System-Software Interfaces for Safety-Related Digital I&C Systems

N. Kececi, C. Smidts, M. Modarres & Y-S. Hu
University of Maryland, Center for Reliability Engineering MD USA

ABSTRACT: The purpose of this paper is to discuss a structured software development process based on a graphical method which relates functional requirements and software specifications to the detailed software design and implementation. This paper presents a systematic logic-based method. The approach may be used to model the functional specifications of digital instrumentation and control systems used for safety purposes. The graphical representation of functional requirements is depicted through a multilevel hierarchical decomposition technique which allows one to (i) show functional interrelations between system and software (ii) map to follow functional information from the system level to the software implementation (requirement, design, code), (iii) address incomplete, inconsistent, and ambiguous requirements.

1 INTRODUCTION

Digital Instrumentation and Control (I&C) technology is expected to enhance the safety and performance of nuclear power plants by offering process control improvements, such as reduced instrument calibration requirements and improved plant condition monitoring displays [Gill et al.1994]. These systems also incorporate self-testing capabilities for reduced maintenance requirements. On the other hand, digital control systems can introduce new failure modes [Thadani et al. 1993].

Software in nuclear power plants, even in safety-critical applications can differ substantially in its functionality and its complexity. Nuclear power plant software spans the range from a programmable system controller implementing one simple function or operation to a digital shutdown system controlling multiple devices and responsible for the detection of transients and the activation of shutdown process. Accordingly, assuring quality for software in nuclear power plants must address a wide range of applications and must fully account for the system level interactions, which exist.

In a nuclear power plant safety system, software will perform two types of functions. The first type is concerned with performing the plant safety functions themselves. These safety functions are identified at the system level and flow down through the software requirements, design and code. The second type is concerned with maintaining the integrity of the system and software elements that perform the primary safety functions, such as error checking and fault tolerance. These integrity functions are partially identified at the system level, but are significantly expanded within the software itself.

There are two general approaches to assuring software quality/reliability: (1) controlling (reviewing) the software development process and (2) verifying the end product. Neither of the classic approaches can produce a highly reliable software product, as noted in a National Research Council [NRC 1995] report. For software, the reliability and quality assurance responsibility is known as software Verification and Validation (V&V or Independent V&V).

Several studies have shown that about 50% of software errors are introduced early in the software development. The later those software errors are detected, the more costly they are to correct. Lutz [1992] has reported results of the study of faults detected during the integration/testing of Voyager and Galileo spacecraft as following: (1) Very few serious errors were introduced during the later stage of the software life cycle. The primary sources of

catastrophic failures are faults in the requirement specification. (2) 197 faults were characterized as cause of catastrophic failure; of these, 3 were coding errors and 194 were traced to problem in the specification of functions and interfaces.

It is clear that correct requirements are necessary for reducing cost and increasing the quality and reliability of software. Moreover, we need to develop mechanisms to map the system level information to the software requirements/design/code, i.e., which system requirements are related to which software requirements. From this perspective, identification of system/software functional relationships to resolve the problems highlighted in this introduction is the purpose of this paper.

2. SOFTWARE REQUIREMENT ANALYSIS TECHNIQUES

Two types of software development specifications exist. The first is the statement of the user's needs in documents referred to as a requirement specification. Clearly, the users must validate these documents since only they know what they want. The software developers draw up the second set of specifications, technical document, which restates the requirements in a form meaningful to the software developers.

Most software projects arise from requirements expressed in natural language, either orally or textually. If these requirements are expressed textually in a requirement specification, then a number of validatory processes can take place. First, the documents can be checked for correct spelling and grammar. Many variants of ambiguity can be detected automatically [Mander 1980]. Experience has shown that some of the reasons why more errors tend to occur in the requirements phase are as follows: (1) Lack of a common terminology and use of natural language: There is no known way in which details of functionality can be extracted from natural language text with any degree of certainty, as noted in the Hennell (1987) study. Contrary to the views expressed in many books [DeMarco 1979, Jackson 1983], functions cannot be deduced necessarily or exclusively from the use of verb. (2) Incomplete and incorrect requirements specifications. (3) Ambiguous and inconsistent requirements specification. In general, no method can be certain to detect occurrences of (2) and (3). The reason for this is that under a particular interpretation inconsistency or consequential errors may not exist. There also appears to be no way to identify those aspects, which have been forgotten, or were not understood as reported by Wingrove (1987).

To deal with inconsistent and incomplete requirements, many approaches to software requirement analysis have been developed over the last few years. According to a survey and assessment of conventional software V&V methods [Groundwater et al.1995], requirements and design techniques consist of four major classes and various subclasses. These major classes of techniques and the total number of individual techniques are as follows:

1. Formal methods are based on a translation of requirements into mathematical form. Eight different techniques were discovered.
2. Semi-formal methods are based on the expression of requirement specifications in a special requirement language. Eleven different individual techniques were discovered.
3. Reviews and Analysis (informal method) are based on reviews by special personnel of the adequacy of the requirement specification according to a pre-established set of criteria and detailed checklists and procedures. Seven different techniques were identified.
4. Requirements Tracing and Analysis Techniques are based on matching of each unique requirement element to design elements and then to the elements of the implementation. Two different techniques were identified.

Formal methods involve mathematical and logical calculations for expressing relationships among data and other objects and the processes, which act upon them. But mathematical verification of requirements does not seem to greatly simplify development.

The Semi-formal methods are less difficult to apply than the formal methods. They often involve rigorous constraints on notations, sequencing, and selection of operator/objects to achieve the goal of guiding the analysis or specification within well defined boundaries. Their major advantage is the philosophy of supporting system-engineering descriptions in a graphical mode, a characteristic that greatly facilitates simulation or animation of requirements and design. But formalization itself cannot guarantee error detection, nor can it prove that the requirement specification is correct. Testing a specification will not find all possible errors.

The Traceability Assessments establish existence (or the lack thereof) of mapping relationships between requirements, design, and coding. The adva n-

tages of the traceability assessments are reported by Groundwater et al. [1995]. These assessments help detect unintended functions and omission, incomplete and incorrect requirements. Detection of unintended functions is important from a safety viewpoint. The non-specified additional functions might lead to unexpected errors and /or safety problems.

It is clear that functional relationships should be addressed at the system level. It is equally clear that traceability analysis is necessary, from system functional requirements to software specifications to detailed design and to implementation.

3.PROPOSED APPROACH

3.1 *A functional modeling framework: goal tree success tree and dynamic master logic diagram –GTST-DMLD.*

Interactions between the basic elements of almost all-physical systems are very complex. For example, the objective of cooling a nuclear reactor core during an accident involving loss of reactor coolant can be achieved through the so-called emergency core cooling systems (ECCS). However, to attain the objective of cooling the reactor core requires harmonious operation of a number of interacting components (the components that are 'part-of' the ECCS) an accident involving loss of reactor coolant can be achieved through the so-called emergency core cooling systems (ECCS). To attain the objective of cooling the reactor core requires harmonious operation of a number of interacting components (the components that are 'part-of' the ECCS). These components perform sub-functions to attain the overall objective of providing adequate cooling. Additionally, this equipment may require support elements (in the form of auxiliary functions provided by people, software and other equipment/functions) to cool, turn on and off, and power ECCS equipment/functions. These supporting elements form another, complementary hierarchy (i.e., the support hierarchy)

Modeling complexity based on hierarchies has been around for many years. GTST-MLD modeling is a functional decomposition framework to describe and model complex physical systems in terms of objects, relationships, and qualities. GTST-DMLD has been used to represent and model all-important modules of system diagnostic software [Modarres et al. 1999] and have been proposed to model the software development life cycle [Kececi et al. 1998]. An example of a partial GTST-MLD model representing the effects of a 'loss of offsite power' event in a nuclear power plant is shown in Figure 1. The lowest level of this tree shows the systems that perform the physical functions. These systems have many redundancies and interdependencies

3.2 A graphical approach to integration of system/software specifications.

This study is an application of the GTST-DMLD modeling to the integration of system functional requirements and software integrity functions. A graphical representation of system functional requirements using the GTST-DMLD framework, as shown in Figure 2, may help to identify interrelationships between system and software specifications.

The approach proposed is a five-step process:

- Step 1. Software Requirement Collection and Grouping: Requirements are collected from the system specification and are grouped dependent upon the system's goals and functions. After that, functional requirements are classified into two groups describing respectively the main functions and the support functions.

- Step 2. Requirement Decomposition: Main and support functions are decomposed hierarchically into sub-functions.

- Step 3. Define the relationships*:* The relationships in the hierarchies show a connection between different nodes of a hierarchy or between nodes of two different hierarchies. The relations can be characterized as logical, physical or fuzzy (this is not to say that these are all the categories of relationships in a system). Each type is explained below.

    1. Logical (Boolean) connectivity relationships: Logical relations are used to show the redundancy and connectivity between various nodes (objects, functions, behaviors, goals and classes). In a logical relationship the states of the input and output nodes are binary. In this case the nodes can either take a binary value of 0 or 1.
    2. Physical connectivity relationships: A physical relation refers to node relations that are described by some physical laws, and are mostly represented by a continuum of values as opposed to binary values in the case of logical relations. Accordingly, physical relationships are analog in nature.

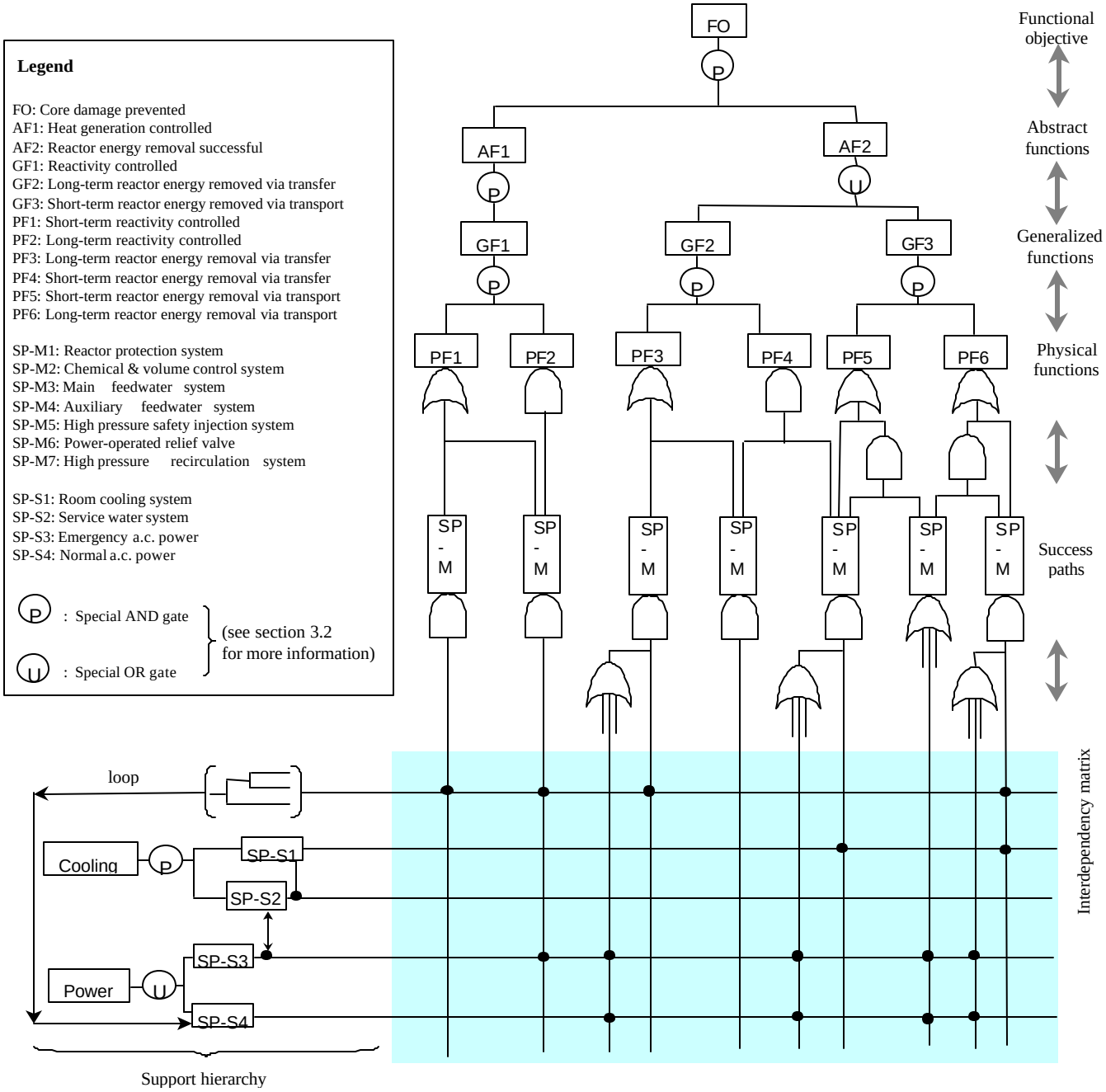

Figure 1 A partial GTST-DMLD Model for a Nuclear Power Plant

3. Uncertain (fuzzy) connectivity relationship: When the relationships are not fully known, a physical-based description is not available, or if available is uncertain. In this case, a fuzzy relation may be most appropriate. More detailed work on the fuzzy relations has been described in Hu Y.S et. al (1994).

- Step 4: Define the logical operators and/or physical macro functions: By using the GTST-DMLD based computer tool, we are able to create different types of macro functions and operators to represent the instrumentation and control system rules. Some examples of macro functions are as follows: math functions, data/time functions, string functions, aggregate functions, data type conversion functions, array functions, system functions, graph functions, hierarchy functions and database functions. The operators in an expression describe what type of action the expression should perform, or how the expression should compare or relate two values. Some examples of operators are Arithmetic and Text Operators, Logical Operators, Comparison Operators, Conditional Operators, Loop Control Operators. Logical, physical and fuzzy gates in the hierarchies can be customized and configured by the users. This flexibility allows us to define, modify and delete logic gates used in defining relationships.

- Step 5: Translation: Translate the natural language requirements into equivalent prepositional expressions using the definitions in step 4.

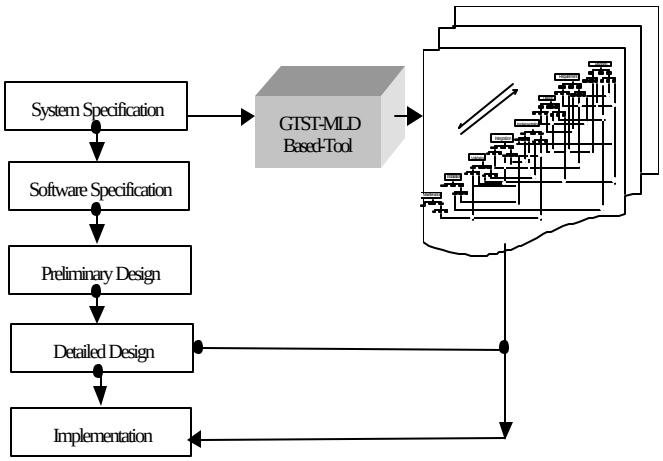

Figure 2. A graphical approach to integration of system/software specifications

3.2 Tools for Analysis

There are two software tools that are currently used for modeling GTST-DMLD models. These are (1) REVEAL-W [Scientech, 1997] and (2) DML-US 98 [Hu Y-S et al.1999].

These tools are designed to help the users build the GTST-DMLD and use it for analysis. Both of these tools are commercial products and have been used extensively for a broad range of applications. While these tools are very helpful, the total approach described in this paper requires additional tool development fully to automate the process.

4. CONCLUSIONS AND FUTURE WORK

This paper has discussed a new graphical technique to integrate the system/software functional requirements, especially for safety-related digital control systems. Using a GTST-DMLD, this paper argues, leads to the following advantages:

(1) After decomposition, each node will only be associated with a limited number of rules. Experts may organize rules hierarchically instead of considering the whole complex system at the same time. Thus, the errors introduced early in the life cycle can be avoided.

(2) A graphical presentation can help identify functional interrelationships between system and software. This can minimize the generation of faults between development life cycle interfaces.

(3) Using a graphical presentation instead of natural language to describe the requirement specification can reduce inconsistencies, incompleteness, ambiguities and better reveals requirements not specified.

(4) It is easy to trace the functional requirements from system level to implementation and vice versa. The GTST-DMLD model and tool can be used as a traceability analysis tool.

In future related work, we will apply the methodology to the Generic Westinghouse Nuclear Reactor Protection System and its software requirements.

7. REFERENCES

1. DeMarco T. 1979. *Structured Analysis and System Specification.* Prentice-Hall.Jones C.B. (1980) *Software Development.* Prentice-Hall.
2. Groundwater E.H., Miller L.A., Mirsky S.M. 1995. *Guidelines for the Verification and Validation of Expert System Software and Conventional Software. Survey and Document of Expert System Verification AND Validation Methodologies* NUREG/CR-6316, SAIC-95/1028., Vol 1-7.
3. Holbert, K., Harmon, D., Rozek, T., and Wilkosz, S. 1994. Nuplex 80+ Advanced Control Complex: Enhanced Safety through Digital Instrumentation and Control. 9th *Annual KAIF/KNS Conference*, April 6-8, 1994.
4. Hennell M.A.1987. *Requirements, Specification and Testing. Software Reliability Achievement and Assessment, Edited by B. Littlewood.* Blackwell Scientific Publication.
5. Hu Y-S, Modarres M.,1994. *Time-dependent system knowledge representation based on dynamic master plant logic diagram.* Proceedings of 2nd IFAC Workshop on Compute Software Struct. Integ. AI/KBS Syst. In Proc Cont, Lund, Sweeden August 1994.
6. Hu Y-S, Modarres M., 1999. *Applying Fuzzy-Logic-Based Hierarchy for Modeling Behaviors of Complex Dynamic System*. System and Software Computing in Nuclear Engineering, Da Ruan ed., Springer-Verlage (in Print).
7. Jackson M.A. 1983. *System Development*. Prentice-Hall.
8. Kececi N., Modarres M. 1998. Software Development Life Cycle Model to Ensure Software Quality. *Proceeding of the 4th International Conference on Probabilistic Safety Assessment and Management*, New York City, USA.
9. Lutz R., 1992. Analyzing Software Requirement Errors in Safety-Critical Embedded Systems. *TR92-27*, Iowa State University
10. Mander K.C.& Presland S.G. (1980) *An Introduction to Specification Analysis*-SPAN.SCM Dept, University of Liverpool Report.

11. Modarres M. 1993. *Functional Modeling of Complex Systems Using a GTST-MLD* Framework. *Proceeding of the 1st International Workshop of Functional Modeling of Complex Technical Systems*, Ispra, Italy.
12. National Research Council (NRC), 1995. *Digital Instrumentation and Control Systems in Nuclear Power Plants: Safety and Reliability Issues, Committee on Application of Digital Instrumentation and Control Systems to Nuclear Power Plant Operations and Safety,* National Academic Press, Washington, DC.
13. Scientech, 1997. REWEAL-W –TM User's Manual, Scientech Corporation, Rockville, MD (www.scientech.com)
14. Thadani A.C., Perch R.L.1993 Consideration of Important Technical Issues for Advance Light Water Reactors. *Proceeding of the 2nd ASME/JSME Joint Conference,* San Francisco, CA.
15. Wingrove A. (1987) *Software Failures are Management Failures.* Center for Software Reliability The City University, London ECIV 0HB