UACEE International Journal of Advances in Computer Science and its Applications – IJCSIA
Volume 3 : Issue 2 [ISSN 2250 – 3765]
Publication Date : 05 June 2013

IMG2XMI : Enhancement of Environment-Independent Data Exchange

Peerapat Thawatsoontorn
Department of Computer Engineering
Chulalongkorn University
Bangkok 10330, Thailand
Destinypeerapat2728@gmail.com

Yachai Limpiyakorn
Department of Computer Engineering
Chulalongkorn University
Bangkok 10330, Thailand
Yachai.L@chula.ac.th

***Abstract*— In this paper, the approach of transformation an image of UML activity diagram to its XMI data format is presented. The images of UML activity diagrams contain three major components, that is, Diagram Layout, Diagram Flow and Data Text. In order to recognize these three components, the modules of OCR, Object detection, and Line detection will be executed. Any text in UML activity diagram images are captured by OCR. On the other hand, the activity diagram elements and arrows are analyzed by the Object and Line detection. Once all the objects and their relations have been detected, an XMI file shall be created. The prototype software called IMG2XMI has been implemented in this research work. It uses an open source library for helping the part of image processing. The proposed approach is also useful for the transformation of other UML diagram images to the XMI format in order to enhance the working environment independence.**

***Keywords*—image processing, UML activity diagram, XMI, software process improvement**

I. Introduction

UML activity diagrams are widely used for describing the behaviors of the systems in various domains. In the area of software engineering, activity diagrams are commonly used to model both procedural computation and organizational processes such as organizational process models, business models, and various workflows. However, it is difficult to create UML activity diagrams that accurately describe the behaviors of the systems. Moreover, modeling the complex systems is resource consuming and error prone. The research [1] thus invented the Action Description Language (ADL), which is a Domain Specific Language (DSL), to automatically create UML activity diagrams. The approach could prevent misconception and inconsistencies in process modeling, as well as ensure that the constructed activity diagrams are conformance to OMG specification [2]. Whereas [1] proposed the preventive approach to generating activity diagrams from ADL scripts, [3] presented the reverse approach to verifying UML activity diagrams from their ADL semantic models. The XMI format of UML activity diagrams is required as the input into the system implemented in [3]. However, different organizations may own different working environments. UML activity diagrams created by different tools may be hardly opened or edited in different workplaces, e.g. that of developers and customers. Documenting as images could be an alternative solution for reviews the diagrams at different sites. This research thus presents a method and implements a component called IMG2XMI to transform the image files of UML activity diagrams to XMI format as part of input preparation step of the system implemented in [3]. This would result in the enhancement of environment-independent data exchange via the tagged language underlying UML diagrams, or XMI.

II. Background

A. *Optical Character Recognition (OCR)*

OCR is the well-known technology used for the conversion of scanned images of handwritten or typewritten, including typewritten on old newspaper [4], into machine-encoded text. Basically, the purpose of conversion is to store the original hard-copy paper document into the electronic information system, so that it can be electronically searched via online services easier. Several OCR software exist in the market either proprietary such as Microsoft Office Document Imaging; or open source such as FreeOCR, Tesseract, SDK, etc. Example OCR applications includes Car License Plate Recognition for record evidence to enforce the traffic rules, and several mobile OCR applications.

B. *Image processing*

An essential technique to analyze or transform images is called Image processing. This technique has been widely used to visualize, measure image patterns or image characteristics. Image processing can be applied for various domains such as face detection, augmented reality, fingerprint scan, etc. Additionally, some techniques are used to recognize meaning of the images such as feature extraction [5].

C. *UML Activity Diagram*

In software development, modeling is the way to visualize your design and check it against requirements before the developer team starts to code. The Unified Modeling Language (UML) is a modeling method that comprises a language and also a procedure for using the language to construct models. Nowadays, UML is widely applied for modeling the systems via a collection of diagrams which are graphical presentation of a set of elements, most often rendered as a connected graph of things and relationships. UML includes nine such diagrams for use cases, static structures (class and object diagrams), behavior (state-chart, activity, sequence and collaboration diagrams), and implementation (component and deployment diagrams). Each of those diagrams is used for describing functionality or behaviors of software systems. In this paper, the focus is on UML activity diagrams which are often used to illustrate the behavior, or process of systems by using notations. In activity diagrams, there are nodes, flows, text, controls such as decision, loop, fork, etc. A node represents an action and a flow describes a sequence of actions contained in the system.

III. Research Methodology

IMG2XMI can be considered as a component of which the functionality is to extract information from activity diagram images to build an XMI file as the input into the system implemented in [3]. IMG2XMI will load the activity diagram image file and prepare some input data. Three main components of the activity diagram images will then be processed consisting of Diagram Layout, Diagram Flow and Data Text. The method to generate the XMI file of the input activity diagram image is shown in Fig. 1.

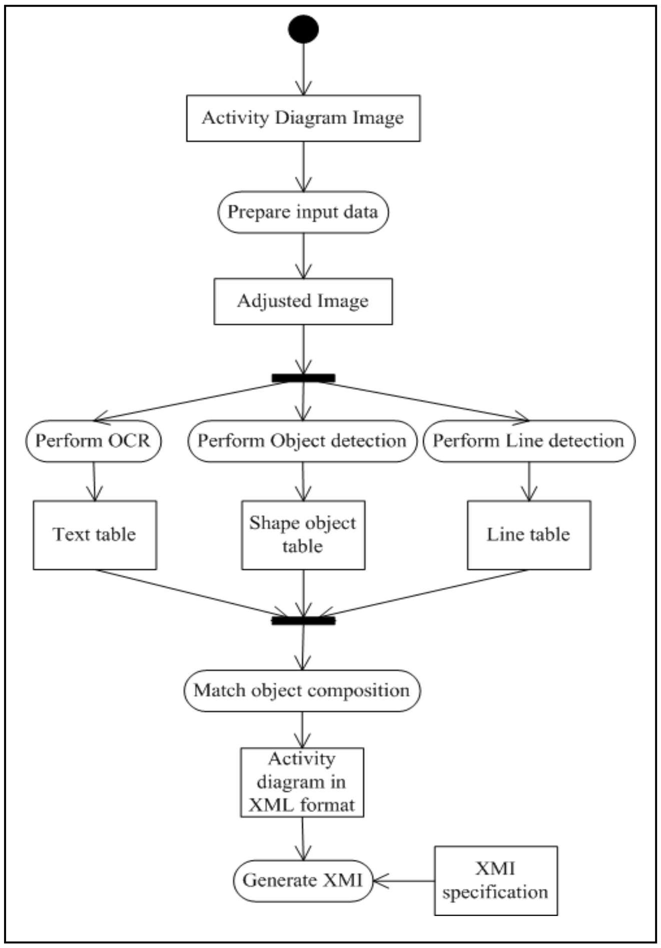


Figure 1. Research methodology.

A. Prepare input data

The first step is to prepare input data from the activity diagram image. The color image will be transformed into gray scale as well. The output from this step will be a 2-dimensional array containing object metadata to be used for object and line detection in following step.

B. Analyze image components

The method to analyze the activity diagram image consists of three sub-processes: 1) text extraction by OCR, 2) shape object detection, and 3) line detection. Each process can be briefly described as follow:

- Performing OCR is to extract the text in the image as well as to keep the location of the text on the image denoted by XY-coordinates (x, y) to be used in the next step, Match object composition.
- Performing object detection will detect the shape of any notations, called node, in the activity diagram image. Moreover, the details of position (x, y) of the CONs (Center Of Node) and metadata for each node will be stored.
- Performing line detection process is responsible for detecting the flow lines, the beginning and end of the line including flow direction, namely up, down, left, or right. Additionally, in order to analyze the flow line, the process needs to recognize the line patterns if it is arrow head or triangle shape, associated with its direction.

C. Match object composition

Once the image components have been analyzed, three sets of data are obtained consisting of the data of text, notation objects, and flows. The next step will do the matching composition by analyzing the position of each component using the Euclidean equation as shown in (1).

$$\text{Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (1)$$

The distance between components will be compared with the defined threshold. If the distance is not greater than the threshold, those components are related, meaning that a node with text inside or a line connected to a node. The output of this process is an XML file, of which the structure is close to XMI format.

D. Generate XMI

This process generates the XMI file based on the XMI specification defined in [3], which conforms to XMI version 2.0. The output of XMI file will be rendered by visualization tools to create the activity diagram equivalent to the input image file.

UACEE International Journal of Advances in Computer Science and its Applications – IJCSIA
Volume 3 : Issue 2 [ISSN 2250 – 3765]
Publication Date : 05 June 2013

IV. Case Study

This section briefly describes how to generate XMI file from the activity diagram image as shown in Fig. 2 using the method explained in the previous section.

Most activity diagrams are created by UML tools. The image of the activity diagram shown in Fig. 2 is created by using the "export graphics" function of ArgoUML [6] tool which is open source software. In this work, Emgu CV [7], and open source SDK are used as the OCR and image processing tools, respectively.

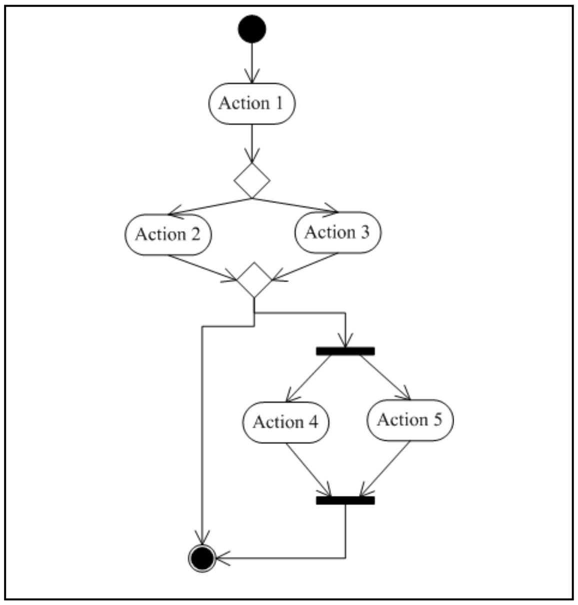


Figure 2. Activity diagram image as a case.

The activity diagram image (Fig. 2) will be loaded into the OCR module for text extraction processing. The output from this module shall be a set of data consisting of text and its XY-coordinates on the 2-dimensional plane. First, an individual character will be recognized. Then, these individual characters will be processed for character concatenation to make single text. After having performed OCR, all characters will be recognized, including their XY position. Characters will be concatenated to a text by considering the distance between their positions. Next, all character positions will be calculated to find center position of the text and its position boundary in order to analyze compositions using Match object composition module. As shown in Table 1, each text has three coordinates: center, lower right corner, and upper left corner. The center XY-coordinates are calculated from

$$X_{center} = (X_{upper} + X_{lower}) / 2,$$

and $$Y_{center} = (Y_{upper} + Y_{down})/2.$$

TABLE I. TEXT TABLE RESULT

Text	XY-coordinates		
	Center	Lower right corner	Upper left corner
Action 1	X_{c1}, Y_{c1}	X_{d1}, Y_{d1}	X_{u1}, Y_{u1}

The input image will also be processed by Object detection module. The functionality of this module is to classify all the object shapes, namely activity nodes, transition nodes, initial node, and final node. Similar to OCR, the output of this process is a set of object nodes associated with their center XY-coordinates of node calculated from the same formula as stated in OCR module. Table 2 shows the information stored for each shape object detected.

TABLE II. SHAPE OBJECT TABLE RESULT

Shape Object	XY-coordinates		
	Center	Upper left corner	Lower right corner
Activity node	X_{ca1}, Y_{ca1}	X_{au1}, Y_{au1}	X_{ad1}, Y_{ad1}
Initial node	X_{cs1}, Y_{cs1}	X_{su1}, Y_{su1}	X_{sd1}, Y_{sd1}
Final node	X_{ce1}, Y_{ce1}	X_{eu1}, Y_{eu1}	X_{ed1}, Y_{ed1}
Fork node	X_{cf1}, Y_{cf1}	X_{fu1}, Y_{fu1}	X_{fd1}, Y_{fd1}
Join node	X_{cj1}, Y_{cj1}	X_{ju1}, Y_{ju1}	X_{jd1}, Y_{jd1}

The final component to be extracted is lines. The Line detection module will be responsible for the recognition of arrow direction and then mark it as head. The information of line objects are stored in the table as shown in Table 3, consisting of the XY-coordinates of head and tail of each line detected.

TABLE III. LINE OBJECT TABLE RESULT

Line Object	XY-head	XY-tail
Line 1	X_{h1}, Y_{h1}	X_{t1}, Y_{t1}

All the data and information obtained from the previous three processes: OCR, Object detection, and Line detection, will be used for further processing of Match object composition module of which the detail is shown in Fig. 3.

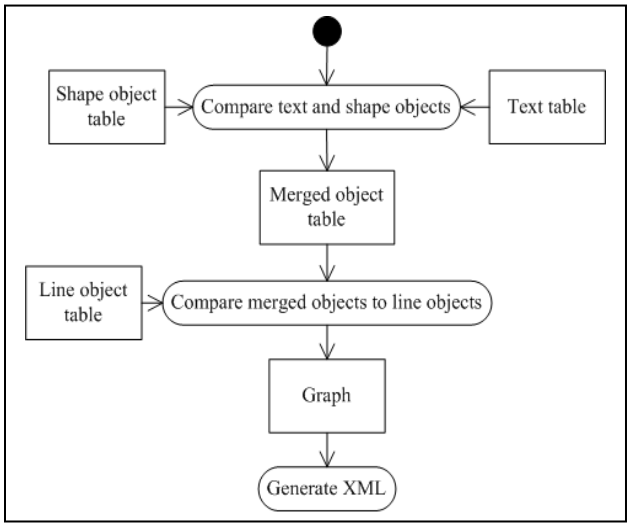


Figure 3. Match object composition procedure.

The procedure of matching composition starts from:

First, text and shape objects will be merged by comparing with each other to confirm that the text and shape objects are nearest. The result of merging shape objects with text contains information stored in the table as shown in Table 4, including the XY-coordinates of center, upper left corner, and lower right corner of each merged object. As the shape objects are the wrapper of text, i.e. their boundary is considered wider compared with the boundary of text, the XY-coordinates of upper left and lower right corners of shape objects are then used as boundary coordinates of merged objects.

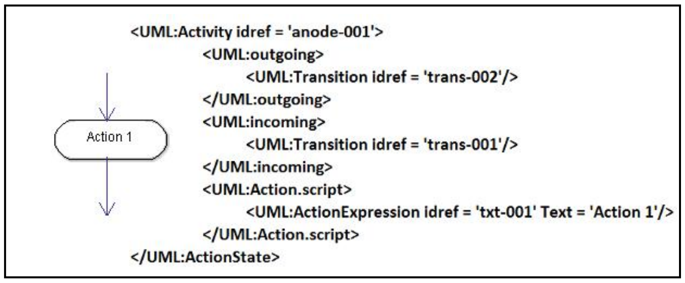
TABLE IV. MERGED OBJECT TABLE RESULT

Merged objects		XY-coordinates		
Text	Shape object	Center	Upper left corner	Lower right corner
Action 1	Activity node 1	X_{cm1}, Y_{cm1}	X_{mu1}, Y_{mu1}	X_{md1}, Y_{md1}

Next, merged objects will then be compared to line objects. The tail or head of the line object which is nearest to the merged object will be connected. Finally, a graph will be constructed by iteratively matching composition. In this work, the stack data structure is used to manage the nested controls.

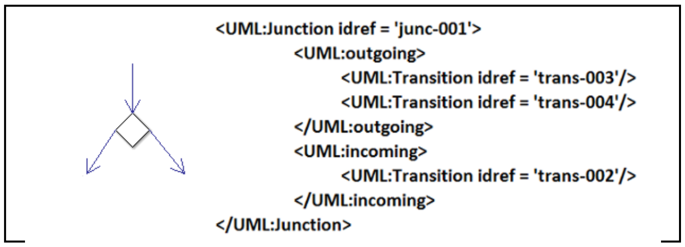
Once the graph has been constructed, the XML associated with each node can be generated. Fig. 4, Fig. 5, Fig. 6, Fig. 7, and Fig. 8 show the XML code associated with the Action node, Junction node, Fork node, Join node, Initial and Final nodes, respectively.

The final step is to generate the XMI file based on the XMI specification defined in [3]. The XMI generation module needs to arrange the XML data to comply with the correct syntax, as well as to edit some attributes, according to XMI specification.



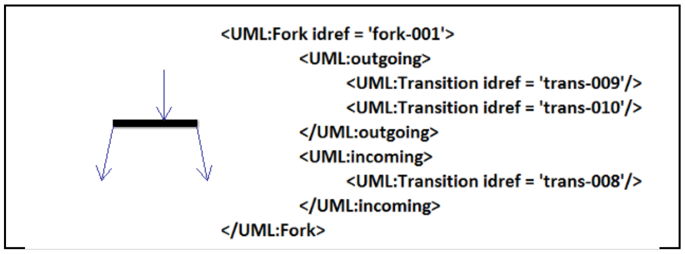
```
<UML:Activity idref = 'anode-001'>
        <UML:outgoing>
                <UML:Transition idref = 'trans-002'/>
        </UML:outgoing>
        <UML:incoming>
                <UML:Transition idref = 'trans-001'/>
        </UML:incoming>
        <UML:Action.script>
                <UML:ActionExpression idref = 'txt-001' Text = 'Action 1'/>
        </UML:Action.script>
</UML:ActionState>
```

Figure 4. Action node transformed to XML.



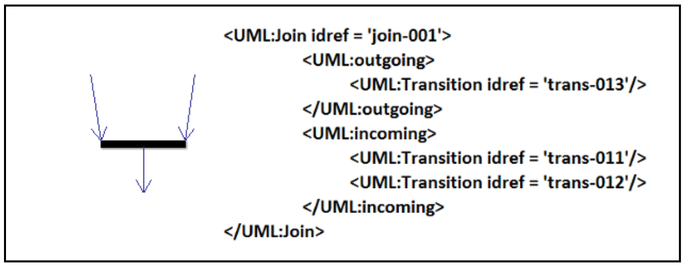
```
<UML:Junction idref = 'junc-001'>
        <UML:outgoing>
                <UML:Transition idref = 'trans-003'/>
                <UML:Transition idref = 'trans-004'/>
        </UML:outgoing>
        <UML:incoming>
                <UML:Transition idref = 'trans-002'/>
        </UML:incoming>
</UML:Junction>
```

Figure 5. Junction node transformed to XML.



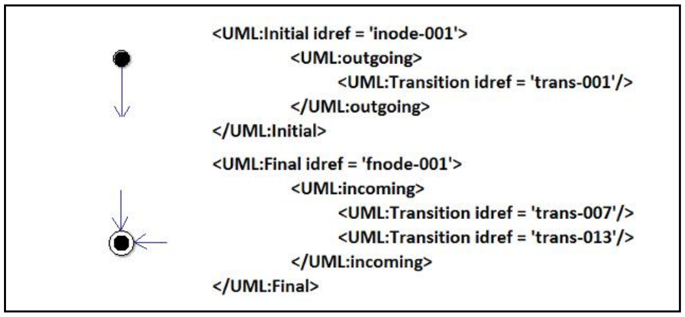
```
<UML:Fork idref = 'fork-001'>
        <UML:outgoing>
                <UML:Transition idref = 'trans-009'/>
                <UML:Transition idref = 'trans-010'/>
        </UML:outgoing>
        <UML:incoming>
                <UML:Transition idref = 'trans-008'/>
        </UML:incoming>
</UML:Fork>
```

Figure 6. Fork node transformed to XML.



```
<UML:Join idref = 'join-001'>
        <UML:outgoing>
                <UML:Transition idref = 'trans-013'/>
        </UML:outgoing>
        <UML:incoming>
                <UML:Transition idref = 'trans-011'/>
                <UML:Transition idref = 'trans-012'/>
        </UML:incoming>
</UML:Join>
```

Figure 7. Join node transformed to XML.



```
<UML:Initial idref = 'inode-001'>
        <UML:outgoing>
                <UML:Transition idref = 'trans-001'/>
        </UML:outgoing>
</UML:Initial>
<UML:Final idref = 'fnode-001'>
        <UML:incoming>
                <UML:Transition idref = 'trans-007'/>
                <UML:Transition idref = 'trans-013'/>
        </UML:incoming>
</UML:Final>
```

Figure 8. Initial and final nodes transformed to XML.

V. Conclusion

This research presents a method and implements the component called IMG2XMI to transform the image of UML activity diagrams to the XMI format. Currently, IMG2XMI supports the transformation of activity diagram image files created from ArgoUML and Modelio. Some limitation is incurred on the current stage of work such as the input could be only the scanned image or typewritten diagram images, non-handwritten. The correctness of IMG2XMI depends on the quality of scanned images. The enhancement of image processing to analyze image more intelligently would be carried out further.

References

[1] C. Narkngam and Y. Limpiyakorn, "Rendering UML Activity Diagrams as a Domain Specific Language – ADL," Proceedings of 24th International Conference on Software Engineering and Knowledge Engineering, San Francisco, USA, Jul 1-3, 2012, pp. 724—729.

[2] OMG Unified Modeling Language™ (OMG UML), Superstructure Ver. 2.4.1 (Aug 2011), http://www.omg.org/spec/UML/2.4.1/Superstructure.

[3] C. Kaewchinporn, and Y. Limpiyakorn, "Semantic Approach to Verifying Activity Diagrams with a Domain Specific Language," Proceedings of 2012 International Conference on Advanced Software Engineering & Its Applications, Jeju Island, Korea, Nov 28-30, 2012, pp. 466-473.

[4] T. Shima, K. Terasawa, and T. Kawashimao, "Image Processing for Historical Newspaper Archives," Proceedings of 2011 Workshop on Historical Document Imaging and Processing, Beijing, China, Sep 16-17, 2011, pp. 127-132.

[5] C. Patil, and V.Dalal, "Content Based Image Retrieval Using Combined Features," Proceedings of the International Conference & Workshop on Emerging Trends in Technology, Mumbai, India, Feb 25-26, 2011, pp. 102-105.

[6] Tigris, ArgoUML: UML diagramming application, argouml.tigris.or

[7] Sourceforge, EmguCV: Opencv in .NET framework, www.emgucv.com