FROM GRAPHICAL USER INTERFACE TO DOMAIN CLASS DIAGRAM: A REVERSE ENGINEERING APPROACH

[1]MOHAMMAD I. MUHAIRAT, [2]RAFA E. AL-QUTAISH, [1]BELKACEM M. ATHAMENA

[1]Al Zaytoonah University of Jordan, P.O. Box: 130, Amman 11733, Jordan
[2]Al Ain University of Science & Technology - Abu Dhabi Campus, P.O. Box: 112612, Abu Dhabi, UAE

E-mail: drmohairat@alzaytoonah.edu.jo, rafa@ieee.org, athamena@alzaytoonah.edu.jo

ABSTRACT

The Graphical User Interfaces (GUIs) of software products are extensively used by researchers and practitioners in Software Engineering field. For Example, they are used for testing, measuring usability, and many other purposes. This paper describes a new reverse engineering approach to transform the GUI into class diagram. However, the correctness of such transformation process is essential for the corrected execution of the overall software. To assure this correctness, the interpreted Petri nets models will be implemented on the proposed transformation processes (i.e. capturing, normalization, and translation processes).

Keywords: *Class Diagram, Graphical User Interface - GUI, Optical Character Recognition – OCR, Petri Nets – PNs, Reverse Engineering, Software Design, UML*

1. INTRODUCTION

Applying the reverse engineering approaches in software engineering is very important and useful. This importance and usefulness are due to the need to go backwardly in the development process to get missed documents especially for legacy software. Moreover, reverse engineering aims at extracting several types of information for existing software and to employ them for comprehension, reuse, or maintenance [1].

However, many research projects were conducted to apply the reverse engineering approaches to enable the reuse of existing process code [2], to combine metrics and program visualization [3], to recover design pattern information from source code [4], to perform testing on GUI [5], to enhance web applications [6, 7, 8], to maintain web sites [9], and to automate the construction of sequence diagrams for dynamic web applications [10].

In this paper the reverse engineering approach will be used to construct the class diagram from the Graphical User Interfaces (GUIs) of software product. However, the class diagrams are very important, necessary, and useful for the software development process. Consequently, many researchers have discussed the importance and usefulness of the class diagrams; for example, Agarwal and Sinha [11] find that only the class diagram and interaction diagram are significantly perceived as user-friendly, that is, the use of such tools is relatively easy, comfortable and clear. In addition, Te'eni *et al.* [12] affirm that fifty three percent of the projects uses class diagrams and fifty six percent represent business processes using one of the appropriate diagrams (e.g., activity, sequence and collaboration), whereas, all other diagrams are hardly and rarely used. Furthermore, the implementation of the class diagram is straightforward in most modern object-oriented programming languages, that is, each of the classes in a class diagram maps naturally into a programming language construct, for example, a Java class or interface [13].

The GUIs of software products are extensively used by researchers and practitioners in software engineering field. For Example, they are used for testing, measuring usability, etc.

This paper describes a new reverse engineering approach to transform the GUI into class diagrams. However, the correctness of such transformation process is essential for the correctness of the execution of the overall software. To assure this correctness, the Petri nets models will be applied on the transformations processes (i.e. capturing,

normalization, and translation processes). Therefore, the transformation processes will be implemented as flowcharts then they will be converted to Petri nets models.

Flowcharts are semi formal tools and thus the Petri nets modeling will be used as an effective graphical tool. Therefore, the flowcharts are not useful in the analysis of the processes. In this paper, we propose a domain of transformations, that is, to go from the semi-formal description using flowchart models to formal description using Petri nets models. In this way the capability to describe the processes behavior is fully exploited, since the processes analysis can be more properly performed in Petri nets domain. However, Petri nets models are widely used to represent and analyze industrial systems [19-21]. The reasons for using the Petri nets models are their formal semantics, graphical nature, expressiveness, and the availability of analysis techniques to prove their structural properties, such as, invariance properties, reachability, deadlock, liveness, etc.

The rest of this paper is organized as follows: Section 2 discusses the related concepts used in this paper, that is, the optical character recognition, class diagram, and Petri nets models. Section 3 explains - in details - the proposed approach, and Section 4 presents a case study on the proposed approach. Finally, Section 5 concludes the paper and presents potential future work.

2. RELATED CONCEPTS

This section introduces a set of related concepts, that is, the concepts which to be integrated to build the intended reverse engineering approach. However, the concepts of the Optical Character recognition (OCR), class diagrams, and Petri nets models will be discussed.

2.1 Optical Character Recognition

Optical Character Recognition (OCR) technology has become an aid for inputting documents quickly by and for users with vision impairments. A complete OCR system consists of a scanner, the recognition component, and OCR software that interact with the other components to store the computerized document in the computer. The process of inputting the material into the computer begins with the scanner taking a picture of the printed material. Then, during the recognition process, the picture is analyzed for layout, fonts, text and graphics. Finally, the picture of the document is converted into an electronic format that can be edited with application software. OCR systems designed specifically for users with visual impairments have modified interfaces that can be used with minimal assistance [14].

In general, OCR systems work as an external device with the user's existing assistive technology. Once the picture is in electronic format, it is accessed for reading and/or editing through the user's Braille, speech or magnification technology. Since some products work better with certain speech or Braille systems, therefore, it is important to note its compatibility with the other products in the user's computer system. Some products, however, have an adaptive device built in. These are referred to as ‘stand-alone reading machines’. Of these, some products have the added flexibility of working either as stand-alone or with a computer [15].

In our proposed approach, the OCR is used to capture the words and/or phrases which in turn represent buttons, text boxes, labels, and any component of a form or frame.

2.2 Class Diagrams

In software engineering, a class diagram as a type of the Unified Modeling Language (UML) is a static structure diagram that illustrates the structure of software by showing the software’s classes, their attributes, and the relationships between the classes [16].

Class diagrams are commonly used to describe the types of objects and their relationships in software. The class diagrams are used to model a class structure and contents using design elements such as classes, packages and objects. In addition, they describe three different perspectives when designing a software product, that is, conceptual, specification, and implementation. These perspectives become evident as the diagram is created and help solidify the design [17]. However, Figure 1 illustrates that the classes are composed of three items, that is, class name, class attributes, and class operations.

Student
name: string age: integer
getName(): string

Figure 1: The Contents of the Class

A class diagram is similar to a family tree in which a class diagram consists of a group of classes and interfaces reflecting important entities of the business domain of the software being modeled and the relationships between these classes. The classes in the diagram represent the members of a family tree and the relationships between the classes are equivalent to relationships between members in a family tree. Interestingly, classes in a class diagram are interconnected in a hierarchical fashion, like a set of parent classes and related child classes under the parent classes. Similarly, a software application is comprised of classes and a diagram depicting the relationship between each of these classes, that is, the class diagram [18].

2.3 Petri Nets Models: An Overview

Petri nets (PNs) model is a graphical-mathematical tool used to represent and analyze various systems for describing the relations between conditions and events, especially for systems with parallel and concurrent activities [21]. However, PNs are currently used in many industrial branches for planning and controlling of production flow, system/software synthesis, etc. The graphic representation of PNs can be understood even for non-technical staff. It allows – for example – to specify such behaviors as parallelism and concurrency, choice, synchronization, memorizing, reading or resources sharing.

The advantage of obtaining a formal PNs model for the software resides in the possibility of use of standard tools for the analysis of PNs, thus, PNs properties can be quickly verified.

A graphical PNs model consists of circles, bars, directed arcs and dots, which represent places, transitions, arcs and tokens, respectively. Besides, PNs model provides qualitative analysis for system properties such as reachability, liveness, boundedness, safeness, conservativeness, and deadlock [19-21].

A flowchart is a common graphic formalism, often used to represent the control structure of programs or workflow systems. It is possible to conveniently map flowcharts into PNs, in which each vertex of the flowchart is replaced by a corresponding PNs fragment [22].

3. THE PROPOSED APPROACH

The proposed reverse engineering approach for constructing a class diagram from GUI will consist of the following processes:

1. Capturing process: it will give us the ability to capture all the GUI forms components and store them in a not-normalized temporary table-.
2. Normalization process: it consists of scanning all records in the temporary table and normalizing it. The result will be new normalized table.
3. Translation process: it will translate the normalized table to class diagram.

Figure 2 shows the above three processes and the inputs and outputs for each process. For example, the input for the normalization process is temporary table and its output is normalized table.

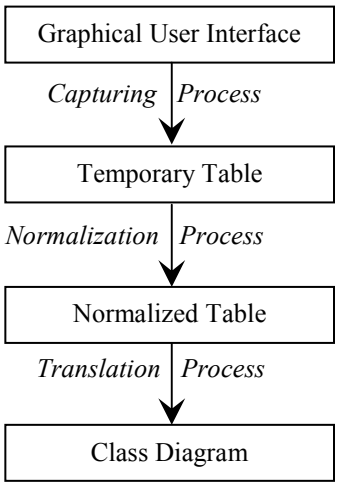

Figure 2: The Entire Transformation Process (Reverse Engineering Process)

As seen above, the capturing process is the first process and it will identify each component and store its information in special table called temporary table. This process consists of the following two sub processes:

1. The capturing sub process; and
2. The storing sub process.

However, Figure 3 demonstrates the algorithm of the capturing and storing sub processes.

As illustrated in Figure 3, our algorithm scans about operation name in operation table. The operation table will consist of all common possible operation names that can be compared with the captured buttons names from GUI. This table can be modified as needed.

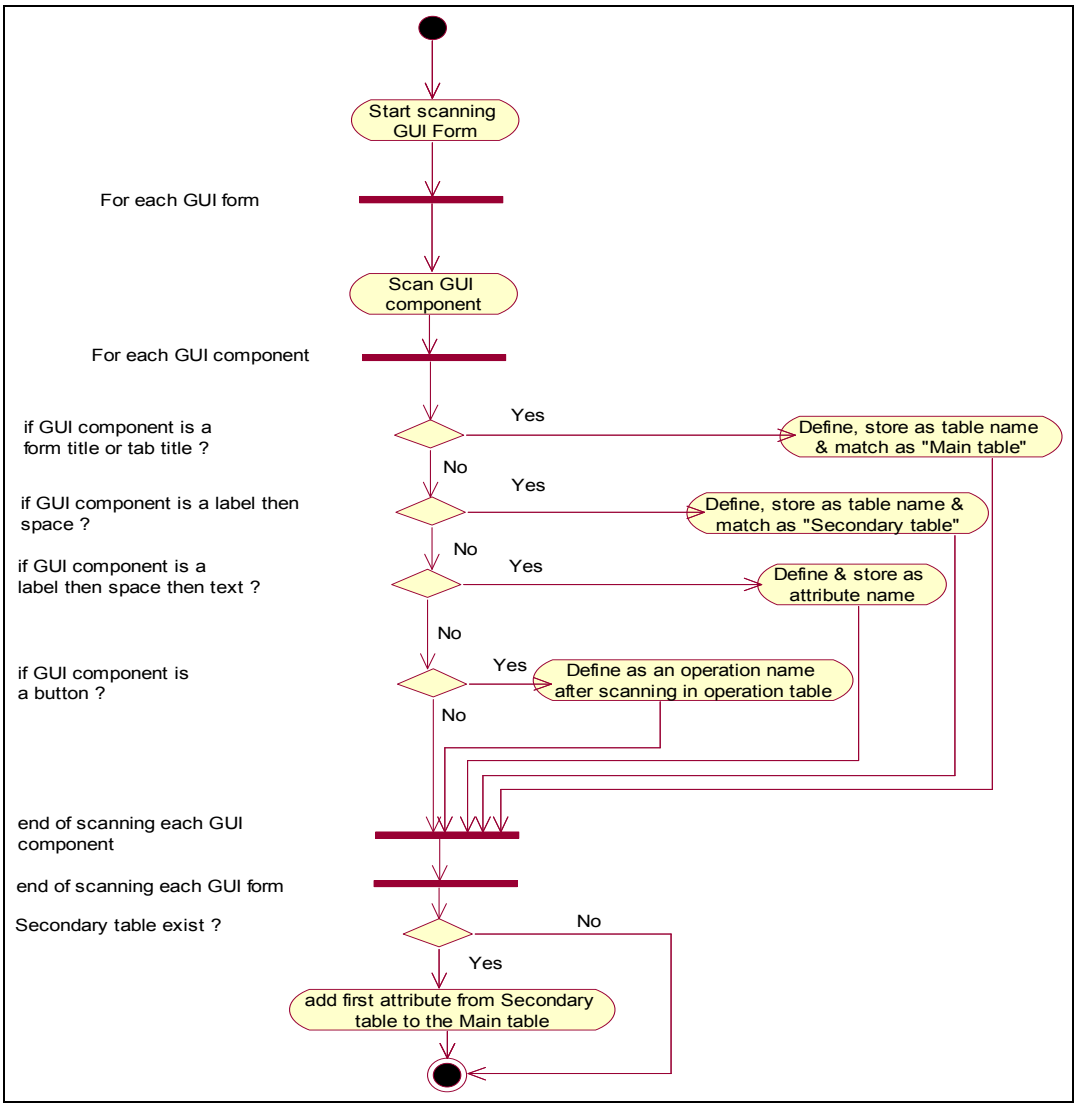

Figure 3: Capturing Process

Table 1: Operations Table

Button name	***Operation Name***
Save, Add, Submit, New, Insert, Create	Insert
Delete, Cancel, Erase	Delete
Update, Change, Modify	Update
Search, Find, Explore, Navigate, Select, Read, print	Select

After executing the capturing sub process the result will be stored on a temporary table which may contain redundancy data, such as names of records, fields and operations. However, Table 2 shows the contents of this table. Then, the explanations of the contents of this table will be discussed to ensure the understandability of the table.

Table 2: Temporary Table

TN_i	FN_i	FN_{i+1}	. . .	FN_m	ON_k	ON_{k+1}	. . .	ON_n
TN_{i+1}								
⋮								
TN_l								

The symbols are used in Table 2 and defined as the following:

TN: Table Name,
FN: Field Name,
ON: Operation Name,
i: Field index,
j: Table name index,
k: Operation Index,
l: Maximum number of table names,
m: Maximum Number of Fields, and,
n: Maximum Number of Operations,

Now, to verify capturing process, the algorithm for the capturing process in Figure 3 will be transformed to its equivalent PNs model. According to the standard transformation rules, the equivalent PNs model of the capturing process is represented in Figure 4. Then, the reachability graph (RG) of capturing process is shown in Figure 5. However, from this reachability graph - Figure 5 - and from the PNs properties [21], we can conclude that the capturing process is bounded, safe, live, reachable and without deadlock.

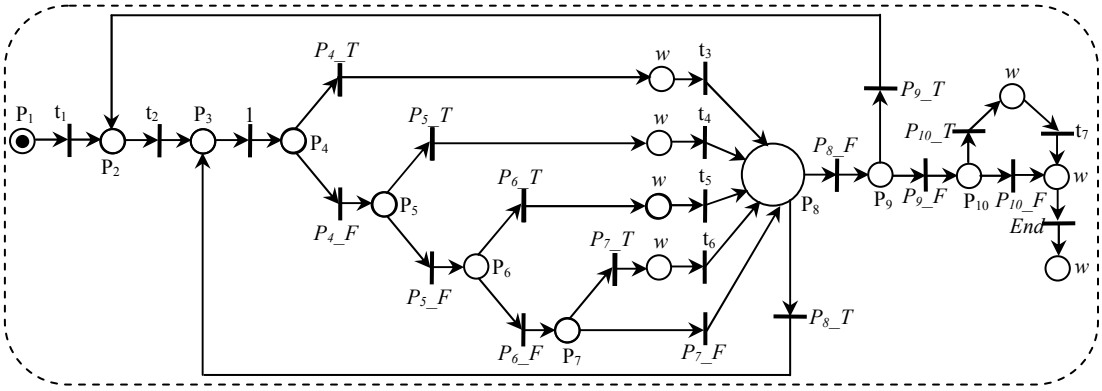

Figure 4: PNs Model of the Capturing Process

Whereas, the symbols in Figure 4 are defined as the following:

P_1 : *Ready to Start*,
$P_2 : (GUI\_F \leftarrow)$,
$P_3 : (GUI\_C \leftarrow)$,
$P_4 : [GUI\_C \wedge (FT \vee TL)]$,
$P_5 : (GUI\_C \wedge L) \rightarrow S$,
$P_6 : [(GUI\_C \wedge L) \rightarrow S] \rightarrow TX$,
$P_7 : (GUI\_C \wedge B)$,
$P_8 : GUI\_C++$,
$P_9 : GUI\_F++$,
$P_{10} : (ST \wedge EX)$,
$t_1 : (SN \wedge GUI\_F)$,
$t_2 : (SN \wedge GUI\_C)$,
$t_3 : [(DC \wedge SC) \rightarrow TN] \rightarrow MT$,
$t_4 : [(DC \wedge SC) \rightarrow TN] \rightarrow ST$,
$t_5 : (DC \wedge SC) \rightarrow AN$,
$t_6 : (SC \wedge OT) \rightarrow ON$,
$t_7 : (FA \wedge ST) \rightarrow MT$,
GUI_F: GUI Form,
GUI_C: GUI Component,
FA: First Attribute,
FT: Form Title,
MT: Main Table,
OT: Operation Table,
ST: Secondary Table,
TL: Tab Title,
B: Button,
DC: Define Component,
EX: Exist,
L: Label,
S: Space,
SC: Store Component,
SN: Scan,
TX: Text, and
AN: Attribute Name.

The second process, that is, the normalization process which starts after finishing the capturing process. This process will be executed using the algorithm in Figure 6.

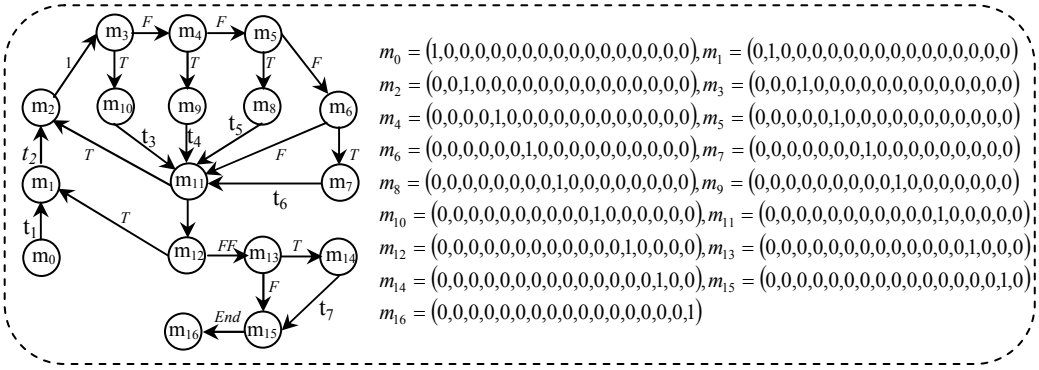

Figure 5: RG of the Capturing Process

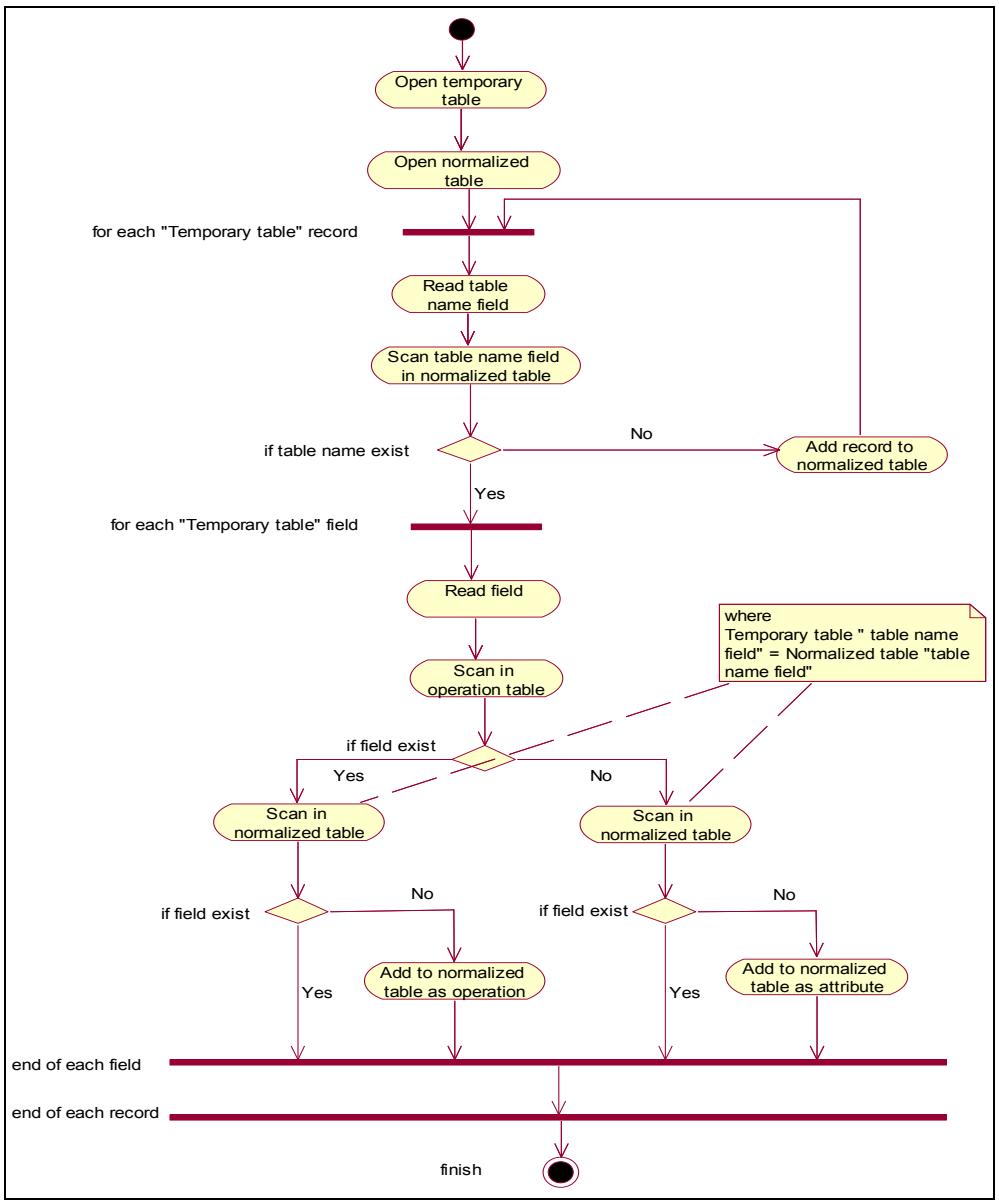

Figure 6: Normalization Process Flowchart

Again, the algorithm of the normalization process needs to be transformed into its equivalent PNs model. Therefore, Figure 7 shows the equivalent PNs model of the normalization process algorithm. Furthermore, based on the PNs properties, Figure 8 illustrates that the PNs model of the normalization process algorithm is bounded, safe, live, reachable, and without deadlocks,.

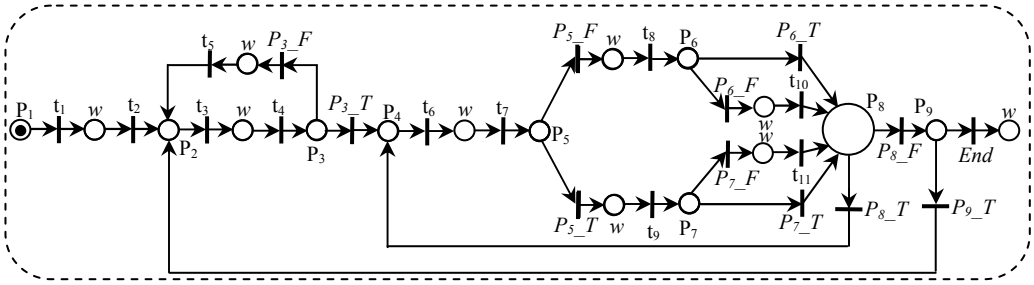

Figure 7: PNs Model of the Normalized Process

However, the following are the definitions of the symbols used in Figure 7:

P_1 : *Ready* to start,
$P_2 : (TT\_RC \leftarrow)$,
$P_3 : (TN \wedge EX)$,
$P_4 : (TT\_FD \leftarrow)$,
$P_5 : (FD \wedge EX)$,
$P_6 : (FD \wedge EX)$,
$P_7 : (FD \wedge EX)$,
$P_8 : (TT\_FD++)$,
$P_9 : (TT\_RC++)$,
$t_1 : (O \wedge TT)$,
$t_2 : (O \wedge NT)$,
$t_3 : (R \wedge TN\_FD)$,
$t_4 : (SN \wedge TN\_FD \wedge NT)$,
$t_5 : RC \rightarrow NT$,
$t_6 : (R \wedge FD)$,
$t_7 : (SN \wedge OT)$,
$t_8 : (SN \wedge NT)$,
$t_9 : (SN \wedge NT)$,
$t_{10} : AT \rightarrow NT$,
$t_{11} : OP \rightarrow NT$,
TT_Rc: Temporary Table Record,
TT_FD: Temporary Table Field,
FD: Field,
O: Open,
NT: Normalized Table,
R: Read,
RC: Record,
TN_FD: Table Name Field,
OP: Operation, and
AT: Attribute.

As a result of execution normalization process, a new normalized table will be build without any redundancy in tables' names, fields' names and operations' names.

The third process (translation process) is divided into three sub processes, that is, the relationship definition, translation to class diagram, and translation to association sub processes. Firstly, Figure 9 represents an algorithm for the process of defining the relationship between records from normalized table and as a result between classes.

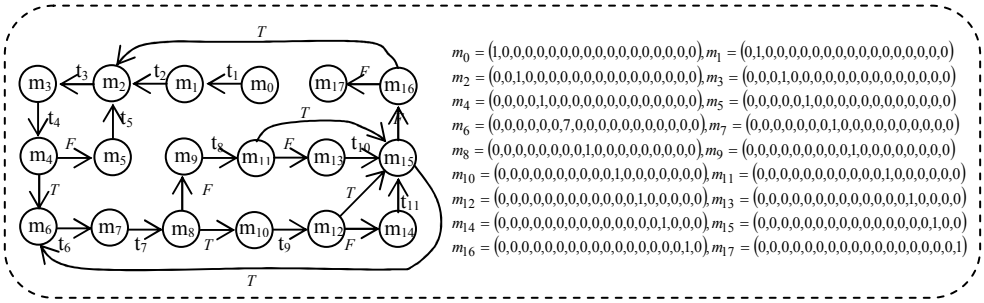

Figure 8: RG of the Normalized Process

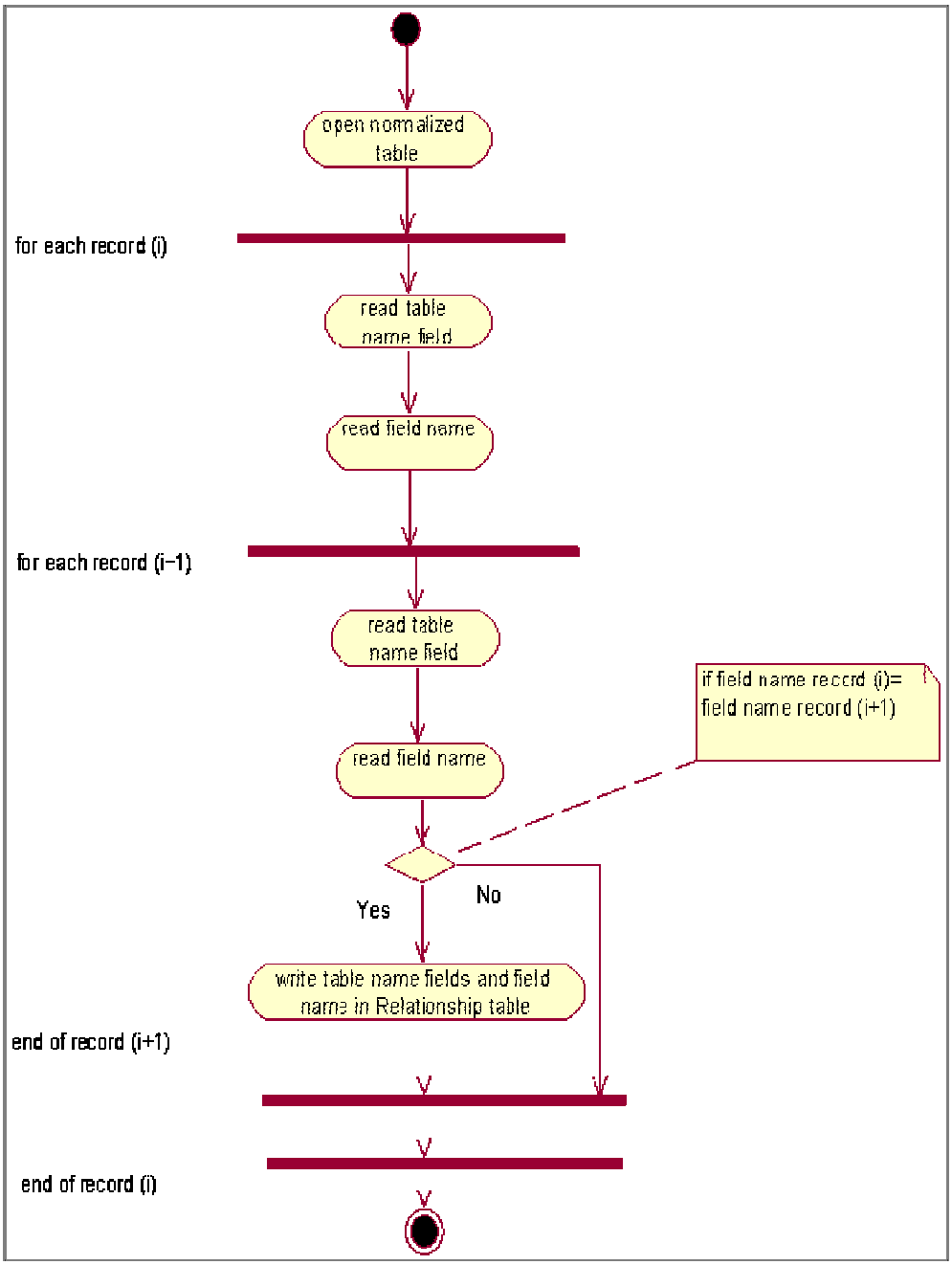

Figure 9: Relationship Definition Flowchart

After executing the algorithm in Figure 9, the result will be stored in a relationship table, as in Table 3.

Table 3: Relationship table

TN	*TN*	*RF*

In Table 3, *TN* represents a Table Name and *RF* represents a Relationship Field.).

Figures 10 and 11 represent the equivalent PNs model and the reachabilty graph of the relationship definition sub process, respectively, From these figures (Figures 10 and 11), we can conclude that this sub process is also bounded, safe, live, reachable, and without deadlock.

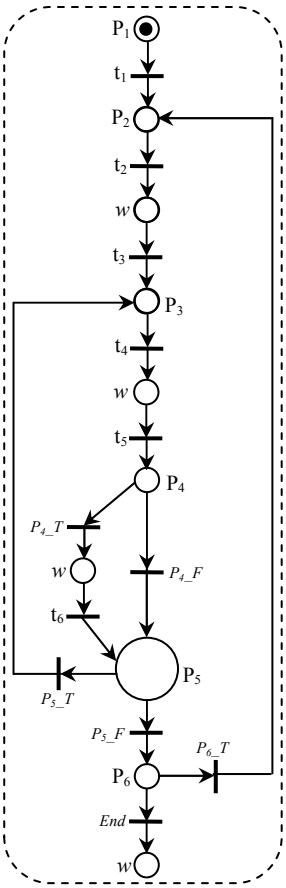

Figure 10: PNs Model of the Relationship Process

The following presents the symbols' definitions for Figure 10:

P_1 : *Ready* to start,
$P_2 : (RC(i) \leftarrow)$,
$P_3 : (RC(i+1) \leftarrow)$,
$P_4 : FN\_RC(i) = FN\_RC(i+1)$,
$P_5 : (RC(i+1)++)$,
$P_6 : (RC(i)++)$,
$t_1 : (O \wedge NT)$,
$t_2 : (R \wedge TN\_FD)$,
$t_3 = (R \wedge FN)$,
$t_4 : (R \wedge TN\_FD)$,
$t_5 : (R \wedge FN)$,
$t_6 : (W \wedge TN\_FD \wedge FN) \rightarrow RT$,
FN_RC: Field Name Record,
W: Write, and,
RT: Relationship Table.

Secondly, the algorithm of the translation to class diagram sub process is shown in Figure 12. Its equivalent PNs model and the reachability graph are illustrated in Figures 13 and 14, respectively.

Finally, the algorithm of the translation to association sub processes is presented in Figure 15, its PNs in Figure 16, and the reachability graph of the PNs in Figure 17. However, the outputs of Figures 12 and 15 will be saved in a meta-data table, from which we will generate the class diagram.

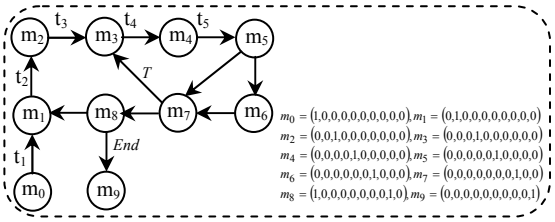

Figure 11: RG of the Relationship Process

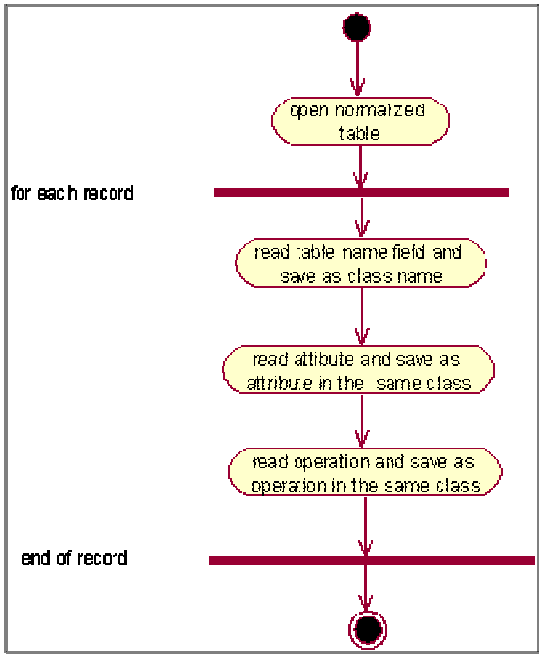

Figure 12: PNs Model of the Translation to Class Diagram Process

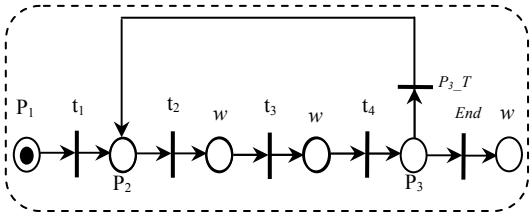

Figure 13: Translation to Class Diagram Process Flowchart

For Figure 13, the following are the definitions of the used symbols:

P_1 : *Ready to start*,

$P_2 : (RC \leftarrow)$,

$P_3 : (RC++)$,

$t_1 : (O \wedge NT)$,

$t_2 : (R \wedge TN\_FD) \rightarrow (SV \wedge CN)$,

$t_3 : (R \wedge AT) \rightarrow (SV \wedge AT \wedge CN)$,

$t_4 : RO \rightarrow (SV \wedge OP \wedge CN)$,

CN: Class Name, and

RO: Read Operation.

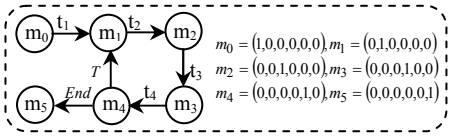

Figure 14: RG of the Translation to Class Diagram Process

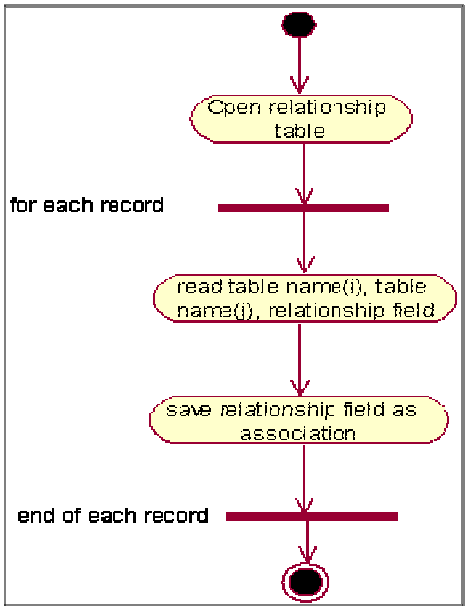

Figure 15: Translation to Associations Process Flowchart

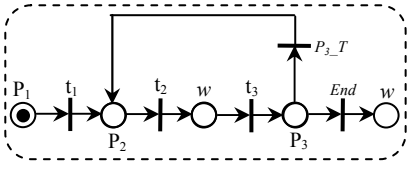

Figure 16: PNs Model of the Translation to Association Process

Figure 16 contains a set of symbols which are defined as the following:

P_1 : *Ready to start,*

$P_2 : (RC \leftarrow)$,

$P_3 : (RC++)$,

$t_1 : (O \wedge RT)$,

$t_2 : (R \wedge TN(i) \wedge TN(j) \wedge RF)$,

$t_3 : (SV \wedge RF) \rightarrow AS$,

RT: Relationship Table, and

AS: Association.

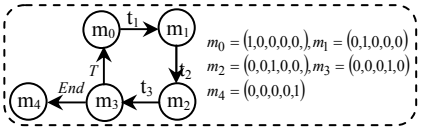

Figure 17: RG of the Translation to Association Process

4. CASE STUDY: ORDERING SYSTEM

An example represented here – the ordering system – is to describe the process of ordering and maintaining information about customers and items. To execute each process, the users have to interact with specific forms; these forms are represented in figures 18, 19 and 20. However, these forms will be used as input to the capturing process of our proposed approach; see Figure 3 for more details about the capturing process.

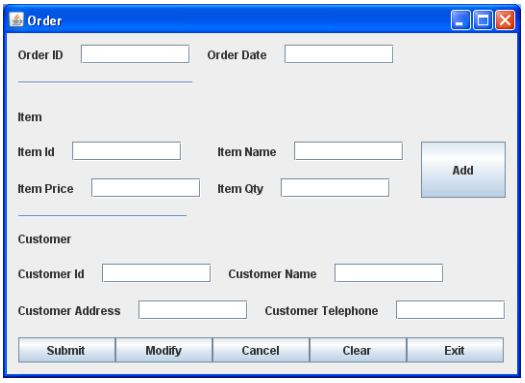

Figure 18: Order frame

Figure 19: Item frame

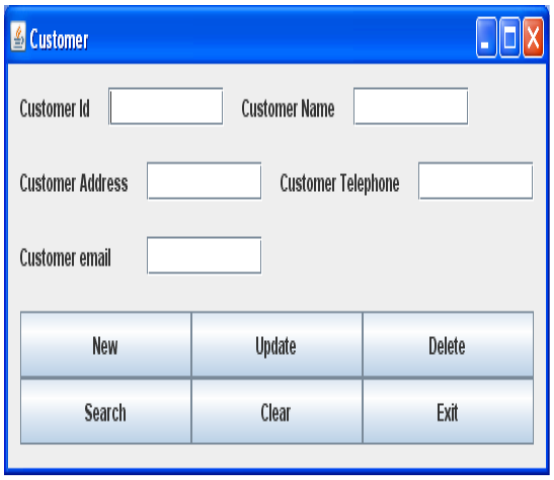

Figure 20: Customer frame

After applying the capturing process algorithm, as in Figure 3 above, the result will be stored in a temporary table, as in Table 4.

As seen in Table 4, the result in the temporary table is not normalized and contains *Order*, *Item*, and *Customer* as Tables' Names (TN), also it contains *ordereID* and *orderDate* as Fields' Name (FN), and *insert* and *update* as Operations' Names (ON), see table 2 above for the places and definitions of the TN, FN, and ON.

Next, Table 5 shows the results after executing the normalization process (as in Figure 6), that is, excluding the data redundancy in Table 4, the result will be stored in a table named normalized table, that is, without any kind of data redundancy.

Table 4: Temporary Table

Order	ordered	orderDate	insert	Update	Delete	itemId	customerId
Item	itemId		itemName	itemPrice		itemQty	Insert
Customer	customerId		customerName	customerAddress		customerTelephone	
Item	item Id	item Name	item Price	item Qty	Item ExpiredDate	item QtyOnHand	Insert	update	delete	select
Customer	customer Id	customer Name	customer Address	Customer Telephone		customer Email	insert	update	delete	select

Table 5: Normalized Table

Order	ordered	orderDate	Insert	update	Delete	itemId	customerId
Item	item Id	item Name	item Price	item Qty	item ExpiredDate	item QtyOnHand	insert	update	delete	select
Customer	customer Id	customer Name	customer Address	customer Telephone		customer Email	insert	update	delete	select

By applying the relationship definition process algorithm – as in Figure 9 – we will determine if there is any relationship between the records from the normalized table. The result of the relationship definition process is stored in Table 6 below. Therefore, we find that there are relationships between the *Order* and *Item* tables, and between the *Order* and *Customer* tables.

Table 6: Relationship Table

Order	Item	itemId
Order	Customer	customerId

Finally, in order to produce the class diagram and draw the relationships between these classes, the translation process to class diagram algorithm and the translation process to association algorithm are applied. Figure 21 presents the produced class diagram based on the GUIs of the ordering system, that is, depends on Figures 18, 19, and 20.

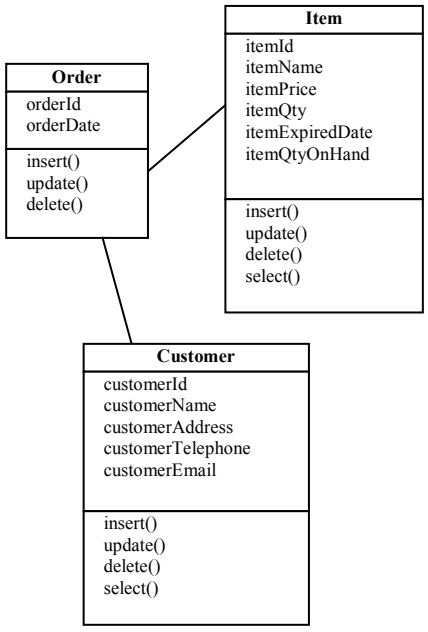

Figure 21: The Produced Class Diagram

5. CONCLUSION AND FUTURE WORK

The Graphical User Interfaces (GUIs) of software products are extensively used by researchers and practitioners in Software Engineering field. For Example, they are used for testing, measuring usability, etc. This paper described a new reverse engineering approach to transform the GUI into class diagrams. This reverse engineering approach will help the practitioners to produce the class diagram backwardly from the GUI to save time to work with the legacy software products.

However, the correctness of such transformation process is essential for the correct execution of the overall software. To assure this correctness, the Petri nets models were implemented on the transformations processes (i.e. capturing, normalization, and translation processes). By implementing the PNs we have assured that the three processes, that is, capturing, normalization, and translation processes are safe and live.

As a future work, the proposed approach in this paper could be used to define all types of relationship between classes, multiplicity and to construct another UML diagrams, such as use case diagram, from the GUI of any software product.

REFRENCES:

[1].H. El Bouhissi, M. Malki, and D. Bouchiha, A Reverse Engineering Approach for the Web Service Modeling Ontology Specifications, in *Proceedings of the 2nd International Conference on Sensor Technologies and Applications (SENSORCOMM'08)*, Cap Esterel, France, August 25-31, 2008, pp. 819-823.

[2].H. Tran, U. Zdun, and S. Dustdar, View-Based Reverse Engineering Approach for Enhancing Model Interoperability and Reusability in Process-Driven SOAs, in *Proceedings of the International Conference on Software Reuse (ICSR'08)*, Beijing, China, May 25-29, 2008, pp.233-244

[3].S. Demeyer, S. Ducasse, and M. Lanza, A Hybrid Reverse Engineering Approach Combining Metrics and Program Visualization, in Proceedings of the 6th Working Conference on Reverse Engineering (WCRE'99), Atlanta, GA , USA, October 6-8, 1999, pp. 175-186.

[4].A. Alnusair, and T. Zhao, Towards a Model-Driven Approach for Reverse Engineering Design Patterns, *the 2nd Workshop on Transforming and Weaving Ontologies and MDE (TWOMDE'09)*, Denver, Colorado, USA, October 4-9, 2009.

[5].A. Memon, I. Banerjee, and A. Nagarajan, GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing, in *Proceedings of the 10th Working Conference on Reverse Engineering (WCRE'03)*, Victoria, BC, Canada, November 13-16, 2003, pp. 260-269.

[6].S. Weijun, L. Shixian, and L. Xianming, An Approach for Reverse Engineering of Web Applications, in *Proceedings of the International Symposium on Information Science and Engineering (ISISE'08)*, Shanghai, China, December 20-22, 2008, pp. 98-102.

[7].G. A. Di Lucca, A. R. Fasolino, F. Pace, P. Tramontana, and U. De Carlini, WARE: a Tool for the Reverse Engineering of Web Aapplications, in *Proceedings of the 1st International Conference on Web Information Systems Engineering (ICWEISE'00)*, Hong Kong, China, June 19-21, 2000, pp. 241-250.

[8].J. Pu, H. Yang, B. Xu, L. Xu, and W. C. Chu, Combining MDE and UML to Reverse Engineer Web-Based Legacy Systems, in *Proceedings of the 32nd Annual IEEE International Computer on Software and Applications (COMPSAC'08),* Turku, Finland, July 28 - August 1, 2008, pp. 718-725.

[9].S. Chung and Y. Lee, Reverse Software Engineering with UML for Web Site Maintenance, in *Proceedings of the 1st International Conference on Web Information Systems Engineering (ICWEISE'00)*, Hong Kong, China, June 19-21, 2000, pp. 157-161.

[10]. M. H. Alalfi, J. R. Cordy, and T. R. Dean, Automated Reverse Engineering of UML Sequence Diagrams for Dynamic Web Applications, in *Proceedings of the International Conference on Software Testing, Verification and Validation (ICSTW'09)*, Denver, CO, USA, April 1-4, 2009, pp. 287-294.

[11]. R. Agarwal, and A. P. Sinha, "Object-Oriented Modeling with UML: A Study of Developers' Perceptions", *Communication of the ACM*, Vol.46, No.9, 2003, pp.248-256.

[12]. D. Te'eni, R. Gelbard, and M. Sade, Increasing the Benefit of Analysis: The Case of Systems that Support Communication, in *Proceedings of the 11th International Conference of the Association Information and Management (AIM'06)*, June 8-9, 2006, pp. 13-27.

[13]. J. M. P. Cachopo, *Development of Rich Domain Models with Atomic Actions*, PhD Thesis, Universidade Técnica de Lisboa, Lisboa, Portugal, 2007.

[14]. H. Bunke, and P. S. P. Wang, *Handbook of Character Recognition and Document Image Analysis*, World Scientific Publishing Company, Hackensack, NJ, USA, 1997.

[15]. B. Kumar, *Optical Pattern Recognition*, Prentice-Hall, USA, 2003.

[16]. Russell Miles, Kim Hamilton, *Learning UML 2.0*, 1st edition, O'Reilly Media, USA, 2006.

[17]. G. Booch, R. A. Maksimchuk, M. W. Engel, B. J. Young, J. Conallen, K. A. Houston, *Object-Oriented Analysis and Design with Applications*, 3rd edition, Addison-Wesley Professional, London, UK, 2007.

[18]. M. Chitnis, P. Tiwari, and L. Ananthamurthy, *The UML Class Diagram: Part 1*, online: http://www.developer.com/article.php/220679, visited on Dec. 5, 2010.

[19]. R. David, and H. Alla, *Discrete, Continuous, and Hybrid Petri nets*, Springer-Verlag, Berlin Heidelberg, Germany, 2005.

[20]. T. Murata, *Petri nets: Properties, Analysis and Applications*, Proceedings of the IEEE, Vol. 77, No. 4, 1989, pp. 541-580.

[21]. R. David and H. Alla, *Petri nets for modeling of dynamics systems-A survey*, Automatica, vol. 30, no. 2, pp. 175–202, 1994.

[22]. S. Achasova, O. Bandman, and V. Markova, *Parallel Substitution Algorithm, Theory and Application*, World Scientific, USA, 1994.

AUTHOR PROFILES:

Dr. Mohammed I. Muhairat received the M.Sc. degree in Computer Engineering from Kharkov State Technical University of Radio Electronics, Ukraine in 1997, and the Ph.D. degree in computer engineering from Kharkov National University of Radio Electronics, Ukraine in 2002. Currently, he is the Department Chair and Assistant Professor of Software Engineering in Department of Software Engineering, Al Zaytoonah University of Jordan. His research interests are in Software Engineering field, such as, Requirements Specification, Software Architecture, Software Development Process, Reverse Engineering, and Formal Methods. He has more than 20 published articles in International Journals and Conferences.

Dr. Rafa E. Al-Qutaish received the B.Sc. in Computer Science and M.Sc. degrees in Software Engineering in 1993 and 1998, respectively. Also, he received the Ph.D. degree in Software Engineering from the School of Higher Technology (ÉTS), University of Québec, Canada in 2007. Currently, he is an Assistant Professor of Software Engineering at Al Ain University of Science and Technology in Abu Dhabi, UAE. His current research interests are in Software Measurement, Software Product Quality, Software Engineering Standardization, Reverse Engineering, Software Comprehension and Maintenance, and Compiler Construction. So far, he has more than 34 published articles in International Journals and Conferences. Dr. Al-Qutaish is a senior member of the IEEE & IEEE-CS, and also a senior member of the IACSIT in Singapore.

Dr. Belkacem M. Athamena was born in Algeria. He received the M.Sc. and Ph.D. degrees in Computer Engineering and System/Software Modeling & Analysis from Annaba University in collaboration with UCL University, Belgium, in 1994 and 2004, respectively. He is currently an Associate Professor at the Department of Software Engineering, Al Zaytoonah University of Jordan. His research interests include System/Software Modeling and Analysis, Fuzzy Logic, Neural Networks, Petri Nets, UML, VVT, Formal Methods, Fault Diagnosis. He has published over 40 papers, chapters in books, and conferences.