The New Approach for Reliability Assessment of Control Systems Software

Martin Jedlicka, Oliver Moravcik, Andrej Elias, Lukas Smolarik

***Abstract*—This paper introduces the new approach for reliability estimation of control systems software. First we provide the basic starting points and definitions required for understanding the approach. Next we define the main parameters and variables, which are necessary for reliability estimation. The last section deals with the model for assessing the reliability of control systems software.**

***Index Terms*— control dependency graph, control system software, function block, reliability.**

I. INTRODUCTION

THE one of the most complex software non-functional requirements, which cannot be missed out when designing software system, is the reliability. The software reliability is the probability that a software system is performing successfully its required functions for the duration of a specific mission profile. In the past years there were introduced several approaches to evaluate reliability of software systems in the early stages of software development at the architecture level using several models (e.g. UML, Petri-nets, etc.)

The adoption of IEC 61499 standard provided model for using some aspects of component-based software also to control systems software. This main fact will be used for reliability assessment of control systems in this paper.

The approach will combine various techniques from software engineering practice (e.g. UML, Sequence diagrams) and also algorithm for directed graph traversing.

II. DEFINITION OF BASIC FACTS

Definition 1: Dependency is the relationship among two or more objects, where change in one or more objects leads to a potential change in one or more objects. [3].

Manuscript received August 8, 2012; revised July 22, 2012.

Martin Jedlicka is with the Institute of Applied Informatics, Automation and Mathematics, Faculty of Materials Science and Technology in Trnava, Slovak University of Technology in Bratislava, Trnava, 91701, Slovak Republic (martin.jedlicka@tcempire.sk).

Oliver Moravcik is with the Institute of Applied Informatics, Automation and Mathematics, Faculty of Materials Science and Technology in Trnava, Slovak University of Technology in Bratislava, Trnava, 91701, Slovak Republic (oliver.moravcik@stuba.sk).

Andrej Elias is with the Institute of Applied Informatics, Automation and Mathematics, Faculty of Materials Science and Technology in Trnava, Slovak University of Technology in Bratislava, Trnava, 91701, Slovak Republic (andrej.elias@stuba.sk).

Lukas Smolarik is with the Institute of Applied Informatics, Automation and Mathematics, Faculty of Materials Science and Technology in Trnava, Slovak University of Technology in Bratislava, Trnava, 91701, Slovak Republic (lukas.smolarik@stuba.sk).

Fig. 1 represents the relationship between two objects *A* and *B*. Dependence is directed edge between related objects. Then we can say that object *B* is dependent on the object *A*.

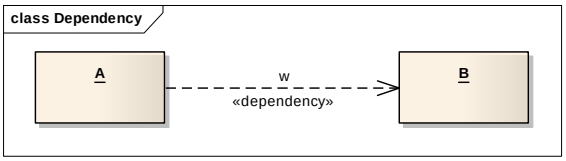

Fig. 1 Dependency relationship between UML objects

Definition 2: **The dependency ratio** (*w*) represents quantitative characteristic how one or more objects depend from one or more objects.

The dependency ratio is a quantitative value of edge, which represents the dependency relationship between two objects. This quantitative value may reflect some degree of software quality characteristics (e.g. reliability) or degree of probability.

Definition 3: **Scenario** is represented by a set of interactions $I=\{i_1,...,i_m\}$ the set of objects $O=\{o_1,...,o_n\}$, where *m* is the total number of interactions and *n* is the total number of objects. The object will further represent component, respectively functional block or its instance.

The scenario is used for analyzing the dynamic behavior of software. The UML sequence diagrams [6] are one of the tools for capturing dynamic behavior. Sequence diagrams are used in the proposed methodology to capture the dynamic behavior of the software of control system which is modeled according to standard IEC 61499.

Definition 4: **Sequence diagram** is two-dimensional graph showing the scenario where the horizontal axis represents a set of objects and the vertical axis represents the time which passes from the top to bottom.

Fig. 2 represents a sequence diagram that contains the horizontal axis of a set of objects $\{o_1,...,o_n\}$, where the order of objects on the horizontal axis is not significant.

On the vertical axis, time goes from top to bottom, where the objects exchanged messages among themselves. The order of message interactions is significant and the first sent message is always on the top of the vertical axis. The dashed line represents the lifeline of the object and the rectangle located on the lifeline represents the activity of the object at the time.

Definition 5: **Interaction $I_S(o_i, o_j)$** represents set of all messages sent from object o_i to object o_j during the execution within a single scenario *S*, which is described by a sequence diagram.

Fig. 2 also shows the set of all messages $\{i_1, i_2, i_3, i_4\}$ within the described scenario.

ISBN: 978-988-19252-4-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

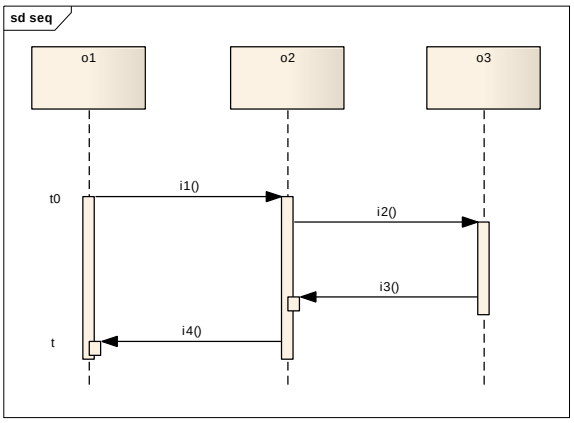

Fig. 2 Sequence diagram

Definition 6: **Interaction matrix** is two-dimensional matrix, where *i* is the row index and *j* is the index column, and the matrix has dimension $N \times N$, where N represents the number of all objects.

	o_1	o_2	…	o_j	…	o_N
o_1						
o_2						
…						
o_i						
…						
o_N						

Fig. 3 Interaction matrix

Definition 7: **Interaction matrix row** contains the numerical values of the interactions for scenario *S*, where the object o_i represent the sender role (NSI = the number of sent interactions).

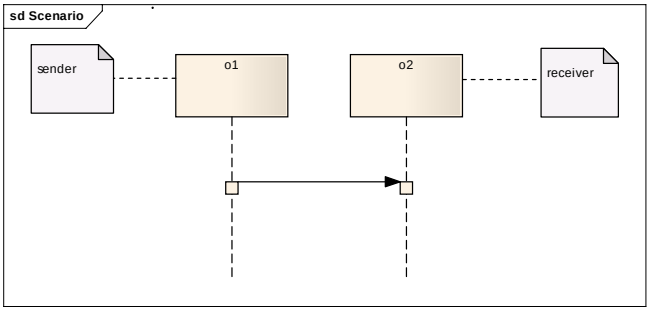

Fig. 4 Object roles in interaction

Fig. 4 shows the roles of object in their possible interactions. The object role in interaction is also represented graphically by direction of interaction - the direction of the arrow from the sender to the recipient.

Definition 8: **Interaction matrix column** contains the numerical values of the interactions for scenario *S*, where the object o_j represents the recipient role (NRI = number of received interactions).

Definition 9: **Interaction matrix cell (i, j)** represents the numerical value of number of interactions between objects o_i and o_j.

Diagonal cells in interaction matrix don't contain any value because they represent the object interaction with itself, and such interactions are not significant for this paper.

Fig. 5 represents the general example of interaction matrix, where value indicated as *A* is the number of interactions sent from object o_2 to object o_1, and also the number of interactions received by the object o_1 from the object o_2.

	o_1	o_2	…	o_j	…	o_N
o_1	0					
o_2	***A***	0				
…			0			
o_i				0		
…					0	
o_N						0

Fig. 5 Interaction Matrix Example

A. Scenario identification

The scenarios represent different modes of behavior in analyzing phase of the control system software, i.e. they model all the possible ways control flow across the software.

Definition 10: Scenario indicated as S_k is a sequence of possible interactions of elements, and $S_k \in S$, where $k=1...|S|$ and *S* represents the total set of application scenarios.

Element is a function block, which has interactions with other function blocks. UML sequence diagrams are the most appropriate for scenario modeling, where function block is then transformed into sequence diagram as a component.

Scenarios Identification may be based on:

1. Existing model of the function blocks network and knowledge of system behavior,
2. Required operational profile.

If there is known model of function block network, we can identify scenarios on behalf of system behavior knowledge which is represented by the function block network model. The scenarios are then analytic output of function block network behavior.

If scenarios are identified from the desired operational profile, they are created during the design phase of control system software using function blocks. UML diagrams, as a tool to support software engineering, can be used in the design phase of control system software. The required operational profile based on the functional requirements of the application.

B. Sequence Diagram Construction

In this step, we have to construct sequence diagram for each scenario. If there exists the system architecture modeled by the function block network, it is necessary to transform its behavior into a sequence diagrams using mapping rules. Following rules are respected when mapping functional blocks into UML elements:

- The name of the element in the sequence diagram is the same as the name of the function block instance;
- Interactions of function blocks in the sequence diagram are representing the data flows from the function blocks model;

ISBN: 978-988-19252-4-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

- One data flow from the function blocks model may occur as the interaction in multiple sequence diagrams depending on the execution scenarios.

Fig. 6 shows an example scenario that was constructed from network of two function blocks *FB_1* and *FB_2*, which for better explanation contains also time characteristics. At the time t_0 function block instance *FB_1* sends an event *request()* to function block instance *FB_2*, which will receive it at the time t_1. Function block instance *FB_2* will perform the event flow processing, execute the algorithm and generate output events in the time interval $\{\{t_1, t_2\}$. Output event *answer()* will be sent by the function block instance *FB_2* at time t_2 and the function block instance *FB_1* will receive it at the time t_3.

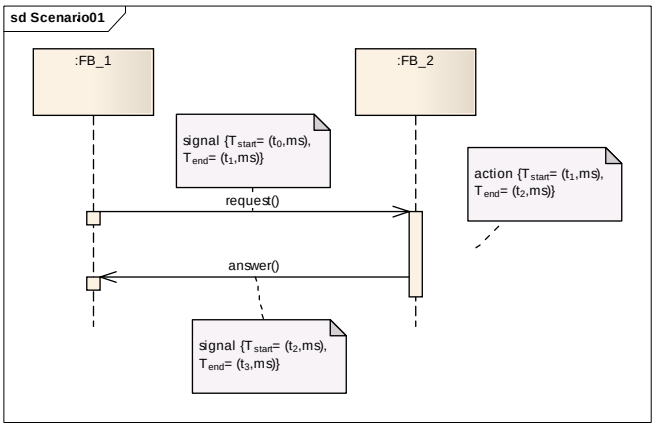

Fig. 6 Scenario example modeled by sequence diagram

C. Interaction Matrix Construction

It is necessary to create a interaction matrix separately for each scenario in this phase and to determine the number of sent interactions (NSI) for each function block, respectively components. The scenario is identified in the upper left corner and particular cells are filled with the NSI values according to sequence diagrams.

Fig. 7 shows an example of function block network, which contains a set of function blocks $FB = \{FB_1, FB_2, FB_3, FB_4\}$ and a set of interactions $I = \{i_1, i_2, i_3, i_4\}$.

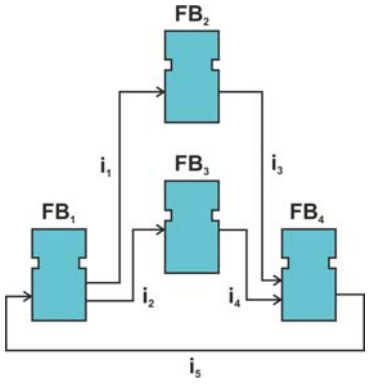

Fig. 7 Function block network example

Two scenarios $S = \{s_1, s_2\}$ were identified during analyzing the software behavior, t. Fig. 8 shows the scenario s_1 and its sequence diagram.

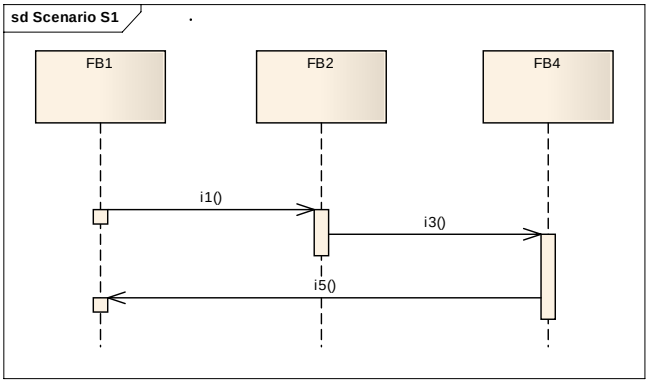

Fig. 8 Example of scenario s1

Fig. 9 shows the interaction matrix of scenario s_1.

s_1	FB_1	FB_2	FB_3	FB_4
FB_1	0	1	0	0
FB_2	0	0	0	1
FB_3	0	0	0	
FB_4	1	0	0	0

Fig. 9 Interaction Matrix of scenario s1

Fig. 10 shows the scenario s_2 and its sequence diagram.

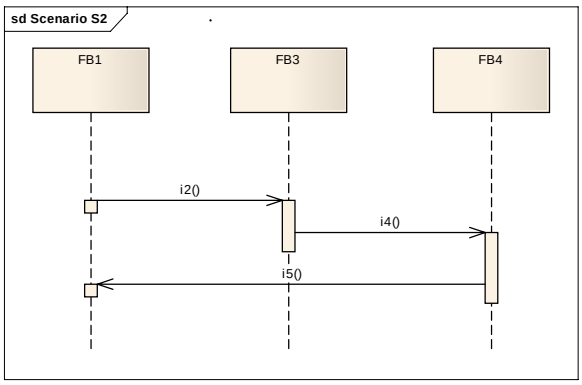

Fig. 10 Example of scenario s2

Fig. 11 shows the interaction matrix of scenario s_2.

s_2	FB_1	FB_2	FB_3	FB_4
FB_1	0	0	1	0
FB_2	0	0	0	0
FB_3	0	0	0	1
FB_4	1	0	0	0

Fig. 11 Interaction Matrix of scenario s2

The NSI values in interaction matrix are important in determining the parameters for further reliability assessment.

III. Definition of Input Parameters to Reliability Assessment

The proposed method for reliability assessment requires defining input parameters as follows:

ISBN: 978-988-19252-4-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

1. Function Block Reliability;
2. Transition Reliability;
3. Transition Probability.

A. Function Block Reliability

Function Block Reliability $\boldsymbol{RFB_i}$ represents ability of function block to perform the required functions under declared conditions for the specified time. The function block reliability value can be determined by various methods from reliability theory:

- Testing,
- Fault injection,
- Reliability growth model.

It is very difficult to estimate reliability of control system software in early stages of software lifecycle, which is modeled by function blocks, because reliability data from system operation are not available. The ready-made function block libraries are very common which are used for application design. Such function block types are pre-tested, so we can assume for reliability assessment purposes that function block reliability is unity.

B. Transition Reliability

Transition Reliability $\boldsymbol{RT_{ij}}$ represents the reliability of transition from one element to another element, so it is probability that the information is correctly transferred from the source function block to the target function block in the direction of execution.

Transition reliability estimation depends on two important parameters: *Interface Reliability* and *Link Reliability*. Therefore: [7]

$$PT_{ij} = \frac{\sum_{k=1}^{|S|} NSI_k(FB_i, FB_j)}{\sum_{k=1}^{|S|}\sum_{m}^{|N|} NSI_k(FB_i, FB_m)} \quad (1)$$

where RI_{ij} is the interfaces reliability of source and target function blocks;
RL_{ij} is link reliability between source and target function block.

Interface Reliability is the probability that two function blocks, which are in interaction, have corresponding interfaces. If there is no interface compatibility, which are in interaction, it may be caused by:

- Mutual incompatibility in structure or sequence of messages which are the matter of the function blocks communication;
- Incompatibility of data formats and message types;
- Replacement of elements roles in the interaction (replacement of source and target function block).

In the proposed method, the interface reliability assessment is the result of correct design of function blocks interfaces which communicate each to other, and this is fully dependent on human factor. For these reasons, the interface reliability is ignored in overall reliability assessment of application.

Link Reliability is the probability that the message is correctly transferred from the source to the target function block. It means that the link reliability depends on the physical connection of function blocks. Especially in a distributed environment it is affected by the appropriate hardware and physical network layer. Typical problems of link reliability are the connection failures, delays or congestion of transmission channels. Analysis of link reliability is beyond the scope of this paper and therefore it will not be further examined.

C. Transition Probability

Transition Probability $\boldsymbol{PT_{ij}}$ is the probability of transition from the function block $\boldsymbol{FB_i}$ to the function block $\boldsymbol{FB_j}$. The transition probability estimation is based on the number of interactions between particular functional blocks. The transition probability is calculates as follows:

$$PT_{ij} = \frac{\sum_{k=1}^{|S|} NSI_k(FB_i, FB_j)}{\sum_{k=1}^{|S|}\sum_{m}^{|N|} NSI_k(FB_i, FB_m)} \quad (2)$$

where $k = 1...|S|$ a $|S|$ is the total number of scenarios;
$m = 1...|N|$ a $|N|$ is the total number of function blocks,
$NSI_k(FB_i, FB_j)$ is the number of sent interactions between function block FB_i a function block FB_j in one scenario k;
$\sum_{m}^{|N|} POI_k(FB_i, FB_m)$ is the number of sent interactions among function block FB_i and all other function blocks (FB_i, ...,FB_i, ..., FB_m) in scenario k,

The sum of transition probabilities of edges from any function block must be unity.

$$\sum_{j=1}^{N} PT_{ij} = 1 \quad (3)$$

In sequence diagrams, which were used to describe the behavior of application, the number of interactions is calculated from interaction matrix of the particular scenarios. The number of interactions in the numerator is equal to the sum of interactions which are between the function block FB_i in sender's role and function blocks FB_j with the role of the receiver in all scenarios, where such an interaction exists.

Transition probability (for example in Figure 12) from the $PT_{FB1 \rightarrow FB2}$ for scenario s_1 is calculated from the NSI value in $Cell(FB_1,FB_2)$, which is in numerator, and the sum of NSI values in $Row(FB_1)$ from all interaction matrixes, which is in the denominator. The NSI value for interaction $FB_1 \rightarrow FB_2$ is null in scenario s_2. For this particular case the transition probability is calculated as follows:

$$PT_{FB_1 \rightarrow FB_2} = \frac{NSI(FB_1, FB_2)}{NSI(FB_1, FB_2) + NSI(FB_1, FB_3)} \quad (4)$$

This means that transition probability depends mainly on the scenarios and also on the number of sent interactions within each scenario.

IV. Reliability Assessment Model

The proposed a model for reliability assessment of the control system software, which is modeled by the function blocks, concerns a specific type of graph - Component Dependency Graph (CDG). This graph is based on control flow graphs, which is classic method for describing the structure, decision points and branching in the code. [5]

ISBN: 978-988-19252-4-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

CDG graph is basically directed graph which describes the dependencies among graph nodes by its edges. Nodes and edges of graph can be rated by one or more parameters depending on the application type. CDG graph is widely used tool by various authors [7] [1] [2] [4].

A. Component Dependency Graph

CDG graph for reliability assessment is probabilistic directed graph, which describes the dependence among the nodes by the edges. The graph nodes and edges are rated by parameters that were mentioned before. CDG graph is based on the possible execution ways among the function blocks according to scenarios.

Definition 11: CDG graph is defined by the tuple $G=\{V, H, s, f\}$, where:

V is the finite set of nodes $V = \{v_i \mid i=1...|V|\}$,

H is the set of directed edges of graph, $E = \{e_i \mid i=1...|E|\}$, $E \subset V \times V$,

s is the start node of graph,

f is the termination of graph.

Definition 12: Node v: $v \in V$ represents function block FB_i and its defined as a tuple $\{NFB_i, RFB_i\}$, where:

NFB_i is the name of the function block FB_i,

RFB_i is the reliability of the function block FB_i.

Definition 13: Function block reliability RFB_i is the probability that function block FB_i execute correctly (faultless) its function.

Definition 14: Directed edge h: $h \in H$ represents a transition of control flow from one function block to another. Directed edge is defined as a tuple $\{RP_{ij}, PP_{ij}\}$, where:

RT_{ij} is the reliability from node v_i to node v_j, respectively from the function block FB_i to function block FB_j,

PT_{ij} the probability of transition execution from node v_i to node v_j, respectively from the function block FB_i to function block FB_j.

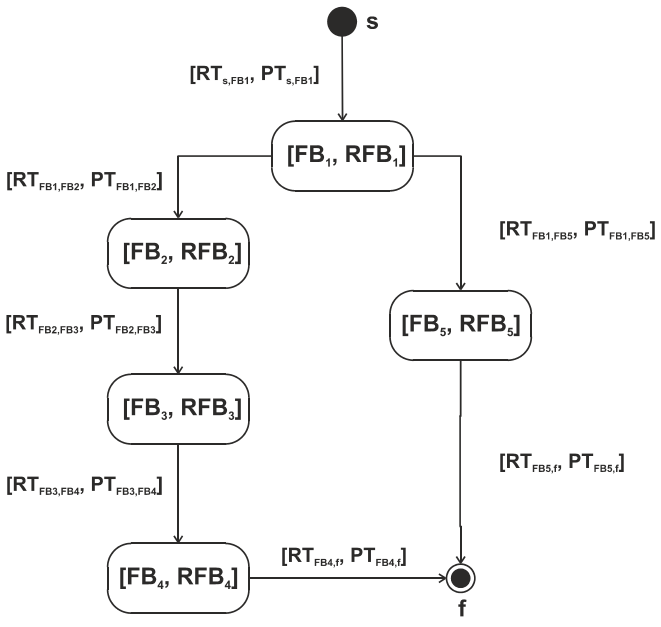

Fig. 12 CDG graph example for reliability assessment

Definition 15: Transition reliability RP_{ij} is probability that the information sent from one functional block FB_i is failure-free received by the second function block FB_j.

Definition 16: Transition probability PT_{ij} is the probability that the execution of required functionality of functional block FB_j is performed immediately after the execution of required functionality of the function blocks FB_i.

Fig. 12 shows the example of control system application, which consists of five function blocks.

B. CDG Graph Construction

The start node of the graph represents the point where signals enter the system, respectively inputs from outside the system. Termination node of the graph is the point where signals exit the system, respectively outputs from the system. Reliability of the start node and termination node is equal to unity.

CDG graph construction algorithm:

1. Identify the functional blocks which are the inputs into the system.
2. Identify the functional blocks which are the outputs from the system.
3. Assign all components from sequence diagrams into the set V, $V \leftarrow \{FB_1, FB_2, FB_3, ..., FB_n\}$, kde $n=1...|N|$ a $|N|$ is the number of functional blocks.
4. Assign into the set W, $W \leftarrow V$.
5. Assign into the w any element from the set W and at the same time remove it from the set W.
6. While the set W is not empty, repeat:
   a. If w is the system input, then add the edge (s,w) into the set of edges E, otherwise
   b. If w is the system output, then add the edge (w,f) into the set of edges E, otherwise
   c. If there is a dependence between $w \in W$ a $v \in V$ in the sequence diagram, then add the edge (w,v) into the set of edges E.
   d. Into w add next element from the set W at the same time remove it from the set W.

Next step is to:

- Assign function block reliabilities RFB_i values to graph nodes,
- Assign transition reliabilities RT_{ij} values to graph edges,
- Assign the transition probabilities (PP_{ij}) for all scenarios using scenario probability and transition probabilities among functional blocks in each scenario according the equation (2).

If all these parameters are set, then it is possible to construct CDG graph according to the earlier definitions.

C. Reliability Assessment Algorithm

The last step after the CDG graph construction is to apply the reliability analysis algorithm. This algorithm incorporates the traversing of directed graph.

The CDG graph traversal algorithm starts from the starting node seeking for all of its descendants. If found child node is not the termination node of the graph, this is serial execution flow and we use multiplication to calculate the reliability. Following these instructions, every path is

ISBN: 978-988-19252-4-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

traversed until the termination node is found. This indicates that either this is the end of the graph traversing or there is still some node in the stack and there is a branching in the graph. If there was a branching in the graph, the algorithm traverse another way in the graph and the calculated value of this parallel path is added to the temporary value of reliability.

Algorithm is described by this pseudo-code:

```
    Input:CDG Graf
    Output: RCelk
 1 Initiation:  RCelk = 0; RTemp = 1;
 2 Begin
 3 s = TStack.Create; {stack creation}
 4 s.push(v1); {adding the first node into the
    stack}
 5 repeat
 6  begin
 7   s.pop(vi,RFBi); {selecting and removing
    the node from stack}
 8   if (vi = f) {if actual node is termation
    node}
 9    then RAll = RAll + Temp
10      else
11       for j:= all neighbours of vi do
12        if EdgeExist(vi,vj) then {for all
    edges from node vi}
13         Begin
14          RTempj = RTempi * RFB(vi) *
    RT(vi,vj) * PT(vi,vj);
15          s.push(vj,RFBj,Rtemp); {add all
    descendants of node V into the stack}
16         End;
17   until s.empty;
18      s.Free;
19    End;
20 End.
```

V. Conclusion

Quality assurance and reliability assessment of technical systems, which include software, is one of the most important but also most difficult tasks of software engineering. The reliability assessment moves to earliest stages of the software development cycle to predict the reliability and identify critical system components. After finding reliability critical software components it is necessary to adopt such design tasks to increase the reliability level by appropriate engineering practices.

Today the automation and control systems software are penetrating from industrial area also to everyday life appliances. Such control systems software becomes still more complex and sophisticated, and it is the challenge to ensure its reliability.

The paper introduced the new approach for reliability assessment of control systems software which can be used in early stages of development. The approach follows the last trends in control systems software using the IEC 61499 standard which provides the model for capturing the behavior and the architecture of control systems software.

The next steps in future work contain the case study elaboration and practical evaluation of the approach.

References

[1] CHANG, J., MA, H. “Modeling the Architecture for Component-Based E-commerce System” in *Proceedings of the 4th international Conference on Formal Engineering Methods: Formal Methods and Software Engineering (October 21 - 25, 2002)*. C. George and H. Miao, Eds. Lecture Notes In Computer Science, vol. 2495. Springer-Verlag, London, 2002, pp.98-102.

[2] CHAUHAN, P., et al. “Automated Abstraction Refinement for Model Checking Large State Spaces Using SAT Based Conflict Analysis” in *Proceedings of the 4th international Conference on Formal Methods in Computer-Aided Design (November 06 - 08, 2002)*. M. Aagaard and J. W. O'Leary (Eds.) Lecture Notes In Computer Science, vol. 2517. Springer-Verlag, London, 2002, pp. 33-51.

[3] DELUGACH, H.S., COX, L.C., SKIPPER, D.J.. Representing Software Component Dependencies Using Conceptual Graphs [Online] Available: http://pdf.aminer.org/000/290/146/representing_natural_language_causality_in_conceptual_grpahs_the_higher_order.pdf

[4] PELIKAN, M., MÜHLENBEIN, H. “The bivariate marginal distribution algorithm” in *Advances in Soft Computing - Engineering Design and Manufacturing (1999)*, London, R. Roy, T. Furuhashi, and P.K. Chawdhry (Eds.), 1999, pp.521-535.

[5] PRESSMAN, R. Software Engineering: A Practitioner’s Approach, 4th ed. New York : McGraw Hill, 1997.

[6] UML. OMG Unified Modeling Language (OMG UML), Superstructure. [online] OMG. Available: http://www.omg.org/spec/UML/2.2/Superstructure/PDF/

[7] YACOUB, S. M., CUKIC, B., AMMAR, H. H. “Scenario-Based Reliability Analysis of Component-Based Software” in *Proceedings of the 10th international Symposium on Software Reliability Engineering (November 01 - 04, 1999)*. ISSRE. IEEE Computer Society, Washington, DC, 1999, s. 22 – 31.

ISBN: 978-988-19252-4-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)