Trends in Applied Sciences Research

ISSN 1819-3579

Academic
Journals Inc.

www.academicjournals.com

Trends in Applied Sciences Research 6 (9): 1046-1054, 2011
ISSN 1819-3579 / DOI: 10.3923/tasr.2011.1046.1054

Synthesizing a Composite Model for Runtime Monitoring and Adapting Goal Oriented Systems

Seyed Morteza Babamir and Abdollah Aghaei
Department of Computer Engineering, University of Kashan, Kashan, Iran

Corresponding Author: Seyed Morteza Babamir, Department of Computer Engineering, University of Kashan, Kashan, Iran

ABSTRACT

In goal oriented requirements engineering, a system is developed through identifying and refining goals. Then, the system developer proceeds with goal identification and refinement to obtain requirements. Responsibility for satisfaction of requirements is assigned to agents. Because of insufficient specifications and change of environmental conditions, goals may be violated during run time. The main aim of this study was to identify goal violation by run time monitoring system. This paper has presented a composite model for: (1) monitoring behavior of software agent and (2) adapting system to requirements if some software agent violates some requirement. To show the effectiveness of the proposed model, we applied it to the CIIP (Continues Infusion Insulin Pump) system. By applying present method, we came to the conclusion that our model presented a way by which one is able to: (1) develop a system by goal-oriented approach, (2) monitor it based on the events and (3) adapt it to the new situation.

Key words: Goal oriented, goal adaptation, requirement engineering, runtime monitoring, verification

INTRODUCTION

Goal-oriented is a requirements engineering method for eliciting, elaborating, structuring, specifying, analyzing and documenting requirements (Van Lamsweerde, 2004). Goals vary from high-level concerns to low-level requirements. To achieve goals, agents such as software components, input/output devices and human should collaborate with each other. The significant decision in the requirements engineering process is assignment of responsibilities for goals to agents. However, agents may fail in achieving goals. In the following violation of a system specification is considered.

System specification may be violated because: (1) environment agents may behave in an unpredictable way at run-time and (2) the post-deployment evolution of environment conditions can make initial valid assumptions are no longer valid. Two complementary approaches can be followed to manage runtime violations of requirements:

- Predicting as many as possible requirements at specification time
- Detecting and resolving such violations at runtime

The first approach called static approach applies method obstacle handling (Van Lamsweerde and Letier, 2000). In this method, having identified goals, we should identify and resolve the causes of dissatisfaction of the goals. While obstacle analysis may highly deliver a

cost-effective method for obtaining robust specifications, identifying complete specification of all possible obstacles may be unachievable. Besides, an overly defensive specification may be too costly to implement and result in unnecessary complexity of the software. It is here that the runtime monitoring comes into play (Sayenko *et al*., 2010) as well as testing (Layachi-badri, 2006; Xin *et al*., 2010) is effective.

Our suggested model has three main steps. In the first step, we specify the system by the KAOS (Van Lamsweerde, 2009, 2008) goal oriented methodology. Then, we identify the possibly violated goals at run-time and generate monitor code by the FLEA (Feather *et al*., 1998) compiler. In the second step, an agent monitor observes the software agent behavior while it is executing. In the three step, the system is adapted if some goal is violated. We use *one point adaptation* method by which the system stops its initial execution path and starts an alternative one. That which of the alternative execution paths should be started is a concern. To deal with this concern, we use alternative sub-goals of the refinement tree. Having adapted the system to an alternative, we change the monitor code to observe the new situation if necessary. Overall, this study discusses runtime violation of goals identified at specification time. The violation may be due to unpredictability of the behavior of environment agents or change of environmental conditions.

GOAL ORIENTED REQUIREMENT ENGINEERING

Goal oriented requirement engineering deals with identifying the goals of a system, refining the goals to sub-goals so far as to identify requirements. Then, it addresses assignment of responsibility for satisfying requirements to the agents. Each goal is an objective that system should achieve it in collaboration with software agents and environment ones. Requirement engineering is a significant step in software development (Kheirkhah *et al*., 2009) and goals have important role in the process (Mylopoulos, 2006) in which the goal refinement is performed by and/or in a refinement graph.

Graphically, a goal is depicted as a parallelogram in and refinement, a goal is gradually refined to a set of connected sub-goals; so, to satisfy the goal, all sub-goals should be satisfied. In an or refinement, each goal is refined to a set of alternative sub-goals so that it is enough that at least a goal is satisfied (Van Lamsweerde, 2009).

An agent is graphically represented as a hexagon. Responsibility an agent for a goal is indicated by pointing an arrow from the agent to the goal or requirement. Graphically, a requirement is depicted as a parallelogram with a bold black border (Van Lamsweerde, 2009).

An operation is an action performed by an agent to achieve a goal by operationalizing a requirement. Graphically, operations are represented as ovals. Operationalization is shown graphically as a circle indicating a requirement should be operationalized. An agent is responsible for an operation and the agent responsibility is shown by a circle including lines that point to the operation for which the agent is responsible (Van Lamsweerde, 2009).

A message is a communication between two agents represented graphically as a pentagon together with a pointer (Van Lamsweerde, 2009).

In requirements engineering, KAOS is a goal oriented methodology defining formally goals in real-time temporal logic formulae (Van Lamsweerde, 2009). Figure 1 shows the KAOS methodology steps. The figure consists of four sub models: Goal, object, responsibility and operational. The responsibility model is obtained from goal model and the operational one is obtained from responsibility one. Figure 1 shows the KAOS method by indicating that how the corresponding sub models are obtained. To construct the goal refinement graph, goals are elicited from available sources while we ask why and how questions (goal elaboration step); objects, relationships and attributes are derived from the goal specifications (object modeling step); agents are identified,

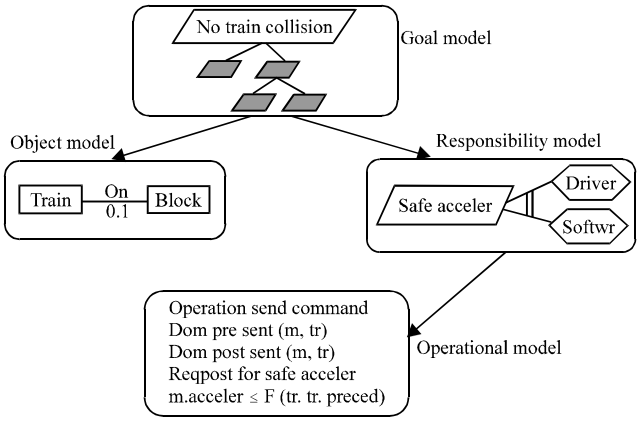

Fig. 1: The KAOS models

alternative responsibility assignments are explored and agent interfaces are derived (responsibility assignment step); operations and their pre-conditions and post-conditions are identified (operational model). These steps can be advanced in parallel with each other or one step can be backtracked to a previous one.

To identify goals from initial document, one can search the document for keywords "objective", "purpose", "intent", "concern", "in order to" and etc. Then, goals are formalized and the object model is derived. To explore alternative responsibility assignments, goals should be formulated from the goal elaboration step. In the refinement graph, assignments of terminal goals to individual agents are captured and/or by responsibility links. Having assigned terminal sub-goals to individual software or environmental agents, we derive each agent from the goal specifications in order to derive agent interfaces. To identify operations, one should identify operations relevant to goals and then define their pre-conditions and pos-conditions. Since goals show specific state transitions, an operation that fires the transition is identified. The goal pre-condition and post-condition captures the state transition. To operationalize goals, one should elaborate the conditions so that the various goals linked to the operation are achieved. For goals assigned to software agents, this step produces requirements on the operations for the corresponding goals to be achieved.

The KAOS model uses temporal operators to formally define the components: o, means next state, ●, means previous state, ◊, means eventually, ◆, means sometime in the past, □, means always in the future, ■, means always in the past, U, means until and W, means unless. Real-time restrictions are indicated by subscripts; e.g., ◊ ≤ nu means "sometime in the future within n time units u". We use the KAOS methodology for development of our system.

RUNTIME MONITORING

Having developed the system, we identify the goals that may are violated at runtime. We use the formal definition of these goals to generate the monitor code. By the FLEA compiler (Feather *et al.*, 1998), we generate the monitor code from formal definition of the goals may be violated.

The FLEA language provides constructs for expressing temporal combinations of events. Run-time code to monitor such combinations is automatically generated by the FLEA compiler. The run-time system comprises an historical database management system equipped with an inference engine and a communication mechanism to gather events and distribute notifications of occurrences

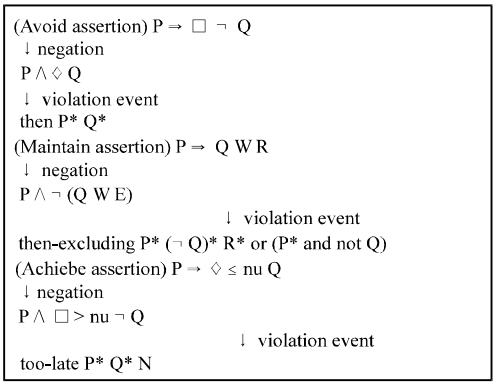

Fig. 2: Translating the KAOS goals patterns to FLEA ones

Table 1: Temporal patterns of FLEA

Syntax	Meaning
Then P Q	An event P followed by an event Q
Then-excluding P Q R	An event P followed by an event Q, without any event R in-between
In-time P Q d	An event P followed by an event Q within time delay d
Too-late P Q d	An event P not followed by an event Q within time delay d

of event combinations. The FLEA introduces events as special relations in which the first parameter is the time at which the event occurs. Other parameters are event attributes. Table 1 shows the various definable temporal patterns in FLEA. Using some transformation rules, the KAOS goal patterns can then be translated into FLEA event definition patterns shown as in Fig. 2.

RUNTIME ADAPTATION OF THE SYSTEM

At run-time, the monitor agent controls the behavior of system agent. If some goal violation occurs, the monitor reports this violation to the change agent. The change agent then adapts system to one safe state so that prevents the system from some failure. To keep the system off some failure, we should replace the execution path of the violated goal with an alternative sub-goal path. To this end, we use refinement tree of the violated goal and use one-point adaptation in adaptation part of our model.

Semantic of the one point adaptation indicates that the system initially is executing by first program named the source program. The system is able to work by alternative way named the target program. When the one point adaptation should be carried out, the source program should be stopped and the target program should be started. This approach is well-suited for systems that have at least two alternative ways.

Zhang and Cheng (2006) presented the semantic of adaptation for Meta-Socket problem. Brown *et al.* (2006) presented a goal oriented approach for the Meta-Socket problem by KAOS methodology. These approaches are compatible with adaptive systems having two different execution modes; however, we use them and adapt it so that we can replace an alternative sub-tree of the goal refinement tree for some violated goal. Figure 3 shows the general one-point adaptation we use for the model which it may be generalized to different systems.

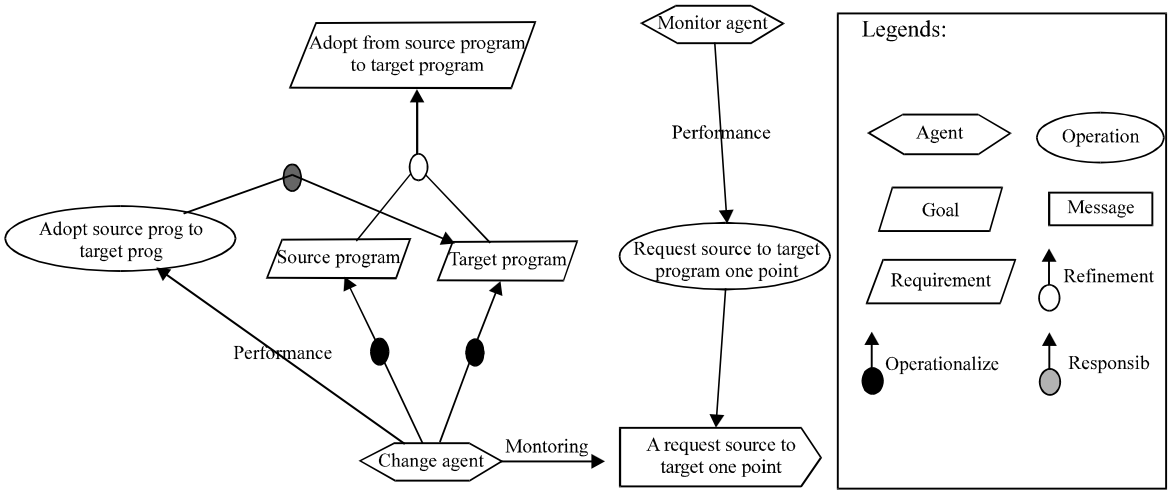

Fig. 3: One point adaptation model

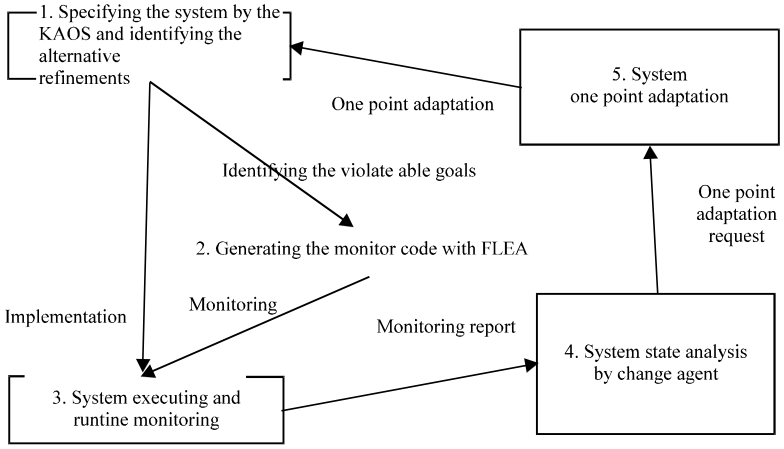

Fig. 4: This study suggested model to monitor and adapt goal-oriented systems

By one-point adaptation, our general model works as follows: (1) the *monitor agent* performs the operation "Request source to target program by one point" and (2) reports the message "A request for source to target by one point". Having observed the monitor agent message, the change agent performs the operation "Adaptation of source-prog to target-prog". The system initially is in "source program" state and it will be in the "target program" state when the goal adaptation is satisfied.

THE SUGGESTED MODEL

Here, we represent run-time monitoring and adaptation model (Fig. 4). The composite model is based on Feather model (Cohen *et al*., 1997) and Brown model (Zhang and Cheng, 2006; Brown *et al*., 2006). But the Feather model specifies no specific method for system adaptation and so their model is not a precise model. Therefore, we should adapt it for different systems. We generate a general one point adaptation model and combine it with the Feather model. This study

adaptation model works with replacing an alternative refinement sub-goal from refinements tree of violated goal. One point adaptation is compatible for such systems stopping the execution of the program at changing alternative sub-goal time and so the systems will work without fail.

The first step of this study model is goal oriented specification of system properties. In this step, goals and requirements of system are specified by and/or refinement tree. Then, responsibility for satisfaction of requirements is assigned to the system agents and the system executes. Another work that should be performed at development time is verification of system so as to identify and resolve the errors that may occur.

The second step is identification of all erroneous states which not only it may be not practical but also it is a costly idea. This is why that we want to identify the goals that just may be violated at run-time. To this end, we design the monitor through formal definition of such goals. The monitor is an agent whose responsibility is monitoring the behavior of system agents and comparing the system behavior with the system specification. We use the FLEA language for translating the formal definition of goals may be violated and then we generate the monitor code.

The third step is the system execution and monitoring the system behavior by the agent monitor. If a goal violation occurs, the monitor reports it to the change agent.

The fourth step is analyzing the system states and requesting adaptation by the change agent if necessary.

The fifth step is the system adaptation by one point adaptation method carried out by the change agent. The system execution through source program stops and the execution through an alternative sub-tree of the violated goal starts (this is the target program). After adapting the system, system conditions may change and so we need to reconstruct the monitor to monitor the system together with new conditions.

STATEMENT OF THE PROBLEM

Here, we aim to show the effectiveness of present method to synthesize the run-time monitor of a safety-critical system called CIIP (Continues Infusion Insulin Pump) one. The system includes a controller, whose software determines diabetic's blood sugar and computes some insulin dose to deliver the diabetic. In order to constant control of the diabetic's sugar, a miniature insulin pump is worn by the diabetic. The system works as follows. The input sensor samples diabetic's blood one time per 10 min to determine the current blood sugar and reports the sugar value, measured in micrograms/milliliter between 1 and 35, to the controller. Then, the controller computes some insulin dose and the needle set delivers it to the diabetic. Developed for depicting the goal-oriented software engineering, the Objectiver tools (http://www.objectiver.com) used to develop present models.

Figure 5 shows the goal refinement tree for the insulin delivery. The goal "give needed insulin" constitutes the root of the tree and its sub-goal is "adapt from auto dose computation to user give dose". We provided this property to adapt some violated goal. The goal is followed for two reasons: (1) The requirement "insulin dose calculus" is achieved by the software agent and (2) The requirement "user give dose" is achieved by the user agent respectively. The goal behind "give needed insulin" may be failed due to change of some condition or due to some behavior during the operation. So, we design the monitor code using the formal definition of goals shown in Fig. 6. The figure, first of all, shows the formal definition of the goal negatively and then regarding the patterns stated in Fig. 2, it shows the generated monitoring events.

The goal "given needed insulin" consists of "achieve" pattern; therefore, we use the pattern shown in Fig. 2 and then generate the monitor code. During the system execution, the monitor

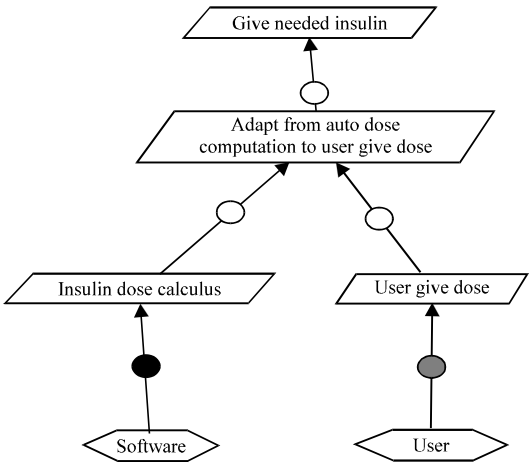

Fig. 5: Refinement tree for the CIIP system

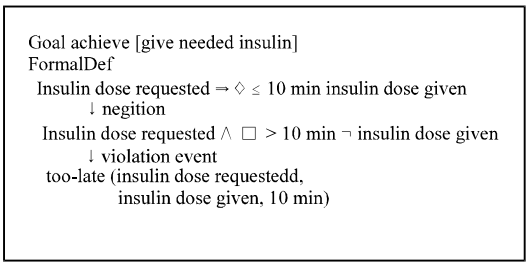

Fig. 6: Monitor code of the CIIP System

observes the software agent behavior and in case of goal violation, the "adaptation request" is sent to the change agent. Here, the change agent (which is a software agent) considers adaptation of the "insulin calculation by software" to "user give dose". Using this method, the program execution is halted by the agent and the system starts to deliver some insulin dose by its user. Figure 7 shows, present suggested model for the CIIP system in which a blood sample is taken one time per 10 minutes. As the model shows, the system can deliver insulin dose through two ways causing the system is well-suited for this study model.

Having observed the software agent behavior, the monitor agent will carry out the "request on point adaptation" operation if it observes some goal violation. Afterwards, the operation reports the message "request on point adaptation". Having received the message, the software agent carries out the operation "adapt insulin dose calculation by software with user give dose". Performing this operation stops insulin delivery by the system and starts it by the user.

Van Lamsweerde and Letier (2000) presented an obstacle handling method to prevent the system failure at the development time. However, because of insufficiency of static approaches, dynamic approaches have come into play. Feather *et al.* (1998) presented a model for monitoring and adapting the goal oriented systems at run-time. In this model, the monitor was created automatically by the FLEA compiler. In the adaptation part of the model, they only presented general suggestions. However, we use the Feather approach in present model to develop the system and automatically generate the monitor code. Another difference is that we use

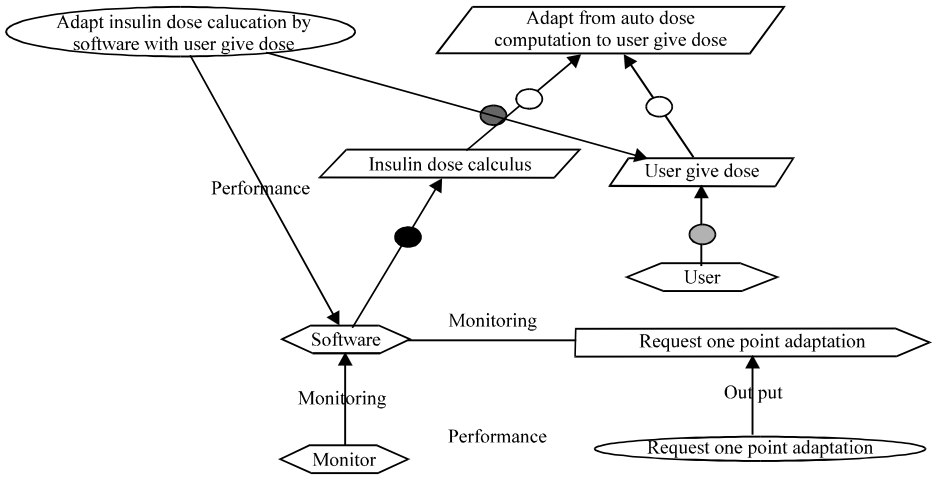

Fig. 7: The suggested model for the CIIP system by three different adaptations

one point adaptation model (Zhang and Cheng, 2006) in the adaptation part. One point adaptation model presented for the adaptive systems having two different execution ways (Brown *et al*., 2006). The Brown *et al*. (2006) have no stress on the monitor part in their model. Accordingly, we changed their model to replace an alternative sub-goal of refinement tree and used it in the adaptation part of present model. Also, Zhang and Cheng (2006) presented an adaptation semantic by a formal approach and then Brown *et al*. (2006) translated their formal models to the goal oriented ones by the KAOS methodology. Present model has integrated the benefits of both models.

CONCLUSION, SUGGESTIONS AND FUTURE STUDY

The one-point adaptation model was suggested for the systems which are naturally adaptive and can work at least in two special ways. We applied the one-point adaptation model in the body of the monitoring system by substituting the alternative sub-goal using the goal refinement tree. We stated a case study to show effectiveness of our suggested model. Feather *et al*. (1998) presented a general model for monitoring and adapting goal-oriented requirements whereas, the major advantage of our model is the stress on the adaptation part and specially the use of the one-point adaptation model. Although Brown *et al*. (2006) presented an adaptation model, our model considered the monitor as an event-oriented one. In the other words, we used the one point adaptation which is appropriate for the adaptive systems and generalizes it for replacing an alternative sub-goal using the refinement tree. Our model, in fact, presented a way to develop the system by goal-oriented approach in order to one can monitor it based on the events and adapt it to a new situation. Moreover, our model integrated the advantages of both models. Applied in our adaptation model, Fig. 7 showed three different adaptations. The choice of adaptation depends on the application requirements.

We exploited the FLEA language for generating the monitor code. This can be resulted in use aspect oriented languages such as the ASPECTJ (Laddad, 2009) language for generating the monitor code. Aspects have been used to software development (Fazal-e-Amin *et al*., 2010). Our main contribution was generalization of a monitoring and adaptation model, where can be used by different systems.

REFERENCES

Brown, G., B.H.C. Cheng, H. Goldsby and J. Zhang, 2006. Goal oriented specification of adaptation requirements engineering in adaptive systems. Proceedings of the International Conference on Software Engineering, May 20-28, Shanghai, pp: 23-29.

Cohen, D., M.S. Feather, K. Narayanaswamy and S.S. Fickas, 1997. Automated monitoring of software requirements. Proceedings of the 19th Conference on Software Engineering, May 17- 23, Boston, MA, USA., pp: 602-603.

Fazal-e-Amin, A.K. Mahmood and A. Oxley, 2010. A review on aspect oriented implementation of software product lines components. Inform. Technol. J., 9: 1262-1269.

Feather, M.S., S. Fickas, A. Van Lamsweerde and C. Ponsard, 1998. Reconciling system requirements and runtime behavior. Proceedings of the 9th International Workshop on Software Specification and Design, Apr. 16-18, Ise-Shima, Japan, pp: 50-59.

Kheirkhah, E., A. Deraman and Z.S. Tabatabaie, 2009. Screen-Based prototyping: A conceptual framework. Inform. Technol. J., 8: 558-564.

Laddad, R., 2009. AspectJ in Action. 2nd Edn., Manning Publication, India, ISBN: 1933988053, pp: 568.

Layachi-Badri, S., 2006. Structural test of a data basis oriented object after phase of conception. Inform. Technol. J., 5: 753-758.

Mylopoulos, J., 2006. Goal-oriented requirements engineering. Proceedings of the 14th IEEE International Requirements Engineering Conference, Sept. 11-15, Minneapolis, USA., pp: 5-5.

Sayenko, V., M. Al-Rawajbeh and A. Golubev, 2010. Continuous monitoring system for estimation the quality of service of computer network. J. Applied Sci., 10: 2034-2040.

Van Lamsweerde, A. and E. Letier, 2000. Handling obstacles in goal-oriented requirements engineering. IEEE Trans. Software Eng., 26: 978-1005.

Van Lamsweerde, A., 2004. Goal-oriented requirements engineering: A roundtrip from research to practice. Proceedings of the 12th IEEE International Requirements Engineering Conference, Sept. 6-10, IEEE Computer Society, Washington, DC. USA., pp: 4-8.

Van Lamsweerde, A., 2008. Requirements engineering: From craft to discipline. Proceedings of 16th ACM SIGSOFT International Symposium on the Foundations of Software Engineering, Nov. 9-14, Atlanta, Georgia, pp: 297-307.

Van Lamsweerde, A., 2009. Requirements Engineering: From System Goals to UML Models to Software Specifications. John Wiley, New York.

Xin, W., H. Feng-Yan and Q. Zheng, 2010. Software reliability testing data generation approach based on a mixture model. Inform. Technol. J., 9: 1038-1043.

Zhang, J. and B.H.C. Cheng, 2006. Using temporal logic to specify adaptive program semantics. J. Syst. Software, 79: 1361-1369.