RESEARCH REPORT

VTT-R-00213-14

Hardware failure modelling methodology for model checking

Authors: Jussi Lahtinen

Confidentiality: Public

Report's title Hardware failure modelling methodology for model checking	
Customer, contact person, address VYR	**Order reference** 33/2013SAF
Project name Safety evaluation and reliability analysis of nuclear automation	**Project number/Short name** 77379 / SARANA
Author(s) Jussi Lahtinen	**Pages** 36/
Keywords Model checking, nuclear power plants, architecture, hardware failure	**Report identification code** VTT-R-00213-14

Summary

Digital instrumentation and control (I&C) systems are challenging to verify. They enable complicated control functions, and the state spaces of the models easily become too large for comprehensive verification through traditional methods. Model checking is a formal method that can be used for system verification. In our earlier work we have primarily focused on verifying the logic designs of systems using model checking. There is, however, need to examine safety systems under certain failure assumptions on the hardware equipment that is used to realize those systems.

In this work we have developed methodology for modelling hardware failures. The developed modelling techniques are highly modular and encapsulated, which results in decreased effort when the model is created. The methodology allows the verification of plant-level properties under various failure assumptions, which has previously been quite difficult.

Confidentiality	Public

Espoo, 26.2.2014

Written by	**Reviewed by**	**Accepted by**
Jussi Lahtinen, Research Scientist	Janne Valkonen Senior Scientist	Riikka Virkkunen Head of Research Area

VTT's contact address
VTT Technical Research Centre of Finland
P.O. Box 1000, FI-02044 VTT, Finland
Phone internat. +358 20 722 4520
Fax +358 20 722 4374

Distribution (customer and VTT)
SAFIR2014 Reference group 2

Preface

This report has been prepared as part of the research project Safety Evaluation and Reliability Analysis of Nuclear Automation (SARANA), which is part of the Finnish Research Programme on Nuclear Power Plant Safety 2011–2014 (SAFIR2014). This report describes the development of modelling methodology for hardware failures. The new methodology allows properties to be verified on the plant-level under certain failure assumptions.

We wish to express our gratitude to the representatives of the organizations and all those who have given their valuable input in the meetings and discussions during the project.

Espoo, February 2014

Authors

Contents

Abbreviations

ACP	AC power system
ADS	Automatic depressurisation system
APU	Acquisition and processing unit
BDD	Binary decision diagram
BMC	Bounded model checking
BWR	Boiling water reactor
CCF	Common cause failure
CCW	Component cooling water system
CD	Core damage
COM	Communications module
CTL	Computation tree logic
DPS	Diverse protection system
ECC	Emergency core cooling system
EFW	Emergency feed water system
FMEA	Failure mode and effects analysis
FTA	Fault tree analysis
I&C	Instrumentation and control
LOCA	Loss of coolant accident
LOFW	Loss of feed water
LOOP	Loss of online power
LTL	Linear temporal logic
MFW	Main feed water system
NPP	Nuclear power plant
PRA	Probabilistic risk assessment
RHR	Residual heat removal system
RPS	Reactor protection system
RPV	Reactor pressure vessel
SAT	Propositional satisfiability problem
SWS	Service water system
VU	Voting unit
V&V	Verification and validation
YVL	Finnish nuclear regulatory guide (Ydinvoimalaitosohjeet)

1. Introduction

The verification of digital instrumentation and control (I&C) systems is challenging because programmable logic controllers enable complicated control functions, and the state spaces of the designs easily become too large for comprehensive manual inspection. Design verification can eliminate design errors that are hard to detect later in the development process. These errors are very expensive to repair, often leading to a major redesign and reimplementation cycle. Typically, verification and validation (V&V) activities rely heavily on subjective evaluation, which covers only a limited part of the possible behaviours of the system. Therefore, more rigorous formal methods are needed; see, for example, [Valkonen et al., 2008] for an overview.

Model checking [Clarke et al., 1999] is a formal method that can be used to verify the correctness of system designs. Internationally, it has been used in the verification of, e.g., hardware and microprocessor designs, data communications protocols and operating system device drivers. Several model checking systems and tools exist. In this work, we have focused on the model checking tool NuSMV. The tool is able to determine automatically whether a given state machine model satisfies given specifications.

In our previous work [Lahtinen et al., 2012c] we have primarily focused on verifying the logic designs of systems using various model checking methodologies. There is, however, need to examine safety systems used in a nuclear power plant on a higher level. Namely, the overall system safety needs to be examined on the plant level. The purpose of this examination is to e.g. evaluate whether the system architecture is designed so that it is sufficiently fault tolerant. The Finnish regulatory guides on nuclear safety (YVL guides) require that all individual safety systems are single-failure tolerant. For some systems it should also be possible to perform the safety function even if any single-component fails and any other component is simultaneously out of operation due to repair or maintenance.

Traditionally, the verification of the plant level architecture has been performed using methods such as fault tree analysis (FTA), and failure mode and effects analysis (FMEA). These techniques, however, do not take into account the actual behaviour of the digital automation systems (i.e. the potential errors in the logic of the automation systems). Probabilistic risk assessment (PRA) methods are being developed to account software faults as well (see e.g. [Authén et al., 2012a]) but the methodology is still under development and there is currently no consensus over the best practices. Model checking can be used to verify the logical designs exhaustively. Thus, it is tempting to try to expand the scope of model checking so that effects of hardware failures to the plant-level behaviour can be exhaustively analysed under certain failure assumptions.

In our earlier work regarding the modelling of hardware failures [Lahtinen et al., 2012b] we developed preliminary methodology for hardware fault models using a small fictitious example model. These fault models included faults in telecommunication links, microprocessor faults, and electrical faults influencing all equipment in a cabinet. The intention of this methodology is to examine the effects of a set of hardware failures to the system's operation. In this paper we improve on this methodology. We have created 1) a failure module that covers all hardware components and their failure modes, 2) link modules that encapsulate information transfer in the model, and 3) a process module for covering the effects of initiating events. The new methodology also covers more plant hardware equipment such as pumps and valves.

The developed modelling technique is highly modular and encapsulated, which results in decreased effort when the model is created. When verification is performed on the plant level, the resulting models tend to become large, and abstraction is typically required in order to be able to analyse certain properties. The methodology we have created allows abstract configurations of the model to be created quite effortlessly by replacing whole modules with more abstract ones.

The methodology intends to bridge the gap between model checking and PRA methods. The methodology is quite compatible with PRA methodology since the case study model was created based on PRA material as input. It allows the verification of system properties under various failure assumptions and “on the plant level”, which has previously been quite difficult. Assumptions on common cause failures can also be made.

We have applied the developed methodology on an example case study. We have used a PRA-model as reference material and created a corresponding model that can be used for model checking. The PRA-model depicts a fictitious nuclear power plant originally made for illustration purposes to demonstrate basic elements of the risk and reliability analysis software Risk Spectrum (trademark of Scandpower Lloyd’s Register). The model has been analysed and improved upon in the DIGREL project [Authén et al., 2010]. The model is a fictive and simplified nuclear power plant (NPP) that has acted as a reference model when different analysis methods (PRA) have been developed. The PRA model represents a boiling water reactor (BWR), which has four-redundant safety systems. The model includes eight different systems of which we have modelled seven. The power distribution system was not modelled in our work. Further information on the used model can be found in Section 3 and in [Authén et al, 2013].

The rest of the paper is organised as follows. In Section 2 we briefly introduce the formal method called model checking. In Section 3 we describe the example system used in the case study. In Section 4 we present the developed methodology for modelling hardware failures. The verification results from the case study are in Section 5, and the conclusions are in Section 6.

2. Model checking

Model checking [Clarke et al., 1999], [Clarke & Emerson, 1981], [Queille & Sifakis, 1982] is a computer-aided verification method developed to formally verify the correct functioning of a system design model by examining all of its possible behaviours. The models used in model checking are quite similar to those used in simulation. However, unlike simulation, model checkers examine the behaviour of the system design with all input sequences and compare it with the system specification. In model checking, at least in principle, the analysis can be fully automated with computer-aided tools. The specification is expressed in a suitable language, temporal logics being a prime example, describing the permitted behaviours of a system. Given a model and a specification as inputs, a model checking algorithm determines whether the system has violated its specification. If none of the behaviours of the system violates the given specification, the (model of the) system is correct. Otherwise, the model checker will automatically give a counter-example execution of the system demonstrating how the specification has been violated.

In this work we have used the model checker NuSMV [Cavada et al., 2010], [NuSMV, 2011], which was originally designed for hardware model checking. NuSMV is a state-of-the-art symbolic model checker that supports synchronous state machine models in which the real-time behaviour must be modelled with discrete time steps using explicit counter variables that are incremented at a common clock frequency. In NuSMV the formal specification can be written as a state invariant clause, or in a more complex specification language such as Linear Temporal Logic (LTL) or Computation Tree Logic (CTL) [Clarke et al., 1999], making it quite flexible in expressing design specifications. In this work we have primarily used state invariant specifications. There are several possible algorithms that can be used for verifying state invariant specifications. In NuSMV, at least three variants are available: 1) a Binary Decision Diagram (BDD) based approach [Bryant, 1986] [McMillan, 1993], 2) a propositional satisfiability solving (SAT) based approach [Biere et al., 1999], [Biere et al., 2006], and 3) an advanced variant of the SAT-based approach called k-induction [Sheeran et al., 2000], [Eén & Sörensson, 2003], in which the state invariants are proved using induction, and the base step and the induction step are reduced to bounded model checking problems. In this work we have used the k-induction algorithm for verifying the specifications.

3. Description of the example system

The case study system used in this work is originally a PRA-model of a nuclear power plant originally made for illustration purposes to demonstrate basic elements of the risk and reliability analysis software Risk Spectrum (trademark of Scandpower Lloyd's Register). The model has been analysed and improved upon in the DIGREL project. The model depicts a fictive and simplified NPP that has acted as a reference model when different analysis methods (PRA) have been developed. The PRA model represents a boiling water reactor (BWR), which has four-redundant (4 x 100%) safety systems. The example model includes eight different systems. The example model should not be interpreted as a representative boiling water reactor, but rather as an example for demonstrating the reliability analysis of representative nuclear safety I&C. The data on the case study model presented on the tables of this paper has been mainly extracted from [Authén et al., 2012a] and [Authén et al., 2013]. See these documents for further information on the used model.

3.1 Safety systems

The example model represents a fictive boiling water reactor (BWR), which has four-redundant safety systems. The example model includes the following systems:

- ACP – AC power system
- CCW – Component cooling water system
- ECC – Emergency core cooling system
- EFW – Emergency feedwater system
- ADS – Automatic depressurisation system
- RHR – Residual heat removal system
- SWS – Service water system
- MFW – Main feedwater system.

The safety systems other than the AC power system, read measurements, and perform calculations on these measurements. On certain conditions the safety systems actuate their dedicated pumps / valves. The list of measurements is presented in Table 1. A measured value such as the RPV water level (reactor pressure vessel) is measured using multiple components with different component IDs. The components are also four redundant: the letter *i* in the component ID is a place holder for identifying the redundancy of the component (1-4). In the table it is also shown whether the measurement belongs to the reactor protection system (RPS) or to the diverse protection system (DPS) and the related signal ID that is used is also shown.

Measurement	**Component ID**	**Limit**		**Purpose**	**RPS**	**DPS**
RPV water level, fine level	RPVi1CL001	L2	Low level	Core cooling protection	RSS04	
	RPVi2CL001	H2	Extra high level	RPV overfilling protection		DSS05

	RPVi2CL001	L2	Low level	Core cooling protection		DSS04
RPV water level, coarse level	RPVi1CL002	L4	Extremely low level	Core cooling protection	RI002 RTB01	
	RPVi2CL002	L3	Extra low level	Core cooling protection		DX001
	RPVi2CL002	L4	Extremly low level	Core cooling protection		DI002
Feedwater system pump suction pressure	MFWi0CP001	L1	Low pressure before feedwater pump	Loss of feedwater supervision		DSS13
Feedwater system room temperature	MFWi0CT001	H1	High room temperature	Leakage supervision		DM005
Containment pressure	RCOi1CP001	H1	High pressure in containment	Leakage supervision	RI005 RTB02	
	RCOi2CP001	H1	High pressure in containment	Leakage supervision		DI005
Condensation pool temperature	RCOi0CT001	H1	High temperature in condensation pool	Residual heat removal	RX003	
Water level in the ECC pump room	ECCi0CL001	H1	Water on the floor	Leakage supervision	RH00i	
Water level in the EFW pump room	EFWi0CL001	H1	Water on the floor	Leakage supervision		DH00i
AC power voltage bus bar ACP-i	ACPi1CE001	L1	Low voltage on bus bar ACP-i	Loss of offsite power supervision	RZ00i	
	ACPi2CE001	L1	Low voltage on bus bar ACP-i	Loss of offsite power supervision		DZ00i

Table 1. Measurements. Adapted from [Authén et al., 2013].

The conditions on which the different actuators of the plant should be operated are presented in Table 2. For example, the pumps operated by the CCW safety system should be started in case of a reactor scram or high temperature in the condensation pool. The table presents the conditions of actuation on the signal level as well. The right-most column in the table presents the default value that is used instead of the measured value in case a fault is detected from a device associated with that measurement.

System	Actuator	Control	Condition for control type	Signal ID	DFLT
ACP	Diesel generator	Start	Reactor scram due to containment isolation or low voltage in respective bus bar	RSS12 + RZ00i + DZ00i	0
		Stop	Manual stop and not active start signal	NOT(RSS12 + RZ00i + DZ00i) * MAN-0iDG01	1
ADS	Pressure relief valve	Open	Depressurisation signal	RADS1 {RTB0}	0
		Close	Manual close and not active depressurisation signal	RADS2 {NOT(RTB00) * MAN-ADSi, i = 1-8}	1
CCW	Pump	Start	Reactor scram or high temperature in the condensation pool	RSS00 + RX003	0
		Stop	Manual stop and not active start signal	NOT(RSS00 + RX003) * MAN-CCW0iPM01	1
ECC	Pump	Start	Containment isolation and no water leakage in the respective pump room	NOT(RH00i) * RI000	0
		Stop	Water leakage in the respective pump room	RH00i	1
ECC	Motor-operated valve	Open	Containment isolation and no water leakage in the respective pump room	NOT(RH00i) * RI000	0
		Close	Water leakage in the respective pump room	RH00i	1
EFW	Pump	Start	Feedwater system isolation, reactor scram due to low water level in reactor or containment isolation and no water leakage in the respective pump room	NOT(DH00i) * (DM000 + DSS04 + DI000)	0
		Stop	Water leakage in the respective pump room	DH00i	1
EFW	Motor-operated valve	Open	Reactor scram due to low water level in reactor, diverse low water level condition or very low water level condition and no water leakage in the respective pump room	NOT(DH00i) * (DSS04 + DX001 + DI002)	0
		Close	Water leakage in the respective pump room or very high water level in reactor	DH00i + DSS05	1
HVA	AC cooler	Start	Start EFW	NOT(DH00i) * (DM000 + DSS04 + DI000)	0
		Stop	Manually	MAN-HVA0iAC01	1
MFW	Pump	Start	Manual start and not active stop signal	NOT(DM000 + DSS05) * MAN-MFWi, i = 1, 2, 3	0
		Stop	Feedwater system isolation or very high water level in reactor	DM000 + DSS05	1
RHR	Pump	Start	Reactor scram or high temperature in the condensation pool and no water leakage in the respective pump room	RSS00 + RX003	0
		Stop	Manual stop and not active start signal	NOT(RSS00 + RX003) * MAN-RHR0iPM01	0
RHR	Motor-operated valve	Open	Reactor scram or high temperature in the condensation pool and no water leakage in the respective pump room	RSS00 + RX003	0
		Close	Manual stop and not active start signal	NOT(RSS00 + RX003) * MAN-RHR0iVM02	0
SWS	Pump	Start	Reactor scram or high temperature in the condensation pool	RSS00 + RX003	0
		Stop	Manual stop and not active start signal	NOT(RSS00 + RX003) * MAN-RHR0iVM02	0
RSS	Control rods		Reactor Scram	RSS {RSS00} + DSS {DSS00}	1

Table 2. Actuator functions. Adapted from [Authén et al., 2013].

The signals of Table 2 are further elaborated in Table 3. The actuation signals in Table 2 are calculated based on other signals and eventually based on some measurements and manual commands. The actuation logic can be deduced from Table 3. As an example, in Table 2 it is stated that EFW pumps are stopped when there is water leakage in the pump room (signal DH00i). In Table 3 we can trace the signal DH00i to a condition “EFWi0CL001-H1 + EFWi0CL002-H1”. Both of these signals are sensors that measure the water level in the pump room. Other signals can be traced in a similar fashion. Table 3 also indicates how the automation signals respond to different initiating events (a loss of coolant accident (LOCA), loss of online power (LOOP), loss of feed water (LOFW) and Transient). “Always” means that the signal always becomes true if the initiating event is true. “Spurious” means that the signal is false during the initiating event and can only become true in case there is a fault in the equipment related to the signal. “Manual” means that the signal can be true during the initiating event if related manual commands are given.

Signal	Description	Condition	LOCA	LOOP	LOFW	Transient
RPS						
RH00i	Isolation of the ECC pump room i	ECCi0CL001-H1 + ECCi0CL002-H1	Spurious	Spurious	Spurious	Spurious
RI000	Containment isolation	2/4*(RI002-i + RI005-i)	<- RI002, RI005	<- RI002	<- RI002	<- RI002
RI002	Containment isolation due to extremly low level in RPV	2/4*(RPVi0CL002-L4)	Always	Always	Always	Always
RI005	I isolation due to high pressure in containment	2/4*(RCOi0CP001-H1)	Always			
RM000	Feedwater isolation	2/4*(RM005-i)	<- RM005	<- RM005	<- RM005	<- RM005
RM005	Feedwater isolation due to high temperature in feedwater system compartment	2/4*(MFWi0CT001-H1)		Spurious		Spurious
RSS00	Reactor scram	2/4*(RSS04-i + SS05-i + SS12-i + SS13-i)	<- RSS04, RSS12 (<- RI000), RSS13	<- RSS04, RSS12 (<- RI000), RSS13	<- RSS04, RSS12 (<- RI000), RSS13	<- RSS04, RSS12 (<- RI002)
RSS04	Reactor scram due to low water level in RPV	2/4*(RPVi0CL001-L2)	Always	Always	Always	Always
RSS05	Reactor scram due to high water level in RPV	2/4*(RPVi0CL001-H2)	Probable	Spurious	Spurious	Spurious
RSS12	Reactor scram due to containment isolation (I- or M-isolation)	2/4*(RI000-i + RM000-i)	<- RI000	<- RI000	<- RI000	<- RI000
RSS13	Low pressure before feedwater pump	2/4*(MFWi0CP001-L1)		Always	Always	
RTB00	Depressurisation of the primary circuit	RTB01 * RTB02		<- RTB01 & RTB02	<- RTB01 & RTB02	<- RTB01 & RTB02
RTB01	Depressurisation of the primary circuit condition 1: extreme low level in reactor (same as I002)	2/4*(RPVi0CL002-L4)	Always	Always	Always	Always
RTB02	Depressurisation of the primary circuit condition 2: high pressure in containment (same as I005) or manual actuation	RTB03 + 2/4*(RCOi0CP001-H1)	Always	Manual	Manual	Manual
RTB03	Manual TB	MAN-TB				
RX003	High temperature in condensation pool	2/4*(RCOi0CT001-H1)	Always	Always	Always	Always
RZ00i	Low voltage in AC bus bar i	ACPi0CE001-L1	Always	Always	Always	Always
DPS						
DH00i	Isolation of the EFW pump room i	EFWi0CL001-H1 + EFWi0CL002-H1	Spurious	Spurious	Spurious	Spurious
DI000	Containment isolation	2/4*(DI002-i + DI005-i)	<- DI002, DI005	<- DI002	<- DI002	<- DI002
DI002	Containment isolation due to extremly low level in RPV	2/4*(RPVi0CL002-L4)	Always	Always	Always	Always
DI005	I isolation due to high pressure in containment	2/4*(RCOi0CP001-H1)	Always			
DM000	Feedwater isolation	2/4*(DM005-i)	<- DM005	<- DM005	<- DM005	<- DM005
DM005	Feedwater isolation due to high temperature in feedwater system compartment	2/4*(MFWi0CT001-H1)		Spurious		Spurious
DSS00	Reactor scram	2/4*(DSS04-i + SS05-i + SS12-i + SS13-i)	<- DSS04, DSS12 (<- DI000), DSS13	<- DSS04, DSS12 (<- DI000), DSS13	<- DSS04, DSS12 (<- DI000), DSS13	<- DSS04, DSS12 (<- DI002)
DSS04	Reactor scram due to low water level in RPV	2/4*(RPVi0CL001-L2)	Always	Always	Always	Always

DSS05	Reactor scram due to high water level in RPV	2/4*(RPVi0CL00 1-H2)	Probabl e	Spurious	Spurious	Spurious
DSS12	Reactor scram due to containment isolation (I- or M-isolation)	2/4*(DI000-i + DM000-i)	<- DI000	<- DI000	<- DI000	<- DI000
DSS13	Low pressure before feedwater pump	2/4*(MFWi0CP0 01-L1)		Always	Always	
DX001	Extra low level in RPV	2/4*(RPVi0CL00 2-L3)	Always	Always	Always	Always

Table 3. Protection functions. Adapted from [Authén et al., 2013] and [Gustafsson, 2012].

Below in Figure 1 is a simplified flow diagram related to the safety systems in the case study model. The flow diagram illustrates some of the valves and pumps related to the safety systems. For most safety systems, only one redundant implementation of the system is shown in the diagram.

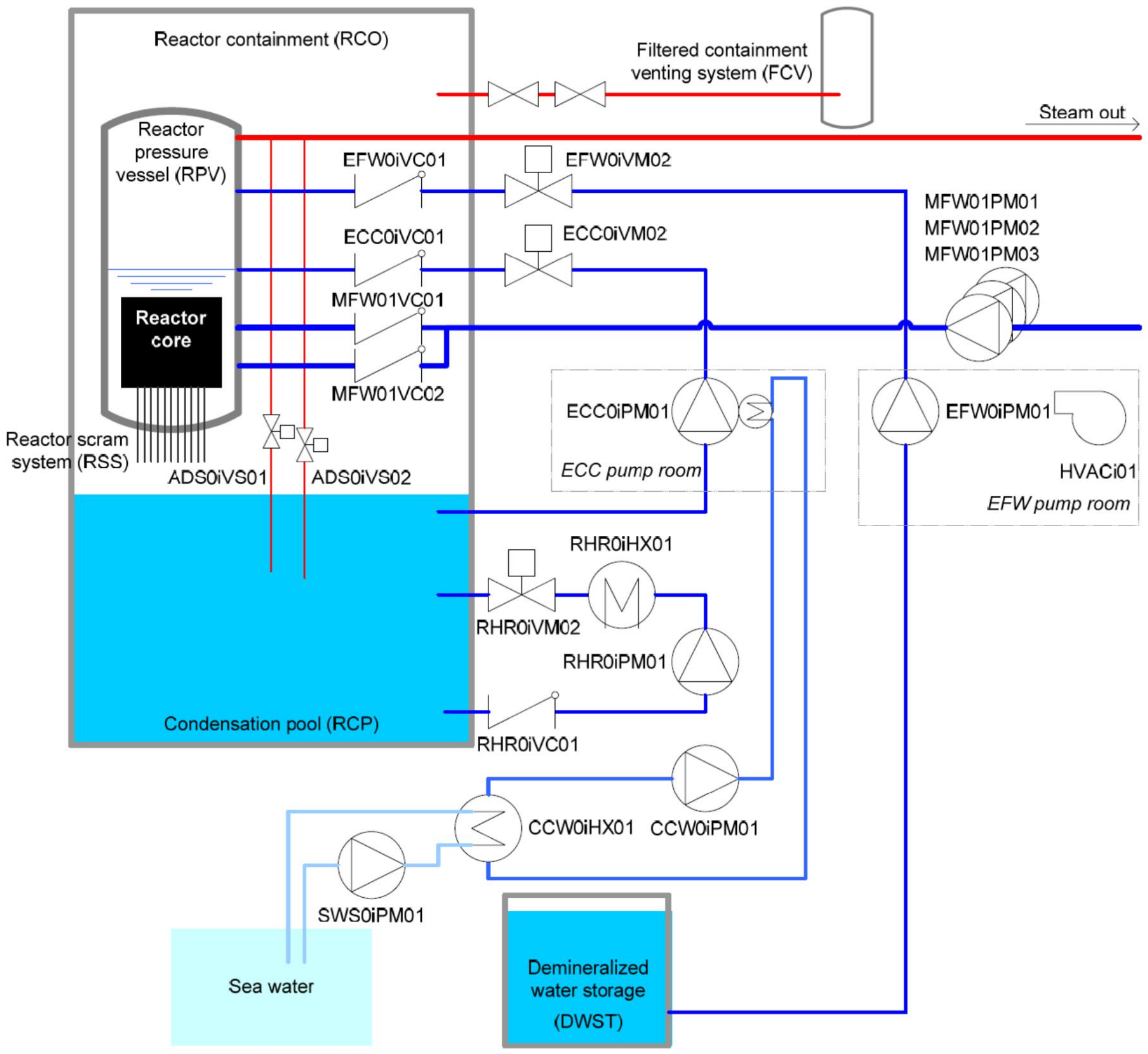

Figure 1. Flow diagram illustrating one redundancy of the safety systems. Adapted from [Authén et al., 2013].

3.2 Safety system requirements

Five initiating events are considered in the case study. The initiating events are presented in Table 4. Depending on the initiating event there are different success criteria for the safety systems. For example, in the case of a large LOCA initiating event both ECC and RHR should operate sufficiently. Sufficient operation means for ECC that in one of the four redundancies, the pump is running and the corresponding valve is open.

Initiating event	MFW	EFW	ADS	ECC	RHR
LOCA – Large Loca	No credit	No credit	Not required	1oo4	1oo4
LOFW – Loss of main feedwater	No credit	1oo4	4oo8	1oo4	1oo4
LOOP – Loss of offsite power	2oo3	1oo4	4oo8	1oo4	1oo4
Transient	2oo3	1oo4	4oo8	1oo4	1oo4
Common cause initiator loss of DC power bus bar	2oo3	1oo4	4oo8	1oo4	1oo4

Table 4. Safety system success criteria. Adapted from [Authén et al., 2013].

Event trees exist for each initiating event as well. From the event trees it is possible to see which systems are required to operate for the plant to survive the initiating event without any core damage. The event tree for the Large LOCA initiating event is in Figure 2. From the event tree we can see that if either of the safety systems (ECC or RHR) fails in all redundancies, the result is core damage (CD). Similar event trees exist for all initiating events. These event trees are not presented in this paper.

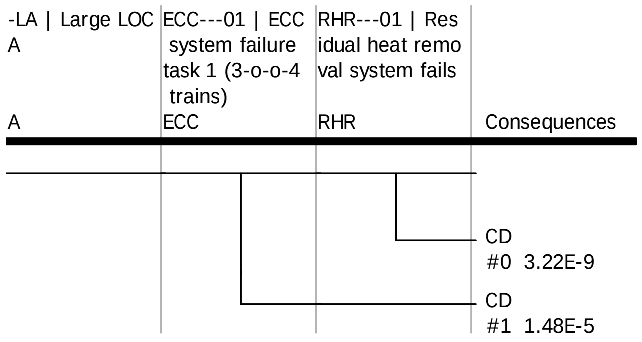
-LA | Large LOCA
A
ECC---01 | ECC system failure task 1 (3-o-o-4 trains)
ECC
RHR---01 | Residual heat removal system fails
RHR
Consequences
CD #0 3.22E-9
CD #1 1.48E-5

Figure 2. Large LOCA event tree

In addition to the successful operation of the front-line safety systems, the supporting systems are required to operate successfully as well. Respective EFW or ECC train is cooled by the component cooling water system (CCW) train, which is cooled by the corresponding service water system (SWS) train.

3.3 Safety system architecture

Most of the safety systems are designed four-redundant. This means that their actuation logic is implemented in four separate acquisition and processing unit (APU) computers. Measurements are separately brought to each APU. The APUs calculate their control signals independently and pass the resulting signals to voting units. Voting units collect the APU control signals together and actuate their dedicated safety device whenever 2 out of 4 of the APU signals indicate that the device should be started. The basic safety system architecture is illustrated in Figure 3.

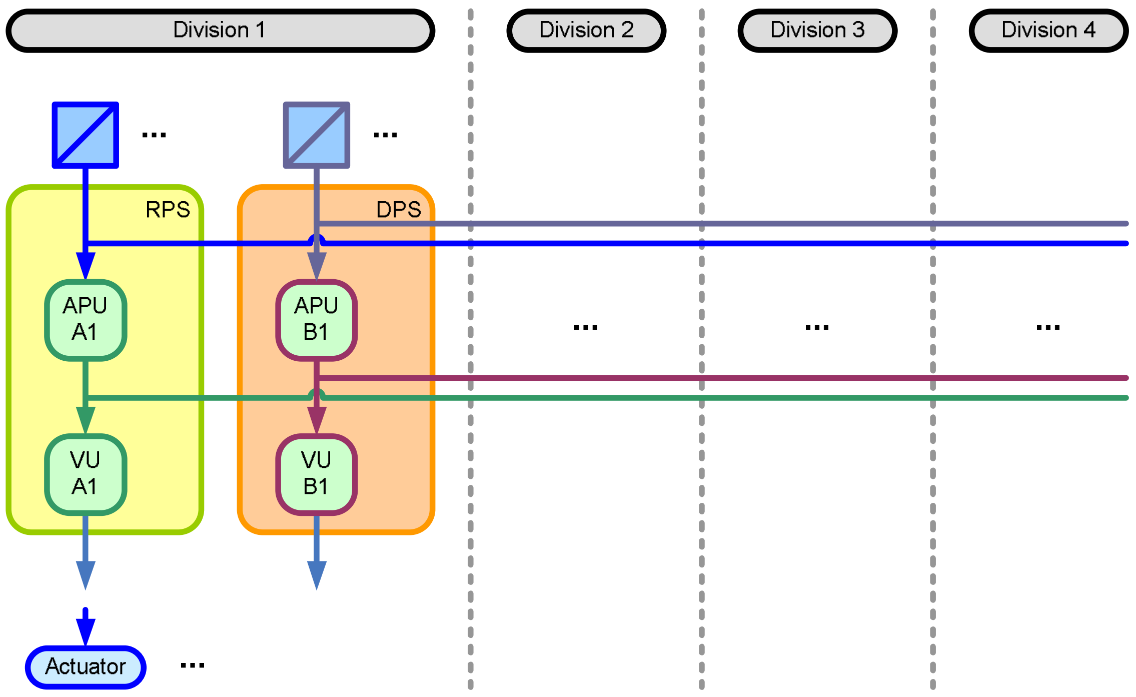

Figure 3. Safety system architecture. Adapted from [Authén et al., 2013].

The safety systems are divided into two separate subsystems: Reactor Protection System (RPS) and Diverse Protection System (DPS), which are implemented on different automation hardware, i.e. there exists four APU computers and voting units for the RPS safety systems, and four APU computers and voting units for the DPS safety systems. The RPS safety systems are: automatic depressurisation system (ADS), component cooling water system (CCW), emergency core cooling system (ECC), service water system (SWS) and residual heat removal system (RHR). The DPS safety systems are emergency feedwater system (EFW), and main feedwater system (MFW). The AC power system belongs to both RPS and DPS.

3.4 I&C failure modes

3.4.1 Instrumentation

Measuring devices have four failure modes that are included in the case study model:

- **Fails high**: the measured value indicates a value that is higher than the correct value.
- **Fails low**: the measured value indicates a value that is lower than the correct value.
- **Drift of value**: the measured value differs from the correct value.
- **Freeze of value**: the measured value incorrectly remains at some value.

3.4.2 Pumps and valves

Pumps have two failure modes:

- **Failure-on-demand**: the pump is given a starting command, and the pump does not start.
- **Spurious actuation**: The pump starts even though no start command has been given.

Valves have one failure mode:

- **Failure to open**: The valve is stuck close, and does not open when an open command is given.

3.4.3 I&C units

The APU computers and voting units are both I&C units that have many hardware components. A simple structural diagram of an I&C unit is in Figure 4. APU computers for example can be set to communicate via the bus link. In this case the relevant modules of the I&C unit in which a failure can affect that communication are the CPU module, the COM module, the subrack module (power supply) and the bus link between the two APUs.

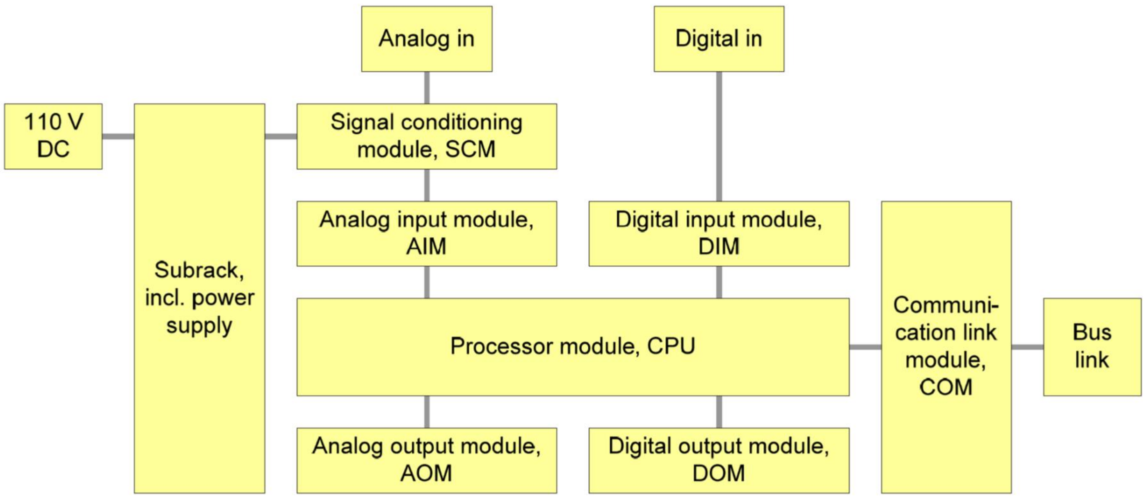
Analog in
Digital in
110 V DC
Subrack, incl. power supply
Signal conditioning module, SCM
Analog input module, AIM
Digital input module, DIM
Processor module, CPU
Communi-cation link module, COM
Bus link
Analog output module, AOM
Digital output module, DOM

Figure 4. I&C unit structure. Adapted from [Authén et al., 2013].

The list of hardware components and their failure modes is in Table 5. All of the components are not currently in the scope of the model. A limited set of hardware components and failure modes is in Table 6. The limited scope corresponds to the level of detail used in modelling of the case study as well.

Hardware Components	Failure Modes
Processor module	Hang
	Communication dropout
	Delayed signal
	Random behaviour
Analog Input Module	Signal fails high/low
	Signal drifts
	Signal hangs/freeze
Analog Input Module, Single channel	Signal fails high/low
	Signal drifts
	Signal hangs/freeze
Analog Output Module	Signal fails high/low
	Signal drifts
	Signal hangs/freeze
Analog Output Module, Single channel	Signal fails high/low
	Signal drifts

	Signal hangs/freeze
Digital Input Module	Signals stuck to current value
Digital Input Module, Single channel	Signal stuck to current value
	Signal fails to opposite state
Digital Output Module	Signals stuck to current value
Digital Output Module, Single channel	Signal stuck to current value
	Signal fails to opposite state
Signal Conditioning Module	Signal fails high/low
	Signal drifts
	Signal hangs/freeze
Communication module	Failure to establish communication
Watch-Dog Timer	Fails to activate
	Activates without computer failure
Backplane	Loss of backplane
Power supply	Interruption
	Short circuit
	Ground contact

Table 5. Full list of I&C unit components and failure modes. Adapted from [Authén et al., 2013].

Hardware Components	Failure Modes
Processor module	Hang
	Communication dropout
	Delayed signal
	Random behaviour
Analog Input Module	Signal fails high/low
	Signal drifts
	Signal hangs/freeze
Digital Input Module	Signals stuck to current value
Digital Output Module	Signals stuck to current value
Communication module	Failure to establish communication
Backplane	Loss of backplane
Power supply	Interruption
	Short circuit
	Ground contact

Table 6. Limited scope hardware components and failure modes. Adapted from [Authén et al., 2013].

3.4.4 Communication link

Communication network (i.e. network buses) has been modelled as communication links in the model. A communication link is always between two entities, and has only one failure mode:

- **Loss of function**: the connection is not working.

4. Model checking methodology for hardware failures

4.1 Case study scope

In this work we have created a model that can be used for model checking. The DIGREL model description has been used as reference material. We have modelled seven of the safety systems that are part of the DIGREL model. The modelled systems are:

- CCW – Component cooling water system
- ECC – Emergency core cooling system
- EFW – Emergency feedwater system
- ADS – Automatic depressurisation system
- RHR – Residual heat removal system
- SWS – Service water system
- MFW – Main feedwater system.

Only the AC power system was not included in the model. This was simply because the inclusion of the power system might require too much effort. The AC power system can be added to the model in future research.

An automation logic design was created for each safety system. The logic was manually designed in a rather straight-forward manner based on the DIGREL model material (Table 2 and Table 3). In addition to these safety system logics, the equipment required to operate and actuate each system was modelled. Each system has redundant sub-systems that were also modelled. The APU computers that implement the design logics are mostly four-redundant. This means that four computers execute the same safety system logic simultaneously. The measurements used by each APU are also typically four-redundant. The different redundancies also transfer information between each other via network buses.

No priority logic that would take care of conflicting signals or the completion of actuation of safety measures was modelled. This kind of functionality was not defined in the reference material either.

4.2 Modelling methodology principles

Some basic principles were followed in the development of the new modelling methodology. One main principle was to keep the methodology as modular as possible. High level of modularization often reduces the amount of manual modelling work when a single functionality is only modelled once and then reused as a module. Modularization also makes the abstraction of the model simpler as individual modules can be easily replaced with more abstract ones. Another modelling principle was that the developed methodology should be compatible with the previous techniques for modelling logic designs, and that the previous methodology for function block based logic designs should remain unchanged.

The applied modelling methodology is illustrated in a general level in Figure 5. The green boxes represent structures of the model (different modules), the blue circles represent variables in the model, and arrows illustrate information flow in the model. The essential ideas governing the modelling methodology are:

- A separate failure module that decides which hardware components experience a failure. The module has as input the information of how many common cause failures and other single failures are allowed simultaneously in the model. Based on this information, the failure module can decide in a non-deterministic manner any combination of failures on the plant level so that the failure assumption given as input is fulfilled. The failure module defines a separate module for each hardware component as well. These hardware component modules are used as parameters for the link modules.

- Link modules are used whenever some piece of information is transferred from one place to another in the plant automation. For example, the output of an APU computer is used as input on the voting unit. In addition to the value output by the APU, the value received and interpreted by the voting unit depends on whether the hardware equipment related to transferring the information have somehow failed. Another example is the measurement of water level and the use of the measured value as input on an APU computer. The read value depends on whether e.g. the measurement device or the input module of the APU has failed, and whether the failure has been detected. The link modules handle this behaviour in the model.

- A process module is used for deciding what the physical parameter values (e.g. reactor temperature, pressure) in the plant are. In our modelling method the only factor limiting the value of these physical parameters is the accident scenario. For example, in a loss of coolant accident (LOCA) the reactor containment water level must rise. The process module is otherwise kept as free as possible. In our model the operational states of pumps and valves do not affect the physical parameters. In other words there are no feedbacks implemented in the model environment. It would be technically possible to model these feedbacks but this would overcomplicate the model. Secondly, the intention in our modelling is not to cover the process aspect of the plant in a very detailed manner. Simulation tools exist for this purpose. In addition, the model checking tool used does not fit well for modelling complex physical processes involved in plant behaviour that require the use of more than basic mathematical operators.

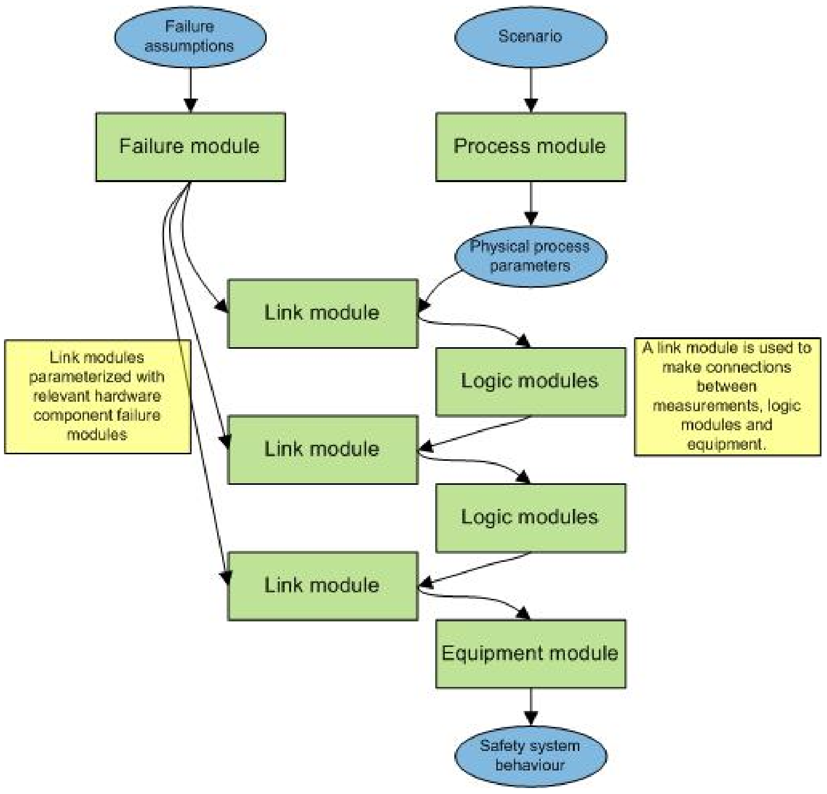
Failure assumptions
Scenario
Failure module
Process module
Physical process parameters
Link module
Link modules parameterized with relevant hardware component failure modules
Logic modules
A link module is used to make connections between measurements, logic modules and equipment.
Link module
Logic modules
Link module
Equipment module
Safety system behaviour

Figure 5. Modelling methodology – information flow in the model.

4.3 Model development

4.3.1 Logic modules

The developed logic modules depict the automation logic that is implemented in the APU computers. The logic was written based on the DIGREL model descriptions, namely Table 2 and Table 3. The logic design was written using a function block based approach because that has been a convention in our earlier work. The logic depicted in the tables could be designed using function blocks: AND, OR, NOT, 2oo4 vote. No timers were added to the design, even though using timers could result in a more realistic model. However, requirements for timings did not exist, and it was decided that there was no need to overcomplicate the model.

In addition to the logic implemented in the APU computers, voting unit logic was also implemented in separate modules. The voting unit modules were simple modules that typically had eight inputs: four start or open commands from the redundant APUs and four stop or close commands from the APUs. The voting unit simply calculated its two outputs, start/open and stop/close using 2oo4 function blocks on the input signals. Potential conflicting control signals (e.g. both start and stop signal true at the same time) were not restricted in any way in these voting units.

4.3.2 Link modules

Link modules are used whenever some piece of information is transferred from one place to another. Link module executes this transfer of information but simultaneously the effects of possible faults affecting the information are taken into account. Since there exists only a rather small number of different type of links (e.g. APU-to-APU, measurement to APU, etc.) it is reasonable to create link type modules that can be parameterized with equipment related to the particular link. In other words, we can create link type modules for a set of links instead of modelling each link separately. This decreases the amount of manual modelling work tremendously. In our case study model, five different link types exist:

- Measurement to APU: The transfer of a physical parameter to the APU computer via the measuring device.
- APU to APU: The different redundancies exchange information with each other via a network bus.
- APU to Voting unit: The control commands calculated by the APUs are transferred to voting units.
- Voting unit to pump: The actuation commands given by the voting units are transferred to a pump.
- Voting unit to valve: The actuation commands given by the voting units are transferred to a valve.

A link type module was written for each link type. The link type modules have as parameter:

- the variable transferred via the link,
- the hardware component modules of relevant hardware, and
- a default value that is used to replace the transferred variable value in case of a failure that is detected.

As an example, the link type module for measurement-to-APU links is below.

```
MODULE LINK_MEAS_APU(in1, measurement, apu, DFLT)
VAR
prevout : boolean;
DEFINE
output1 := case
       apu.backplane_or_powersupply_status != OK : FALSE;
       apu.digital_input_status = stuck_to_current_detected :
       DFLT;
       apu.digital_input_status = stuck_to_current_undetected :
       prevout;
       measurement.status = fail_high_detected : DFLT;
       measurement.status = fail_low_detected : DFLT;
       measurement.status = drift_detected : DFLT;
       measurement.status = freeze_detected : DFLT;
       measurement.status = fail_high_undetected : TRUE;
       measurement.status = fail_low_undetected : FALSE;
       measurement.status = drift_undetected : ! in1;
       measurement.status = freeze_undetected : prevout;
       TRUE : in1;
       esac;
ASSIGN
init(prevout):=FALSE;
next(prevout):= output1;
```

The parameter *in1* refers to the variable transferred by the link module, *measurement* is the measuring device from which the value is received, *apu* refers to the APU computer receiving the information, and *DFLT* is the related default value. The transferred variable value is a Boolean variable. The TRUE value of the variable means that the threshold related to the measurement has been surpassed. The FALSE value of the variable means that the physical value is still below the related threshold. The module consists of a single case clause that defines the value of *output1*. The case clause goes through all possible failure modes of the measuring device and the APU that can influence how the variable is read and interpreted in the APU logic. In case of a detected failure (failure modes attached with “*_detected*”) the module uses the *DFLT* value for *output1*. In case of non-detected failures the output is changed according to the failure mode. For example, in the fail-high failure mode the measured value is replaced with TRUE indicating that the measurement is above the respecting limit. Two of the failure modes are such that the variable value freezes to the previous value. This has been modelled using a separate variable *prevout*. In case of a freeze failure the output is set to variable value experienced in the previous time point. The link type module described above could be instantiated using e.g. the following expression:

```
LINK_RPV20CL001-H2_APU1 : LINK_MEAS_APU(processmodule.RPVi0CL001-
H2, failuremodule.MEAS_RPV20CL001-H2, failuremodule.APU1,
RPVi0CL001-H2_DFLT);
```

4.3.1 Failure module

All the failures of the model are modelled in a separate failure module. To be more specific, failures have been modelled using a set of modules. A single aggregate failure module is used to collect and store all instances of modelled hardware components and connections. Each hardware component and network connection has its own module. A hardware component module exists for APU computers, connections, pumps, valves and measurements. In addition, a separate common cause failure module is used for modelling CCF’s. The CCF module creates instances of CCF’s that have been modelled. Figure 6 illustrates the model structure related to modelling failures. Each box represents a module in the model. The arrows indicate that the module creates an instance of another module that the arrow is pointing at.

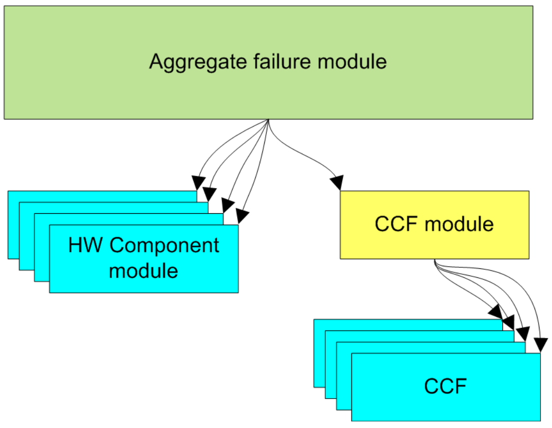

Figure 6. The model structure for modelling failures

4.3.1.1 Hardware component modules

Each hardware component type has its own module. Below is the hardware component type module for APU’s and voting unit computers.

```
MODULE APU_failuremodule(id, CCFmodule)
VAR
FROZENVAR processor_status : {OK, hang_detected, hang_undetected,
dropout_detected,dropout_undetected, delayed_detected,
delayed_undetected, rand_detected, rand_undetected};
```

```
FROZENVAR digital_input_status : {OK, stuck_to_current_detected,
stuck_to_current_undetected};
FROZENVAR digital_output_status : {OK, stuck_to_current_detected,
stuck_to_current_undetected};
FROZENVAR backplane_or_powersupply_status : {OK,
loss_of_function};
DEFINE

nro_faults :=  toint(processor_status != OK)
        + toint(digital_input_status != OK)
        + toint(digital_output_status != OK)
        + toint(single_failure_in_backplane) ;
single_failure_in_backplane := (backplane_or_powersupply_status !=
OK) & ! CCFmodule.APU_backplane_CCF.realizes;

ASSIGN
init(backplane_or_powersupply_status):= case
        CCFmodule.APU_backplane_CCF.realizes & (id in
        CCFmodule.APU_backplane_CCF.affected_ids) :
        CCFmodule.APU_backplane_CCF.failure_mode;
        TRUE : {OK, loss_of_function};
esac;
```

The module has as parameter the hardware component id number and the CCF module. Because the APU computer has many elements that can fail, each element has its own variable. The following APU elements have been modelled: processor, digital input module, digital output module, backplane and power supply. The variables are of type FROZENVAR which in the NuSMV modelling language means that the variable value cannot change after the initial time point. In our failure modelling, we assume that faulty equipment do not suddenly become non-faulty and vice versa. As an example, the variable that covers backplane and power supply failures has two possible states: either there are no failures in these parts of the APU (status is *OK*) or these parts have failed in which case the only failure mode experienced is *loss_of_function*. The other APU parts have several failure modes out of which some can be divided into two subcategories: detected failures and undetected failures. The failure modes are named accordingly. In addition to these variable declarations the hardware component module also calculates the number of experienced failures in the particular APU since several simultaneous failures can be possible. Since common cause failures have influence on the status of individual components the module also uses *init* clauses to force the related variables to the values dictated by the effective common cause failures.

4.3.1.2 The aggregate failure module

The aggregate failure module creates instances of the hardware component modules. All the hardware in the plant is instantiated including APU/VU computers, connections, pumps, valves and measurements. The aggregate failure module has as parameter the failure assumptions used in the model. This includes the number of simultaneous common cause failures and single failures that are allowed in the model. These inputs simply restrict the failure module so that overly many simultaneous failures are not possible. All hardware component module failures are added up and an INVAR clause is added to restrict the module to handling only those situations that are according to the failure assumptions. The relevant model code is below.

```
nro_of_faults := measurement_faults + APU_faults + VU_faults +
connection_faults + equipment_faults;
INVAR nro_of_faults <= nro_of_allowed_failures;
```

4.3.1.3 Common cause failures

The aggregate failure module also creates an instance of the common cause failure module. This module is a data structure collecting all instances of common cause failures. The common cause failure module also calculates the realizing common cause failures together and makes sure that the number of simultaneous CCF:s is not against the given failure assumptions. The relevant model code is below.

```
MODULE CCF_failuremodule(nro_of_allowed_CCFs)
VAR
APU_backplane_CCF : CCF({1,2,3,4}, loss_of_function);
APU_APU_connections_CCF : CCF({1,2,3,4,5,6}, loss_of_function);

DEFINE
nro_of_CCFs := toint(APU_backplane_CCF.realizes)
+ toint(APU_APU_connections_CCF.realizes);

INVAR nro_of_CCFs <= nro_of_allowed_CCFs;
ASSIGN
```

A separate CCF module has been defined for modelling a common cause failure. The module has as parameter a set of id numbers that are affected by the CCF, and the failure mode related to that CCF. This module does not include any calculations; it simply exists for data structuring purposes. The model code for the CCF module is below.

```
MODULE CCF(ids, failuremode)
VAR
realizes : boolean;
DEFINE
affected_ids := ids;
failure_mode := failuremode;
ASSIGN
```

4.3.1 Process module

The process module decides on the values of the physical parameters of the plant. These values are the actual physical values independent from the measurement values that may diverge from the actual values. These physical parameters have been modelled mainly as Boolean variables instead of real valued variables. This is because the physical parameters are typically compared only against a single limit value. From the model perspective it only matters whether the physical parameters are below or above this limit. This behaviour can be achieved using only Boolean variables.

The process module has as parameter the scenario that is under examination. The scenario is one of the following: LOCA, LOOP, LOFW, TRANSIENT, or FREE. The first four are accident scenarios that force certain physical parameters to have a particular value. For example, in all accident scenarios the reactor water level becomes low. Consequently, the corresponding variables in the model shall also indicate that the reactor water level is low. The process module consists of clauses that implement these kinds of rules for all scenarios. In case of the FREE scenario the physical parameters of the plant experience no restrictions what so ever: the values of the parameters are selected non-deterministically, and all combinations and sequences of variable values are possible. Below is the model code for reactor containment pressure. In case of the LOCA scenario the pressure is always high. In other scenarios any value is possible.

```
init(RCOi0CP001-H1) := case
        scenario = LOCA : TRUE;
        TRUE : {TRUE, FALSE};
esac;
next(RCOi0CP001-H1) := case
        scenario = LOCA : TRUE;
        TRUE : {TRUE, FALSE};
esac;
```

The process module is created so that it has as little functionality as possible. This is intentional. For model checking it is advisable to not make the process module too detailed. The reasons for this are mainly that the model checking tool does not fit too well for modelling the physical process in a very detailed level. The NuSMV is not fit for modelling complex physical behaviour very accurately. Modelling the process in a detailed way also causes the state space of the model to become very large, which is not wanted. Additionally, if the environment model has more detail it becomes more probable that some behaviour may be left uncovered in the analysis. In model checking it is more useful to create rather simple over-approximated environment models, in which all kind of behaviour is possible. In this way, the truth value of safety properties is preserved, since if there is no unwanted behaviour in a model with a rather free environment, the safety properties are also true in a more restricted environment model. For these same reasons we have not included any feedback to the process module. Feedback tends to overcomplicate the model too much. It would be technically possible to add certain simple feedback, e.g. if the pumps and valves of a certain safety system are operational and open, the water level in the reactor containment should rise. These kinds of feedbacks were not implemented in the model.

4.3.2 Equipment modules

Separate modules were created for pumps and valves as well. In this model the modules were not exactly necessary since these modules merely pass the on and off commands received from the voting units via corresponding links. There are no priority issues involved in this model, which makes the modules quite unnecessary. The modules were nonetheless created for future compatibility handling, and for the possible addition of some priority logic in the future, for example if manual commands are added to the model, some prioritization with the automation commands is probably needed.

4.3.3 Detected and undetected failures

The detected and undetected failures were modelled as separate failure modes following the reference material. The detection of failures is perceived by the link modules that replace the signal values with a predetermined default value in case of one of the detected failure modes is present. In a realistic I&C system the detected failures can be detected by the I&C unit equipment, and in case of such a detection the signal might be marked e.g. using a status bit. The replacement of the signal value with some predetermined default value could also be part of the I&C application logic. In this case study, the status bits were not used or modelled since such behaviour and logic was not described in the reference material, and the modeller did not want to diverge too far from the input material descriptions. Technically, a status bit implementation would be possible, but the resulting model would probably become somewhat more demanding to verify.

4.3.4 Scenarios

The PRA analysis uses various initiating events whose consequences are evaluated. An initiating event is an event that creates a disturbance in the plant and has a potential to lead to core damage, depending on the successful operation of the various mitigating systems of the plant. [IAEA, 1993] Various initiating events were also modelled in our case study in order to follow this approach used in the PRA world. Four initiating events (LOCA, LOOP, LOFW, and Transient) were modelled using a single enumerative variable that determines the used scenario. The value of the scenario variable forces certain process parameters to some values in the model. In addition to the initiating events the variable *scenario* could also have the value *free*, in which case any of the physical process parameter values were allowed. In what follows, the initiating events are gone through in more detail:

- **LOCA**. A loss of coolant accident means that the primary coolant is not available due to e.g. damage in the primary coolant circuit. In the simplified model, this means that the reactor pressure vessel water level is low, the reactor containment pressure becomes high and the temperature in the condensation pool becomes high.
- **LOOP**. Loss of online power. In the simplified model the initiating event leads to low water level in the reactor pressure vessel, high temperature in the condensation pool, and low pressure before feed water pump.
- **LOFW**. Loss of feed water may occur due to e.g. main feed water pump failures. The effect on the model scope is low pressure before the feed water pump, low water level in the reactor pressure vessel, and high temperature in the condensation pool.
- **Transient**. Transient initiating events are those which introduce disturbance in normal plant operation without the loss of primary coolant. These events require the shutdown of the reactor. [IAEA, 1993] Effects on process parameters in the model are reactor pressure vessel water level decrease, and high temperature in the condensation pool.

4.4 Abstractions and other measures for verifying properties

The case study model becomes very large and complex, which makes the verification of properties challenging. Especially the link modules cause the model to become very complex. However, there are several abstractions and model checking techniques and parameters that make the verification of some properties on this very large model feasible. This section goes through the main techniques that are relevant in our case study and can be used to enable the verification of properties on the model within a sensible time limit. The verification results are in Section 5.

4.4.1 Program slicing and interface modules

If only one of the subsystems is addressed in a given requirement it is often possible to only include that subsystem in the model. This means that only part of the model needs to be examined to verify the property. Program slicing can be performed manually to obtain the reduced model i.e. manually edit the model file so that unnecessary parts are removed, and the interfaces to those parts modified so that the model remains functional. In our earlier work (see e.g. [Lahtinen et al., 2012a]) we have also applied the compositional minimization technique to abstracting function block based designs. In compositional minimization certain modules of the model are replaced with interface modules. An interface module contains no function blocks, and the outputs of the module are defined as free variables. The inputs of an interface module are left intact because of technical issues allowing compatibility with the model, but the inputs have no influence on the outputs. Definition of the output variables as free variables is a complete over-approximation of the module, i.e. no restrictions on the behaviour of the module are set.

In this case study, interface modules can be created for the modules that encompass the logical functionality of the I&C systems. Technically interface modules could be created for other modules as well (for example the link modules) but it is not clear yet whether this is practical and useful. In practice, the use of interface modules makes the construction of abstract model configurations quite quick and effortless. For each module in the system, the modeller just has to choose which version of the module is used (the concrete version or the interface version), and the final model is created via a computer script that constructs the model according to the selections.

4.4.1 Compositional verification

In addition to the compositional minimization type approach it may be possible to utilize other compositional verification techniques as well. The most relevant in this context is probably assume-guarantee type reasoning [Păsăreanu, 1999] [Rushby, 2001]. In this technique, an assumption is made of the environment of a module. The assumption is then verified on the environment. When the assumption is known to hold, we can check if a specification is true in an individual module under this assumption. If the specification is true in the individual module, it is also true in the whole system. It may be possible to use this approach in our case study as well. If a particular requirement cannot be verified on the full model it could be verified as two or more sub-requirements. Each sub-requirement would be verified using only part of the model, and if all sub-requirements were true, this would also indicate that the original property is true in the whole model.

4.4.2 Link module abstractions

In our case study the link modules cause a significant portion of the complexity of the model. The link modules can, however, be abstracted as well. Using interface modules instead of concrete link modules is one option that was not really rigorously explored in the case study. Several instances of the link modules are created in the model, and it seems that using interface modules might result in too much abstraction in order to get reasonable results. In some models replacing link modules with interface modules may be practical.

Instead of interface modules we used a lighter abstraction. We created simplified versions of the link type modules that still covered a large portion of the functionality of the link modules but the complex failure behaviour was simplified.

As an example consider the concrete version of the link type module for measurement-to-APU links below.

```
MODULE LINK_MEAS_APU(in1, measurement, apu, DFLT)
VAR
prevout : boolean;
DEFINE
output1 := case
        apu.backplane_or_powersupply_status != OK : FALSE;
        apu.digital_input_status = stuck_to_current_detected :
        DFLT;
        apu.digital_input_status = stuck_to_current_undetected :
        prevout;
        measurement.status = fail_high_detected : DFLT;
        measurement.status = fail_low_detected : DFLT;
        measurement.status = drift_detected : DFLT;
        measurement.status = freeze_detected : DFLT;
        measurement.status = fail_high_undetected : TRUE;
        measurement.status = fail_low_undetected : FALSE;
        measurement.status = drift_undetected : ! in1;
        measurement.status = freeze_undetected : prevout;
        TRUE : in1;
        esac;
ASSIGN
```

```
init(prevout):=FALSE;
next(prevout):= output1;
```

The link type module has one internal variable *prevout* which complicates the state spaces of the models, in which the link is used because the value of the internal variable depends on the value of the *output1* of the module on the previous time point. In the abstract version of the module the internal variable *prevout* can be replaced with a free variable *random* which can choose its value non-deterministically at every time point. The abstraction is an over-approximation and preserves the truth value of universal properties (e.g. safety properties). If some bad behaviour is non-existent in the abstract model, then the bad behaviour is also non-existent in the more concrete model as well. The resulting abstract link type module is below.

```
MODULE LINK_MEAS_APU(in1, measurement, apu, DFLT)
VAR
random : boolean;
DEFINE
output1 := case
        apu.backplane_or_powersupply_status != OK : FALSE;
apu.digital_input_status = stuck_to_current_detected : DFLT;
apu.digital_input_status = stuck_to_current_undetected : random;
        measurement.status = fail_high_detected : DFLT;
        measurement.status = fail_low_detected : DFLT;
        measurement.status = drift_detected : DFLT;
        measurement.status = freeze_detected : DFLT;
        measurement.status != OK : random;
        TRUE : in1;
esac;
ASSIGN
```

4.4.3 Exclusion of liveness properties

In the case study we have thus far concentrated on verifying basic state invariant properties only. The reason for this is that verifying LTL or CTL properties in general seems to take too much time on the model. Bounded model checking (BMC) can be performed on the model but most BMC techniques can only output counter-examples to specifications and not give any proofs.

One exception to this is the incremental k-induction algorithm implemented in NuSMV. The algorithm can in many cases also prove that specifications are true. The algorithm supports only state invariant properties, but seems to be quite fast compared to the other NuSMV model checking algorithms. The utilization of the algorithm forces us to exclude liveness properties from the case study. The invariant checking algorithm could also be used for checking liveness properties if a liveness to safety reduction could be applied on the model. We leave this for future work.

5. System verification results

We were able to verify several formal specifications on the case study model. The specifications were formalized from the list of requirements in *Table 7*.

Number	Requirement
1	In case of a LOCA initiating event, the plant safety systems shall fulfil the related success criteria.
2	In case of a LOFW initiating event, the plant safety systems shall fulfil the related success criteria.
3	In case of a LOOP initiating event, the plant safety systems shall fulfil the related success criteria.
4	In case of a TRANSIENT initiating event, the plant safety systems shall fulfil the related success criteria.
5	In case of a LOFW/LOOP/TRANSIENT initiating event, the EFW safety system shall start a pump and open a valve in at least one of the four redundant subsystems.
6	In case of a LOCA/LOFW/LOOP/TRANSIENT initiating event, the ECC safety system shall start a pump and open a valve in at least one of the four redundant subsystems.
7	In case of a LOOP/TRANSIENT initiating event, at least two out of the three MFW pumps shall be started.
8	In case of a LOFW/LOOP/TRANSIENT initiating event, at least four out of the eight ADS release valves shall be opened.
9	In case of a LOCA/LOFW/LOOP/TRANSIENT initiating event, at least one out of the four RHR pumps shall be started.

Table 7. Requirements checked on the case study model

The first four requirements are plant-level requirements that require the inclusion of several safety systems in order to be verified. The requirements 5 through 9 are requirements for individual safety systems.

The plant-level requirements are based on the success criteria on which the plant survives a given initiating event.

The plant survives a LOCA event if one train of the ECC system and one train of the RHR system operates. In our model an ECC train operates when the pump is running and the valve is open. In addition, the supporting systems must operate on the same redundancy (SWS and CCW pump running). The RHR train operates whenever the pump is running.

The plant survives a LOFW event when ECC and RHR operate on at least one train and the ADS operates on at least four out of eight trains. Additionally, the EFW system contributes to the plant cooling but it is not absolutely required for the success criteria to be fulfilled. For ECC and EFW systems the supporting systems must also be operating (CCW, SWS).

The plant survives a LOOP and a Transient event when ECC and RHR operate on at least one train and the ADS operates on at least four out of eight trains. Additionally, the EFW and MFW systems contribute to the plant cooling but they are not absolutely required for the success criteria to be fulfilled. For ECC and EFW systems the supporting systems must also be operating (CCW, SWS).

The plant level requirements are such that if no failures are assumed they should be true. Depending on the requirement several hardware failures can be tolerated. We checked the requirements using several different failure assumptions: no single failures assumed, one single failure allowed, two single failures allowed, and three single failures allowed. CCF failures were not assumed.

The requirements are such that they can be written as state invariants. In this case study, we were limited to simple state invariants because they can be verified using a bounded model checking method called k-induction that is much less computationally demanding than more traditional BDD-based model checking methods. The k-induction method could be used to verify all the specifications in this case study. In NuSMV the method can be invoked using the *check_invar_bmc_inc* command. A bound of 2 was sufficient for verifying the properties of the case study. The k-induction method could prove the properties on bound 1. We also used the NuSMV parameters *–dynamic* and *–coi* for better performance.

In what follows we briefly go through each requirement and present the corresponding formalized invariants, and the abstractions and assumptions that were used in the verification.

Requirement 1.

- Only the parts of the system covering the systems (RHR, ECC, SWS and CCW) need to be included in the model. Because of this, other safety system logics were replaced with interface modules.
- The fulfilment of the property depends on whether the ECC system is operating. The ECC system is stopped whenever a high water level is detected in the ECC pump room. Because of this we assume that the water level does not rise. This assumption does not prevent faulty measurements of the water level.
- For link modules we use the abstract link modules as described in Section 4.
- Different failure assumptions were made. As an example the resulting formalized property assuming two single failures is:

```
INVARSPEC (failures = 2 & CCFs = 0 & scenario = LOCA & !
processmodule.ECCi0CL001-H1 & ! processmodule.ECCi0CL002-H1  ->
LOCA_No_Core_Damage);
```

Requirement 2.

- Only the parts of the system covering the systems (EFW, ADS, RHR, ECC, SWS and CCW) need to be included in the model. Because of this, other safety system logics were replaced with interface modules.
- The fulfilment of the property depends on whether the ECC system is operating. The ECC system is stopped whenever a high water level is detected in the ECC pump room. Because of this we assume that the water level does not rise. This assumption does not prevent faulty measurements of the water level. For the same reasons, we also assume that the water level does not rise in the EFW pump room.
- The ADS valves open automatically if there is high pressure in the reactor containment. Otherwise the valves must be opened manually. Because of this, we assume that one of these conditions (high pressure or manual open command) is also true.
- In our model all of the ADS valves are manually opened using a single switch. We assume that the switch itself does not fail because otherwise a single failure in the switch could prevent the system from operating. In a more realistic model, there would probably be a separate switch for each valve.
- For link modules we use the abstract link modules as described in Section 4.
- Different failure assumptions were made. As an example the resulting formalized property when no failures are assumed is:

```
INVARSPEC (failures = 0 & CCFs = 0 & scenario = LOFW & !
processmodule.ECCi0CL001-H1 & ! processmodule.ECCi0CL002-H1 & !
processmodule.EFWi0CL001-H1 & ! processmodule.EFWi0CL002-H1 &
(processmodule.MAN_DEPRESSURISATION_VALVEi_OPEN  |
processmodule.RCOi0CP001-H1)  &
failuremodule.MEAS_MAN_DEPRESSURISATION_VALVEi_OPEN.status = OK
-> LOFW_No_Core_Damage);
```

Requirement 3.

- For the verification of this property, no interface modules were used.
- The assumptions for the systems EFW, ECC and ADS are the same as in requirement 2.
- The MFW system is prevented from operating if a high water level is detected in the reactor pressure vessel or if there is high temperature in the feedwater system room. We assume that these parameters are not true.
- The MFW system starts with a manual command. We assume that the manual command is also given.
- In the model, only a single manual switch was modelled for all three MFW pumps. We also assume that the switch does not fail.
- For link modules we use the abstract link modules as described in Section 4.
- Different failure assumptions were made. As an example the resulting formalized property when a single failure is assumed is:

```
INVARSPEC (failures = 1 & CCFs = 0 & scenario = LOOP &
processmodule.MAN-MFWi & ! processmodule.MFWi0CT001-H1 & !
processmodule.RPVi0CL001-H2 & failuremodule.MEAS_MAN-
MFWi.status = OK & ! processmodule.ECCi0CL001-H1 & !
processmodule.ECCi0CL002-H1 & ! processmodule.EFWi0CL001-H1 & !
processmodule.EFWi0CL002-H1 &
(processmodule.MAN_DEPRESSURISATION_VALVEi_OPEN  |
processmodule.RCOi0CP001-H1)  &
failuremodule.MEAS_MAN_DEPRESSURISATION_VALVEi_OPEN.status = OK
-> LOOP_No_Core_Damage);
```

Requirement 4.

- For the verification of this property, no interface modules were used.
- The assumptions for the systems EFW, ECC, ADS, and MFW are the same as in requirement 3.
- For link modules we use the abstract link modules as described in Section 4.
- Different failure assumptions were made. As an example the resulting formalized property when three failures are assumed is:

```
INVARSPEC (failures = 3 & CCFs = 0 & scenario = TRANSIENT &
processmodule.MAN-MFWi & ! processmodule.MFWi0CT001-H1 & !
processmodule.RPVi0CL001-H2 & failuremodule.MEAS_MAN-
MFWi.status = OK & ! processmodule.ECCi0CL001-H1 & !
processmodule.ECCi0CL002-H1 & ! processmodule.EFWi0CL001-H1 & !
processmodule.EFWi0CL002-H1 &
(processmodule.MAN_DEPRESSURISATION_VALVEi_OPEN  |
processmodule.RCOi0CP001-H1)  &
failuremodule.MEAS_MAN_DEPRESSURISATION_VALVEi_OPEN.status = OK
-> TRANSIENT_No_Core_Damage);
```

Requirement 5.

- For the verification of this property, only the part of the model concerning the EFW system and the supporting systems (SWS, CCW) was included.

- We assume that the water level in the EFW pump room remains low, since otherwise the function is prevented.
- For link modules we use the abstract link modules as described in Section 4.
- Different failure assumptions were made. As an example the resulting formalized property when no failures are assumed is:

```
INVARSPEC  (CCFs = 0 & failures = 0 & (scenario = LOFW |
scenario = LOOP | scenario = TRANSIENT) & !
processmodule.EFWi0CL001-H1 & ! processmodule.EFWi0CL002-H1 ->
EFW_1oo4);
```

Requirement 6.

- For the verification of this property, only the part of the model concerning the ECC system and the supporting systems (SWS, CCW) was included.
- We assume that the water level in the ECC pump room remains low, since otherwise the function is prevented.
- For link modules we use the abstract link modules as described in Section 4.
- Different failure assumptions were made. As an example the resulting formalized property when no failures are assumed is:

```
INVARSPEC  (CCFs = 0 & failures = 0 & (scenario = LOCA |
scenario = LOFW | scenario = LOOP | scenario = TRANSIENT) & !
processmodule.ECCi0CL001-H1 & ! processmodule.ECCi0CL002-H1 ->
ECC_1oo4);
```

Requirement 7.

- For the verification of this property, only the part of the model concerning the MFW system was included.
- The MFW system is prevented from operating if a high water level is detected in the reactor pressure vessel or if there is high temperature in the feedwater system room. We assume that these parameters are not true.
- The MFW system starts with a manual command. We assume that the manual command is also given.
- In the model, only a single manual switch was modelled for all three MFW pumps. We also assume that the switch does not fail.
- For link modules we use the abstract link modules as described in Section 4.
- Different failure assumptions were made. As an example the resulting formalized property when no failures are assumed is:

```
INVARSPEC  (CCFs = 0 & failures = 0 & (scenario = LOOP |
scenario = TRANSIENT) & processmodule.MAN-MFWi & !
processmodule.MFWi0CT001-H1 & ! processmodule.RPVi0CL001-H2 ->
MFW_2oo3);
```

Requirement 8.

- For the verification of this property, only the part of the model concerning the ADS system was included.
- The ADS valves open automatically if there is high pressure in the reactor containment. Otherwise the valves must be opened manually. Because of this, we assume that one of these conditions is also true.
- In our model all of the ADS valves are manually opened using a single switch. We assume that the switch itself does not fail because otherwise a single failure in the switch could prevent the system from operating.
- For link modules we use the abstract link modules as described in Section 4.

- Different failure assumptions were made. As an example the resulting formalized property when no failures are assumed is:

```
INVARSPEC  (CCFs = 0 & failures = 0 & (scenario = LOFW |
scenario = LOOP | scenario = TRANSIENT) &
(processmodule.MAN_DEPRESSURISATION_VALVEi_OPEN  |
processmodule.RCOi0CP001-H1) -> ADS_4oo8);
```

Requirement 9.

- For the verification of this property, only the part of the model concerning the RHR system was included.
- For link modules we use the abstract link modules as described in Section 4.
- Different failure assumptions were made. As an example the resulting formalized property when no failures are assumed is:

```
INVARSPEC  (CCFs = 0 & failures = 0 & (scenario = LOCA |
scenario = LOFW | scenario = LOOP | scenario = TRANSIENT) ->
RHR_1oo4);
```

Some of the variables used in the formal specifications (e.g. *ECC_1oo4*) were calculated using simple case clauses:

```
ECC_1oo4 :=  case
               ECC_PM01_RUNNING & ECC_VM01_OPEN &
               CCW_1_OPERATING_NORMALLY : TRUE;
               ECC_PM02_RUNNING & ECC_VM02_OPEN &
               CCW_2_OPERATING_NORMALLY : TRUE;
               ECC_PM03_RUNNING & ECC_VM03_OPEN &
               CCW_3_OPERATING_NORMALLY : TRUE;
               ECC_PM04_RUNNING & ECC_VM04_OPEN &
               CCW_4_OPERATING_NORMALLY : TRUE;
               TRUE : FALSE;
    esac;
```

The system level variables such as *LOCA_No_Core_Damage* were then calculated based on the related event trees (LOCA event tree is illustrated in Figure 2) and case clauses:

```
LOCA_No_Core_Damage := ECC_1oo4 & RHR_1oo4;
```

The verification results, and verification times for the requirements under different failure assumptions are presented in Table 8.

Requirement number	Assumed single failures	Result	Verification time
1	0	TRUE	82 s
1	1	TRUE	79 s
1	2	TRUE	78 s
1	3	FALSE	76 s
2	0	TRUE	78
2	1	TRUE	87 s
2	2	TRUE	86 s

2	3	FALSE	83 s
3	0	TRUE	106 s
3	1	TRUE	103 s
3	2	TRUE	108 s
3	3	FALSE	108 s
4	0	TRUE	101 s
4	1	TRUE	96 s
4	2	TRUE	92 s
4	3	FALSE	78 s
5	0	TRUE	72 s
5	1	TRUE	76 s
5	2	TRUE	71 s
5	3	FALSE	64 s
6	0	TRUE	61 s
6	1	TRUE	63 s
6	2	TRUE	63 s
6	3	FALSE	58 s
7	0	TRUE	81 s
7	1	TRUE	82 s
7	2	FALSE	63 s
8	0	TRUE	65 s
8	1	TRUE	59 s
8	2	TRUE	61 s
8	3	FALSE	52 s
9	0	TRUE	29 s
9	1	TRUE	29 s
9	2	TRUE	31 s
9	3	FALSE	30 s

Table 8. Verification results and times for the model checked properties

The requirement 7 is single-failure tolerant but does not hold when two simultaneous single failures are assumed. Other requirements are true even if two single failures are assumed and not true when three single failures are assumed.

We can also see that the verification times for requirements 3 and 4 are slightly higher than for other requirements. Also the verification times for requirements 1 and 2 are higher than for requirements 5 through 9. This is supposedly due to leaving out unneeded model parts in requirements 5 through 9, and the use of interface modules in requirements 1 and 2.

We can also see that the failure assumption does not seem to have a lot of influence on the verification times.

6. Conclusions

In this work, we have presented new methodology to model failures in a system and tried out the methodology in a case study covering several safety systems of a nuclear power plant. The new models take into account the hardware configuration of a system and the various failure modes of the different hardware components.

To test the hardware failure modelling methodology, an imaginary system was modelled as a case study. The modeled example includes seven many-redundant safety systems. The automation logic of the systems was interpreted based on PRA notations. In addition to the design logics of the I&C systems, the hardware equipment realizing these functions has been included in the model. Failure modes of automation hardware were also modelled according to the reference PRA material.

Using the developed methodology, we managed to verify several system properties using model checking. When we have proved with a model checker that the system model satisfies its formally specified properties, that is a mathematical statement of the correctness of two formal objects. We of course have to consider the possibilities of modelling errors in both the model as well as the formal properties, as well as the possibility of bugs in the model checker implementation. For further discussion on the topic, see e.g. [Kuismin & Heljanko, 2013].

The performed case study system has previously been used in many PRA studies. Our case study model is entirely based on the same reference material that is used as input in PRA analyses. Because of this, the case study has given a better understanding on the two approaches. It is possible that an even tighter integration between these two approaches is possible. A single well-defined system-level model of the plant could be used for both PRA and model checking.

The main contribution of the work is the novel methodology for modelling hardware failures in an automation system. In practice, this means that in addition to the logic modules we have traditionally modelled, we can also create 1) a failure module that covers all hardware components and their failure modes, 2) link modules that encapsulate information transfer in the model, and 3) a process module for covering the effects of initiating events.

The methodology intends to bridge the gap between model checking and PRA methods. It allows the verification of system properties under various failure assumptions, which has previously been quite difficult. Assumptions on common cause failures can also be made. It is possible to verify fault-tolerance requirements on the plant level as well.

However, the developed methodology has some drawbacks. When complex I&C systems are modelled on this level of detail the resulting model can become very large, and the verification of typical properties on the whole model is very challenging. Currently we are limited to verifying simple safety properties. The model is so large that the typically used BDD-based model checking algorithm cannot be used. Another algorithm called k-induction can handle the large model better, but this algorithm can only handle simple state invariant properties (safety properties). More complicated system properties could be verified by using a liveness-to-safety reduction (see [Kuismin & Heljanko, 2013]) that transforms the more complex system property into a simple state invariant property.

In addition, the case study logics are quite simple. If e.g. more timing or feedbacks are added to the model, the state space will become even larger. Some modular abstraction or compositional verification techniques can be used to make the verification more feasible in some instances. In this work we have already used some of these techniques.

The model checking methodology for large and complex systems has been previously addressed in the SARANA project through a specific algorithm that is based on the modular composition of the model. It is still an open question whether a similar approach could be used for models that encompass the hardware failures of the system as well. We leave this question for future work.

References

[Authén et al., 2010] Authén, S., Björkman, K., Holmberg, J.-E., Larsson, J. Guidelines for reliability analysis of digital systems in PSA context, Phase 1 Status Report, NKS Report, NKS-230, December 2010.

[Authén et al., 2012a] Authén, S., Holmberg, J.-E. Reliability analysis of digital systems in a probabilistic risk analysis for nuclear power plants. Nuclear Engineering and Technology. Vol. 44 (2012) No: 5, 471 – 482, 2012.

[Authén et al., 2012b] Authén, S., Gustafsson, J., Holmberg, J.-E. Guidelines for reliability analysis of digital systems in PSA context, Phase 2 Status Report, NKS Report, NKS-261, February 2012.

[Authén et al., 2013] Authén, S., Gustafsson, J., Holmberg, J.-E. Guidelines for reliability analysis of digital systems in PSA context, Phase 3 Status Report, NKS Report, NKS-277, March 2013.

[Biere et al., 1999] Biere, A., Cimatti, A., Clarke, E. M. & Zhu, Y. Symbolic model checking without BDDs. In: Proc. of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99), 1999.

[Biere et al., 2006] Biere, A., Heljanko, K., Junttila, T., Latvala, T. & Schuppan, V. Linear Encodings of Bounded LTL Model Checking. Logical Methods in Computer Science 2(5:5), pp. 1–64, 2006.

[Bryant, 1986] Bryant, R. E. Graph-Based Algorithms for Boolean Function Manipulation. IEEE Trans. Computers 35(8), pp. 677–691, 1986.

[Cavada et al., 2010] Cavada, R., Cimatti, A., Jochim, C. A., Keighren, G., Olivetti, E., Pistore, M., Roveri, M. & Tchaltsev, A. NuSMV 2.5 User Manual. FBK-irst, 2010.

[Clarke & Emerson, 1981] Clarke, E. M. & Emerson, E. A. Design and synthesis of synchronization of skeletons using branching time temporal logic. In: Proceedings of the IBM Workshop on Logics of Programs, Vol. 131 of LNCS, Springer, pp. 52–71, 1981.

[Clarke et al., 1999] Clarke, E. M., Grumberg, O. & Peled, D. A. Model checking. Cambridge MA: MIT Press, 1999. 314. ISBN 0-262-03270-8.

[Eén & Sörensson, 2003] Eén N, Sörensson N. Temporal induction by incremental SAT solving. Electr Notes Theor Comput Sci ;89(4):543-60, 2003.

[Gustafsson, 2012] Gustafsson, J. Reliability analysis of safety-related digital instrumentation and control in a nuclear power plant. Degree project in Risk and Safety, Second Level. Royal Institute of Technology, January 2012.

[IAEA, 1993] International Atomic Energy Agency IAEA. Defining initiating events for purposes of probabilistic safety assessment. IAEA-TECDOC-719. IAEA, Vienna, September, 1993.

[Kuismin & Heljanko, 2013] Kuismin, T., Heljanko, K. Increasing Confidence in Liveness Model Checking Results with Proofs. In Bertacco, V., Legay, A., eds.: Haifa Verification Conference. Volume 8244 of Lecture Notes in Computer Science., Springer (2013) 32-43

[Lahtinen et al., 2012a] Lahtinen, J., Björkman, K., Valkonen, J., Niemelä, I. Emergency Diesel Generator Control System Verification by Model Checking and Compositional Minimization. In Proceedings of 8th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2012), Kučera, A., Henzinger, T.A., Nešetřil, J., Vojnar, T., and Antoš D. editors. pp. 49-60. Znojmo, Czech Republic, October 25-28, 2012.

[Lahtinen et al., 2012b] Lahtinen, J. Launiainen, T., Heljanko, K. Model checking methodology for large systems, faults and asynchronic behaviour. SARANA 2011 work report. VTT Technology 12, 2012. Available online: http://www.vtt.fi/inf/pdf/technology/2012/T12.pdf

[Lahtinen et al., 2012c] Lahtinen J., Valkonen J., Björkman K., Frits J., Niemelä, I., Heljanko K. Model checking of safety-critical software in the nuclear engineering domain. Reliability Engineering & System Safety, 105:104-13, 2012. doi:10.1016/j.ress.2012.03.021.

[McMillan, 1993] McMillan, K. L. Symbolic Model Checking, Kluwer Academic Publ., 1993.

[NuSMV, 2011] NuSMV Model Checker v.2.5.4, 2011. http://nusmv.irst.itc.it/.

[Păsăreanu et al., 1999] Păsăreanu, C. S., Dwyer, M. B. & Huth, M. Assume-Guarantee Model Checking of Software: A Comparative Case Study. In: Proceedings of the 5th and 6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking, Dennis Dams, Rob Gerth, Stefan Leue, and Mieke Massink (Eds.). Springer-Verlag, London, UK, Pp. 168–183, 1999.

[Queille & Sifakis, 1982] Queille, J. P.; Sifakis, J. Specification and verification of concurrent systems in CESAR, International Symposium on Programming. In Lecture Notes in Computer Science Volume 137, 1982, pp 337-351, 1982.

[Rushby, 2001] Rushby, J. Formal Verification of McMillan's Compositional Assume-Guarantee Rule. Technical Report, University of Minnesota, Minneapolis, 2001.

[Sheeran et al., 2000] Sheeran M, Singh S, Stålmarck G. Checking safety properties using induction and a SAT-solver. In: Jr. WAH, Johnson SD, editors. FMCAD; vol. 1954 of Lecture Notes in Computer Science. Springer. ISBN 3-540-41219-0, p. 108-25, 2000.

[Valkonen et al., 2008] Valkonen, J., Karanta, I., Koskimies, M., Heljanko, K., Niemelä, I., Sheridan, D. & Bloomfield, R. E. NPP Safety Automation Systems Analysis – State of the Art. VTT Working Papers 94, VTT, Espoo. 62 p. 2008. http://www.vtt.fi/inf/pdf/workingpapers/2008/W94.pdf