Extracting UML models from images

Conference Paper · March 2013

DOI: 10.1109/CSIT.2013.6588776

CITATIONS
20

READS
13,115

2 authors:

B. Karasneh
Leiden University
10 PUBLICATIONS **143** CITATIONS
SEE PROFILE

Michel R. V. Chaudron
Eindhoven University of Technology
263 PUBLICATIONS **4,187** CITATIONS
SEE PROFILE

Extracting UML Models from Images

Bilal Karasneh
Leiden Institute of Advanced Computer Science (LIACS)
Leiden, The Netherlands
bkarasne@liacs.nl

Michel R.V. Chaudron
Dept. of Computer Science and Engineering
Chalmers and Gothenburg University, Sweden
chaudron@chalmers.se

Abstract — **Software modeling is an active field of research. In this field, UML is considered the standard for software modeling. Studying UML models is important to understand their effectiveness in software development. Ideally, researchers want to study UML models from documentations from industrial software projects. Given their limited availability, we resort to collecting UML models from internet. A big problem for studies of UML models is that UML models are published mostly in image formats (such as JPEG). These images do not include the model information that is available for UML models when saved in a CASE (Computer Automated Software Engineering) tool format or the XML-based version of the UML interchange format: XMI. Current CASE tools cannot recognize information from images. In this paper we propose the Img2UML tool to solve this problem. The Img2UML extracts UML Class models from images such that these models can be loaded into CASE tools for further study. The Img2UML tool exports UML Class models into XMI files which can be read with the StarUML CASE tool. We performed a validation which shows that Img2UML successfully handles a large class of UML Class images.**

Keywords—UML Class models, Image recognition, XMI

I. Introduction

UML is used for modeling software because it shows a high level description of a system. UML models are created during different stages of the development process and also during maintenance. We recognize the following needs for extracting UML models from images. Firstly, in software projects, UML models are typically made during the early (design) phases of a project. As projects progress, emphasis in developer activity shifts to coding and developers tend to ignore updating the UML model. One reason for the lack of updating is that the UML model has been copied from a CASE tool and pasted in a software design document that is created using a word-processing tool. In such a work-processing tool the UML model is now represented in an image format and does not allow updating. Often, the software design document is used in subsequent development, but the CASE-tool version of the model is neglected, leaving a project only with an image-format version of their design. Clearly it is desirable to recover the UML model from such image formats for updating and maintenance.

Secondly, empirical studies in software engineering aim to study artefacts from software development projects. For such empirical studies that use source code, the huge number of open source software projects has been a much used source of data. Contrary to common belief, it turns out that UML modeling is used in open source projects, but that UML models are stored in image formats. Through our Img2UML tool we will be able to open up open source projects as a much needed source of UML models for empirical researchers in software modeling.

Nowadays there are many UML CASE tools, which give many features like creating and modifying UML models, exporting models into XMI files, export models into images and automatically generating the user code backbone.

UML CASE tools typically can work with different file types for storing UML models. Unfortunately, these file types cannot be exchanged with another one. These file types can be divided into 3 kinds: First, native tool format – this is exclusive to the CASE tool and cannot be opened from another UML CASE tool. Second, XML Metadata Interchange (XMI) format: XML files contain information about the data they contain and how its structure. XMI are XML files that store all information of a UML model. XMI files can be loaded by the CASE tool by which they were created. However, each CASE tool uses its own XMI structure as a result of which is not necessarily recognizable by other CASE tools. Third, image formats: CASE tools can export UML models into an image to publish the model. Exporting a model into an image allows a model to be viewed by stakeholders that do not own a CASE tool. However, until now CASE tools do not support extracting UML models from images.

In this paper we propose the Img2UML tool, which automatically extracts UML Class models from images, without the need to redraw the model using a CASE tool.

Class models are the most common of the UML models. They play a central role in describing software structure, and provide a static description of system components [13]. Also Class models are the bridge between software design and the implementation. Recognizing class models from diagrams presents several challenges: these diagrams may draw from a large set of symbols, they typically include a fair amount of text, and relationships between classes may be drawn in a large variety of manners [3].

This paper is organized as follows. Section II, discusses related work. An overview of the Img2UML tool is presented in section III. Section IV explains the approach used. Some implementation aspects of Img2UML are presented in section V. In Section VI, we present the validation. Results are discussed in section VII. Section VIII discusses limitation of

This work is supported by Erasmus Mundus Partnership JOSYLEEN

Img2UML. Finally, the conclusion and future work are in Section IX.

II. RELATED WORK

We can categorize engineering diagrams depending on how they were made into two categories: hand-drawn and made via modeling/CASE-tool (using predefined geometric shapes). Tools for recognition of hand-drawn images are usually called sketch recognition tools. Although both categories seem close, sketch- recognition tools use significantly different approaches.

In [7] authors showed that the interpretation of engineering drawing is complex and theory-weak process. They mentioned that systems supports this technology is still difficult to put into engineering applications.

In [1], authors mentioned two motivations of engineering diagram recognition. Firstly, the amount of engineering diagrams images in diverse design and education related scenarios are non-trivial. Secondly, engineering diagram recognition enhances the supportive value of diagrams in design. They proposed four functional models for recognizing computer-made and hand-made engineering diagrams.

In [8] authors presented a system to recognize a large class of computer-made engineering drawings which include flowcharts, logic and electrical circuits, and chemical plant diagrams. In [9] authors proposed a framework for hand-made engineering drawings recognition using a case-based approach.

In [3-6], authors proposed an approach for recognizing UML class diagrams from whiteboard sketches. This approach fixes a problem of current CASE tools interfaces. Creating models using CASE tools interfaces takes more time than sketching the same model. By adding the time that users need to learn how to use a CASE tool interface to create a model, CASE tools interfaces distract users from their specific purpose. Using sketching software, gives users a large freedom to draw as they like. In [3], authors show two limitations of CASE tools. First, developers worked independently on separate computers. Second, developers are forced to learn how to use CASE tools. The authors proposed a tool which recognizes sketches of UML Class models, Use Case models and Sequence models. In [4], authors propose a sketch tool to recognize Class models, Use Case models and Sequence models. In [5], authors propose a tool which supports sketching Class models. Text in the Class model has to be entered using a virtual keyboard. The tool supports exporting model to XMI.

In [6], authors are interested in how to use UML models in the process of designing of software. Their tool supports drawing of hand-made Class models. They used a combination of geometry-based and graffiti-based approach to balance mistakes in free sketching and recognition accuracy. They use two predefined symbols: diamond and triangle. Problems in accuracy of recognition of symbols are explained. Some symbols in UML Class models are not supported.

In general, sketching tools are an easy and fast way to create and (re)draw UML Class models. However, the size of sketched UML Class models becomes bigger than UML Class models created by CASE tools - especially text written by hand becomes larger than when typed using a keyboard. Sketching tools need large tablet or the area of sketching should be big enough for relax feeling in drawing. Although sketching tools have benefits, as in [2], when a large number of Class models needs to be entered, sketching tools take more time and effort for redrawing models. Also, the algorithms that are used in recognizing UML in sketching-tools typically make use of information regarding the movement and order of drawing elements in the diagram. This information is not available in our setting. As a result these algorithms cannot simply be transferred to our setting. Our approach differs from engineering diagram recognition in through the fact that we enrich the recognized shapes with meaning (class, method, attribute). This surpasses the level of recognizing the geometric shapes in engineering diagrams.

III. OVERVIEW OF IMG2UML

For recognizing Class model images, we have to recognize shapes, symbols, lines and text. Also we need to identify the role of each diagram element in the model.

For example, texts in a Class model could be a class name, an attribute for class number three, a function for the first class, comments, etc. Recognizing Class model images without recognizing the text would make the tool of less practical value. Fig. 1 shows an UML Class model image downloaded from the Internet. It contains 10 classes, each class has attributes and functions. Redrawing the model in Fig. 1 using any CASE tool, requires 8 to 10 minutes. Typing the text needs

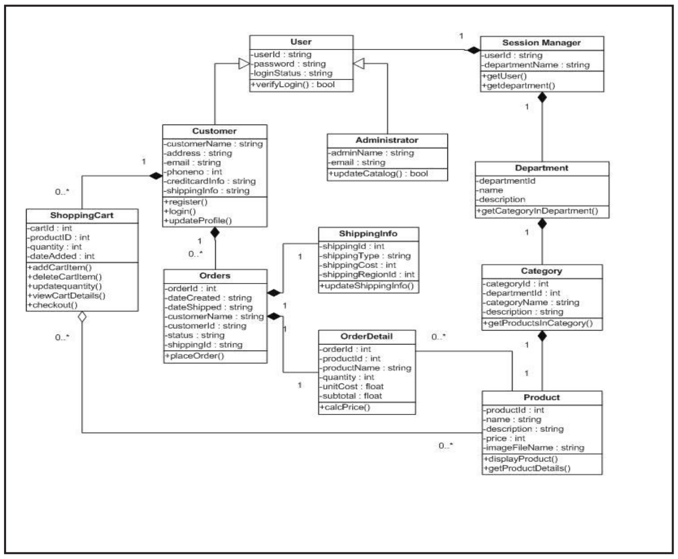

Figure 1. UML Class model.

some more time. So it is useful to use a method that can recognize text automatically. Therefore, we need to define methods for recognizing classes, relations between classes and texts in UML Class model images, and methods that determine which role texts play in the diagram.

Finally, we need to save information extracted from the image into a file. This file should be compatible with a CASE tool. Therefore we decide that Img2UML shall be able to produce output in the XMI format which is meant to be a standard XML-based exchange format for UML models. Fig. 2 shows the design of the Img2UML tool.

Due to the fact that most UML CASE tools support XMI, Img2UML exports Class model images to XMI file. Because it is impossible to export UML Class model images to a format that is compatible with all UML CASE tools, one UML CASE tool and its XMI structure studied carefully to generate output compatible with XMI.

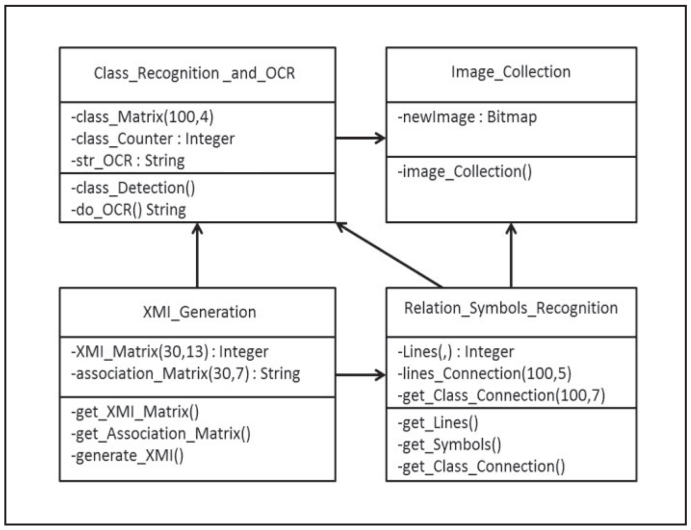

Figure 2. Class model of the Img2UML software

This XMI file may not work perfectly with other CASE tools, but UML Class model images can be loaded using this CASE tool. In this case, Img2UML In particular, we test this by loading the XMI into the StarUML tool. More details about this are in Section III.

IV. Approach of Img2UML

In this section we describe the design and some implementation aspects of Img2UML in detail. In Fig. 3 shows an overview of the tool.

A. *Input Images*

Img2UML can read nine types of image formats which are the most common images formats found on the internet: BMP, EMF, EXIF, JPG, JPEG, GIF, PNG, TIFF, WMF. These kinds of images could be black and white or colored images.

Img2UML can load one image at a time, or a set of multiple images as input. The images in the set may be of varying size, type and color, and may originate from different UML tools.

The Img2UML tool converts all images to the BMP format. The remaining image processing is subsequently standardized to work with this one image type. After conversion to BMP, a grayscale filter is applied. The next step in the process is segmentation.

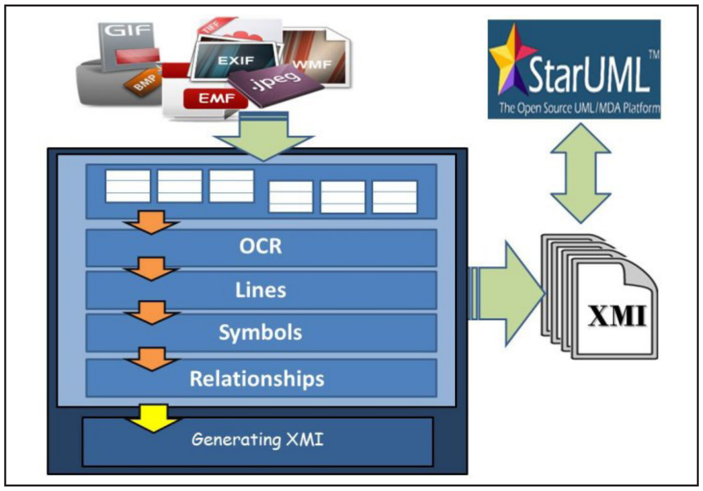

Figure 3. Overview of Img2UML.

B. *The image processing technique*

The processing is consists of four consecutive parts:

1) Detecting Classes in the image: the first process technique is detecting classes in the image. For detecting classes, rectangles are detected in the image. Rectangles are detected by using Aforge.NET Framework image processing library after some quality improvements of the images. These improvements consist of applying multiple filters on the image like Gaussian Sharpen, Grayscale and Threshold.

2) Recognizing text in the classes: Rectangles that represent class may have three different parts that can each contain text: class name, attributes names and functions names. For identifying these parts, the system tries to detect (two or three) rectangles that are contained in a larger rectangles. By definition, the uppermost rectangle contains the class name. The middle rectangle contains attribute names and the lowest rectangle contains functions names.

To recognize text automatically, Microsoft Office Document Imaging (MODI) is used as optical character recognition (OCR) library.

3) Detecting Relationships: Dependencies between classes are depicted in class diagrams through lines that connect rectangles that represent classes. Across different images, a range of styles of drawing such connecting lines is found: straight, hooked (horizontal and vertical), diagonal, curved, solid, dotted. Our tool can detect straight connections, but not curved connections. In Fig. 4 and Fig. 5 class2 and class3 connect to class1 via an aggregation relationship. Img2UML recognizes relationships as presentated in Fig. 4 only, hence not the style where two lines are merged into one.

We assume that each line connect two classes. Some CASE tools allow connecting more than two classes in the same line – using the same relationship.

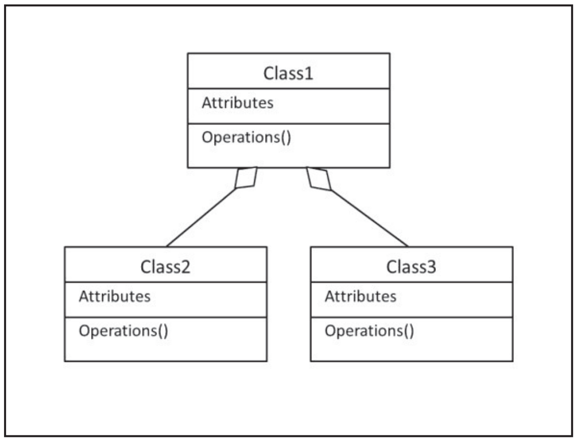

Figure 4. Kind of Association as generalization in Class model.

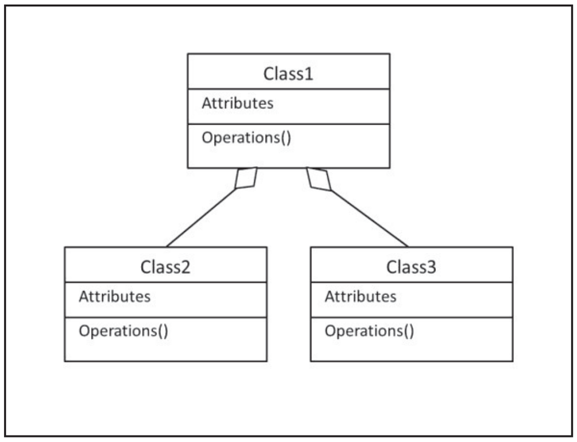

Figure 5. Another presentation of the generalization relationship in class diagram

In this process we just detect which classes have relationships with each other without detection of the kind of the relationships. Relationships detection is divided into 3 parts: solid line detection, dashed line detection and connecting segments of lines.

a) Solid line detection: This step is divided into two parts: solid straight lines detection and solid diagonal lines detection. Solid straight lines detection contains horizontal lines and vertical lines. The tool starts detecting solid straight lines whose length is greater than or equal to 20 pixels. This value is choosen to reduce problems of detecting symbols lines as straight lines. Experimenting with many different values for this lenght showed that using a value of 20 produced the best result. All solid straight lines that are detected are stored in a 2-dimensional array called “lines”. Lines which are smaller than 20 pixels are saved in another 2-dimensional array called “checked solid lines”. This array is used to check if a line segment is a part of another line already detected before. When image resolution is low, some lines could be detected as two or more lines.

Solid diagonal line detection is divided into two parts: left solid diagonal lines and right solid diagonal lines. Again, the length of solid diagonal lines is greater than or equal 20. This value is chosen after testing many different values and it resulted in the best performance. All solid diagonal lines detected are stored in a lines array. Any solid diagonal lines detected whose length is less than 20 is ignored.

Detecing solid diagonal lines is more difficult than detecting solid straight lines. Especially because some CASE tools present relationships as ‘staired’ solid diagonal lines. This makes the detection more difficult and could separate a line into multiple line fragments.

b) Dashed lines detection: Dashed lines detection is seperated into two parts: dashed straight lines detection and dashed diagonal lines detection. Img2UML only supports dashed straight lines detection. Dash straight lines are divided into two parts: horizontal dashed lines and vertical dashed lines. Each dashed line detected contains at least two dashes seperated with one space. The length of each space must be at least three pixels, and lengths of dashes vary. Also the length of the first dash and last dash are usally smaller than other dashes in a dashed line. Dash lines detected are stored in a lines array.

c) Connecting segments of lines: Here detected lines (for both solid and dashed separately) that are close to other lines are connected to form connections between classes. This part is divided into two parts: merging lines to make one line and connecting lines to connect classes. First merging lines to make a line. After a line is detected, proximity to nearby lines is checked. In solid line detection, a checked solid lines array was made which contains small lines detected. Those small lines could be a part of other lines in the lines array. This case may occur because of poor resolution of the image. For example, in Fig. 1 the class Product is connected to the class ShoppingCart via one horizontal line and one vertical line. The horizontal line could be detected as two lines, one is bigger than the other. So, by checking both ends points for each checked line and matching it with ends points for each line in the lines array we can find connection between lines. A connection is discovered when an end point for one line is close (within some threshold) to an end point for another line in the checked line array. When a connection is found between two lines, the end point of the lines are determined and used as endpoints of a new line. In this way, lines can ‘grow’ increasingly longer. A second case is where lines are connected to classes. For the same example, in Fig. 1 the class Product is connected to the class ShoppingCart via one horizontal line and one vertical line. By connecting the horizontal line and the vertical one, the result becomes a new line with one end point near class Product and other end point near class ShoppingCart. The same check is applied to all solid lines and dashed lines encountered.

4) Detecting UML Class model symbols: In this stage we detect the type of relations between classes detected earlier in the process. In class models we have four kinds of relationships: associations, generalization, dependency and realization. Association relations have four kinds of associations: association, direct association, aggregation and composition. The UML notation for class model has six symbols that need to be dectected for determining seven kinds of relationships. UML class model symbols are small geometric shapes. We use shape recognition algorithms from the Aforge.NET Framework image processing library [12] to recognize some symbols, and for other symbols we use some geometric-based operations to recognize them.

Segmentation is the main part for processes 1, 3 and 4. Segmentation is used to change the representation of an image into something that is easier to analyze. Until now, there is no general solution for image segmentation [14], so we use segmentation and geometry-based approaches to enhance the accuracy of recognition. The resolution of images affects the accuracy of the recognition process. The resolution of the images in our test set is reported in the appendix.

C. Generating XMI file

After recognizing classes and relationships, all model information is extracted from the image and needs to be organized. XMI is the most used format used for this purpose. OMG defines the XMI standards based on the XML standard [10]. XMI files can be generated in different ways depending on the version of XMI and the structure used to organize model information. Most CASE tools support multiple XMI standards for loading UML models, yet may use their own structure to export UML model to XMI. For generating an XMI file that is compatible with a CASE tool, the versions and the structure of XMI used in the CASE tool should be known and understood. For this study we use the StarUML CASE tool [16] as a reference: the XMI file generated by Img2UML must be compatible with the StarUML CASE tool. StarUML is an open source UML/MDA platform. XMI version 1.3 is used. For each image from the input one separate output XMI file is generated.

V. IMPLEMENTATION OF IMG2UML

For the implementation of Img2UML; VB.NET is used as programming language, AForge.NET as image processing library compatible with .NET and Microsoft Office Document Imaging (MODI) is used as OCR library.

Fig. 6 shows the Img2UML user interface. The tool has four buttons, two buttons for selecting images, one for selecting one or multiple images and other for selecting an image directory.

When selecting one image; the image will be displayed in size 540x500 pixels. When two or more images are selected, these will be displayed in a table, each image in a size of 150x100 pixels. One other button is for exporting the selected UML Class model images to XMI files. This button becomes enabled after selecting one or more images. The last button is to exit the application.

After pressing "Start Exporting XMI", the tool shows the progress of the image recognition process: detecting classes, lines and symbols.

Recognition problems could hinder the generating of XMI file. When an XMI file is generated for an image, a success icon is shown on top of the image. Otherwise a fail icon is shown on top of the image. Fig. 7 shows the successful exporting of a UML Class model image to XMI file.

When multiple images are selected, images are presented in a table that consists of (at most) 5 columns for each row. Fig. 8 shows the results of exporting 13 images. In Fig. 8, the tool shows that 10 XMI files are generated successfully for 10 UML Class model images out of 13 input images.

To overcome recognition mistakes, StarUML can be used to correct mistakes in a UML Class model manually by importing the XMI file generated from Img2UML.

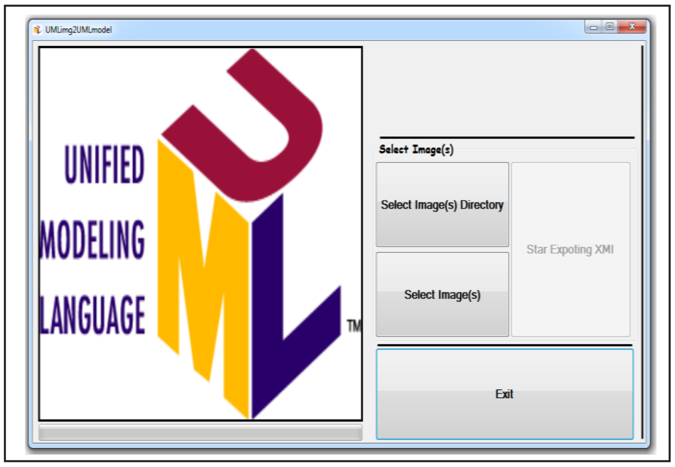

Figure 6. Img2UML interface.

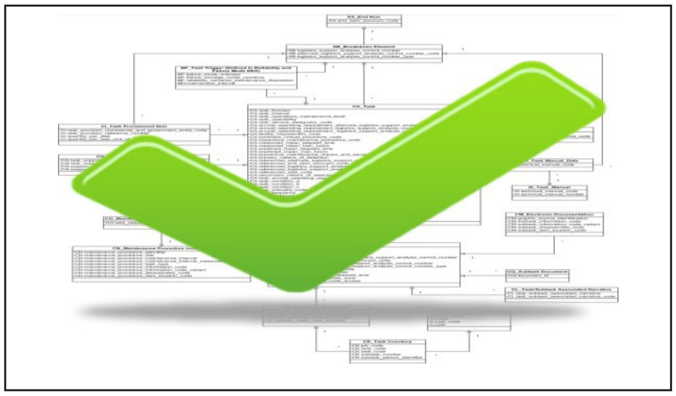

Figure 7. Img2UML interface.

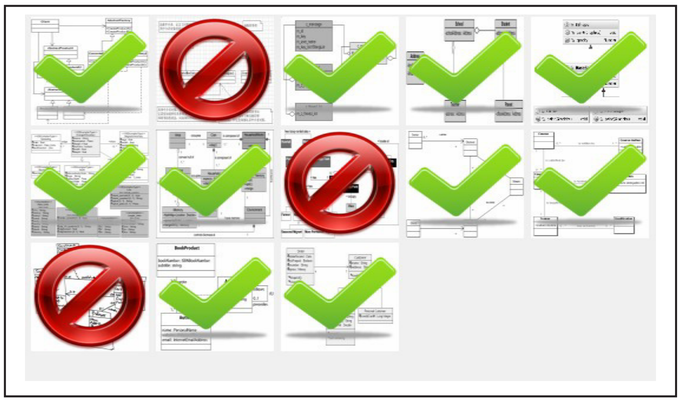

Figure 8. Img2UML interface.

VI. VALIDATION OF UMLIMAGE2UMLMODEL

For validating our tool, we choose 10 different images which vary in size, type and resolution. Images used for the validation are available in [15].

The results of the validation are shown in Table I. Images properties for images used in the validation are available in Table II in Appendix A.

The first image contains 10 classes, 13 relationships and 13 symbols. Img2UML detects all classes, relationships and symbols successfully. For image number five, it has seven classes, 10 relationships and 10 symbols. The tool detects seven classes, and nine out of 10 relationships. Moreover, it detects two additional relationships which do not exist in the original image. As a result it reports a total of 11 relationships detected.

The tool successfully detected eight symbols out of 10, and detected other symbol incorrectly. The total number of detected symbols is nine. In this image there are four errors: two relationships and two symbols.

In image number 10, the tool successfully detected 18 classes, 18 relationships out of 20 with one additional relationship which does not exist in the original image. The total number of detected relationships is 19. For symbols, 12 symbols are detected out of 20 successfully, and five symbols are detected incorrectly. Total number of symbols detected is 17. For image number 10, there are nice detection errors.

VII. RESULTS DISCUSSION

In our experiments class-detection accuracy is 100%, relationship accuracy is 97% and symbols accuracy is 85%. We use image filtering, segmentation and geometric-based approach to recognize classes, relations and symbols. The accuracy of detecting classes is higher than that for relations and symbols, because classes are quite big and regular shaped which makes them easier to recognize.

For detecting lines, the accuracy is also high. Many solutions were used to overcome recognition problems that arise from crossing lines, straight dashed lines and connected straight lines, diagonal lines and dashed lines. There are some additional relationships detected which do not exist in the original image. In our sample of diagrams, this relates to the use of shadows around classes. For some classes that have thick shadows, the inner parts of the rectangles are detected, and the outer parts are detected as lines which are finally interpreted as a relationship from the class to itself.

For detecting symbols, the accuracy is lower than for class detection and lines detection because symbols are small and there are some complications in distinguishing between symbols. Also image resolution affects the detecting of symbols. When there is some text around symbols, the accuracy of detecting the kind of symbol becomes lower than symbols that have no text around it.

Text on relationships may cause some problems in detection. After applying some filters and segmentation such text may become part of (dotted) lines. In the experiment it affects detection of some symbols which are recognized incorrectly. In another case, text at a line causes the line to be cut which affects the accuracy of the line detection.

All additional relationships which do not exist in the original images are relations from a class to the same class. Classes' shadow and text on relationships cause this problem in images 5, 6, 8, 9, 10.

VIII. LIMITATIONS OF UMLIMAGE2UMLMODEL

Img2UML has some limitations which are described next.

A. *Failed XMI file creation*

The image resolution affects the accuracy of recognition. Problems in image recognition may occur at class recognition, relationships recognition or text recognition. Not all problems

TABLE I. VALIDATION RESULTS

	Classes		Relationships			Symbols			Total errors
	Original	*Detected*	*Original*	*Correct detected*	*Total Relationships detected*	*Original*	*Correct Detected*	*Total symbols detected*	
1.	10	10	13	13	13	13	13	13	0
2.	11	11	12	12	12	12	12	12	0
3.	8	8	8	8	8	8	8	8	0
4.	10	10	11	11	11	4	4	4	0
5.	7	7	10	9	11	10	8	9	4
6.	14	14	15	15	17	13	12	12	3
7.	9	9	9	9	9	13	12	12	1
8.	9	9	9	9	12	5	5	7	5
9.	13	13	12	12	13	12	8	11	5
10	18	18	20	18	19	20	12	17	9

in recognition processes prevent the creating of XMI output. For example, if OCR failed in recognizing attributes names in a class, then in the XMI output of this class will appear without attributes. In other cases, if a class is not detected and this class contains a lot of attributes and operations; this case could prevent generation of an XMI file. Most problems arise from the failure to distinguish between text and UML Class model symbols or text and lines. This problem was encountered when a huge of number of UML Class model symbols is detected or a huge number of lines is detected. When no XMI file is generated, the tool shows the fail logo on the image.

B. CASE tools use own standards

CASE tools present information of UML Class model in different ways and Img2UML follows the StarUML standard. This causes problems for extracting information from images created by a CASE tool used different presentation standards. For example, to present private or public attributes or operations, some CASE tools like StarUML use (+) for private and (-) for public, but other CASE tools use special symbols to present it. Img2UML recognizes (+) as private and (-) as public based on OCR, but cannot recognize special symbols from other tools. In particular, other tools may use pictures to denoted public and private properties. This requires dedicated recognition techniques for those symbols.

C. OCR and Relationships recognition

OCR is one of the most difficult challenges for the tool. The problem of OCR is in general also considered as one difficult problem in the image recognition domain in general and is still under improvement.

OCR recognition accuracy also varies between images. OCR errors occur in recognizing numbers of attributes and operations, and names of attributes or operations. Also Sometimes OCR recognizes an open bracket followed by a close bracket “( )” as “O” character or “int” as “inl” and so on which make the distinguishing process not optimal.

Until now Img2UML cannot reliably detect diagonal dashed lines. Also it cannot detect text on relationships. This text causes problems in the recognition techniques which considers this text sometimes as dashed lines or as UML Class model symbols. If the image does not contain text on relationships, the opportunity to export the image into XMI becomes higher.

D. Differences between images in types and properties

Img2UML still has some problems that are common in digital image processing. Images vary between each other in type, size and resolution. If for a large amount of elements that are found in the image, the system cannot determine which role they play in the diagram, the system decides not to produce XMI output.

IX. Conclusions and Future Work

In this paper we present the Img2UML tool and its underlying methods. This tool can convert images of UML Class models (from various image formats) to UML Class models (represented in XMI). Engineers, developers, researchers, teachers and students may find this tool useful for extracting model information from UML Class model images. Img2UML saves time and effort. Img2UML accepts one or more UML Class model images which may vary in type, color and size. Then it processes them and generates an XMI output file for each of them. We hope this tool and its extensions helps for empirically studying UML Class models. XMI files generated by Img2UML are compatible with the StarUML CASE tool which is an open source CASE tool. XMI generated by the tool can be used in software design metrics tools like SDMetrics [11] to use their features for design measurement and design comparison. Discussion of the tool approach and the implementation is presented. Validation of Img2UML shows that the tool works better for detecting classes and relationships than for detecting symbols. The tool still needs improvements to reduce recognition problems and recognition mistakes. StarUML can be used for correcting recognition mistakes.

For future work, we will extend the tool to support generating XMI files for UML Class model images which are compatible with other CASE tools. Also we consider making this tool available as an online application. Finally we hope to make an online database for UML Class models. We also plan to extend Img2UML to support UML Use Case model images and UML Sequence Diagram images.

References

[1] L. Fu, L. Kara, “From Engineering diagrams to engineering models: Visual recognition and application,” Computer-Aided Design, vol. 43, issue 3, pp. 278-292, March 2011.

[2] S. Lahtinen and J. Peltonen, "Enhancing Usability of UML CASE-Tools with Speech Recognition," In proceedings of IEEE Symposium on Human Centric Computing Languages and Environments, pp. 227-235, 2003.

[3] E. Lank, J. Thorley and S. Chen, "An interactive system for recognizing hand-drawn UML diagrams", Proceedings of the IBM Center for Advanced Studies Conference, CASCON 2000, Mississauga, Ontario, Canada, November 2000, pp. 1 - 15.

[4] E. Lank, J. Thorley, S. Chen, D. Blostein, “On-line Recognition of UML Diagrams,” Proc. Sixth Int’l Conf. Document Analysis and Recognition, Seattle, Washington, Sept. 2001, 356–360.

[5] T. Hammond, R. Davis, Tahuti: a geometrical sketch recognition system for UML class diagrams, in: Proceedings of the AAAI Symposium on Sketch Understanding, AAAI Press, 2002, pp. 59–66.

[6] L. Qiu. Sketchuml, “The design of a sketch-based tool for uml class diagrams,” In Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications, ED-MEDIA, 2007.

[7] S. Ablameyko, S. Uchida. Recognition of engineering drawing entities: Review of approaches. International Journal of Image and Graphics, vol.7, no.4, pp.709-733, Oct. 2007.

[8] Y. Yu, A. Samal, S.C. Seth, A system for Recognizing a Large Class of Engineering Drawings, IEEE Trans. on PAMI, 19(8), pp 868-890, 1997

[9] L. Yan and L. Wenyin. Engineering drawings recognition using a case-based approach. In ICDAR, Edinburgh, volume 2886, pages 190–194, 2003.

[10] Object Management Group - http://www.omg.org/

[11] http://www.sdmetrics.com/

[12] Aforge.NET Framework – Computer Vision and Artifitial Intelligence http://www.aforgenet.com/

[13] A. Maraee, M. Balaban, "Efficient Recognition of Finite Satisfiability in UML Class Diagrams: Strengthening by Propagation of Disjoint Constraints," In the Second International Conference on Model Based Systems Engineering, MBSE'09, Israel.

[14] V. Vashisht, T. Choudhury and T. Prasad, "Sketch Recognition using Domain Classification," In: IJACSA - International Journal of Advanced Computer Science and Applications, Special Issue (2011) , p. 1-9.

[15] http://www.liacs.nl/~bkarasne/Img2UML/

[16] StarUML - The Open Source UML/MDA Platform http://staruml.sourceforge.net/en/

Appendix A:

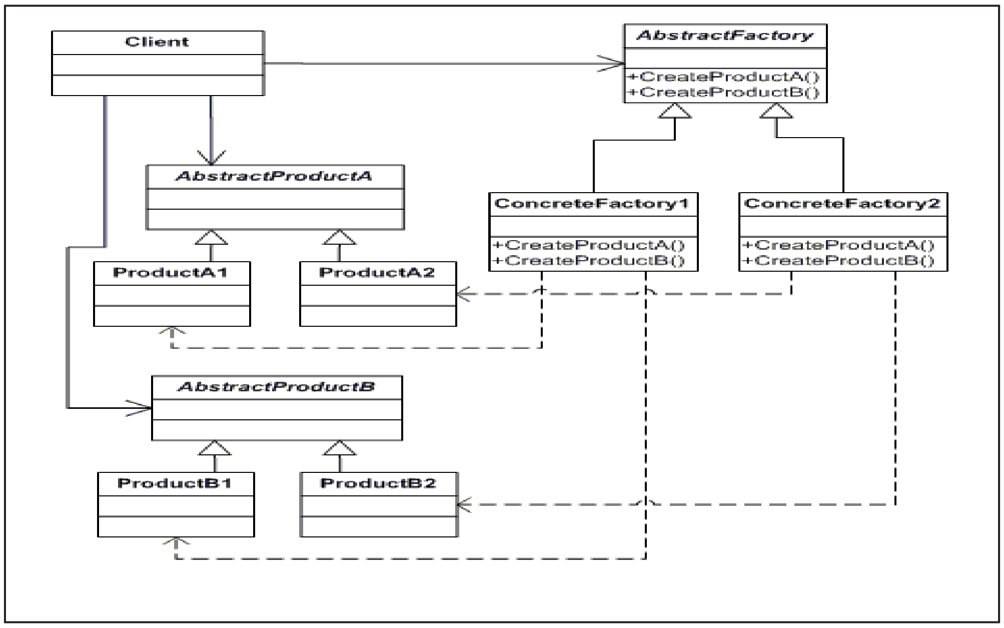

Figure 9. UML Class model image(1).

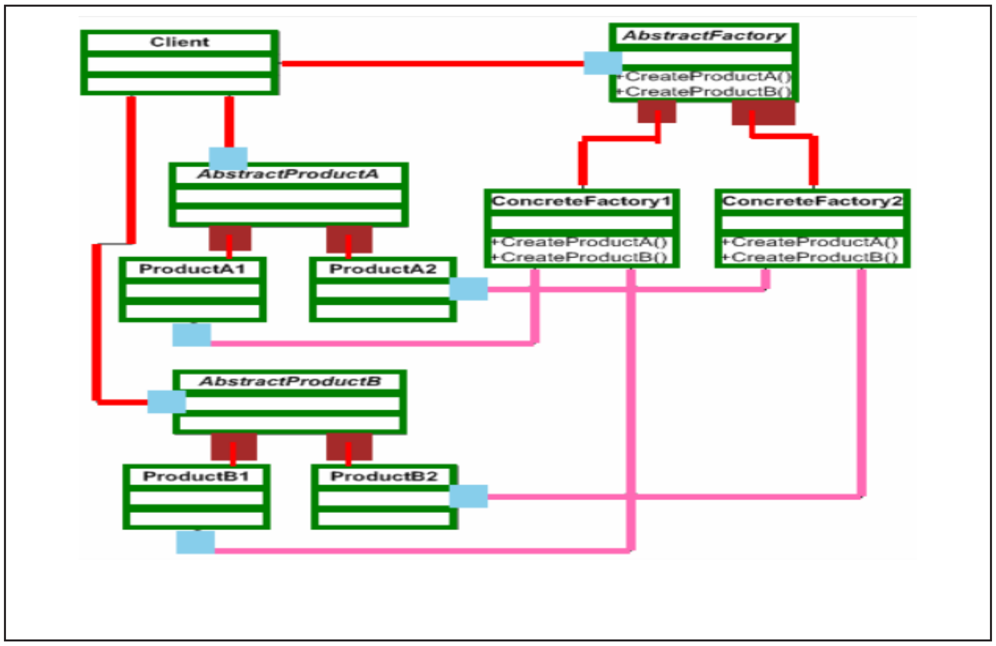

Figure 10. Recognition of UML Class model image (1)

TABLE II. IMAGE PROPERTIES

	Image type	**Dimensions**		**Horizontal Resolution (dpi)**	**Vertical Resolution (dpi)**	**Bit Depth**
		Width (pixels)	**Height (pixels)**			
1.	GIF	437	471	96	96	8
2.	JPEG	700	700	96	96	24
3.	PNG	598	534	96	96	32
4.	Bitmap	488	442	96	96	24
5.	JPEG	675	508	96	96	24
6.	GIF	600	972	96	96	8
7.	JPEG	535	363	295	295	8
8.	JPEG	721	877	96	96	24
9.	PNG	1288	1184	96	96	24
10	PNG	1107	1519	96	96	24