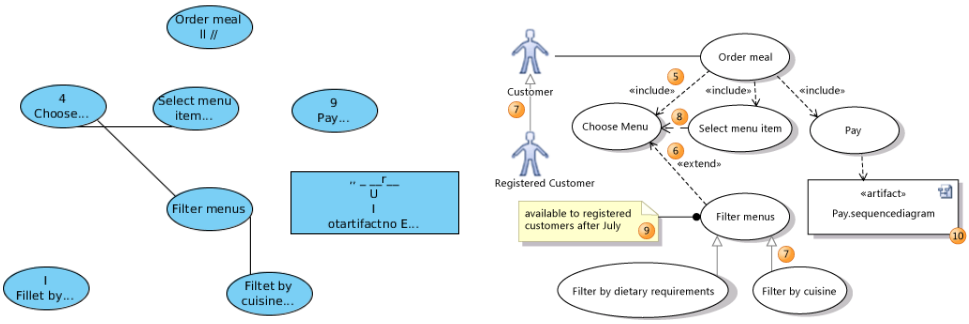


Figure 17: 1.png; left a screenshot of the result of the imported XMI file in Visual Paradigm, on the right the original image

### 4.5.5 Character recognition

Character recognition on images of the third testset were quite the same as in the second testset (Chapter 4.4.5). 4.png contained very small characters, only some characters got recognized well.

[15] Relations that are recognized twice but only appear once in the input image.

| | Actors | | Relations actor-use case | | Relations use case-use case | | System borders | | Use cases | |
|---|---|---|---|---|---|---|---|---|---|---|
| Input image | Software | Original | Software | Original | Software | Original | Software | Original | Software | Original |
| 1.png | 0 | 2 | 0 | 1 | 3 | 8 | 1 | 0 | 7 | 7 |
| 2.jpg | 0 | 3 | 0 | 4 | 0 | 0 | 0 | 2 | 3 | 3 |
| 3.jpg | 0 | 2 | 0 | 5 | 5 | 4 | 0 | 0 | 6 | 6 |
| 4.png | 7 | 6 | 9 | 17 | 8 | 2 | 0 | 1 | 8 | 12 |
| 5.jpg | 3 | 3 | 3 | 6 | 7 | 13 | 0 | 0 | 10 | 12 |
| 6.png | 1 | 2 | 0 | 2 | 0 | 0 | 1 | 1 | 0 | 2 |
| 7.jpg | 7 | 4 | 0 | 37 | 37 | 0 | 0 | 0 | 23 | 23 |
| 8.png | 1 | 1 | 4 | 20 | 0 | 0 | 0 | 0 | 8 | 20 |
| 9.png | 0 | 1 | 0 | 2 | 4 | 5 | 0 | 0 | 6 | 6 |
| 10.png | 2 | 2 | 10 | 24 | 11 | 3 | 1 | 1 | 14 | 21 |
| 11.jpg | 4 | 5 | 0 | 19 | 0 | 0 | 3 | 0 | 2 | 17 |
| 12.jpg | 0 | 4 | 0 | 4 | 7 | 12 | 0 | 0 | 11 | 11 |
| 13.jpg | 0 | 5 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 4 |
| 14.jpg | 5 | 8 | 39 | 21 | 18 | 0 | 0 | 0 | 9 | 11 |
| 15.png | 2 | 2 | 0 | 0 | 0 | 0 | 5 | 2 | 3 | 3 |
| 16.png | 2 | 2 | 1 | 4 | 1 | 3 | 0 | 0 | 6 | 6 |
| 17.jpg | 0 | 0 | 0 | 0 | 2 | 5 | 1 | 1 | 6 | 6 |
| 18.jpg | 2 | 2 | 3 | 2 | 0 | 4 | 0 | 0 | 5 | 5 |
| 19.jpg | 2 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 2 | 2 |
| 20.jpg | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 1 | 3 | 12 |
| 21.jpg | 1 | 1 | 3 | 3 | 4 | 7 | 0 | 0 | 9 | 9 |
| 22.png | 0 | 2 | 0 | 6 | 2 | 2 | 0 | 1 | 5 | 5 |
| 23.jpg | 0 | 3 | 0 | 5 | 2 | 5 | 0 | 0 | 5 | 5 |

Table 6: Results of testset 3

| | Actors | | Relations actor-use case | | Relations use case-use case | | System borders | | Use cases | |
|---|---|---|---|---|---|---|---|---|---|---|
| Input image | Missed | Ghost | Missed | Ghost | Missed | Ghost | Missed | Ghost | Missed | Ghost |
| 1.png | 2 | 0 | 1 | 0 | 5 | 0 | 0 | 1 | 0 | 0 |
| 2.jpg | 3 | 0 | 4 | 0 | 0 | 0 | 1 | 0 | 2 | 1 |
| 3.jpg | 2 | 0 | 5 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 4.png | 1 | 2 | 9 | 2 | 2 | 4 | 0 | 0 | 4 | 0 |
| 5.jpg | 1 | 1 | 4 | 0 | 9 | 5 | 0 | 0 | 2 | 0 |
| 6.png | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| 7.jpg | 4 | 7 | 37 | 0 | 0 | 37 | 0 | 0 | 2 | 0 |
| 8.png | 0 | 0 | 16 | 0 | 0 | 0 | 0 | 0 | 12 | 0 |
| 9.png | 1 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 10.png | 1 | 1 | 14 | 0 | 1 | 10 | 0 | 0 | 7 | 0 |
| 11.jpg | 1 | 0 | 19 | 0 | 0 | 0 | 0 | 3 | 15 | 0 |
| 12.jpg | 4 | 0 | 4 | 0 | 6 | 0 | 0 | 0 | 0 | 0 |
| 13.jpg | 5 | 0 | 8 | 0 | 0 | 0 | 0 | 1 | 3 | 1 |
| 14.jpg | 7 | 4 | 13 | 31 | 0 | 18 | 0 | 0 | 2 | 0 |
| 15.png | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 |
| 16.png | 0 | 0 | 3 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 17.jpg | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 |
| 18.jpg | 0 | 0 | 0 | 1 | 4 | 0 | 0 | 0 | 0 | 0 |
| 19.jpg | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 20.jpg | 4 | 0 | 31 | 0 | 0 | 0 | 1 | 0 | 10 | 0 |
| 21.jpg | 0 | 0 | 0 | 0 | 4 | 1 | 0 | 0 | 0 | 0 |
| 22.png | 2 | 0 | 6 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 23.jpg | 3 | 0 | 5 | 0 | 4 | 1 | 0 | 0 | 0 | 0 |

Table 7: Missed relations and ghost relations in testset 3

# 5 Conclusions

In this chapter we describe the conclusion of our work, and the improvements that can be done on the software. The main question; How to convert a binary UML use case diagram (image file) to a more generic form in an automated way? (see Chapter 1.1) is answered through this thesis. The other (maybe more important) question: If it is possible to convert a use case diagram to a more generic format in an automated way is this method also a reliable method? is answered in Chapter 4 and this chapter.

Overall we think our software works as we expected (except for the relation recognition) because in testset 1 the mean recognition percentage is around 90% on use cases, actors and system borders. The software can be used to convert simple use case diagrams to XMI files that can be imported into Visual Paradigm or Enterprise Architect (Chapter 5.6). The software is still a bit too insufficient to convert more advanced use case diagrams to XMI files as seen from the results on testset 2 and 3.

## 5.1 Recognition of use cases

We think use case recognition works well since 92% of all use cases of testset 1 are recognized. In testset 2 and 3 the success rate is a lower because of images that contain use cases that are not ellipse shaped, some use cases are rectangular shaped. Since the software is developed to recognize ellipse shaped use cases (Chapter 3.1) it is not a surprise that these rectangular shaped use cases will not be recognized. Something that was a bit of a surprise was the bad recognition of the use cases described in Chapter 4.4.1.

## 5.2 Recognition of actors

The results of testset 2 and 3 are a lot worse than the results of testset 1. The number of ghost actors found (especially in testset 3, see 4.5.2) is not really what we expected to happen. Recognition of other shaped actors and rotated actors could be added to the software in the future.

## 5.3 Recognition of relations

Relation recognition definitely needs some work; as seen in Table 5 and Table 7 there are a lot of ghost relations found or the relations are not recognized at all. We believe this can be improved by changing the parameters of the `HoughLinesP()` function or by using another algorithm to find the lines (relations).

Directed relations, include relations and extend relations that have been recognized (see Chapter 4.4.3), are not really surprising. With relation recognition we look for lines in the image (more about that in Chapter 3.3), images containg directed relations, include and extend relations still have these lines. They may have some extras added like an arrow or the `<< include >>` string but there is also a line (relation) just as with a normal relation that the software recognizes.

A large amount of double relations have been recognized in testset 3 (more about double relations in 4.5.3). We found out that the double relations appeared by a bug in our software, the bug has been fixed after we found out the bug was there.

## 5.4 Recognition of system borders

Most system borders have been recognized in testset 1; in less than 10% of the images errors were found. This error percentage is higher on testset 2 and especially on testset 3. This is not a surprise since testset 3 contains comment boxes, system borders that are a little bit rotated and use cases that are quite big and rectangular.

System borders that are (partly) rotated, or the system borders in images that are (partly) rotated will not be recognized, maybe there could be an extension on the recognition algorithm that checks if the input image is rotated.

## 5.5 Recognition of characters

The character recognition works quite well if the font that has to be recognized is big enough (as described in our validation criteria, Chapter 4.2) and the software finds the right location of the actor name and system border name (more about that in Chapter 3.5). In some cases characters got replaced, this is not too big of a problem because editing text is quite easy in most CASE tools. Actor names and sytem border names are recognized poor, this is because the recognition is done in a hard-coded area (see Chapter 3.5 for more information) and the software does not check if the actor name or system border name is really there. To improve the recognition algorithm for actor names and border names, there could be searched for groups of pixels along the actor or system border to guess the location of the names. Of course guessing the location of the names is not optimal but it could be more effective than our approach.

## 5.6 Generated XMI files

The generation of XMI files works well because we were able to import all XMI files that we generated without any errors into Visual Paradigm. All objects that had been recognized were presented in Visual Paradigm.

We also found out that we were able to import the XMI files into Enterprise Architect. The difference between importing the diagram into Visual Paradigm and Enterprise architect is a visual representation of the diagram in Visual Paradigm. In Enterprise Architect the objects of the use case diagram are only loaded into the software. To make the visual representation of the diagram it requires some manual labour. XMI generating for various tools would be a convenient feature that could be added to the software although this is not a solution for the different XMI files that get accepted by the various CASE tools, it would be better if every CASE tool would accept the XMI standard created by OMG.

## 5.7 User interface

In our experience the graphical user interface has quite some advantages over the command line interface. Browsing for input images is a lot easier and faster, also the graphical interface gives a better overview on what the software has actually recognized.

Some improvements on the graphical user interface could be useful. For example; it would be handy if the user is able to correct misrecognized characters in the software itself rather than in the CASE tool. Manual deletion of ghost objects[16] would also be a huge improvement, as seen in testset 3 there are quite a lot of ghost actors. Deletion of ghost objects could also be done in the CASE tool itself obviously, but since our software shows where (ghost) objects are recognized, it is easier and less time consuming to delete them before loading the XMI file into the CASE tool rather than after loading them into the CASE tool and then searching for them.

# 6 Future work

Future work on our software is also (partly) described in Section 5. The user experience on the software can be improved with various extensions on the software; for example: editing the text of the use cases inside the user interface or recognition of more UML diagrams. Improvements on use case diagram recognition in terms of speed could also make for a better user experience. Releasing the software for the OS X operating system could attract more people to use the software.

Research shows that a lot of CASE tools use their own XMI standard (Section 1.2), it would be convenient if all CASE tools would apply the same XMI standard. Future research could show why CASE tools apply their own standard and if this is really required. Maybe research can also show that these XMI 'standards' can be converted (by developing software) to the XMI standard OMG created.

[16] Objects that are detected by the software but are non-existing in the input image.

# References


[1] Ohst, D. and Welle, M. and Kelter, U.; Differences between versions of UML diagrams. In SIGSOFT Softw. Eng. Notes, vol. 28 (2003)

[2] Dobing, B., Parsons, J.: How UML is Used. In Commun. ACM, vol. 49 (2006)

[3] W3Techs.com - Usage Statistics of Image File Formats for Websites: http://w3techs.com/technologies/overview/image_format/all. August 2013

[4] Karasneh, B., Chaudron, M.R.V.: Extracting UML Models from Images. In CSIT2013, Amman, Jordan.

[5] Persson, A., Gustavsson, H., Lings, B., Lundell, B., Mattsson, A., Ärlig, U.: OSS tools in a heterogeneous environment for embedded systems modelling: an analysis of adoptions of XMI. In: Proc. of the 5th Workshop on Open Source Software Engineering (May 2005)

[6] Lundell, B., Lings, B., Persson, A., Mattsson, A.: UML Model Interchange in Heterogeneous Tool Environments: An Analysis of Adoptions of XMI 2. In: Wang, J.,Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, Springer, Heidelberg (2006)

[7] Eichelberger, H., Eldogan, Y., Schmid, K.: A Comprehensive Survey of UML Compliance in Current Modelling Tools. In: SE 2009. LNI, vol. 143

[8] OpenCV, HoughLinesP(); http://opencv.willowgarage.com/documentation/cpp/imgproc_feature_detection.html#cv-houghlinesp

[9] OpenCV, fitEllipse(); http://opencv.willowgarage.com/documentation/cpp/structural_analysis_and_shape_descriptors.html#cv-fitellipse

[10] OpenCV, HoughCircles(); http://opencv.willowgarage.com/documentation/cpp/imgproc_feature_detection.html#cv-houghcircles

[11] OpenCV, findContours(); http://opencv.willowgarage.com/documentation/cpp/imgproc_feature_detection.html#cv-houghcircles