Generating a Language-Independent Graphical User Interfaces from UML Models

Amany Shatnawi[1] and Raed Shatnawi[2]
[1]Department of Computer Science, Jordan University of science and Technology, Jordan
[2]Department of Software Engineering, Jordan University of science and Technology, Jordan

Abstract: *The cost of the software development is high and there is a need to automate parts or all activities of the software development to reduce the development costs. In this work, the User Interface (UI) design is automated and UIs are generated for language-independent code from Unified Modeling Language (UML) diagrams. These diagrams are used to generate both the content of the UIs and the navigation through the use interfaces. Based on end-user feedback, the UML diagrams and the UI prototype can be iteratively refined. To demonstrate this work, a tool that automates the generation of UI prototype is built. The tool generates a prototype that is coded using an eXtensable Markup Language (XML) called the UI Markup Language (UIML). The proposed approach is validated and UIs are generated for two case studies.*

Keywords: *UI prototyping, UIML, UML, language-independent UI.*

Received September 20, 2013; accepted May 6, 2014; published online June 11, 2015

1. Introduction

Software systems are becoming larger and more complex and the cost of software development is a success factor. There is a need to have a set of Computer Aided Software Engineering (CASE) tools that automate software development and reduce the software development cost. Various tools already exist to automate software development activities such as software design, automatic code generation and testing automation. Software engineers use iterative methods to build quick prototypes to represent the stakeholder needs. They usually build prototypes to gather the feedback from customers and end-users at early stages of the software development cycle. A prototype is an initial version of a software system that can be used to help find more about the software requirements. In addition, creating a prototype helps developers to control the software development costs and allows the stakeholders to experiment the software in early stages of the development process [23]. The software prototype can serve many purposes. In the requirement engineering process, prototypes can help the developers in the elicitation and the validation of the system requirements, allow users to see how a system supports their work and propose new or modify system requirements. Furthermore, as the prototype is developed, it may reveal errors and omissions in the proposed requirements. A functional description of the software specifications may seem well-defined and useful. However, when specifications are reviewed, the users may find incorrect and incomplete requirements. In the system design process, developers build a prototype to check the feasibility of a proposed design and to support User Interface (UI) design. Prototyping can also, serve as an early input to produce the user documentations and to train end users even before the release of the software. In human-computer interaction, prototypes are built to provide early hands-on experience to users and to assess various usage environments [24].

Several papers have discussed different methods for generating UI prototypes from Unified Modeling Language (UML) diagrams [4, 5, 10, 11, 12]. However, in these studies the UIs were generated for a particular language (e.g., Java, eXtensable Markup Language (XML), HTML and C++) and a particular platform (e.g., Windows or Linux). Each language has its own syntax, abstractions and applications. For example, HTML is used to develop web pages (document style), while JavaScript is used to handle user events. These languages are not necessarily platform-independent. Using a specific language to implement UIs may cause many problems with peripherals of different screen sizes such as handheld devices [2]. Therefore, software developers need to design and build a completely separate UI prototype for each platform. The aim of this research is to propose a new approach to generate a language-independent UI from UML diagrams. We propose to use UML because it is easier to understand and communicate using graphical notations [8, 22]. We aim to build a tool that implements the underlying approach and then exercise it on real software models. The tool generates the UI prototype in an XML-based language called the UI Markup Language (UIML). The research objectives are summarized as follows: Identify the necessary UML diagrams that could be used in the process to develop Graphical User Interfaces (GUI) without the need to add any extended diagrams; build a tool (UI-gen), as a result of the proposed approach, that automates the generation of UIs. The tool produces UIs represented in an XML-based language called UIML, which provides a general-purpose presentation of UIs that is implementation-independent from operating systems and platforms [1, 19] and demonstrate the feasibility of our approach by running the generated UI prototypes for some target platforms. In the rest of this paper, we discuss the related work in section 2, we provide the details of the research approach in section 3 and we discuss the results and the case studies in section 4 and finally we conclude our work in section 5.

2. Related Works

Different methods have been suggested for deriving the UI from the specification of the application domain. Nichols has suggested a new system (personal universal controller) for automatic generation of two types of UIs, graphical and speech interfaces for Personal Digital Assistants (PDAs) or mobile phones using the abstract specification [19, 20]. Almendros and Iribarne [5] have proposed a new method for mapping use case models into graphical UI design and they used the use case model to identify the requirements of the system and used the activity diagrams to describe use cases. In another study, Almendros and Iribarne [4] described how they modeled UIs by using specialized UML diagrams. They defined a new kind of UML diagrams for modeling UIs based on use case diagrams. Elkoutbi *et al*. [11] have suggested a new requirement engineering process to generate a UI prototype from the scenarios and formal specifications of an application under development. In another study, Elkoutbi and Keller [10] suggested another process that generates a UI prototype from scenarios and yields a formal specification of the system as high-level Petri nets. Xia and Zhang [25] introduced the generation of UIs only from use cases that combine both the concept of modeling using a use case model and the concept of the data flow diagram. The previous approaches have many limitations. First, some approaches assume changes in UML language which is a standard language for modeling software systems and developers may not be aware of these changes. Second, the portability of the generated UIs is very weak; they assume a language dependent UIs on specific platforms which limits the work to few languages. On the other hand, the proposed approach in this work generates the UI prototype in an XML-based language which provides a general-purpose presentation of UIs that are implementation-independent of operating systems and platforms. We do not define any extended UML diagrams, so our approach can be integrated with existing CASE tools without further modifications.

3. Research Methodology

In this section, we propose an iterative prototyping process. The process starts with analyzing and designing UML models. Figure 1 shows the activities of the UI development process. The process starts with the developers designing UML diagrams that represent the requirements of the system under development. The UML diagrams then are used as input to generate the prototype throughout a series of transformations that produces the UIML code. The UIML code represents the UI in two perspectives: Content and structure (user navigation). We use special tools to render the UIML code. The resulting UIs can be evaluated and refined for the next iteration if needed. In the following, we describe the main activities in this process.

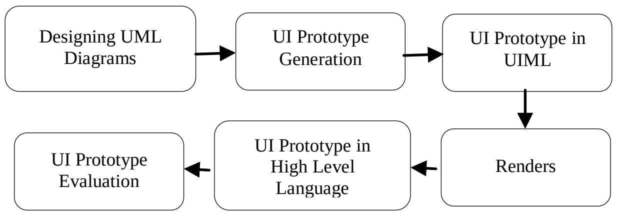


Figure 1. The process of generating UI prototype.

3.1. Designing UML Diagrams

The system analysis and design is composed of a set of UML diagrams that are annotated with UI objects. UML is considered simultaneously with graphical UI design [17, 21]. In this work, the use case diagram is used to visualize the interactions between users and the system. The details of a particular use case are visualized using a collaboration diagram, which also, depicts the relationship between a use case and the elements of the class diagram. The class diagram and the state transition diagrams are vital to discover the static (interface contents) and dynamic (navigation map) aspects of the prototype. A UI is composed of many interface objects that are derived through multi-step process. The users interact with the system using different interface objects, for example inserting or receiving data. The system analyst draws the use case diagram that represents the different interactions between the actors and the system. For each actor, the analyst draws the collaboration diagrams for all use cases based on objects in the class diagram. To ease prototyping, we assume that the system analyst should add a stereotype <<enduser>> to the actors that represent a personal end user; otherwise an actor can be a device or an external system. The analyst should draw the class diagram for the system to show data and functions of the system. The system analyst should add a stereotype <<UI>> for each class that is related to interface objects. The analyst should model all interactions with interface objects using one or many State Diagrams (SD). The system analyst should show the link between a state and a use case by adding the name of the use case as a stereotype for its relevant state. A state transition consists of action, guard and event. The analyst should add to each state an action that is composed of a name and a list of parameters. The analyst can describe user interactions by adding one of the stereotypes <<inputdata>>, <<outputdata>>

and <<useraction>>. An event is used when triggered by a specified guard condition. These UML diagrams are visualized using ArgoUML. ArgoUML provides the building blocks for serializing UML data textually, by exporting the UML model to an XML-based language, i.e., XMI. XMI is an open industry standard that applies XML to abstract systems such as UML models. It is used to capture and express the relationships in UML models, while discarding most of the visual details of a particular UML diagram.

3.2. UI Prototype Generation

After completing the UI modeling, the analyst exports the XMI file for all models and uses it as an input to generate a prototype. A prototype has two different components, static (interface content) and dynamic (navigation between UIs).

3.2.1. UI Content

Interface content is generated from information in state transitions based on rules which are adapted from [7, 14, 16]. There are three steps to generate interface contents from state transitions. First, generating a Directed Graph of Transitions (DGT). Second, masking interactive transitions. Third, generating prototype screens from the interactive transitions signals.

- Generating a DGT: This operation derives a DGT in a SD. Nodes in the DGT represent the transitions of the SD. If the target state of a transition T1 is the source state of transition T2, then there is an edge between node t1 and node t2 in DGT. In addition, a list of initial nodes (initialNodeList) is generated that contains the entry nodes of each DGT, for example, t1 in Figure 3 represents the entry node. The entry nodes are the transitions that come from the initial state of the SD. Figures 2 and 3 show a SD of login operation and their representation in the DGT.

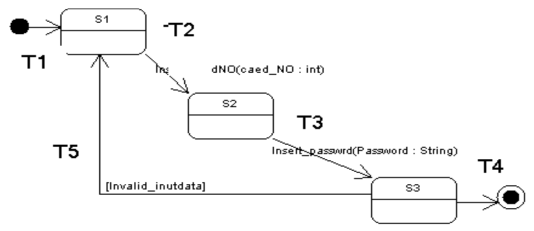


Figure 2. Login state transition diagram for three states (s1, s2, s3).

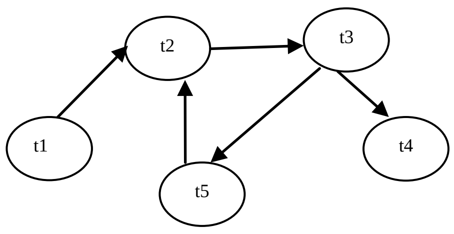


Figure 3. DGT of login SD.

- Masking Interactive Transitions: Transitions in the SD can be divided into two types. Interactive transitions and non-interactive transitions based on the transition signals. the non-interactive transitions have no effect on the UI contents, therefore all nodes that comes from non-interactive transitions are removed from the DGT, as well as the incoming and outgoing edges of this node.
- Generating Prototype Screens from Masked DGT: Figure 2 shows an example of how a structured SD looks like. Each node in the DGT is converted to a user screen based on its signal (action, event and guard). To produce UI contents, if the signal is a <<useraction>> then the parameter type is not important and a button widget is generated for all user interactions. if the signal is an <<inputdata>> or <<outputdata>> then the content depends on the parameter type as shown in Table 1.

Table 1. The mapping of parameter types to graphical widgets.

Signal	Parameter Type	Graphical Widget
<<useraction>>	Not Important	A Button
<<inputdata>>	A Primitive or a String	An Enabled Text Field
	Enumeration Values <= 6*	A Group of Radio Buttons
	Multiple Values (e.g., array)	An Enabled List
	A Boolean	An Enabled Check Box
<<OutputData>>	No Data	A Label Widget and the Caption is the Action Name
	A Primitive Data Type or a String	A Disabled Text Field
	Multiple Values (e.g., array)	A Disabled List

*6 radio buttons are arbitrary choice of the authors

3.2.2. UI Navigation

The dynamic aspects of a UI are modeled in a masked DGT. The navigation has two types: Intra-navigation (within a masked DGT) and inter-navigation (between several masked DGT). For intra-navigation, each node in the masked DGT represents a screen. Navigations between screens are specified based on three conditions starting from the initial node: If the node has one outgoing edge then navigation goes directly to the next screen; if the node has more than one outgoing edge then a menu screen is generated to switch between different screens and if there is no an outgoing edge, then there are two choices either the navigation reached the last possible screen (leaf screen) or the system navigates to another masked DGT (inter-navigation).

For inter-navigation, we integrate the masked DGT's using the use case diagram. The integration is done based on the following rules:

- *Rule* 1: If there is an end-user actor connected to more than one use case, a screen is generated that has a menu providing as options the different navigations for each associated use case.
- *Rule* 2: If there is a generalization/specialization relationship between two actors p and q (p is special kind of q), then all prototypes that are generated for the q actor will be inherited by the actor p.
- *Rule* 3: If there is a generalization/specialization relationship between use cases, then an interface for each special uses case is generated. The parent has a menu that navigates to all special ones.
- *Rule* 4: If there is an include relationship between two use cases and both of them are associated with

the actor, then the prototype of the included use case will be inherited by the other use case.

- *Rule* 5: If there is more than one actor then a new screen will be generated. This screen has a menu providing, as options, the different navigations for each actor.
- *Rule* 6: Finally, the main screen of each actor can be derived in two ways: First, if the actor is associated with multiple use cases, then the screen that is generated by using the (rule 1) is considered the main screen. Second, if the actor is associated with only one use case, then the screen that is derived from this use case is considered the main screen.

3.3. UIML Prototyping and Rendering

We use a tool to render the generated platform-independent UI prototype into high level languages like C#, Java and C++. We use open-source software, liveview[1], to render the UIML documents of the generated UI prototype into C# language [18]. UIML can have several renderings into several programming languages [3, 13, 15]. We use the .NET framework, C# language to build the UI-gen tool which automates the generation of UIs. UI-gen takes UML diagrams that are presented in an XMI file as an input and produces UIs in a UIML. In the appendix we show examples of both the XMI for a UML diagram and the XMI that represents UIML language.

4. UI Prototype Evaluation

Two case studies are evalutated, the Automatic Teller Machine (ATM) and the Wheel system.

4.1. The ATM Case Study

The ATM provides many services that are specified in UML diagrams [6]. Figure 4 represents the ATM use case diagram. The main use cases in the ATM are: A session is started when a customer inserts an ATM card number. Then, ask the customer to enter his/her password. If the card number or the password is invalid an error screen is displayed. A transaction use case is started within a session when the customer chooses a transaction type from a menu of options. A withdrawal transaction asks the customer to insert account number to withdraw from and to insert the amount of money. A deposit transaction asks the customer to insert the account number to deposit to and to insert the amount of money. A transfer transaction asks the customer to insert two account numbers (from and to) and the amount of money. An inquiry transaction asks the customer to insert the account number to inquire about. Figure 5 describes part of the basic structure of the class diagram. Figure 6 describes the SD of the session class. These diagrams are saved as XMI files which are exported into UI-gen. The tool transforms these files into UIML files using the proposed approach. A UIML renderer is then used to generate platform-specific interfaces. The results of the renderer are shown in Figures 7 and 8. Figure 7 shows the main screen welcome screen, the login screen which asks the end-user to insert a card number. After clicking next, a password screen appears. After the log in screen, the menu options screen appears. The next screen asks the user to choose one among several use cases. In Figure 8, we show the screens related to the withdrawal use case.

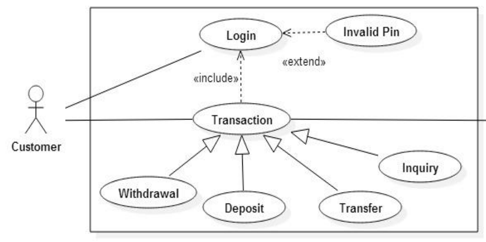


Figure 4. ATM use case diagram.

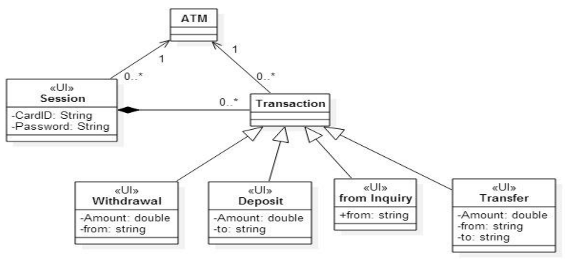


Figure 5. ATM class diagram.

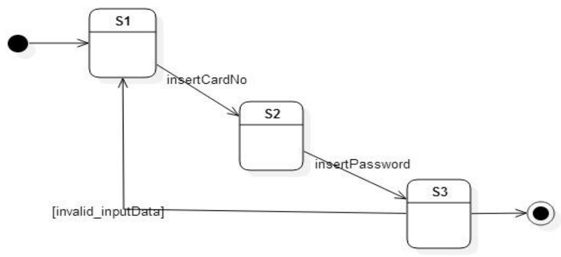
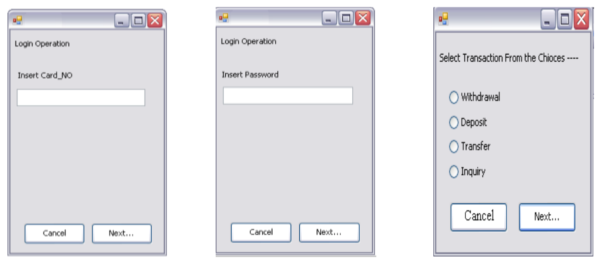
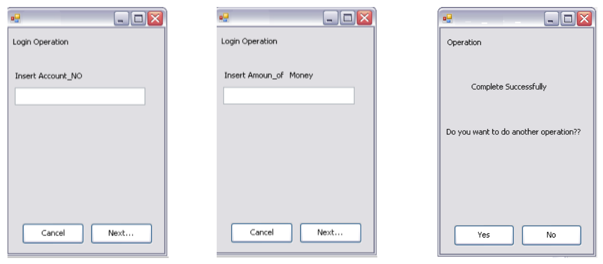


Figure 6. Session SD.



a) Welcome screen. b) Login screens. c) Menu options.

Figure 7. the main transaction



a) Insert account number. b) Insert money amount. c) Completion confirmation.

Figure 8. The scenario of withdrawal (three screens).

[1] http://research.edm.uhasselt.be/~gummy/#info

4.2. The Wheel System Case Study

The requirement specifications of the Wheel system were proposed originally in [9]. The specifications of this system are described using UML diagrams. Figure 9 show the main use case in the Wheel system. The tool transforms UML diagrams to UIML files. We used the .NET renderer again to generate C# interfaces. The results of the renderer are shown in Figures 10 and 11.

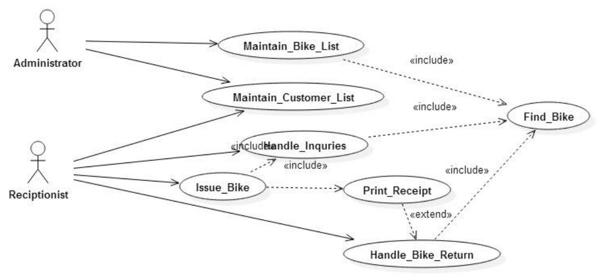
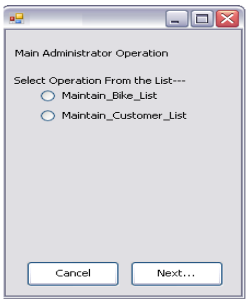


Figure 9. Wheel system use case diagram.

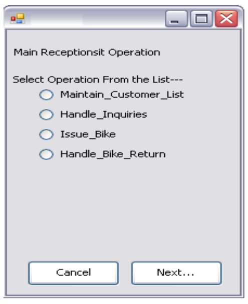


Main Administrator Operation

Select Operation From the List---

- Maintain_Bike_List
- Maintain_Customer_List

Figure 10. Select customer type screen in wheel system.



Main Receptionsit Operation

Select Operation From the List---

- Maintain_Customer_List
- Handle_Inquiries
- Issue_Bike
- Handle_Bike_Return

Figure 11. Receptionist choice screen in wheel system.

5. Conclusions and Future Work

In this research, we provided an underlying approach to generate UIs from UML diagrams. The approach analyzed the UML diagrams that represent an initial understanding of the system under development. To validate this approach, we built a tool UI-gen to transform UML diagrams specified in XML-based language into a UI language UIML. This tool provides a general-purpose presentation of UIs that are implementation-independent from operating systems and platforms. The resulting prototype is generated from UML diagrams. The process of generating these UIs should help developers in producing a rapid prototype that can help analysts in their negotiation with customers. The generation of the UIs requires no considerable efforts from the analysts and designers of the system under development. We successfully evaluated the tool on two case studies, the ATM and Wheel systems.

Limitations and Future Work: User prototypes can be used as a means of facilitating discussion about an existing or to propose a new system. Therefore, the models need not to be complete or correct for the purpose of prototyping. Finally, The case studies under consideration are from academia and may not represent all software application domains. In the future, we will define new UIML generic vocabulary and a renderer for many computer languages.

References

[1] Abrams M. and Helms J., “User Interface Markup Language (UIML) Specification Version 3.1,” *Technical Report, Oasis UIML Technical Committee*, 2004.

[2] Ali M., “A Transformation-based Approach to Building Multi-Platform User Interfaces using a Task Model and the User Interface Markup Language,” *PhD Dissertation, Virginia Polytechnic Institute and State University*, Virginia, USA, 2004.

[3] Ali M., Pérez-Quiñones M., Abrams M., and Shell E., “Building Multi-Platform User Interfaces with UIML,” *in Proceedings of the 4th International Conference of Computer-Aided Design of User Interfaces*, Valenciennes, pp. 255-266, 2002.

[4] Almendros J. and Iribarne L., “An Extension of UML for the Modeling of WIMP User Interfaces,” *the Journal of Visual Languages and Computing*, vol. 19, no. 6, pp. 695-720, 2008.

[5] Almendros J. and Iribarne L., “Designing GUI Components from UML Use Cases,” *in Proceeding of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pp. 210-217, 2005.

[6] ATM., available at: http://www.math-cs.gordon.edu/courses/cs211/ATMExample, last visited 2011.

[7] Bodart F., Hennebert A., Leheureux J., Provot I., and Vanderdonckt J., “A Model-based Approach to Presentation: A Continuum from Task Analysis to Prototype,” *in Proceedings of the 1st Eurographics Workshop on Design, Specification, Verification of Interactive Systems*, Carrara, Italy, pp. 77-94, 1994.

[8] Bouabana T. and Belmesk M. “Integration of the Association Ends within UML State Diagrams,” *the International Arab Journal of Information Technology*, vol. 5, no. 1, pp. 7-15, 2008.

[9] Britton C. and Doake J., *A Student Guide to Object-Oriented Development*, Elsevier Butterworth–Heinemann, 2005.

[10] Elkoutbi M. and Keller R., "User Interface Prototyping based on UML Scenarios and High-Level Petri Nets," *in Proceedings of 21st International Conference*, *ICATPN 2000 Aarhus*, Denmark, pp. 166-186, 2000.

[11] Elkoutbi M., Khriss I., and Keller R., "User Interface Prototyping using UML Specifications," available at: http://www.iro.umontreal.ca/~keller/Suip/bookChapter.pdf, last visited 2000.

[12] Huzarr Z. and Loniewskil G., "Deriving Prototypes from UML 2.0 Sequence Diagrams," available at: http://ceur-ws.org/Vol-252/paper09.pdf, last visited 2007.

[13] Luyten K., Thys K., Vermeulen J., and Coninx K., "A Generic Approach for Multi-Device User Interface Rendering with UIML," *Computer-Aided Design of User Interfaces V*, pp. 175-182, 2007.

[14] IBM. Systems Application Architecture: Common User Access-Guide to User Interface Design-Advanced Interface Design Reference, 1991.

[15] Luyten K. and Coninx K., "Uiml.net: An Open UIML Renderer for the .Net Framework," *in Proceedings of the 5th International Conference of Computer-Aided Design of User Interfaces*, Dordrecht, pp. 259-270, 2004.

[16] Maher T., "Automated Generation of the User Interfaces," *Department of Computer Science and Computer Engineering*, *La Trobe University*, Melbourne, Australia, 1994.

[17] Martinez A., Estrada H., Sanchez J., and Pastor O., "From Early Requirements to User Interface Prototyping: A Methodological Approach," *in Proceedings of the 17th IEEE International Conference on Automated Software Engineering*, pp. 257-260, 2002.

[18] Meskens J., Vermeulen J., Luyten K., and Coninx K., "Gummy for Multi-Platform User Interface Designs: Shape Me, Multiply Me, Fix Me, Use Me," *in Proceedings of the International Working Conference on Advanced Visual Interfaces*, Napoli, Italy, 2008.

[19] Nichols J., "Automatically Generating User Interfaces for Appliances," available at: http://www.cs.cmu.edu/~jeffreyn/papers/doccon_paper.pdf, last visited 2004.

[20] Phanouriou C., "UIML: A Device-Independent User Interface Markup Language," *PhD Dissertation*, *Virginia Polytechnic Institute and State University*, 2000.

[21] Pinheiro P. and Paton N., "A UML-based Design Environment for Interactive Applications," available at: http://www.cs.man.ac.uk/~norm/papers/uidis01.pdf, last visited 2001.

[22] Rajabi B. and Lee S., "Consistent Integration between Object Oriented and Coloured Petri Nets Models," *the International Arab Journal of Information Technology*, vol. 11, no. 4, pp. 406-415, 2014.

[23] Sommerville L., "Software engineering. Professional Computing Series," available at: http://www.acm.org/about/se-code, last visited 2007.

[24] Stangl H., "Script: A Framework for Scenario-Driven Prototyping," *PhD Dissertation*, Technische Universität München, 2012.

[25] Xia B. and Zhang Y., "A Mapping Method of using the Compound Use Cases to Generate User Interface," *Open Journal of Applied Sciences*, vol. 2, no. 3, pp. 180-183, 2012.

Amani Shatnawi received her BSc and MSc degrees in computer science from Jordan University of Science and Technology. Currently, she is a PhD student and a research assistant in Computer Science Department at Utah State University, USA. She is interested in software modeling, search engines of temporal XML data.

Raed Shatnawi is currently associate professor in Jordan University of Science and Technology. He received PhD and MSc from University of Alabama, USA. He has published many ISI Journals in the software engineering field.