A Technique of Software Safety Analysis in the Design Phase for PLC Based Safety-Critical Systems

Seo-Ryong Koo and Chang-Hwoi Kim
I&C/HF Research Division, Korea Atomic Energy Research Institute
111 Daedeok-daero 989beon-gil, Yuseong-gu, Daejeon, 34057, Korea
**Corresponding author: srkoo@kaeri.re.kr*

1. Introduction

As a software verification and validation should be performed for the development of PLC based safety-critical systems, a software safety analysis is also considered in line with entire software life cycle [1]. The purpose of safety analysis, which is a method of identifying portions of a system that have the potential for unacceptable hazards, is firstly to encourage design changes that will reduce or eliminate hazards and, secondly, to conduct special analyses and tests that can provide increased confidence in especially vulnerable portions of the system. A hazard is a state or set of conditions of a system (or an object) that, together with other conditions in the environment of the system (or object), will lead inevitably to an accident (loss event) [2]. The main hazards in a nuclear reactor are the possibility of a rapid, energetic fission reaction and the release of radioactive fission products, which are waste products of the fission reaction.

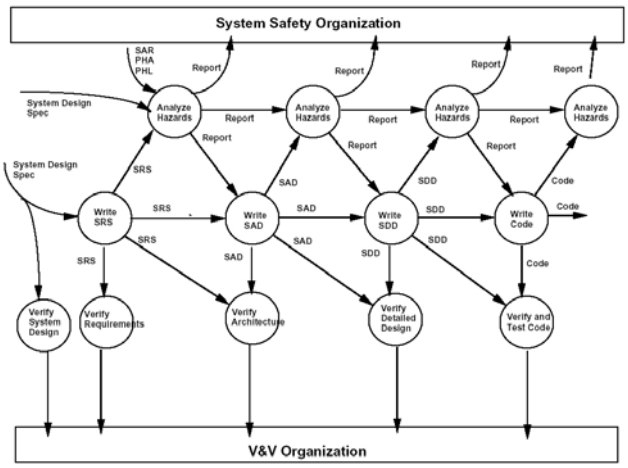

Fig. 1. Software Hazard Analysis within the Software Life cycle.

In the development of NPP systems, software hazard analysis should be a defined aspect of the software life cycle. Fig. 1 shows the general approach with respect to the technical development activities (namely, the requirements, architecture, design, and code), as well as the activities of V&V and hazard analysis. Although V&V is not considered part of hazard analysis, the results of V&V may be of use. For instance, we used the results of various software hazard analyses to appropriately change the design of the protection system and architecture of the software, and to identify the portions of the software that required increased attention to quality. As shown in Fig. 1, in line with the software life cycle, software hazard analysis consists of requirements hazard analysis, design hazard analysis (actually, it is divided into architectural design and detailed design), and code hazard analysis.

Currently, in NPP Instrumentation and Control (I&C) systems, a Programmable Logic Controller (PLC) is being considered for safety-class hardware. In this study, for the PLC-based safety-critical software, we newly propose a software design safety analysis technique based on the NuFDS (nuclear FBD-style design specification) approach. The NuFDS approach is composed of a software design specification technique and a software design analysis technique.

2. Fault Tree Synthesis

Several procedures have been proposed for the automatic synthesis of fault trees. In the synthesis, the fault tree is built by matching the inputs and outputs of mini-fault trees. We now describe a technique for generating fault trees from the architecture specification of the NuFDS approach. To systematically translate the NuFDS specification into fault trees, we define a template for constructing fault trees and we used the template to intuitively compose the architecture specification [3].

This section describes how to generate a fault tree from the architecture specification of the NuFDS. As an example, we used a bistable processor (BP) of the KNICS digital plant protection system [4]. The KNICS digital plant protection system has three subsystems: a reactor protection system (RPS), an engineered safety features-component control system (ESF-CCS), and an automatic test and interface processor (ATIP). The RPS, which comprises a bistable processor (BP) and a coincidence processor (CP), uses trip logic to ensure that the plant can be safely shut down in emergency situations; for example, when there is an increased reactor temperature or loss of coolant. The ESF-CCS, which has similar features to the RPS, attempts to reduce the influence of other reactor accidents. The ATIP, on the other hand, periodically tests the RPS to ensure the ongoing safety of the plant. Of the components of the KNICS digital protection system, the BP is the one we selected to exemplify the fault tree

synthesis. The fault tree synthesis comprises the following four steps:

Step 1: Define top event of fault tree.

The first step of the fault tree synthesis is to define the highest level of the architecture module as a top event of the failure mode. In Fig. 2, an architecture design block is used to show a software architecture specification of the BP. As shown in Fig. 2, the top event is simply defined as 'FAILURE OF RPS_BP1', which means the failure mode of the BP in channel 1 of the reactor protection system.

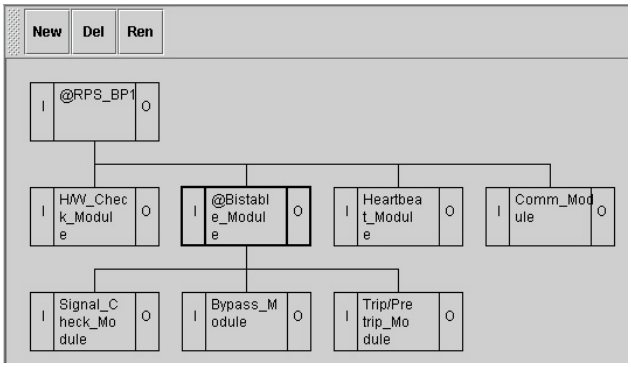

Fig. 2. Software Architecture Specification of BP.

Step 2: Compose intermediate events.

In the FTA, the intermediate events are pseudo events between the top event and the leaf nodes in the tree. For our fault tree synthesis, the second step composes these intermediate events directly. Fig. 3 shows the results of the second step. This fault tree is first transformed from the architecture specification shown in Fig. 2.

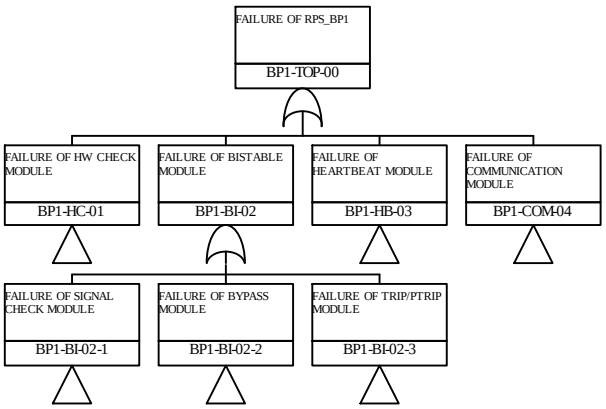

Fig. 3. Composition of Intermediate Events in Step 2.

Step 3: Transform basic events into template.

The third step is to define basic events and to transform those basic events into the template for the fault tree synthesis described in [3].

If there are any events that have no more sub-modules, those modules are considered as basic events in the fault tree. In Fig. 2, the basic events in the fault tree are *H/W_Check_Module*, *Heartbeat_Module*, and *Comm_module*, which are the lower level modules of RPS_BP1, along with *Signal_Check_Module*, *Bypass_Moudule*, and *Trip/PreTrip_Module*, which are the lower level modules of Bistable_Module. These basic events should have the failure modes defined in [3] as a template. In step 3, therefore, the basic events should be directly transformed into the template for our proposed fault tree synthesis. Fig. 4 shows the completed fault tree transformed from the architecture specification of the BP.

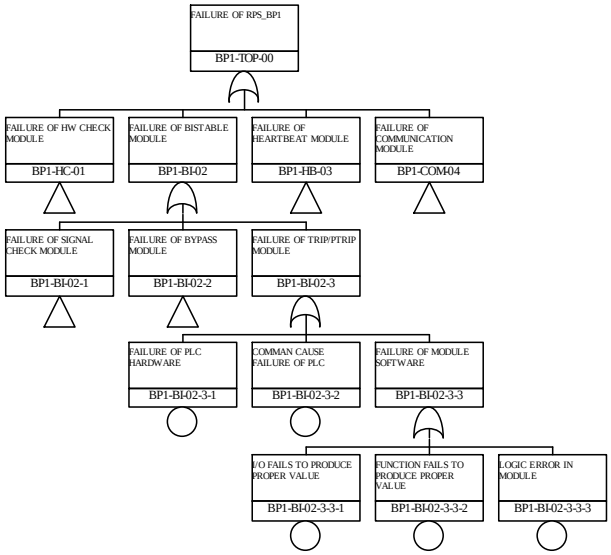

Fig. 4. Transformed Fault Trees.

Step 4: Software design safety analysis.

The final step of the fault tree synthesis is a software design safety analysis on the basis of the generated fault trees. In this step, both qualitative and quantitative analyses are possible with the aid of the generated fault trees. However, we focus on the qualitative analysis in this study because the data on the failure probability of the software modules for the quantitative analysis is not yet well defined. The basic purpose of the qualitative analysis is to reduce the tree to a logically equivalent form that shows the specific combinations of basic events sufficient to cause the top event. The final goal of the analysis is to find the minimal cut sets. These cut sets, which cannot be reduced in number, represent the basic events that will cause the top event—that is, a cut set does not contain another cut set. Moreover, cut sets are defined in such a way that if even one event in the cut set does not occur, the top event will not occur.

3. Conclusions

For the design and implementation phase of the PLC based systems, we proposed a technique for software design specification and analysis, and this technique enables us to generate software design

specifications (SDSs) in nuclear fields. For the safety analysis in the design phase, we used architecture design blocks of NuFDS to represent the architecture of the software. On the basis of the architecture design specification, we can directly generate the fault tree and then use the fault tree for qualitative analysis. Therefore, we proposed a technique of fault tree synthesis, along with a universal fault tree template for the architecture modules of nuclear software.

Through our proposed fault tree synthesis in this work, users can use the architecture specification of the NuFDS approach to intuitively compose fault trees that help analyze the safety design features of software. A translated fault tree can help users to easily identify software hazards; it also helps them to find software weaknesses by analyzing cut sets. Consequently, we can analyze the safety of software on the basis of fault tree synthesis.

REFERENCES

[1] U.S. NRC, NUREG-6430: Software Safety Hazard Analysis Version 2.0, UCRL-ID-122514, 1995.
[2] Nancy G. Leveson, SAFEWARE – System Safety and Computers, Addison-Wesley Publication Company, New York, 1995.
[3] Seo Ryong Koo and Poong Hyun Seong, “Software design specification and analysis technique (SDSAT) for the development of safety-critical systems based on a programmable logic controller (PLC)”, Reliability Engineering & System Safety, Volume 91, Issue 6, Pages 648-664, June 2006.
[4] Chang-Hwoi Kim, Joo-Hyun Park, and Dong-Young Lee, “Development of a Safety I&C Systems for NPP”, Transactions of Korean Nuclear Society Meeting, 2005.