# The verification methodologies for a software modeling of Engineered Safety Features–Component Control System (ESF-CCS)

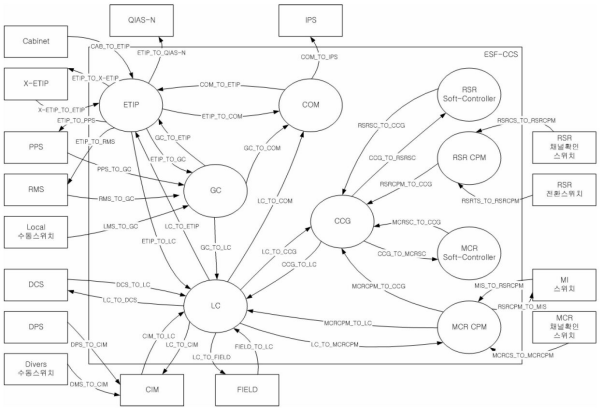
Young-Jun Lee, Se-Woo Cheon, Kyung-Ho Cha, Gee-Yong Park, Kee-Choon Kwon
*Korea Atomic Energy Research Institute (KAERI)*
*P.O. Box 105, Yuseong, daejeon, Korea*

## 1. Introduction

The safety of a software is not guaranteed through a simple testing of the software. The testing reviews only the static functions of a software. The behavior, dynamic state of a software is not reviewed by a software testing.
The Ariane5 rocket accident[7] and the failure of the Virtual Case File Project[8] are determined by a software fault. Although this software was tested thoroughly, the potential errors existed internally.
There are a lot of methods to solve these problems. One of the methods is a formal methodology. It describes the software requirements as a formal specification during a software life cycle and verifies a specified design.
This paper suggests the methods which verify the design to be described as a formal specification. We adapt these methods to the software of a ESF-CCS (Engineered Safety Features-Component Control System) and use the SCADE (Safety Critical Application Development Environment) tool for adopting the suggested verification methods.

## 2. The interface of the ESF-CCS software

ESF-CCS is a system which operates Engineered Safety Features System Components after receiving an initial signal of a ESF from a Plant Power System (PPS) and Radiation Monitoring System (RMS)[5]. Also, it performs control functions of every safety related component including the component related to a ESF-CCS. ESF-CCS consists of a Group Controller (GC), Loop Controller (LC), ESF-CCS Test and Interface Processor (ETIP), Cabinet Operator Module (COM), Communication Device and a Man-Machine Interface (MMI) Component. Figure1 shows the interface of the ESF-CCS software.




[Figure 1] The interface of ESF-CCS software

## 3. The verification methods for the ESF-CCS

ESF-CCS defines the software behavior with the models which are described in the Software Requirement Specification and the Software Design Specification. We will perform a verification about these models to confirm whether these software behaviors are operated exactly or not.
The verification about a software is tired through a three-stage process as a Model Semantics, Simulation, and Formal Verification about software models. The three-stage process is as follows:

1) Model Semantics

Specified models are thoroughly checked before a simulation code or target code is generated. The syntax checking, semantic checking, and cycle detection are performed to review the model semantics.

- Syntax checking

Syntax checking evaluates whether the model is syntactically correct with respect to the graphical and textual formalism used in the Language.

- Semantic checking

Semantic checking determines whether the model conforms to the Language semantics. For instance, the model topology must be consistent, with no orphan states or missing connections.

- Cycle detection

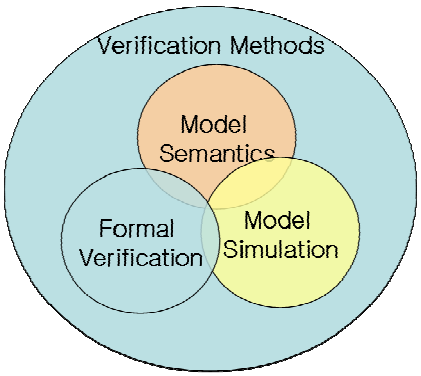
When designing the State of a software, although we may create a model that is syntactically correct, it may nevertheless be inconsistent. When this happens, because of the instantaneous emission of signals, the concurrent presence of signals may end up in algebraic loops. In this case, we will have to modify our design when such cycles are detected.

2) Model Simulation

Simulation is one of the most common methods of a verification used since the 1960s[6].
We use a simulation to run interactive sessions to dynamically check a model. It grasps whether the software is operated normally and then creates the expected outputs or not.

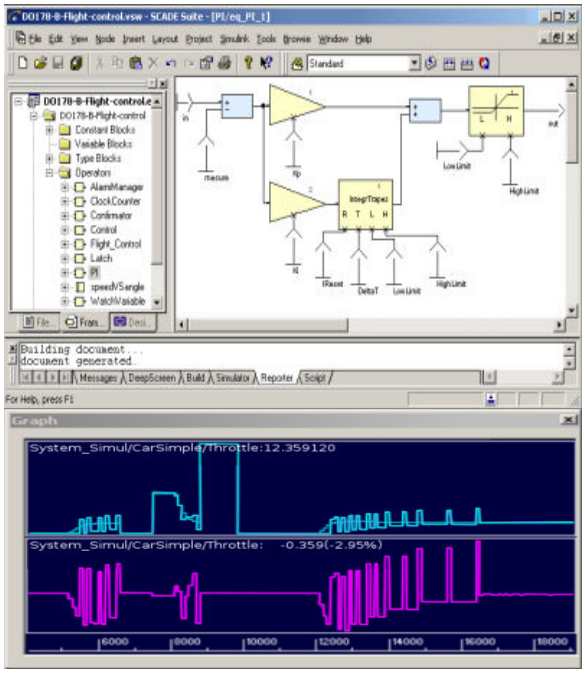
3) Formal Verification

Formal verification is a systematic process of ensuring, through exhaustive algorithmic techniques, that a design implementation satisfies the requirements of its specification. By using a formal verification method, all possible executions of the design are mathematically analyzed without the need to develop simulation input stimulus or tests.




## 4. SCADE Tool

We performed the suggested verification methods with the SCADE (Safety Critical Application Development Environment) tool. SCADE Tool was developed by the Esterel-technology Corporation. It is an integrated development environment for developing a safety critical embedded software[9].
The SCADE uses the Lustre language graphically and supports Safe State Machine (SSM) to present control flow. It also supports a syntax checking, semantic checking, and cycle detection function and it can do a simulation. The design verifier in SCADE can verify the behavior of software models. Figure2 shows the editor and simulator of the SCADE.




[Figure 2] The SCADE tool

## 5. Conclusion

ESF-CCS is an engineered safety system which is used in a safety system of a plant power system and is being developed through a software lifecycle process. To represent the functions and behaviors of a software, a model specification language is used for a software requirement specification, software design specification. We tried to do a model verification to confirm whether the software has a safety feature and a reliability or not.
We suggested 3-stage process to do a verification about the software models and tried to utilize these methods with the SCADE tool. The 3-stage processes consist of a model semantic, simulation and a formal verification. A model semantic checks a syntax and semantic of a specified model. A simulation checks the functions of a model dynamically. A formal verification analyzes an abnormal state of a software model after checking its behaviors.
The verification of the 3-stage process guarantees that the software model is implemented exactly and is operated errorlessly.

## REFERENCES

[1] USNRC Reg. Guide 1.152, Rev.02, Feb. 2006, “Criteria for Programmable Digital Computers System Software in Safety Related Systems of Nuclear Power Plants”
[2] IEEE Std. 7-4.3.2, 2003, “Standard Criteria for Digital Computers in Safety System of Nuclear Power Generating Stations”
[3] IEEE Std. 829-1998, “IEEE Standard for S/W Test Documentation”
[4] IEEE Std. 1008-1987, “IEEE Standard for Software Unit Testing”
[5] KAERI, “ESF-CCS Software Requirement Specification” KNICS-ESF-SRS221
[6] Douglas L. Perry, Harry D. Foster, “Applied Formal Verification” McGraw-Hill, Electronic Engineering
[7] Peter B. Ladkin, “The Ariane 5 Accident: A Programming Problem?”, http://www.rvs.uni-bielefeld.de/publications/Reports/ariane.html#References
[8] Harry Goldstein, “Who Killed the Virtual Case File?”. IEEE Spectrum September 2005.
[9] SCADE User Manual, Esterel-Technologies.