Architectural-Level Risk Analysis FOR UML Dynamic Specifications[1]

Alaa Ibrahim, Sherif M. Yacoub, Hany H. Ammar[1]
Department of Computer Science and Electrical Engineering,
West Virginia University
Morgantown, WV26506-6109
Ibrahim@csee.wvu.edu, yacoub@csee.wvu.edu, hammar@wvu.edu

ABSTRACT

Risk assessment is an essential process in managing software development. Performing risk assessment in the early development phases enhances the resource allocation decisions. Several methodologies for risk assessment were developed, mostly based on subjective judgment. In this paper we use the Unified Modeling language (UML), and a dynamic metrics based methodology developed in earlier work. We use commercial tools such as Rose RealTime modeling and simulation environment to obtain simulation statistics for which an automated architectural-risk assessment environment can be developed. We use Microsoft Excel sheets and Macros in the development of this environment. The dynamic metrics based methodology is a heuristic risk assessment methodology that is based on dynamic complexity factors and severity analysis. Model execution is used for obtaining dynamic complexity and dynamic coupling measures for all architecture elements. Severity analysis is performed using Failure Mode and Effect Analysis. Heuristic risk factor for each architectural component is obtained. A component dependency graph is constructed and traversed to obtain the overall system/subsystem risk factor.

1. INTRODUCTION

The process of risk assessment is useful in identifying complex modules that require detailed inspection, estimating potentially troublesome modules, and estimating testing effort. According to NASA-STD-

[1] This work is funded in part by grants to West Virginia University Research Corp. from the National Science Foundation Information Technology Research (ITR) Program grant number CCR-0082574, and from the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) managed through the NASA Independent Verification and Validation (IV&V) Facility, Fairmont, West Virginia.

This work was performed at the Software Architecture and High Performance Computing Research Lab at West Virginia University under the direction of Dr. Ammar

8719.13A [5] risk is a function of: the possible frequency of occurrence of an undesired event, the potential severity of resulting consequences, and the uncertainties associated with the frequency and severity. The STD-8719.13A standard [5] defines several types of risks, for example, availability risk, acceptance risk, performance risk, cost risk, schedule risk, etc. In this study, we are concerned with *reliability-based risk*, which depends on the probability that the software product will fail in the operational environment and the adversity of that failure.

In this paper we present how the methodology presented in [8] is automated. The methodology is based on:

1. Dynamic metrics: presented by Yacoub, Ammar and Robinson [9] where component complexity and connector coupling factors are derived from simulating scenarios based on the system scenario profile. A brief description is presented in section 1.1.
2. Component Dependency Graphs (CDG): introduced by Yacoub, Cukic and Ammar [10] and adapted by Yacoub and Ammar [8] where a CDG Risk traversal algorithm is presented. A brief description of the CDG and the risk aggregation algorithm is presented in section 1.2.
3. Severity analysis: Based on MIL_STD_1629A where the worst case consequence of a failure is considered, and the severity is determined by the degree of injury, property damage, system damage, and mission loss that can occur. The Failure Mode and Effect Analysis (FMEA) technique is a systematic approach that details all possible failure modes and identifies their resulting effect on the system as presented in the NASA Technical Standard. NASA-STD-8719.13A [5]. In [8] severity indices ($svrty_i$) of 0.25, 0.50, 0.75, and 0.95 were assigned to minor, marginal, critical, and catastrophic severity classes respectively.

The UML-RT model is built and simulated using Rose Real Time, from which log files are made available for extracting the required parameters. We use Microsoft Excel sheets and Macros in the development of the automated environment together with Rose Real Time. The methodology derives heuristic risk factors for components and connectors from dynamic metrics and severity analysis (equation 1), and the system/subsystem overall risk factor is obtained from the traversal of the CDG.

$$hrf_i = cpx_i \times svrty_i \qquad \textit{Eq. 1 (source [8])}$$

where $0 <= cpx_i <= 1$, and $0 <= svrty_i < 1$ are the normalized complexity level (dynamic complexity for components or dynamic coupling for connectors) and severity level for the architecture element respectively (source [8]). The first step in the Risk assessment methodology for dynamic specifications is to derive the complexity factors (component complexity and connector coupling) using simulation and Dynamic Metrics. The next step is to derive severity factors for components and connectors using Failure Mode and Effect Analysis and simulation techniques. Developing heuristic risk factors for components and connectors using equation 1 is the third step. Constructing a CDGs for risk assessment purposes and traversing the graph using the risk aggregation algorithm presented later in this section, is the final step where the product is the system/subsystem overall risk factor.

1.1. Dynamic Metrics

The complex dynamic behavior of many real-time applications motivates a shift in interest from traditional static metrics to dynamic metrics. Active components are sources of errors because they execute more frequent and experience numerous state changes. Therefore there is a higher probability that if a fault exists in an active component, it will easily manifest itself into a failure. For risk analysis at the architecture level, the risks of a failure are the interest. Hence, the motive to assess the complexity of components and connectors as expected at run-time using dynamic metrics, arises.

In the risk analysis, the dynamic metrics defined by Yacoub, Ammar, and Robinson [9] are used to obtain complexity factors for each architecture element. A complexity factor for each component is obtained using

the dynamic complexity metric for the statechart specification of that component. A complexity factor for each connector is obtained using the dynamic coupling metric for the messaging protocol of that connector.

1.2. Component Dependency Graphs

Component Dependency Graphs (CDGs) are introduced in [10] as probabilistic models for the purpose of reliability analysis at the architecture level. CDGs are directed graphs that represent components, component reliabilities, link and interface reliabilities, transitions, and transition probabilities. CDGs are developed from scenarios. One way to model scenarios is using UML *sequence diagrams*. By using sequence diagrams, we are able to collect statistics required for building CDGs, such as the average execution time of a component in a scenario, the average execution time of a scenario, and possible interactions among components. Figure 1 illustrates a simple CDG example consisting of four components, C_1, C_2, C_3, and C_4.

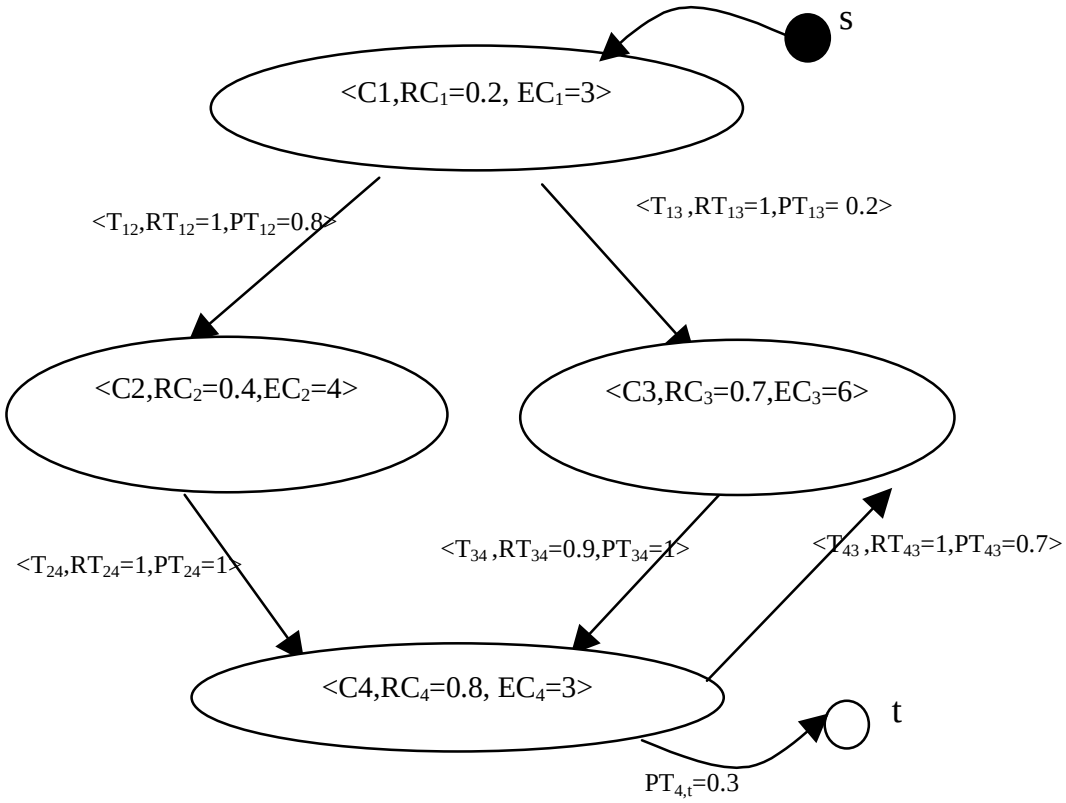

Figure 1 A Sample CDG *1* (source [8])

A CDG is defined as follows:

CDG=<N,E,s,t>; where *N* is set of nodes, *E* is set of edges, and *s* and *t* are the start and termination nodes, i.e. *N = {n}, E ={e},*

$n = < C_i, RC_i, EC_i >$; where C_i is the name of the i^{th} component, RC_i is component reliability, and EC_i is average execution time of a component C_i

$e = <T_{ij}, RT_{ij}, PT_{ij}>$, where T_{ij} is transition from node n_i to n_j in the graph, RT_{ij} is transition reliability, PT_{ij} is transition probability.

1.3. The Risk Analysis Algorithm

The architecture risk factor is obtained from aggregating the risk factors of individual components and connectors. Assuming that a sequence of components are executed, then the risk factor for that sequence of execution is given by:

$$HRF = 1 - \pi_i(1-hrf_i) \qquad \text{Eq. 2(source [8])}$$

After constructing the CDG model, the risk of the application can be analyzed as the function of risk factors of components and connectors using the following risk assessment algorithm (2).

Algorithm
Procedure AssessRisk
Parameters
consumes CDG, AE_{appl},(average execution time for the application)
produces $Risk_{appl}$
Initialization:
$R_{appl} = R_{temp} = 1$ (temporary variables for (1-RiskFactor))
Time = 0
Algorithm
push tuple $<C_1, hrf_1, EC_1 >$, Time, R_{temp}
while Stack not EMPTY do
pop $< C_i, hrf_i , EC_i >$, Time, R_{temp}
if Time > AE_{appl} or $C_i = t$; (terminating node)
$R_{appl} += R_{temp}$;(an OR path)
else
$\forall < C_j , hrf_j , EC_j > \in$ children(C_i)
push ($<C_j, hrf_j , EC_j>$, Time += EC_i , R_{temp} = $R_{temp}*(1-hrf_i)*(1-hrf_{ij})*PT_{ij}$) (AND path)
end
end while
$Risk_{appl} = 1- R_{appl}$
end Procedure AssessRisk

Figure 2 Risk Aggregation Algorithm (source [8])

The algorithm expands all branches of the CDG starting from the start node. The breadth expansions of the tree represent logical "OR" paths and are hence translated as the summation of aggregated risk factors weighted by the transition probability along each path. The depth of each path represents the sequential execution of components, the logical "AND", and is hence translated to multiplication of risk factors (in the

form of (1-hrf_i)). The "AND" paths take into consideration the connector risk factors (hrf_{ij}). The depth expansion of a path terminates when the summation of execution time of that thread sums to the average execution time of a scenario or when the next node is a terminating node.

2. THE AUTOMATED ENVIRONMENT

2.1. Background

UML was explicitly born as an "open" project [3], with the potential of embedding additional notations and tools to satisfy specific design requisites. Along this trace, Rational Software [4](the UML originator) and ObjecTime Limited [2](the Real-Time Object Oriented Modeling "ROOM" originator) collaborated in defining UML for Real-Time [1,6] (UML-RT); an extension of UML optimized for real-time embedded software development. ROOM was introduced to study the dynamic aspects of applications modeled as concurrently executing objects with complex dynamic behavior. ROOM models are intended for simulating the application execution scenarios and complex object behavior. UML specification provides a State Machine package as a sub package of the behavioral elements package. UML state machines formalism is a variant of Harel Statecharts and it incorporates several ROOMcharts concepts and ROOMcharts are a valiant of ROOM modeling language [7]. Dynamic analysis can be conducted on executable design models using several tools such as Rational Rose Real-Time from Rational Software Inc. and ObjecTime Developer from ObjecTime Inc., and hence the dynamic behavior of applications can be verified and assessed. In [6] deriving the set of architectural constructs that integrate ROOM notation in UML were presented. These architectural constructs are derived from general UML modeling concepts using UML extensibility mechanisms. Table 1 provides a summary for these extensions. As a brief description of the basic constructs used in modeling the system structure and component behavior: Three principal constructs; Capsules, Ports and Connectors are used to explicitly describe the system structure. Where in a Capsule collaboration diagram, Capsules and Ports are stereotype roles, and Connectors are association roles. Behavior is modeled using Protocols and state machines. A Protocol specifies the desired behavior over a connector and compromises a set of participants, each participant plays a specific ProtocolRole. A Protocol state machine specifies valid communication sequence and is the standard UML state machine. Capsule behavior is defined in UML state machine where the stereotype (ChainState) is a state that is used in case of transitions that are split into a transition that terminates on the boundary of the state and a transition that propagated into the state (in case of hierarchical state machines).

Metamodel Class	Stereotype
Collaboration	Protocol
ClassifierRole	ProtocolRole
Class	Port
Class	Capsule
State	ChainState

Table 1 Summary of UML Extensions for ROOM, source [6]

Figure 3 shows a Capsule named *Top_Level_Capsule* and its Structure Diagram. The Structure Diagram of *Top_Level_Capsule* contains two Capsules: *First_Capsule* and *Second_Capsule*, each with one port named *Port_1*. *Port_1* in *First_Capsule* is assigned a ProtocolRole *Protocol_1* and *Port_1* in *Second_Capsule* is assigned a ProtocolRole *Protocol_1~*, which is the conjugate of *Protocol_1*. As mentioned earlier a Protocol

defines the flow of messages between ports. Messages are categorized into incoming and outgoing messages. In a conjugated Port the messages defined in the Protocol as incoming messages are defined as outgoing in the ProtocolRole assigned to the Port, and like wise the outgoing messages are defined as incoming messages in the Protocol Role assigned to the Port. A connector connects the two ports and works as a media for message delivery.

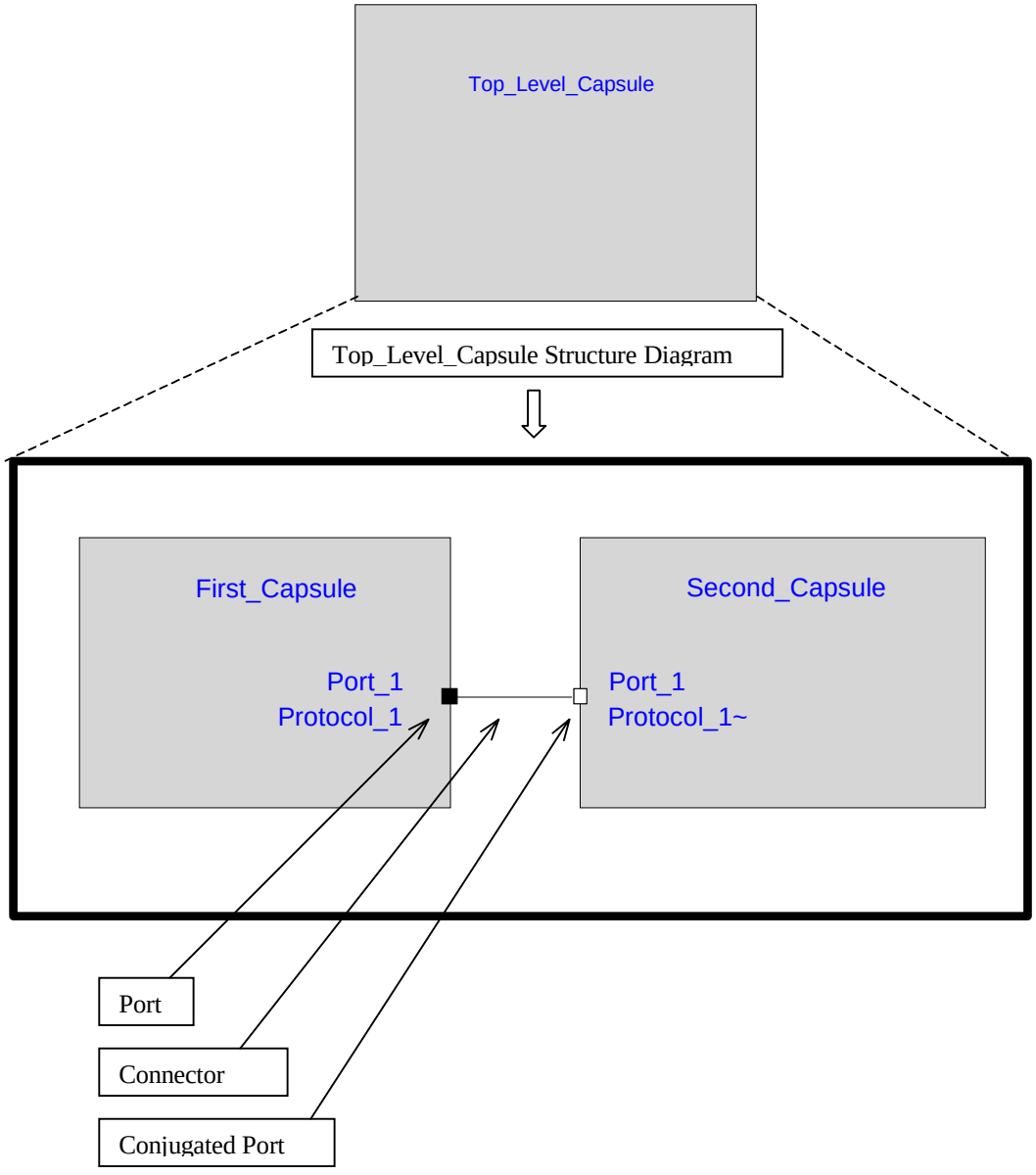

Figure 3 A Capsule (*Top_Level_Capsule*) and its Structure Diagram

Figure 4 shows the State Diagram of *Second_Capsule*. *Second_Capsule* has two states *S_1* and *S_2* and two transition *t_top* and the initial transitions that define the initial state. *S_1* is a macro state that is expanded into another State Diagram shown in figure 5. S_1 has two states and three transition, *t_1*, *t_2* and the initial transition. *t_2* is a transition top a ChainState. Each transition is configured with a message that defines its firing conditions, except transitions from ChainStates like *t_top*.

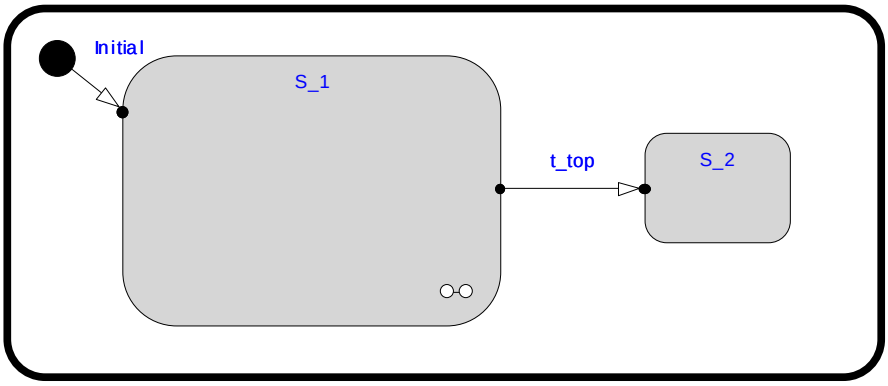

Figure 2.1 State Diagram of *First_Capsule* (top level)

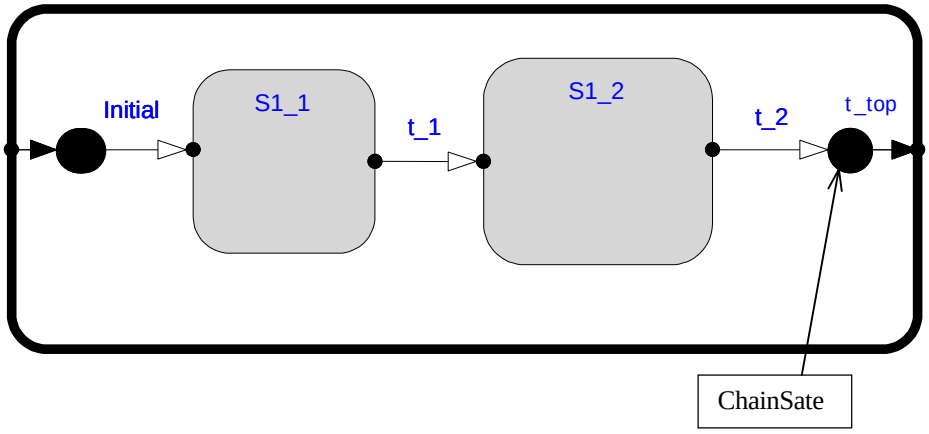

Figure 5 State Diagram of the macro state *S_1*

2.2. Environment

Figure 6 shows a block diagram of the products and processes in the proposed environment for automated risk assessment. Circles and ovals denote inputs/outputs to be processed/produced by the processes and activities shown.

Architecture modeling is performed using the UML simulation environment provided by Rose Real Time In this process the system/subsystem is modeled using the UML-RT artifacts consisting of use-case diagrams, sequence diagrams, Capsule structure diagrams, and state diagrams. The Capsule structure diagrams specify the architecture of the systems based on Capsules, Ports, Protocols, and Connectors. Capsules represent components in the system architecture, Ports are interfaces for Capsules, Protocols specify the messages communicated across Ports, and Connectors represent the media for messages transferred between ports. State diagrams (containing state charts) define the dynamic behavior of Capsules.

The UML simulation environment consists of an Observer Capsule defined as an external observing entity. The Observer component is not part of Rose Real Time; we defined this component in order to facilitate the automation process. This component is not part of the UML model; it is mainly responsible for setting and initiating consecutive simulation runs, detection of constraint/requirements violations and the production of the violation report. These violations represent detected failures during the simulations. The observer is modeled using state charts based on the expected dynamic behavior of the components as depicted in the sequence diagrams.

The analyst provides simulation settings at the start of the simulation. These settings consist of variations for variables that represent timer and delay value for real-time activities on successive runs managed by the observer. They also capture the different settings for the input stimuli that simulate sequences of scenarios. The simulation Log and the violation report produced from the simulation are fed to the analysis tool (Microsoft Excel Macro). The Microsoft Excel Processing Macro analyzes the log file and produces timing diagrams and a violation table. The violation table consists of detected violations or failures and their occurrence time. The timing diagrams are provided to help the analyst identify the severity level of the detected failure in terms of meeting deadlines. The Excel Processing Macro also produces an Excel sheet for normalized component complexity for each component, an Excel sheet for normalized connector complexity for each connector, and an Excel sheet for the CDG. The values hrf_i and hrf_{ij} are identified in a later stage during the execution of the Risk Macro. Severity Ranking is obtained from the severity analysis performed by the analyst using the violation table and timing diagrams as diagnostics for effect analysis and the simulation settings. Feeding the Severity ranking, complexity factors and CDG to the analysis tool (Microsoft Excel Risk Macro), Risk factors for each component and connector are obtained and the CDG is traversed to obtain the system/subsystem overall risk factor HRF.

3. CONCLUSION AND FUTURE WORK

The methodology presented in [8] has the following benefits: it is applicable early at the *architectural-level* and hence it is possible to identify critical components and connectors early in the lifecycle. The methodology uses dynamic metrics that covers the fact that a fault in a frequently executed component will frequently manifest itself into a failure. The methodology is based on *simulation* of UML-RT models. Simulation helps in: performing Failure Mode and Effect Analysis procedures and observing the timing diagrams. The presented automation environment shows how Rose Real Time can be used in fast and efficient deployment of the methodology.

The above methodology and its automation were applied to the Cardiac Pacemaker case study. Yet future research could experiment with applying the methodology to larger case studies with multiple subsystems to compare the aggregated risk factors of individual subsystems.

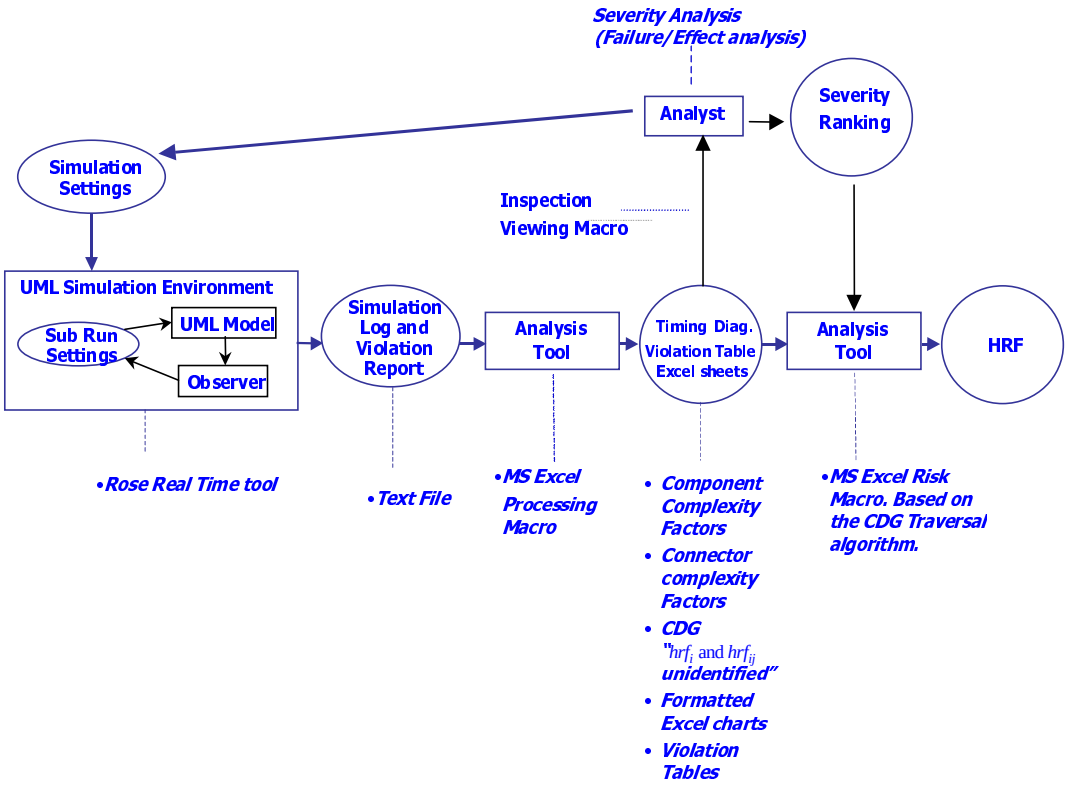

Figure 6 The Automation process-product diagram

4. REFERENCES

[1] Lyons, A. "UML for Real-Time Overview", ObjecTime, Ltd., White Paper. http://www.ObjecTime.com/otl/technical/umlrt.html

[2] ObjecTime Ltd., Kanata, Ontario, Canada, http://www.ObjecTime.com

[3] Object Management Group, Inc., Needham, MA, USA. http://www.omg.org.

[4] Rational Software Corporation, Cupertino, CA, USA. http://www.rational.com

[5] Software Safety, NASA Technical Standard. NASA-STD-8719.13A, September 15, 1997 http://satc.gsfc.nasa.gov/assure/nss8719_13.html

[6] Selic, B. and Rumbaugh, J. "Using UML for modeling complex Real-Time systems", ObjecTime, Ltd., White Paper. http://www.ObjecTime.com/otl/technical/umlrt.html

[7] The Unified Modeling Language v1.3. http://www.rational.com/uml/resources/documentation/index.jsp

[8] Yacoub, S., Ammar, H. "A Methodology for Architectural-Level Risk Assessment using Dynamic Metrics", Proc. of the 11th International Symposium on Software Reliability Engineering, ISSRE'00, IEEE Comp. Soc., October, 2000.

[9] Yacoub, S., Ammar, H. and Robinson, T. "Dynamic Metrics for Object Oriented Designs", Proc. of the Sixth International Symposium on Software Metrics, Metrics'99, Boca Raton, Florida USA, November 4-6 1999, pp.50-61.

[10] Yacoub, S., Cukic, B. and Ammar, H. "Scenario-based Reliability Analysis of Component-Based Software", Proc. of the Tenth International Symposium on Software Reliability Engineering, ISSRE'99, Boca Raton, Florida USA, November 1-4 1999, pp.22-31.