THE GENERATION OF FORMAL SPECIFICATION AND ANALYSIS MODELS OF LARGE-SCALE SOFTWARE SYSTEMS BASED ON COLORED PETRI NETS

K. Lateef, H. H. Ammar, V. Mongulothu, and T. Nikzadeh
Department of Electrical and Computer Engineering
West Virginia University, Morgantown, WV 26506, USA

August 5, 1996

Abstract

This work addresses the problems related to software development and verification of large-scale information systems. The objectives of this work is to develop techniques for generating formal requirement specification and analysis models. These models can be used by analysts to detect potential problems, thus preventing these problems to become part of the design.

The process of developing requirements specifications for a large-scale information system in industry consists of several stages [1]. In the first stage, the system requirements are compiled based on the software system description supplied by the customer. In the second stage, requirements engineering is utilized to develop the requirements specification documents. Requirements engineering is defined in [2] as the process of eliciting, analyzing, and encoding (i.e., documenting) the requirements of the system. These tasks are carried out in a concurrent and iterative manner until a set of adequate, correct, and complete requirements specifications is realized.

Requirements supplication languages (or conceptual grammars for requirements specifications) are classified by Fraser and Kumar in [2] into two major classes: formal specification and informal specifications. Informal specifications mostly used in industry are Structured Analysis [3], or Object-Oriented Analysis [4]. Most commercial Integrated Computer-Aided Software Engineering (ICASE) tools support both of these methods. Informal specification languages use combination of graphics and semiformal textual grammars to describe and specify software systems requirements. These languages by nature are ideal for developers environment as they make it convenient for customer and developer to communicate with each other and refine the customer description to a set of informal requirements documents. One of the drawbacks is that these languages tend to be imprecise and ambiguous.

While informal software specifications are useful in the developer domain, there is clearly a need to use formal specification languages in the requirements analysts domain [5]. Examples of formal specification environments include VDM, Z, and Petri Nets. The colored Petri Nets modeling environments in particular are especially useful in conducting rigorous analysis of dynamic behavioral properties such as concurrency analysis and performability analysis. However, the complexity and rigor of the CPN notation renders model development for large application projects to be a complex and time consuming process.

The work presented here addresses the problem of integrating formal specification and analysis tools based on CPNs (such as Design/CPN) with informal ICASE requirements specification tools (such as Teamwork/SA-RT). A semantics mapping approach is presented. The approach maps hierarchical models developed in the Structured Analysis and Real-Time (SART) notation to formal hierarchical models in CPN notation. The mapping rules in our approach are much more simple compared to those presented in [6] and [7] for High Level Petri Nets. The work presented in [6] and [7] can not be scaled to large systems.

For many years developers have been using informal techniques such as SART for requirements modeling and specifications. Maier [6] has listed several advantages of using Hatley-Pirbhai's SART methodology. This methodology has been used in large industrial projects for many years. Object-Oriented nota-

tions are still evolving and yet to be unified. A unified modeling language is under development at Rational Software Systems, Inc. Which is expected to evolve as a consistent and comprehensive modeling language for object-oriented development.

The semantics mapping approach from SART to CPN is implemented to map models developed using Teamwork/SA-RT to Design/CPN. The implementation is based on the Case Data Interchange Format (CDIF) standard for exchanging information among tools. CDIF is widely supported by most ICASE tools.

A large-scale example is presented to illustrate the utility of this approach. This example is derived from the requirements document of the Flight Operations Segment (FOS) of the NASA's Earth Observing System project conducted by NASA Goddard [8-11].In the following paragraphs, the Commanding component model of FOS is briefly described and simple performability analysis scenarios are discussed.

Description of the FOS Commanding Model

FOS lends itself to a large model. The current version of the SART model consists of 85 diagrams. This includes 48 Data Flow Diagrams (DFDs) used to define the data flow specifications, and 37 State Transition Diagrams (STDs) used to define the control flow specifications. The description of processes in the top level DFD goes to a maximum of five levels deep in the hierarchy.

The Commanding module plays an important role in FOS functions. Commanding has to respond to various signals within a given time limit. Therefore this module becomes a good candidate for detailed verification and analysis. The Commanding function performs four major tasks. These are listed as follows.

1. Generate and verify real-time commands. This is accomplished by the functions $Build_Spacecraft_Realtime_Command$ and $Verify_Command$. This is based on NASA specifications [11] sections 6.5.1.1.4 and 6.5.1.3.4.

2. Merge and uplink the pre-planned and real-time commands to EDOS. The functions $Merge_Command$ and $Transmission_Command$ are responsible for this job. This is based on NASA specifications [11] sections 6.5.1.1.4 and 6.5.1.3.4.

3. Receive and evaluate the command status. This is done by functions $EvaluateSpacecraft_Commands_Status$ and $Receive_Commands_Status_Data$. This is based on NASA specifications [11] sections 6.5.1.1.4 and 6.5.1.3.4.

The automatic retransmission is also provided when an unsuccessful transmission occurs. This is managed with the help of the function $Count_Transmission_number$. This is based on NASA specifications [11] section 7.3.4.2.

In order to fulfill the above tasks, $Commanding$ is decomposed into several functions. These functions are listed below:

1. $Build_Spacecraft_Realtime_Command$
2. $Verify_Command$
3. $Merge_Command$
4. $Transmission_Command$
5. $Evaluate_Spacecraft_Command_Status$
6. $Receive_Command_Status_Data$
7. $Count_Transmission_number$

Out of the seven functions, three were more complex requiring lower level DFDs and STDs. These are $Build_Spacecraft_Realtime_Command$, $Verify_Command$ and $Evaluate_Spacecraft_Command_Status$.

The Commanding module specification is automatically translated to a Design/CPN model. The hierarchy of the CPN model of commanding consists of three levels. The CPN model has a total of nine pages with components haveing a one-to-one correspodance with the components of the SART model.

Performance and Performability Analysis

T he CPN model can capture both static and dynamic modeling aspects of the system characteristics. This is specially true in the early design stages when functional modules are relatively large and where knowledge of their execution behavior may be imprecise. As the design progresses and the modules detailed designs are further resolved, the estimates of their behavior and execution resource characterization become more precise. A CPN model gives the definition and the order of execution of the various functional components.

System execution scenarios which provide the definitions of the external inputs to the model are needed for each simulation of the model. Performance measures for the total system and components are obtained. Simulation of the system is performed to view its performance characteristics.

The following scenarios can be identified to assess the performance and/or Performability of different designs of the commanding process model.

1. Performance under normal sequential execution: In this senario, the CPN model of commanding is simulated for $Operator_Command_Input$. This input will activate the $Builds_Spacecraft_Realtime_Command$ (BSRC) function where the command is successfully built and validated. The $Verify_Command$ function will then verify the authority of the command and produce a $Valid_Realtime_Command$. This is merged with a $Valid_Preplanned_Command$ obtained from the $Integrated_Command_Load$ input. The merged commands form an $Uplink_Data_Stream$ which is uplinked to the space craft through the successful execution of the $Transmission_Command$ function. The status data from the space craft is received and evaluated to indicate a successful reception as the $Spacecraft_Commands_tatus$. This will in turn be used by the $EOC_Controller$ to activate BSRC to build and validate the next $Operator_Command$ input. The simulation of this scenario produces measures on the throughput and total execution time of the operator commands. Utilization measures for the resources such as the Project Database (PDB), and the uplink communication channel can also be obtained. The bottle-neck function can also be identified.

2. Performance under pipelined normal execution: In this case a sequence of operator commands are pipelined through the system. Several functions are concurrently active to process the command sequence. The performance improvement under this pipelined design can be accessed.

3. Performability analysis of the pipelined design: In this case, the failure and repair activities of the system functions are also simulated. This is accomplished by assigning failure probabilities to the system functions such as BSRC and $Verify_Command$. These failure probabilities are used in the code segments associated with the respective transitions to simulate the activity of a failure with which an estimated repair time is attached. The performance of the pipelined design under failures and repairs can then be assessed in the simulation.

References

[1] Olle, W.T., et al "Information systems Methodologies: A Frame work for understanding," Addison-Wesley 1988.

[2] Fraser, M.D., K. Kumar "Informal to formal requirement specification language: Bridging the gap," IEEE Trans. On Sofware Engineering, Vol.17, No. 5, May 1991, pp 454.

[3] Yourdon, E. ,Modern Structured Analysis, Printice-Hall, 1989.

[4] Rumbaugh, J., et al, Object-Oriented Analysis, Printice-Hall, 1991

[5] Miriyala K., and Harandi M. T. "Automatic Derivation of formal software specification from informal description," IEEE Transaction on Software Enginering, Vol 17, No. 10, October 1991, pp 1126-1142.

[6] Pezze, M. At al "Giving Semantics to SA/RT by means of High-Level Timed Petri Nets," The Internationa Journal of Time Critical Computing Systems, Vol. 5, No. 2/3, May 1993.

[7] Ghezzi et al, "High-Level Timed Petri Nets as kernel for executable specifications," The Internationa Journal of Time Critical Computing Systems, Vol. 5, No. 2/3, May 1993.

[8] EOS Ground System Architecture Description Document, April 30, 1993. By Systems Engineering and Security, Inc.

[9] Functional and Performance Requirements Specification for the Earth Observing System Data and Information System (EOSDIS) Core System. Revision A and CH-01.

[10] ECS Operations Concept Document for the ECS Project, August 1993. By Hughes Applied Information Systems, Inc.

[11] Flight Operations Segment (FOS) Requirements Specification for the ECS Project, Volume 1: General Requirements, November 1994. By Hughes Applied Information Systems.