Tampere University

Prasun Biswas

MODEL-TO-MODEL TRANSFORMATION OF NUCLEAR INDUSTRY I&C LOGIC TO ASSIST MODEL CHECKING

Master Thesis
Faculty of Engineering and Natural Sciences
Examiner: Prof. Eric Coatanea
Examiner: Prof. Valeriy Vyatkin
April, 2020

ABSTRACT

Prasun Biswas: MODEL-TO-MODEL TRANSFORMATION OF NUCLEAR INDUSTRY I&C LOGIC TO ASSIST MODEL CHECKING
Master of Science Thesis
Tampere University
Master's Degree Program in Automation Engineering
April 2020

The demand for electricity has increased proportionately with massive urbanisation and industrialisation. Nuclear energy is a strong candidate which can be one of the solutions to cater to this massive demand for energy. Nuclear resources have the benefit of enormous energy density, low carbon footprint, cheap operating cost and production reliability. Even though it is deemed as a dependable and economically viable option, it is limited by safety concerns, unfortunate accidents can cause monumental and long-lasting consequences. On the other hand, if critically examined, thoroughly tested and flawlessly implemented, the decision-makers can opt for nuclear source. Thus utilising nuclear resources will call for an error-proof instrumentation and control system to observe and ensure safe operation. Model verification plays a vital role in critical analysis of I & C system, it checks all the possibilities the system may reach, and thus provide a model to develop an I&C safety system.

An industrial operation may suffer from unknown component failure and design error, but its safety-critical system must be able to strictly prohibit these undesired events in the system before causing any major accident. “Model Checking” is a mathematical deterministic tool for logic design verification, which has been proven to be effective for detecting design errors in the system. Granted that, Instrumentation and control system starts with logic design at the preliminary phase, a model checking tool can efficiently identify design faults by exhaustive analysis. NuSMV is such a tool, which provides simpler syntax, that can represent the system as logical states with simple data structures. Analysts write SMV files to represent the system available as proprietary non-standard machine-readable diagrams. This thesis proposes an automation step towards diagram import in a verification tool and implements an intermediate data representation.

This thesis provides a perception of various technologies relevant to broader authentication process of a safety system covering from design to verification tools. The state-of-the-art model-checking practice is discussed briefly. Subsequently, a number of logical instrumentation and control diagrams, drawn in Microsoft VISIO tool, are analysed and processed to automatically create an intermediate component network consisting of Function Block elements. A significant effort is spent to partially generate NuSMV code from the retrieved component data to assist the model checking of the system. Finally, the thesis is concluded with a synopsis of the work done and future development scope.

Keywords: Model transformation, Microsoft VISIO, Nuclear power plant, Instrumentation and Control, Function Block Diagram, Formal verification, Model checking, NuSMV.

PREFACE

The work conducted in the last six months to accomplish this master thesis has been an exciting, challenging and fulfilling experience. This would not have been possible without the supervision and support from a few people.

The work presented in the thesis was a part SEARCH SAFIR2020 project. This thesis was carried out jointly in the department of Automation Technology and Mechanical Engineering at Tampere University and in VTT Technical Research Center of Finland Ltd.

I would like to express my sincere gratitude to Professor Eric Coatanea and Professor Valeriy Vyatkin under whose supervision the thesis was carried out. I would like to thank Professor Eric Coatanea for his valuable scholarly assistance that guided me through the various difficulties I encountered throughout the thesis work. I was able to improve the content by a long margin with his constructive criticism.

I would like to express my gratitude to Antti Pakonen, Nikolaos Papakonstantinou and Emmanuel Ory for their constant support. I am thankful to them for the crucial guidance and assistance I received throughout the work.

I would also like to thank my friend Palash Halder, Joe David, Salman Azim and Sonja Vannas. My gratitude to them for helping me through the difficult times I faced during not only during the master thesis work but also throughout the master degree program.

I would like to express my deepest gratitude and appreciation towards my family for their endless love and support I had throughout my life.

Finally, I would like to thank the Department of Automation Technology and Mechanical Engineering at Tampere University and VTT Technical Research Center of Finland Ltd. For supporting me with all essential facilities.

Tampere, 29th April, 2020

Prasun Biswas

CONTENTS

LIST OF FIGURES

.

LIST OF SYMBOL AND ABBREVIATIONS

I&C	Instrumentation and Control
XML	Extensible Markup Language
POI	Poor Obfuscation Implementation (by Apache software)
VISIO	Micorsoft VISIO drawing tool
LTL	Linear Temporal Logic
CTL	Computational Tree Logic
NPP	Nuclear Power Plant
PLC	Programmable Logic Controller
POU	Program Organization Units
IEC	International Electrotechnical Commission
FBD	Function Block Diagram
ST	Structured Text
SFC	Sequential Function Chart
SMT	Satisfiability Modulo Theory
MMT	Model-to-model Transformation
M2T	Model-to-text Transformation
T2M	Text-to-Model Transformation
JTL	Janus Transformation Language
ATL	Atlas Transformation Language
PSL	Property Specification Language
BDD	Binary Decision Diagram
XDGF	XML Diagram Format
VSDX	Visio Diagram File format
SMV	Symbolic Model Verifier
NuSMV	A verification tool based on SMV

1. INTRODUCTION

This chapter outlines an introduction to the work presented in this thesis. Besides providing the context of the research, this chapter clarifies the scope and focus of the work.

1.1 Motivation

The nuclear power plant has some inherent advantages such as the ability to produce a massive amount of energy, low carbon footprint, dependability of power generation. Unlike, the renewable alternatives, rate of power generation from nuclear power plant does not directly depend on weather conditions like the intensity of sunlight or wind, which is essential to maintain a constant base load of electricity demand. The Nuclear power sector has all the required expertise and technology to be used on large scale production. As an example, in USA, nuclear sources account for 21% of the electricity produced whereas coal contributes 41%, natural gas 24%, renewables 12% and petroleum only 1% [1]. There is still a great deal of anxiety surrounding nuclear energy despite the current level of adoption due to the devastating consequence of accidental failure.

The key concerns regarding nuclear energy are safety, security, waste management, and massive initial expense. Due to the huge emphasis on safety, verification of plant's logical operation designed as Instrumentation and control (**I&C**) systems is very important. Instrumentation is defined as the discipline of measurement and control of the industrial manufacturing production process variables such as pressure, level, temperature, flow, PH etc.[2]. Observing and controlling the process variables with the I&C system is a crucial task for a nuclear power plant [3]. "Formal verification" method is a reliable technique to find out glitches in the I&C safety system.

The nuclear I&C diagrams are often first drawn with generic drawing software, a non-standard design tool that lacks modelling capability, and transferred to dedicated development tools later. The development tools are not usually based on standardized programming languages, but vendor-specific solutions. The thesis will play a role to process such nuclear specific non-standard safety I&C diagram to create a model of the components and their input/output connections. The generated model of storing

1.2 Research questions

Model-checking technologies were initially developed during 80’s decade of the last century to fill up the vacuum of reliable verification methods. Model-checking went through a rigorous transition to be established among the foremost tools of logic-check in hardware and software engineering. This bridged the gap between theoretical proof and practical engineering application. Nuclear safety I&C system verification is one such instance of model checking use case. The key objective of this thesis is to assist model checking of non-standard I&C safety diagrams by transforming those as a step towards importing the functional diagrams into a model checker for verification.

The I&C diagrams under research consideration are non-standard Microsoft VISIO files. from which, VISIO shapes with certain features needed to be extracted. From the shape data, Function block components required to be modelled to integrate with a model checker in future. Features of those modelled Function blocks should follow a meta-model. Typically, meta-model can be regarded as a guiding structure for process, rules, constraints, models etc. For this instance of the VISIO I&C diagrams, a meta-model is a guideline for attributes and relationships of processed diagram components.

To form a concrete research structure, the research questions are developed in a few steps. First step: Identify vision and achievable goal. Second step: enlist probable obstacle. Third step: breakdown current problems and probable solution. Final step: use “first principle thinking” to reason up and build an effective and creative solution. Initial case studies led to the emergence of the following research questions that are meant to be answered in this study:

Question 1: How to process machine-readable non-standard application logic design for I&C safety system (in Microsoft VISIO file) in a programming environment?

Question 2: How to create a meta-model that helps flexible integration of different tools where I&C diagram is part of the process?

Question 3: How to build model instances by performing transformation of VISIO I&C diagram to follow a scalable schema presented as a meta-model?

Question 4: How to build a visualization of created function blocks by diagram transformation to prove that layout of the original diagram is protected, which can also help to visually Identify fault in transformation?

Question 5: How to test the scalability of model transformation from the source Visio diagram by creating a use case where the input for a model checking tool is generated from processed I&C diagrams?

1.3 Goals and contributions

The main focus of this master’s thesis is to develop a transformation methodology to model functional components contained as an I&C system and its functions derived originally from a non-standard nuclear specific function block diagram. In this context, function means operations represented by the shapes in VISIO I&C diagram which could be as simple as logical AND or more complex ones symbolized with various grouped shapes. The shapes have interconnection, which altogether forms a network. The modelled functions should conform to a meta-model. The meta-model should be formulated in a way that will be able to represent all the logical units and relevant properties of the I&C diagram. The functions should be preserved as intended in the original diagram along with the connection and layout of the components.

1.3.1 Achieve automation

This thesis is focused on the development of a program to import nuclear safety I&C diagrams which exist as Microsoft VISIO files. However, Microsoft VISIO is only used as a visual representation tool for drawing the logical I&C diagrams while it lacks any capability of analysing, testing and simulating the I&C models. To verify the I&C diagram an analyst still has to use verification tool such as a model checker and manually construct the function blocks. For a graphical model-checker tool, redrawing the whole grid of I&C logic is the current way in practice. The thesis intends to process the elements from VISIO diagram to identify and model function block elements which can be used to create an importable file in a format supported by model checker to automate the process.

1.3.2 Improve procedural efficiency

The current practice of verification of the I&C diagram by model checking is a time consuming work due to the manual nature of the process. A graphical interface for model checker create a description of the I&C system (with model checker such as SPIN, NUSMV etc.) and then define specification against which the correctness of the model has to be tested. If no such tool is used, writing description of the model in the syntax of formal model verification language is an even more labour-intensive task. This manual process of description also has more chance of introducing error in coded definition and specifications. Any partial or full automation of I&C diagram import to a verification tool is a step up in the direction of faster and error-free verification. Leastwise, automated script generation for a suitable verification language will be an improvement of the overall efficiency in the procedural steps.

1.3.3 Facilitate an extensible model

In the case of NPP specific I&C diagrams, standardized and analysis capable modelling tools are not necessarily used in the incubating stage of the design process. When a diagram is created with a generic illustration tool (e.g. Microsoft Visio), drawing error can be introduced besides the errors in logic. As a step towards flexible integration of multiple tools involved in the verification (e.g. drawing and model-checking tool), development of a meta-model as standard is essential to maintain uniformity of the model during transformation. The developed meta-model should be extensible to fit in further development endeavour. Moreover, the codebase for diagram processing and model transformation should be modular and extensible to easily incorporate identification and modelling of new components in a diagram. During the development, identification of the I&C function blocks and connections from the source Visio diagrams is a key focus. The identified components from the diagrams follow the developed meta-model, where the layout is preserved to be imported and integrated into a graphical model-checker for verification. Extensible nature of the program and meta-model is an added benefit.

1.4 Research Methodology

Prior to stepping into the implementation part of the thesis, theoretical background study was completed to grasp the general idea which was particularly helpful to formulate research questions and pinpoint the goals of the research. After the initial study, scientific literature related to state-of-the-art were studied to gain a better insight into the relevant technologies. The following topics were studied as a part of the literature review in the scope of the thesis.

- Instrumentation and control system in industrial application.
- Model-to-model transformation approach and tools.
- Special model transformation tool vs general-purpose programming language.
- Formal verification by model checking and related frameworks.

After the upper mentioned studies and primary research of several viable tools, a framework was proposed and a development strategy was finalized. At this point, potential tactics were envisioned, a roadmap was proposed to allocate effort to the specific parts of the whole task of research and development. Than trial of the tools were conducted to check usability by selecting a few basic diagrams and programming languages as a testbed.

Following points are a brief presentation of key parts of the implementation.

- Most suitable programming tool selection for the purpose.
- Analysis of I&C diagram and creating a proof of concept
- Creation of prototype Function Block identification program
- Experimentation on automated NuSMV script generation.
- Scaling up the capability to handle larger diagram.

1.5 Thesis outline

After the introduction in this current chapter, a description of the required theoretical concepts is written in **chapter 2** containing the overview of I&C architecture, formal verification language and tools, PLC program standard and model-to-model transformation.

Chapter 3 describes state-of-the-art in the field of model transformation and formal verification. Considerable examples of attempts to combine model transformation and formal verification is discussed.

In **Chapter 4** the research methodology is discussed briefly. Several methods that were undertaken to handle the nonstandard I&C diagram processing is mentioned.

Chapter 5 expands on the actual implementation process. This chapter covers the concepts behind software architecture of the implementation, class design, technical choices made throughout phase by phase.

Chapter 6 contains the illustration of the achieved result in this process. A comparison between result goal to obtain at the primary phase and actual achievement from the development in terms of accuracy performance is written in this chapter.

Chapter 7 is the summary of the implementation. Future possible improvement scopes are also discussed briefly. This chapter followed by references.

2. THEORETICAL BACKGROUND

This chapter presents a brief overview of the theoretical ideas that are relevant to the span of the thesis. It intends to provide with primary knowledge on the terms and concepts required to get an insight into the work done for the accomplishment of the thesis.

To start with, section 2.1 introduces the concepts of instrumentation and control (I&C). Section 2.2 unfolds the idea of model-checking as a way of formal verification. The tool of choice “NuSMV” for model-checking is also explained briefly. PLCOpenXML as a standard and possible utilities in the work scope of the thesis is explained in section 2.3. Finally, section 2.4 briefly discusses some basic concepts of model transformation and provides examples of such tool.

2.1 Instrumentation and Control

Instrumentation is an umbrella term for indicating, gauging and recording physical quantities with measuring instruments [4]. s the term implies, an instrumentation system consists of diverse instruments beginning from very simple thermal sensors to very complex GPS sensors.

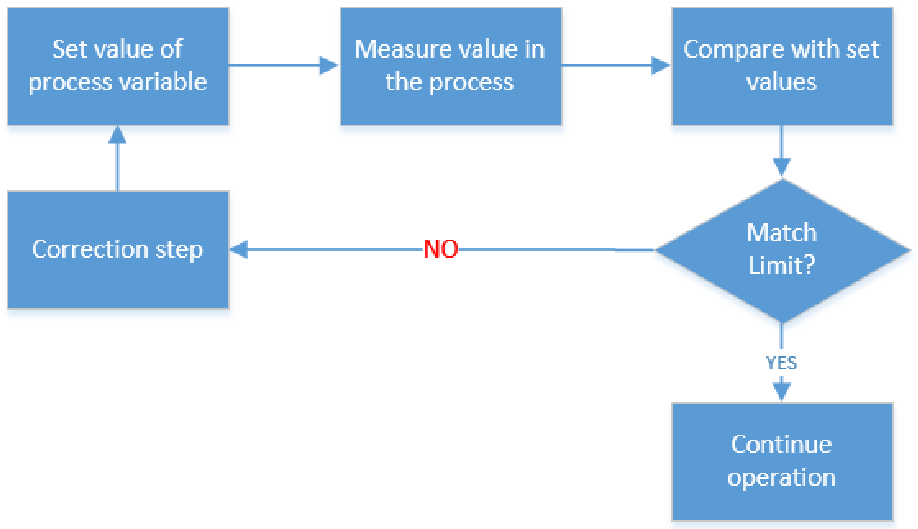


Figure 1. *Basic steps of a typical control system*

Control system encompasses a broader scope of systems that commands, leads, manages, or regulates the behaviour of other devices or systems using control loops [5]. I&C is a body of engineering that deals with the measurement of field parameters through a myriad of sensors and controls actuators depending upon defined operation condition and safety functions. Usually, an I&C system is a combination of electrical, mechanical/pneumatic and electronics devices. I&C systems have been used in a broad range of applications such as household electromechanical devices, automotive, chemical, laboratory, aircraft, manufacturing industry, power plant etc.

Accounting for the consequence of the Nuclear Power Plant (NPP) failure, “safety” becomes top priority by default. Hence, I&C architecture is established in the core of the NPP logical design approach [6].

2.1.1 History and evolution of I&C system

Historically Instrumentation and Control have several phases of evolution starting from the pre-industrial era towards the era of industry 4.0. It is considered that James Watt invented the first I&C system in a very basic form. This was a measurement system of engine speed through rotation of two metal ball attached to flywheel of the engine coupled with a steam control valve to restrict the speed. The first industrial revolution, driven by the steam engine, kick-started initial development of I&C system. The second industrial revolution propelled electricity and mass production of automobiles, cranked up the development of I&C system at the beginning of the 20th century.

For a long period sensors were mechanical which started to change slowly with the invention of electrical instrumentation devices. I&C was enhanced by pneumatic measurements. The third industrial revolution transformed the electrical and electronic apparatus. For example, an advanced measuring system could take input from a variety of thermocouples and delivered linear outputs [7]. It was possible to remotely mount indicator in large mimic panels and to collect data on paper chart recorders. This unlocked a new world of small analogue electronics for sensor design and signal processing [8].

Although this development did not preclude the prolonged use of an analogue interface e.g. 4-20mA, it would be true to indicate that the old technology could now be slowly decommissioned in favor of newer development [9]. All instrumentation that is integrated into the digital supply chain will need to have some technology embedded that provides not only configuration parameters, but connectivity. Adopting configurable systems would be a safe wager as the size and power of embedded processing coupled with its reduced expense, even though It is not easy to predict the protocols required.

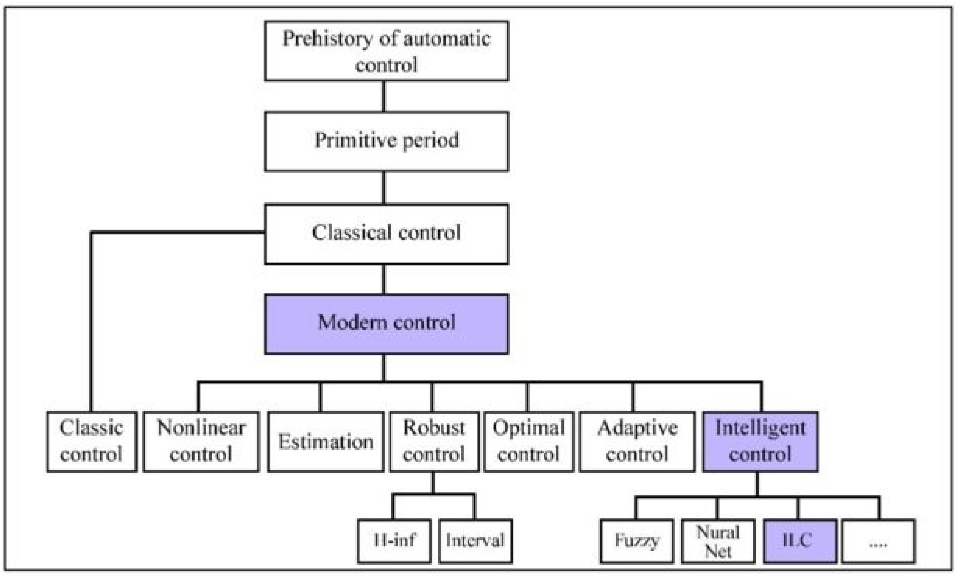


Figure 2. *Different era of I&C system development [10]*

Since the extent of obtainable data surged, and the estimation was that the growth would be exponential, a reduction of raw data transfer became vital. Communication bandwidths were still very limited, especially in case a wireless radio network was considered. Edge computing came to rescue with the power of pre-processing of raw data into manageable packets. The power of embedded processing was increased which provided an instrument with the ability to run a configurable data compression algorithm in its own edge-processing. Later there was a big leap in advanced transmitter system that can be used to control other instruments. Sending data directly to a computerized controller (PLC DSC, SCADA system etc.) was easier than before, where data can be interpreted into readable values to control other system components and process [11].

2.1.2 Significance of the I&C system

A physical operation on an industrial scale, such as factory, mining or power plant, is intense and complicated. Along with performance and cost-efficiency, a crucial focus is the safety and reliability of the operation. Complexity arises from the large-scale use of physical devices and assets like sensors, processor, actuators, boiler, reactor etc. To ensure safety and dependability of plant, I&C system is put in operation. Conceptual modelling, detailed design, troubleshooting and maintenance are some of the vital segments of I&C engineering solutions to establish safety credibility. Modern I&C systems are capable of very accurate measurement and control. Additionally, the systems can prevent system overloads and detect problems in equipment during runtime. They are also capable of performing complex mathematical calculations that ensure smooth efficient operation of the units.

2.1.3 Architecture of a typical I&C system

I&C architecture contains field devices, connecting components (for communication and operation), processors etc. Field devices belong in the front line of the instrumentation control loop to measure process variables (e.g. temperature and pressure sensors). They are individually connected to separate cable leading to a junction box. Individual cables exiting from multiple junction boxes are re-connected to multicore cable if required. These are connected to the interface of marshalling cabinet which has important functions of cross-wiring between a various number of field signal input and channels. In case of voting application (ex: 2 out of 4 inputs) of safety devices, I/O signal are required to be allocated in different slots to avoid common fault cause. Often, I/O modules and controllers are somewhat proprietary arrangement own by the vendors. So, cross-wiring in the marshalling cabinet is managed according to vendor requirements. Finally, the flow connects to a superlative controlling centre like system cabinet, devoid of any more difficult arrangement in between [11].

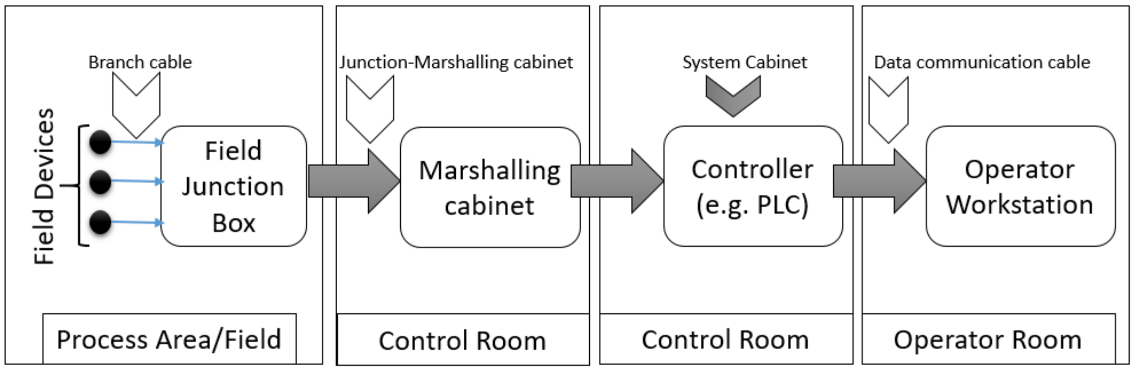


Figure 3. *Usual communication of a I&C system [11]*

2.1.4 I&C in Nuclear Power Plant

The I&C system of the high-temperature reactor consists of the instrumentation, control equipment and safety protection systems [12]. The system and plant operation personnel together function as the main operating mechanism of a nuclear power plant. Several very important actions (e.g. basic parameters sensing, performance monitoring, information integration, automatic plant adjustments etc.) are performed by this system. To perform these operations I&C system is divided into various operating layers or segments such as equipment, modules, subsystems, redundancies, systems, etc. [13]. Each of these layers consists of a variety of sensors, electromechanical actuators and computing components. Plant operation personnel can understand the health of NPP by analysing the accurate and appropriate performance data provided by the I&C system.

It also responds to failures and off-normal events, thus ensuring the goals of efficient power production and safety. If an abnormal condition arises, the I&C system can limit operation to go into a safe shutdown state. As changed mindset and government compliance put efforts to make safety as the first priority against profits and production I&C is becoming an integral part of safety-focused industries. Nuclear I&C has both automatic control logic along with manual input is a number of cases. So, in short, NPP has three major functions needed to be done using I&C system: measurement, regulation and protection to enable a safe and reliable power generation [13].

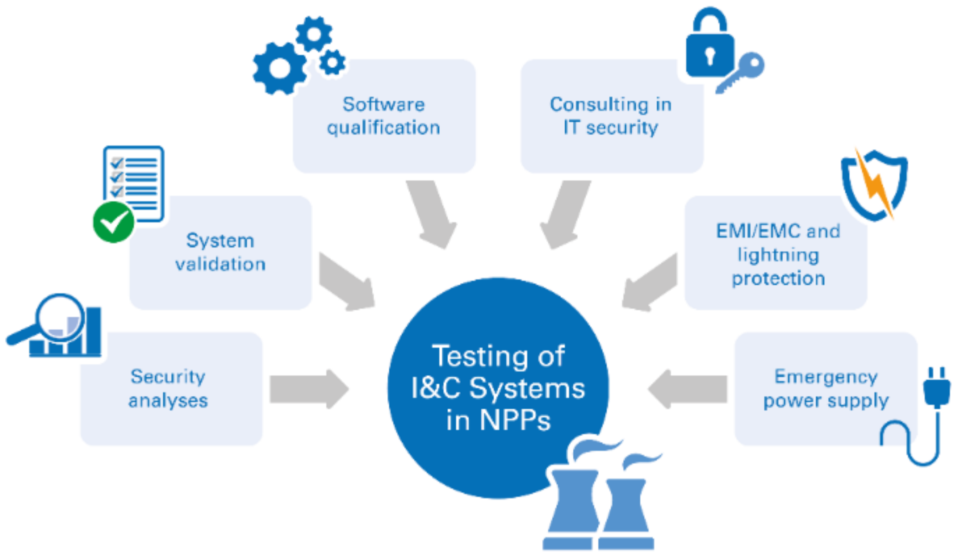

Software qualification
Consulting in IT security
System validation
EMI/EMC and lightning protection
Security analyses
Testing of I&C Systems in NPPs
Emergency power supply

Figure 4. *Aspects related to Nuclear I&C system [14]*

An adequate and optimized design has certain overall benefits such as [13]:

- Minimized cost of construction (e.g. lower cable runs).
- Lowered cost of the life cycle (e.g. adaptability with modern standardization).
- Operation optimization with engaging improved technology
- Protection of investments by limiting uncertainty and defence from accidents.

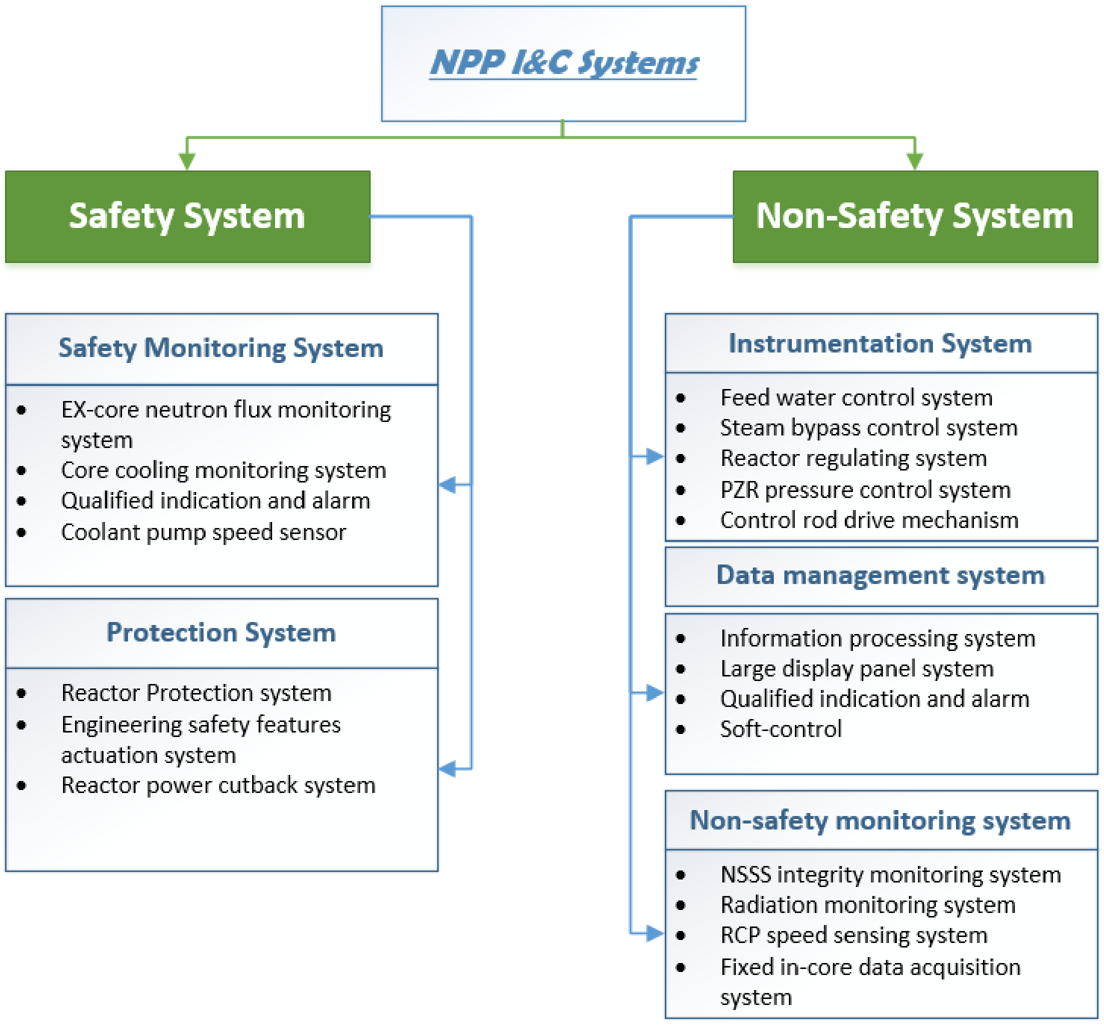

NPP I&C Systems
Safety System
Non-Safety System
Safety Monitoring System
EX-core neutron flux monitoring system
Core cooling monitoring system
Qualified indication and alarm
Coolant pump speed sensor
Protection System
Reactor Protection system
Engineering safety features actuation system
Reactor power cutback system
Instrumentation System
Feed water control system
Steam bypass control system
Reactor regulating system
PZR pressure control system
Control rod drive mechanism
Data management system
Information processing system
Large display panel system
Qualified indication and alarm
Soft-control
Non-safety monitoring system
NSSS integrity monitoring system
Radiation monitoring system
RCP speed sensing system
Fixed in-core data acquisition system

Figure 5. *Categorization of I&C tasks in NPP[15]*

2.2 Model-Checking

Many of the devices that we see around us have a controller that is loaded with codes to operate the devices. For example, a controller in a water pump can regulate the flow of water, turn on or off depending on the input parameter such as water level. There are a massive number of such parameters that are taken into account to operate a NPP and the operation reliability of the I&C system can be examined for fault using a technique named “model checking”.

To justify the reliability of a control system it is essential to satisfy a set of requirements. As a manual approach to check the fulfilment of the requirements, one can set test cases and check the output. However, when the number of connected components increases, performing such manual verification becomes increasingly difficult. Here, Model-checking presents a better alternative solution for analysing the reliability.

Model-checking is a mathematical model based verification technique developed for formal verification of the correct functioning of a system's model against its required formal specifications [16]. In this approach, a mathematical model of the system needs to be constructed and the requirements need to be written in a formal notation. The model is a translation of the physical system and it can be declared as "verified" when the requirements defined by formal notation are satisfied. If not satisfied than a counterexample is generated by the tool which is used to trace the error in the system. it eliminates or at least minimizes the unnoticed error which might occur in manual verification.

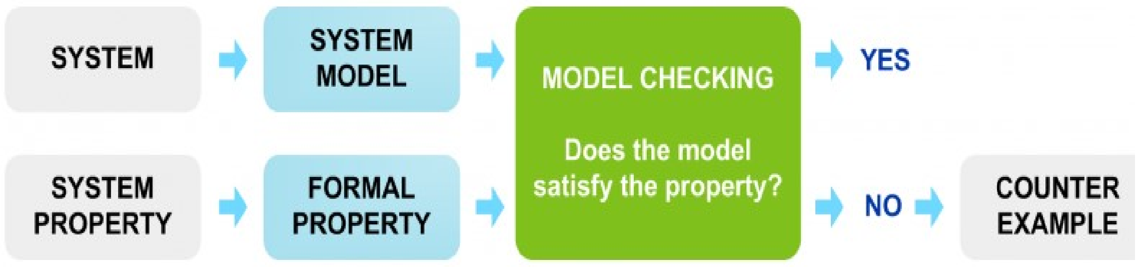


Figure 6. *Model-checking of a system against requirements [17]*

One advantage is the availability of automated tools to verify a formulated model against requirements. Model-checking is applicable for both software and hardware by spending a very reasonable amount of effort to exhaustively validate safety-critical systems and still be tremendously effective, especially in the design phase of the I&C system.

2.3 Programmable Logic Controller

Programmable Logic Controller (PLC) is a special-purpose control system that works as an industrial field computer to digitally operated equipment such as machines, switches or other controllers. Before PLC, relay and cam timer-based electromechanical industrial control systems, were slow, cumbersome to deploy, hard to rewire to execute changed requirements. The inflexibility and maintenance difficulty was addressed and led to an alternative solution. To substitute these earlier controllers PLC was developed. This is a very high-speed device that can perform a fast scan cycle, can handle a large number of inputs/outputs, can interface with another controller and easy to reprogram.

2.3.1 PLC program and hardware structure

The IEC 61131-3 defines a model of memory and program that follows modern software design features [18]. This model integrates concepts such as structured programming, top-down design, formal software interfaces, hierarchical organization, and program encapsulation. PLCs usually consists of a microprocessor (CPU), memory and IO peripherals. In addition to that, PLC has a programming interface. This can be programmed with multiple supported programming languages, and programs are stored in the non-volatile memory. Figure 7, shows the architecture of a PLC.

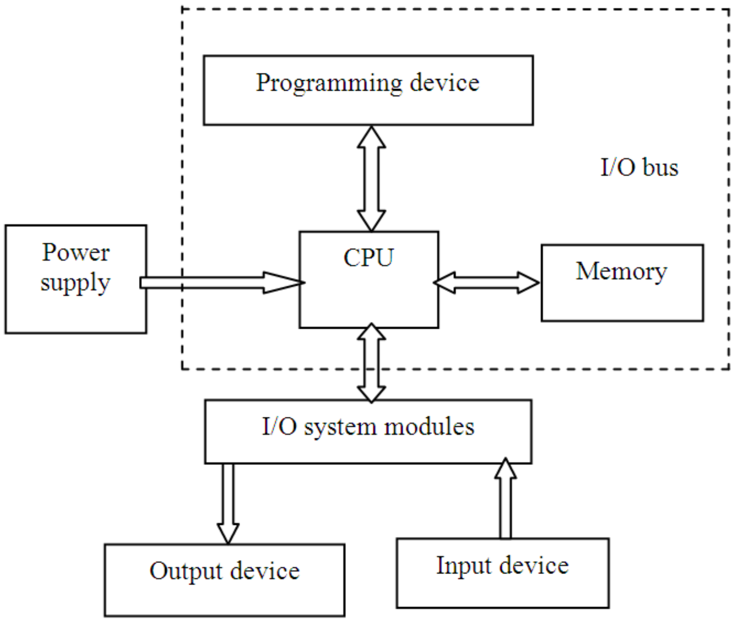


Figure 7. *PLC hardware structure[15]*

Sensors, actuator or controllers are connected through the ports with PLC, PLC controls outputs based on the input by sequentially executing instructions sets on each CPU. Typically, the interconnection of multiple function blocks makes up a program also referred to as Program Organization Unit (POU). Besides the function blocks, the program holds declarations of physical inputs/outputs and any variable local to the program. A program can read and write to peripheral channels, global variables and also can communicate with other programs [18].

There are three categories of POUs: functions, functions blocks, and programs each of which has a declaration part. The simplest type of POUs is a function. They accept input parameters and return an output value without retaining data. External and global variables declared outside its own POU are not accessible by function [18].

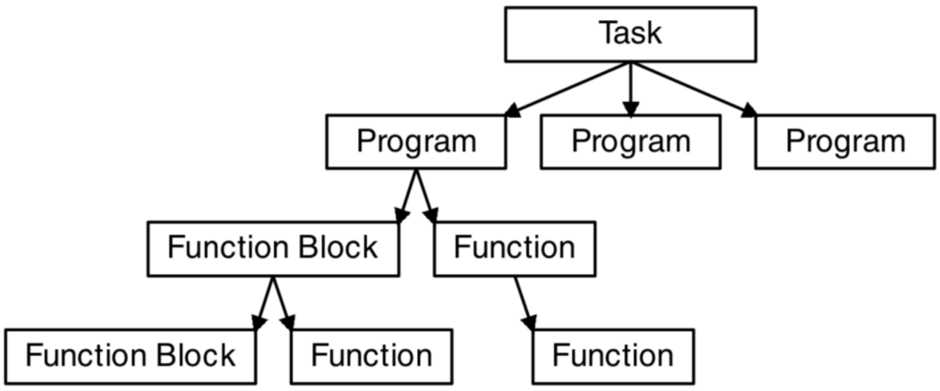


Figure 8. *PLC program structure[18]*

Function blocks POU can be regarded individually as both a function and as an object when in comparison to the object-oriented programming language as a reference. Function blocks can access external/global variables and can hold static variable values that are not lost after the execution of a block. Main POUs in PLC is a program which has major similarity to function blocks. The main distinguishing factor is that in programs global and external variables can be declared while variable declared in programs can be allocated to a physical address, e.g., the memory address for PLC inputs and outputs.

2.3.2 IEC 61131-3 and PLCOpen XML

A common standard for the PLC programming language, IEC 61131-3, was published by the International Electrotechnical Commission (IEC) which is an international standards organization for all fields of electrotechnology. The standard was an important step to solve early day's problem of vendor-specific PLC program and serves as a guideline for PLC programming. Five programming languages, which were in use before, were selected for adoption by the standard and PLC manufacturers were suggested by IEC to conform to at least one of those recommended languages. These languages are instruction list (IL), structured text (ST), ladder diagram (LD), function block diagram (FBD) and sequential function chart (SFC) IL and ST are textual programming language and the rest are graphical programming language.

But languages are only a part of a complex development environment consists of simulation, debug version control and documentation tools. Since the release of the standard, developers wanted the ability to exchange programs, libraries and projects between development environments. PLCopen, an independent organization, had decided to develop logical and graphical interfaces towards all these tools. The primary goal is to transfer a control project according to IEC 61131-3 standard with minimized additional work, from one development tool to another without losing information even if it's incomplete or contains an error. It allows the migration of a software project between different hardware platforms [19]. It uses XML as the technology. An XML file can be generated by numerous modelling and simulation tools and enhanced by verification, documentation, and version control tools [20].

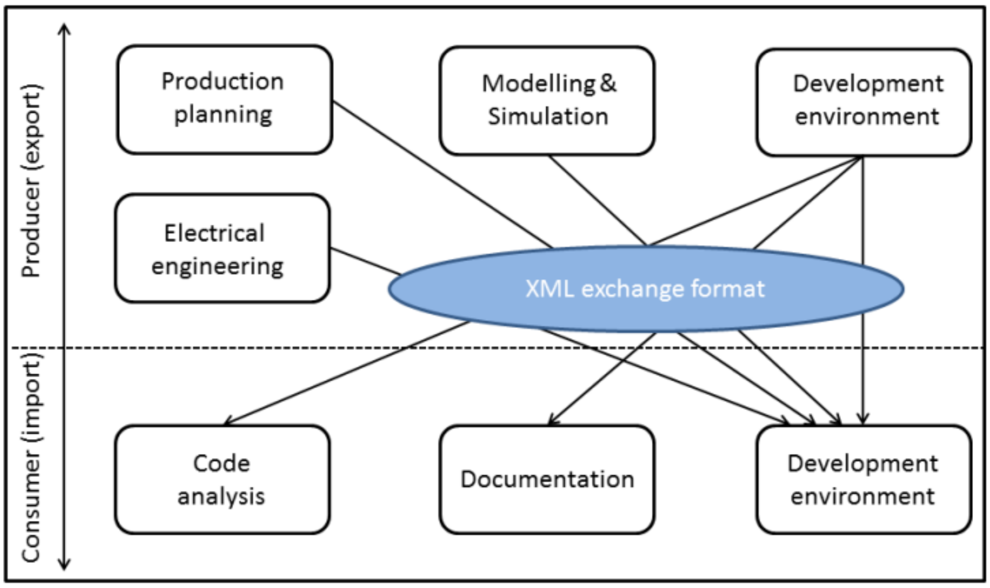


Figure 9. *Exchangeability representation in PlcopenXML [19]*

In this thesis, the provided diagrams for analysis are a representation of I&C safety functions having similarity with the structural representation of IEC 61131-3 function block diagram (FBD), despite having differences in practical applications. FBD is a type of graph where the variables, functions or function blocks are presented as nodes and the edges represents the connections between the variables in the graph and the input/output variables in POUs. In practical design, often FBDs are divided and distributed into several networks which makes it easier to structure the control flow of POUs.

2.4 Model transformation

The term “Model Transformation” can be very subjective but in a generic concept, this is an automated process of creating a modified model from an existing model. The prime utility of model to model translation considered to be making sure that a set of generated models is consistent with a set of definition of the targeted model. Automation of the model creation has the main purpose of saving vast time and effort while minimizing the induced error where it’s possible. There are various classifications of model-to-model transformation usage and approach of model transformation. However, the basic procedure is a program that will take at least one input model for processing to generate another model, which follows some transformation rules. The output of model transformation can be of several different models depending on automation process demand. The transformation rules should be developed in a structure to perform the transformation. In that process, the generated model must follow a standard “meta-model”. The meta-model is constructed with strict requirements that guide what the features and attributes of the generated model will be [21].

The following figure 10 10 is an example of research done, where a model transformation tool (ATL) is used to transfer data that follows a book meta-model and transferred into data that follows a different publication meta-model. After the first stage of transformation, the generated model data is then transferred to a format suited for web publication using another model-to-text transformation tool (Acceleo). This example shows two stages of a one-directional model transformation use case.

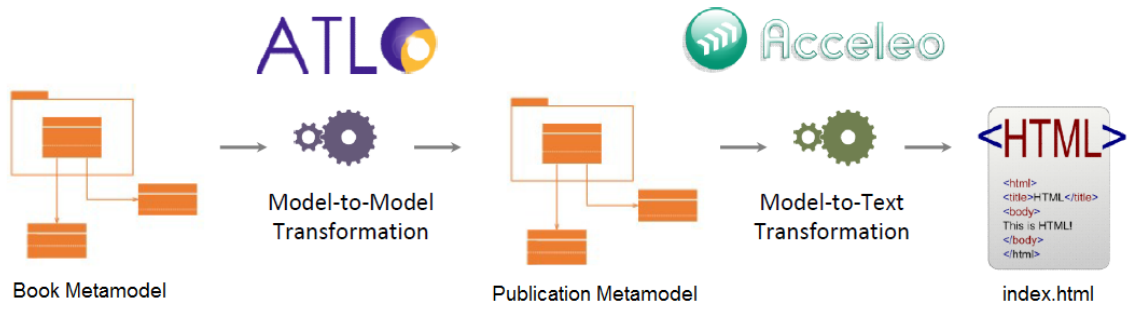


Figure 10. *Example of a practical Model transformation [22]*

3. STATE OF THE ART

This chapter represents the insight gained after reading the publications on works done in the vast field of model transformation and formal verification by narrowing it down to the related topic of this thesis. Of those topics, major concepts are presented as a snapshot from the researches. The relationships among those researches were portrayed with the endeavours to combine the capabilities of model transformation and formal verification techniques. Improved understanding of the broader scope helped me to develop The state-of-the-art of this thesis, presented in the last part of this chapter.

3.1 Model Transformation: Formation and Flow

Model-driven engineering (MDE) is a broad field of evergrowing development that, in basic building block, creates a system by utilizing models. Several motives behind MDE are growth in productivity, simply design process, minimize communication complexities, increase inter-system compatibility. The overall process can benefit from increased efficiency by leveraging automation as a structural part of Model-driven development (MDD). Here, Model-to-model transformation (MMT) comes in to play to support a large spectrum of key aspects involved in digital data-driven activities. To mention a few activities optimization, abstraction, analysis, migration, refactoring, synchronization and debugging can reap the benefit of MMT. Model to code generation is a good example of MMT use case but not limited only here [23].

The remarkable development of standard and language to enlarge the scope of MDD and MMT has resulted in wide industrial adoption (e.g. Ericsson, Airbus). Over 60 software tools and frameworks have practical existence in the field of both academia and business. Most of those tools were developed to serve different requirements, hence, carries different abilities and also limited in this sense. To rightly choose the perfect tool for practical use, one has to get a deep insight into their capabilities and compare the variation of transformation approach within those tools. Researcher [23] undertaken analytical steps to define the criteria of classification in many facets.

In earlier research on the nature of modelling, a model was described by Rothenberg et al. [24] as an abstract manner of dynamic representation of reality on which many experimental operations can be conducted. Aiming at the operations, many languages were developed that varies in abilities. Brambilla et al. [25] divided those into two categories: General purpose modelling (GPM) and Domain-Specific Modeling (DSM) languages. Connotation and combination of elements of a model defined by the semantics of these languages. Structure and elements of a model, guided by a meta-model, can be defined by abstract syntax of any modelling language. Relationships among elements of a model are represented by concrete syntax. The Concrete syntaxes can be graphical (e.g. Sirius, Graphiti) or textual (e.g. EMFText, Xtext) but a combination of both is not unheard of. Even though only meta-model based tools were in the focus of the state-of-the-art studies for this thesis but, there are non-meta-model based tools too.

Model transformation in Six Facets: For careful evaluation of the tools, six facets were developed by Nafiseh et al.[24] categorized a number of tools based on their primary field of utility. Those facets of requirements are general, model-level, transformation level, user experience, collaboration support, and run-time. Impartial assessment regarding these six facets is conducive to find the right tools for MMT.

- The key consideration of the "general" category is not technical, mostly business and performance-related miscellaneous aspects are taken into account. For example, security, resource support [26], technical support and licensing issues belong in this category.
- The aspects related to modelling are examined in "model-level" facet. Finding DSM or GPM language support for a tool by assessment of specific needs of the model and meta-model is the major focus. The assessment is based on activity, use case, package & components, communications, compatibility with standard/specification etc. [25].
- The traits of model transformation languages are included in the "transformation-level" facet. By studying language syntax, input/output type support, cardinality, rule organization, validation, direction, traceability, concurrency etc. are analyzed under this category.
- User experience is a very important consideration in utilizing a tool or language to ease the process of model transformation. Considering based on this facet selection of a tool would depend on the user interface, syntax editor, workspace & project management, programming style and level of automation [23].
- In order to facilitate migration and extendibility of a tool "collaboration support" facet evaluates a variety of tools based upon following notions: Reusability technique, interoperability, extensibility, teamwork support for multiuser etc. How a selected tool would be scaled up in size and scope is depended on these factors.

- The final facet of tool selection is based upon features and aspects related to the runtime environment of a tool. Operating system platform, execution environment, execution model and external dependencies of a tool is carefully examined for final selection, all of which are very crucial for smooth operation a model transformation process.

To obtain a general idea of how each category of tools work it is important to be familiar with the underpinning approaches that are utilized by the tools. Many research and development initiatives to perform specific model transformation needs produced various MMT software and transformation approaches. The terminology “transformation”, encapsulates broad concepts but to be specific, based on criteria of transformation data type three following categories can be formulated.

- **Transformation for abstraction:** This is a transformation that aims at the creation of an abstract behavioural model obtained from the base model. Usually overall behaviour portrayed by this model from the requirements while refraining from implementation details of the original model [27]. This transformation typically regarded as a part of “refactoring” process where the transformation from a complex and refined model to an abstract model can be very challenging.
- **Transformation for model creation:** Creation of a new model by a transformation of an original model is typically a straightforward process in terms of complexity of transformation approach. In general, there is synchronization between source and target model because both models preserve the same data while the model is formed with different attributes. A concrete mapping of properties between models can be helpful to achieve bidirectional transformation in this approach [27].
- **Transformation for code generation:** Execution this type of transformation can be characterized by implementation semantics association from source model and target code. During implementation at primary stage scope of both source model/language and target language are defined. After that, the translation process to create the target language from the source model is described where semantics are preserved in the translation mechanism [27].

Typically in high level, a depiction of a model can be basic text or another complex model which can divide MMT tools in three ways: (i) Model-to-model (M2M), (ii) Model-to-text (M2T), (iii) Text-to-model. First two categories of tools (figure:11) are relevant to this thesis and hence, were subject to thorough study.

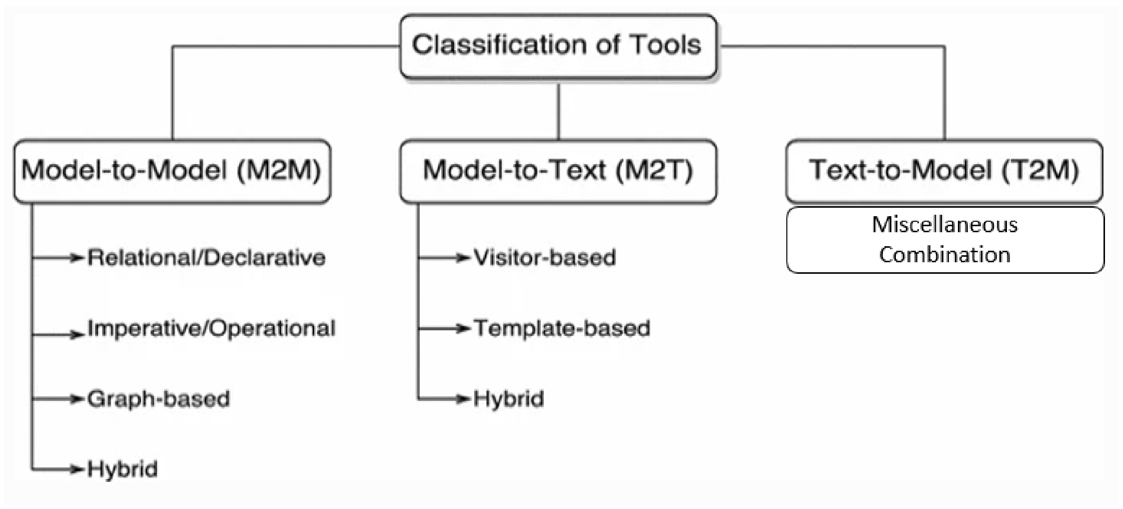


Figure 11. *Classification of tools based on input and output*

To convert a model to another, M2M transformation mechanism follows certain approaches. One favourable way is to directly map the relationships that occur between input and output elements. This is denominated as “relational approach” which is widely used in tools such as UML-RSDS, JTL, QVTr [23][28] etc. Moreover, Some such tools (in low level) utilize pattern matching within an array of variables. Instead of focusing on relations, several tools (e.g. MetaEdit+) uses a chronologically executable list of rules focusing on “when and how” the execution has to be conducted. This is called “imperative approach”. In addition to the mentioned approaches, there are many tools (GReAT, GROOVE etc.) developed based on algebraic graph transformation [29]. On a model as an input graph, some predefined transformation rules are executed on each vertex/edge (represents component and relations) to generate an output graph model. Negative application rules handle prohibited conditions. Beside the mentioned approaches some tools (e.g. VIATRA) also use a hybrid of all approaches. M2T transformation usually follows “visitor-based approach” or “template-based approach” but sometimes hybrid of these two. In tools (e.g. Kermeta2) developed on a visitor-based approach, each element belongs to a tree/graph structure and a predefined pattern of text is generated if a node is visited. On the other hand, in a template-based approach what text need to be generated is described as a template. If a model matches the template then a text output is created. Example of such a tool is Acceleo while MetaEdit+(MERL) is an instance of a tool that uses a hybrid of both approaches [23].

Repository, query and comparison: In the case of industrial use, large models needed to be preserved in a repository. Eclipse Modeling Framework (EMF) is an example of such a repository which was developed using Java Modeling Framework. There are other frameworks too but some commercial tools (e.g. MetaEdit+) prefer to use own framework. When a model is placed in a repository an essential requirement is to be able to find a model by querying. In many cases, such operation is conducted based on object constraint language or pattern of graphs. After the requirements of storing and searching models is taken care of, developers can now compare and manipulate the models based on specific needs. Usually, a comparison is done to find resemblance and variance of the model's attributes. MetaEdit+ is one example that supports model comparison [25][30][31].

Interoperability of transformation languages: The notion of Interoperability is referred to as the capability of enabling information transfer among multiple systems that belong to separate domains [32]. In advanced use cases, interoperability aims at generating transformation programs in a target language by translating from the source. Obtaining interoperability facilitates the goal of achieving extensibility of an MMT system architecture. Quest of achieving interoperability resulted in the development of framework standard such as Query/Views/Transformation (QVT) which aims to find compatibility within MOF, UML, OCL etc. One way to achieve the goal is to create an interpreter to produce target language from the source language. If more than two languages are involved in the scenario than creating a common intermediate language is a viable direction. In experiments, a satisfactory level of accuracy was achieved in the case of languages with similarity in structure. One such example is the translation to Operational Mapping from ATL [33].

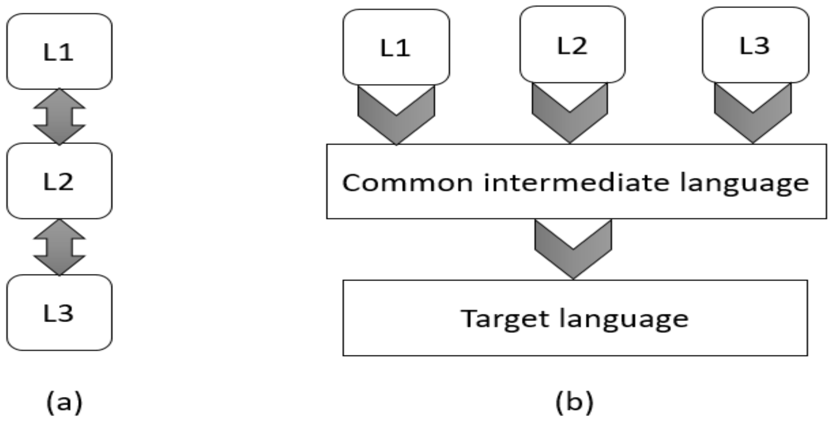


Figure 12. *Direction and intermediate steps in the transformation.*

However this a very diverse field that encompasses a substantial variety of tools and languages. During translation or merger effort, many conflicting conditions need cautious handling where complexity increases with the large scale and variation. So, it is not unexpected that a general unification of tools to gain interoperability is not highly probable

[34]. But, on a narrower scope obtained interoperability and extensibility can enable distributed and saleable transformation solution [35].

Reuse, consistency and correctness: Depending on the direction of transformation, unidirectional and bidirectional transformation are two basic categories mentioned in section 2.4 with an example of two-step unidirectional transformation. If the transformation always receives one type of input to generate one type of final model as output that the process can be considered unidirectional. However, the bidirectional model transformation is more capable and more complex. In this approach, one model can be either input or output and will produce the final output. In this type of transformation, there must be a consistency of crucial relationship rules, which is followed by both models, framed by the meta-model. Alexander et al.[36] researched on consistency and completeness which has key motive to fine-tune the transformation. There are many deciding factors upon which performance can be measured. The priority can be on semantic correctness, syntactic accuracy, robustness and determinism assessed based on the cardinality of the system. There could be many ways to keep track of the performance such as multiplication of factors (score= ”elements” x “no. of completion” x “no. of error”).

Alexander et al.[36] also researched on the utility of human guidance in the transfer process. But, most large models are data-intense, complex and sometimes obscure. So better possible, if the model transformation can be configured dynamically. IBM was successful to develop one such engine and acquired a patent for it. Once the interoperability, correctness and consistency are in place it opens up the possibility of easily modify, transfer, transform and reuse of models.

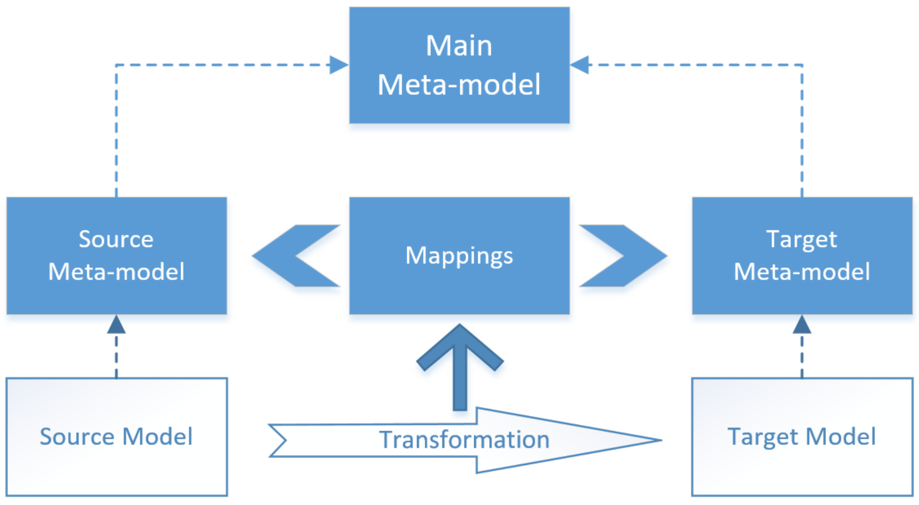


Figure 13. *Basic rules of automated model transformation*

Model transformation approach:

A system has some specific properties and behaviours to function which can be presented in an abstract form referred to as "model". A meta-model represents the core nature of properties that a structure of a concrete model must follow. If meta-model is considered as a program class, a model will be equivalent to its instance [37]. In the context of model-driven-engineering, model transformation is a technique to modify and build models in an automated manner. Here, model-to-model transformation (MMT) will be a bridging technique or a frame to map input model to target output that can be another model, source code etc. [38]. It is very essential to pick the object's behaviour perfectly for concrete modelling of a system.

Here comes another categorization of transformation, endogenous and exogenous. If the same meta-model is followed by both source and target model, then the transformation is referred to as endogenous. Whilst in exogenous type, each of the source and target model complies with separate meta-models [39]. Reverse engineering and code generation would be good example use cases of exogenous transformation.

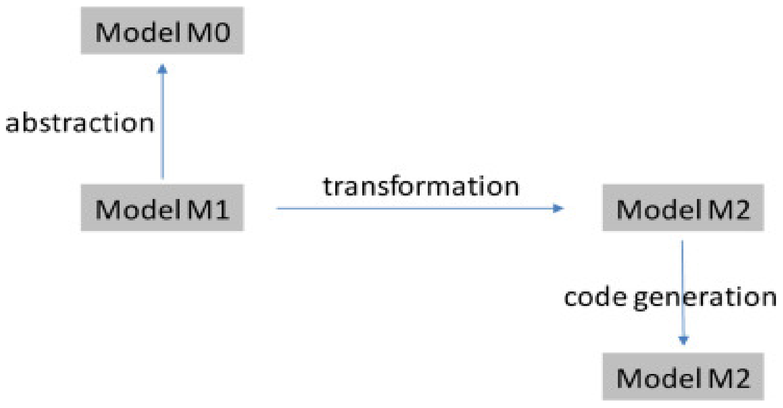


Figure 14. *Combination of multiple-level model transformation[39]*

Many different fields of business and research have used MMT, for example, mathematics, engineering computation, natural science, medicine etc. Growth of model-driven engineering (MDE) enabled the development of model-driven architecture, of which, MMT is a central aspect [40]. The progress led to the origination of various framework, software, add-ons and language to support MMT and MDE. Eclipse Modelling Framework (EMF) is one such notion of a framework that can serve widely used MDE tasks like model and meta-model operation, model to text creation and model to model transformation [39]. IBM Rational Rhapsody Developer Rules Composer is an example of an add-on developed using EMF that can facilitate exogenous transformation. In case of a dynamic and distributed system, adoption of Unified Modelling Language (UML) is prominent in industries [41]. UML is very useful to define system requirements and qualitative behaviour through an assortment of notations and have an advantage in visual modelling technique.

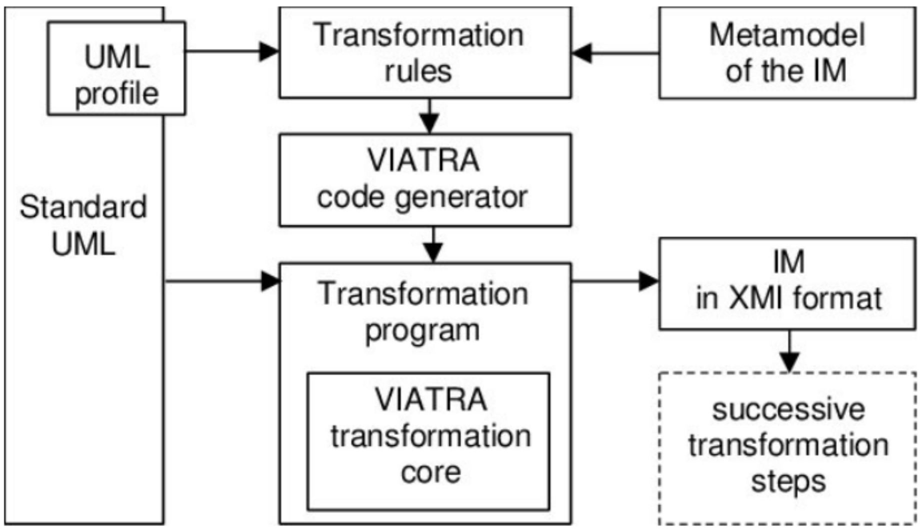


Figure 15. *Steps in VIATRA transformation application from system described as UML profile [91]*

There are a variety of transformation languages developed to serve a specific purpose and cover more generic needs. For example: Atlas Transformation Language (ATL), Janus Transformation Language (JTL), Kermeta, VIATRA, UML-RSDS, MetaEdit+ etc. To mention a few different capabilities of these tools, VIATRA can assist transformation based verification, ATL supports unidirectional source to target model creation, JTL supports bidirectional transformation etc. “MetaEdit+” is a tool consists of a workbench that can design modelling language and an automated modeller that can use the definition of the modelled language create full code from a model designed in the tool’s interface. In [42], Gabriel et al. showed that modernization of real legacy can be assisted by model-driven platform i.e. MoDisco.

3.2 Formal method: approach, analysis and application

Famously quoted by E.W. Dijkstra “testing shows the presence, not the absence of bugs” which implies that a verification identifies fault rather than aiming for proving correctness. The idea of building a proof by testing is at the heart of formal verification. A modelled system has some properties which can be declarative in nature. Properties capture requirements, they can be expressed as a precise mathematical form and manifest time-based behaviour. A mathematical prover can expose bugs by verifying the properties. Bugs can be of many types. There can be an error in the design, fault in formalization of the design intent, fault in grasping the design intents or can be trivial mistakes of missing or misplaced constraints in inserting values of the attributes.

Alternatively, a system can be verified by simulating or checking against testbench. But in these methods, the type of error spotted are “expected”, which means the software only finds errors that those were programmed to find. In contrast, the formal verification methods such as model checking explore through all possible states that a defined system can reach, which in turn can reveal unnoticed violations of constraints expressed as requirements. NASA spotted 197 error in the primary design of Galileo and Voyager shuttlecraft by formal verification, only 3 of which were caused by programming mistakes. Half of the problems were originated by wrong requirement definition and a quarter of the faults were introduced by design faults [43].

Due to safety and security requirement expansion, Formal methods have been applied in various industries. For example: (i) Software development [44], (ii) Integrated Circuit (IC) [45], (iii) aviation controller [46], (iv) controller in aeroplane [47] and (v) Nuclear I&C system [48]. Several researchers investigated the usability of formal methods in the growing sector of IoT and autonomous vehicles for safety.

The aviation industry applied formal verification in multiple fields related to instrumentation and control in aviation. Chen et all.[47], illustrated that model checking can find bugs in the programme that controls interference-driven slats and flaps of an aeroplane based on using PROMELA verification tool. In [49], it was demonstrated that the reliability of GNSS based satellite navigation system, which is very important for trajectory, can be verified with probabilistic model-checker. Furthermore, parameters (e.g. trajectory, landing time, runway selection etc.) that are handled by the air traffic controller, are also subject to formal verification [50]. Besides technical issues, pilot-centric factors such as an instruction guide for pilot activities or Gravity induced loss of consciousness (GLOC) monitoring system were verified and improved using formal model checking [51][52].

Formal verification method has been successfully applied in the nuclear power plant safety assessment. I&C system is the backbone of NPP safety, on which the researcher [53] applied model-checking with NuSMV symbolic model checker to successfully identify design errors and conflicting conditions. To reduce the long processing time occurs due to the state-space explosion, the symmetry within the I&C safety system was used. It was proven that detailed fault tolerance in a safety-critical system can be verified using a formal method [53]. Moreover to the safety-critical system, model checking has been in extensive use in the field of system-on-chip (SOC/IC) and software industries. For instance, System Verilog is a hardware description tool where an integrated circuit can be described in terms of conditional state transition within the range of computational units. Formal verification can discover issues in transition relation by analysing System Verilog assertions [54]. In the software world, the scope of using these techniques is limitless but due to cost constraints only utilized where error cause severe consequence such as online money transfer using blockchain [55] and manufacturing industry involving Multi-Agent System (MAS) [56].

There are several categories of formal verification, model-checking is just one segment of that and have seen a wide range of usage due to a few key advantages mentioned below [57]:

- Unlike other techniques such as “theorem proving”, model checking does not require proof (theorem formed based on requirements) of a system.
- System definition and requirements specification process can be automatic. Automation opens the scope for faster processing.
- A counterexample is generated if a state violates requirement. To debug the system the counterexample can be used for tracing the conflicting condition.
- From the beginning of system design, model-checking can be applied because of its ability to work on a partial system specification.
- The use of temporal logic can minimize the effort to specify system properties.

Nevertheless, model-checking has disadvantages too. Describing, system specification can be strenuous and understanding system requirements using temporal logic can be very problematic. Some bad constraints in design and properties along with Illegal stimulus can create a lot of spurious failures. Also, model-checking can output success in case of an empty model, which, is a failure condition in fact [54]. But biggest of all drawbacks is the “state-space explosion”. This refers to the excessive number of reachable transition states produced by a system that it can no longer be handled by model-checker. Due to the staggering number of states, in comparison with an informal method, a formal method can be very slow, some research case showed slow down in a factor of 3000 [58]. Typically, all the explored states are stored and tracked (to avoid exploring the same state repeatedly) by model-checker in memory during runtime. This number can be several hundred thousand but eventually, for a large system, the number can be too large to handle with computation capability feasibly [57].

Suggested solutions to manage a satisfactory are useful in many scenarios but also has some shortcomings. One solution is lossy storage of explored states in a suitable structure after compressing but, it is not easy to get the retrieve the compressed data by reversing the process [57]. Some of these issues can be partially resolved by introducing abstraction, invariants, coverage techniques to understand the scope of what is being verified and why. Another solution is to partially store from the set of explored states but it is not easy to algorithmically determine which states are important to store. To use abstraction techniques, research in [98] found that analysis of live components on observers location (observer created based on the pattern of transition occurrence) is a useful way to identify the important components in the process. However, the accuracy seems to depend proportionally upon the number of locations that are covered by observers [59]. Some research worked on the invariant selection and proposed way of accelerating model-checking by learning based on interpolation [60]. Besides algorithmic optimization, there are successful attempts to speed-up model-checking by combining computation power of multiple devices. For instance, CUDA technology (developed by NVIDIA) were used to boost up LTL model-checking speed [61].

To handle the voluminous state space of a real safety-critical system, several approaches are practised in verification. Of those, few are mentioned below.

- Manage the high level of system and requirement with abstracted details using formal methods [62].
- Within the system model, associate formal methods only for the most critical segments related to safety.
- Examine models with discretised and minimized range of variables.
- Utilize the “divide and conquer” method to verify a system hierarchically.
- Develop automation of the process of verification [62].

Besides model-checking, there are other techniques of formal approaches to functional verification. Theorem proving, deductive verification and equivalence checking are a few such capable techniques [63]. Following points gives a brief introduction to these techniques.

- **Theorem proving** is a way of mathematically proving the correctness conditions of a specific system concerning the formal specifications described mathematically. Model-checking cannot be directly applied for verifying infinite-state due to undecidability but theorem prover performs better on such unbounded data structure [93]. Mostly automated theorem provers are optimized and geared towards achieving proficiency in mathematical scopes. Theorem provers and program analyser are not the most suitable fit because several features are not suitably mapped into automated theorem prover logic structure but mandatory requirements for a good program analyser. Even though these tools are very capable for application on a mathematical system but they are not capable to supply counterexample in case of failed queries. ZAPATO and SIMPLIFY are two examples of such theorem provers [64].
- **Deductive verification** also referred to as program proving, is a process that turns correctness of a program into a mathematical statement, and after that attempts to prove it. The statements are denoted as verification conditions. Theoretically, any code and specifications are subject to use this method, constrained only by the user’s mathematical ability. Even though older tools were prone to errors, modern tools are much improved to a point that it is possible to mechanize the verification process. This has some challenges, uniquely defining correctness statement for a program is hard where it is likely to introduce bug such as program execution without stepping into a fatal error. This scenario may fail to terminate program which is undesired in case of a safety-focused system [65].
- **Equivalence checking** is a method of determining whether two systems have equivalency in a semantic sense. This system verification issues have a long history in theoretical computer science. A set of reference logics are required to be compared using theorem with respect to the logic equation generated from the logic gate level of netlists. If a system needs to be augmented for verifying sequential behaviour, equivalence checking is constrained. Furthermore, for this method to be effective, a golden reference model is necessary [63]. ATEC and FormalPRO are examples of such equivalence checking tool.

For special purposes, a combination of multiple verification techniques can be utilized. Figure 16 represents a system that applies theorem proving and model checking in combination to produce requirement specifications from source code as input.

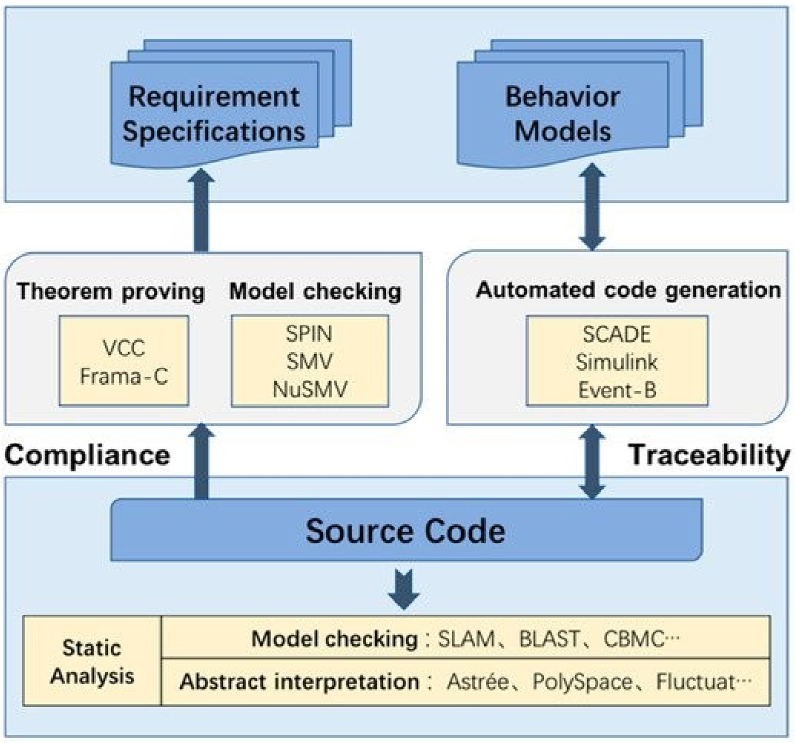

Requirement Specifications
Behavior Models
Theorem proving
Model checking
VCC
Frama-C
SPIN
SMV
NuSMV
Automated code generation
SCADE
Simulink
Event-B
Compliance
Traceability
Source Code
Static Analysis
Model checking : SLAM、BLAST、CBMC…
Abstract interpretation : Astrée、PolySpace、Fluctuat…

Figure 16. *Combination of multiple verification techniques [63].*

Generally, in operating parts of the system, there can be a composite cost function involving contradictory probable requisites. In an ideal case, these can be minimized with optimal decision rule, for instance: occurred delay in fault detection vs faulty alarm. Recognition and sequencing of different function setup and process mode favourable to a specific outcome or fault can be highly utilized to identify the occurrences of failure. Without proper knowledge of plant dynamics and relevant stats, sequential tests may not be very helpful [66]. The counterexamples generated by the violated cases still have very tricky use to identify if the violation is caused by an actual fault in the system design or modelling mistakes or inaccurately written logical formulas or erroneous oversimplified description of system requirements in natural language [67].

At current practice only the actual model checking part is automated, but prior steps can be automated too if the system description in logical terms is created by following standardized machine-readable structures and formats. Some research conducted on model checking for safety-critical system shows that it is possible to develop a unifying approach of “Model-checking for safety properties” of a parameterized system [68]. A “parameterized system” is a family of sub-systems where instantiation by n creates n copies of instantiation which are individual systems. Quantitative risk analysis can be enhanced if model checking is combined with fault tree analysis. The functional combination is very useful to provide formal, qualitative and automated assistance.

3.3 Selection of Model-checking tools

There is no single ultimate tool for verification across all domain but the unifying concept of all is mathematical logic [52]. Based on the algorithm used, model checker tools are of four major kinds. Those are: Bounded model checker (e.g. EBMC, CBMC), Explicit state model checkers (e.g. CADP, SPIN), constrain satisfaction model checker (e.g. ProB) and Symbolic model checkers (e.g. NuSMV, SMV). Some tools supports multiple algorithms. Moreover, classification can depend on synchronous vs asynchronous or discrete-time vs continuous-time operation.

- Bounded model checking is a way to exploit SAT/SMT (Satisfiability modulo theory) for verifying properties of programs, where, a program in a broad sense can be distributed system, protocol etc. Bounded model checkers work on the notion of states described by the values of variables. They can use boolean formulas to represent traces within the transition of a system. Bounded model checking is very suitable for finding out redundancies in design and closing coverage holes in automated test case generation [69].
- Explicit state model checkers, transition system related to a model specification is described using explicit representation. Such tools, for example, SPIN checks if the properties are verified after compiling. Whilst, the compilation does not create a global state transition graph, it assumes that for every verifiable property the transitions are computed [70].
- For verification of a formula, Constrain satisfaction model checkers use logic programming. ProB is a model checker that can work on B language where requirements are constructed in a form of class/modules, named as abstract machines. The state variables, function of the variables and invariant constraints of the variables is encapsulated by abstract machines. Generalized substitution language is used to define the operations. LTL and CTL can be used to write the properties of ProB [70].
- In symbolic model checker, Boolean logic is used to model a system and to represent transition states. The system is divided into a modular structure to allow abstraction and redundancy where time-dependent elements and modules are modelled for operating in discrete steps of time because NuSMV doesn’t support continuous time. Time steps have fixed length where inputs are at first, sampled and then updated for output during every time step [71]. LTL and CTL can be used to define properties in these tools.

While verifying a system’s design specification with model-checking, it is compared parallel with the description of a model environment and description of the system interaction with the environment. Without any assumption about the environment, it allows simple models to have behaviours that are less biased and independent of the system under verification. This approach leads to a safer model checking result. Because, if the system verifies the specification under more liberal environment conditions, it is better suited to satisfy specification in a more restricted environment.

The level of abstraction used in the methodologies is highly dependent on system description, but the main objective is to emphasize on the logical functions leaving the hardware aside while opting for over-approximation of the model so that it has more behaviours than the actual physical system. The typical errors discovered by model-checking are such as (i) events beyond the defined logic, ex: erroneous transmission of the sensor signal and failed hardware, (iii) unexpected order of physical event occurrence most of which are raised from unaccounted scenarios by designer [67].

Multiple research analysed various cases including a changeover switching unit for busbar, an industrial arc protection system, an emergency cooling system of nuclear reactor, an emergency control system of diesel generator, embedded control software of uninterruptible power supply, stepwise shutdown system etc. They recommended a methodology based on their findings where semantic methodologies were created to make a model with function block diagram for two model-checking tools: NuSMV and UPPAAL. [39][41]. For this thesis, NuSMV is the tool of choice. Following two sections contains a discussion on this tool.

3.3.1 CTL and LTL in NuSMV

NuSMV is a deterministic model checking tool developed by extending and re-implementing SMV for additional features [16]. To begin with, a finite state model of a system is constructed along with a requirement in a format of temporal logic notation [72]. During verification, the efficiency of the process can be amplified in various cases by calculating large numbers of states in a phase. The model checking method is noted to be “symbolic” if a set of transition relations of states are represented as logical formulas which are the base for state space traversal. Instead of logical formula other convenient and supported data structure for example Binary decision diagrams (BDD) can be applied. BDD.

NuSMV supports CTL and LTL specification, which are a subset of CTL*. CTL stands for “Computation Tree Logic” which is a branching-time logic used extensively by the model checkers [73]. CTL can specify condition such as when a program should evade the execution of a state if a set of initial condition is satisfied. In CTL description time is modelled as an undefined future state that belongs as different paths in future to a tree-like structure. Any of the future paths can be determined as an outcome by model checking tool based on safety properties.

LTL, stands for linear temporal logic, is a modal temporal logic where modalities refer to time [74][75] and subsequently it is a fragment of first-order logic [76][77]. With LTL, it is possible to encode formulae about the future of paths, (e.g., a condition in always true state, or a condition will only be true if it met another condition, etc). LTL does not permit branching time and quantifiers, which is supported by CTL [78][79]. Sometimes, LTL is called propositional temporal logic, abbreviated as PTL [80].

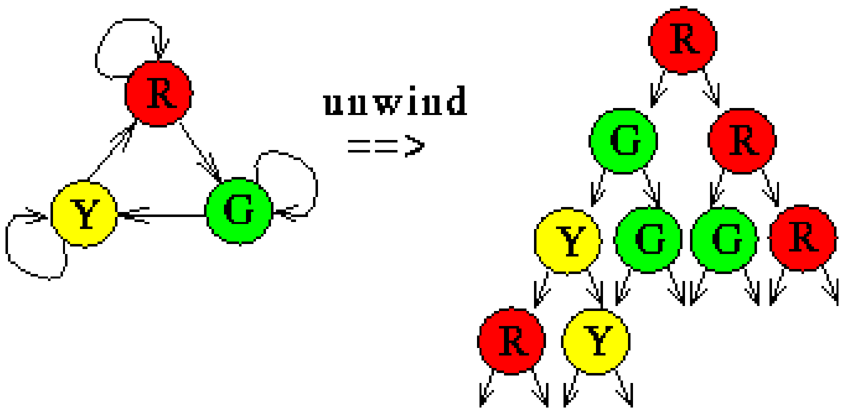
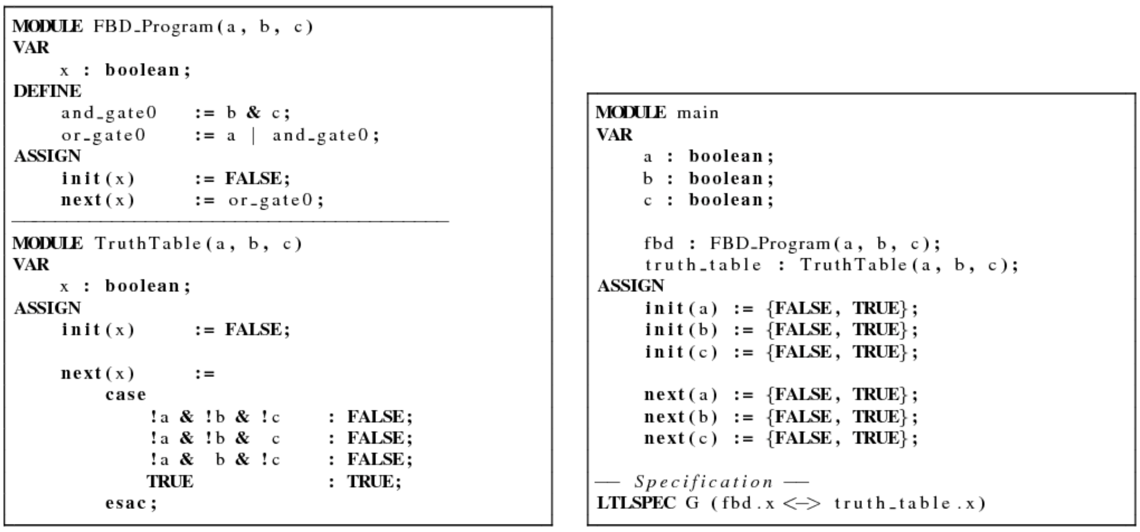


Figure 17. *Unwinding steps of a state transition graph [73]*

An unwinding process of a graph to create a tree is shown in Figure 17 where all possible combination of transition states are modelled and illustrated as paths. Computation tree also referred to as formulae, is developed from a model where branching of the tree is defined by logical operators. At any given state, either true or false is the possessed value in each formulae [81]. Whether a state of formulae is "truth" is determined by the sub-formulae. This is a recursive function in which, if a "truth" value is achieved for all initial state of the system, then the formulae is satisfied. A counterexample is generated if formulae are not satisfied.

3.3.2 Structure of NuSMV model

The system is divided into a modular structure to allow abstraction and redundancy where time-dependent elements and modules are modelled for operating in discrete steps of time because NuSMV doesn't support continuous time. Time steps have fixed length where inputs are at first, sampled and then updated for output during every time step [71].



```
MODULE FBD_Program(a, b, c)
VAR
    x : boolean;
DEFINE
    and_gate0    := b & c;
    or_gate0     := a | and_gate0;
ASSIGN
    init(x)      := FALSE;
    next(x)      := or_gate0;
----------------------------------------
MODULE TruthTable(a, b, c)
VAR
    x : boolean;
ASSIGN
    init(x)      := FALSE;

    next(x)      :=
        case
            !a & !b & !c     : FALSE;
            !a & !b &  c     : FALSE;
            !a &  b & !c     : FALSE;
            TRUE             : TRUE;
        esac;
```

(a) Module declarations

```
MODULE main
VAR
    a : boolean;
    b : boolean;
    c : boolean;

    fbd : FBD_Program(a, b, c);
    truth_table : TruthTable(a, b, c);
ASSIGN
    init(a) := {FALSE, TRUE};
    init(b) := {FALSE, TRUE};
    init(c) := {FALSE, TRUE};

    next(a) := {FALSE, TRUE};
    next(b) := {FALSE, TRUE};
    next(c) := {FALSE, TRUE};

-- Specification --
LTLSPEC G (fbd.x <-> truth_table.x)
```

(b) Main module and spec.

Figure 18. *Example of NuSMV description of a system [82]*

Main features of NuSMV are listed below

- **Functionalities**: NuSMV can analyse a system's specification expressed by CTL and LTL as defining toolset. This enables a synchronous and asynchronous system to be modelled using NuSMV [83].
- **Architecture:** In NuSMV model architecture different components and functionalities are isolated in separate modules. Modules can have interfaces, which allows to modify and extend NuSMV with reduced effort [84].
- **Implementation quality**: NuSMV tool is **POSIX** compliant, written in **ANSI C** and debugged using **Purify** to identify memory leakage [25]. Besides, the BDD package, developed at Colorado University, is used in the core of NuSMV which provides the capability to create an interface with state-of-the-art solvers [84].
- **Additional benefits:** NuSMV has several advantages over SMV, for example, textual interaction shell, invariant analysis, method partitions, LTL and PSL (Property Specification Language) model checking, SAT-based bounded model checking, etc. [83].
- **Usage in research:** Besides practical applications, NuSMV is a widely accepted tool in various researches. **VTT**, a research institute that facilitates the development for this thesis, used NuSMV for model-checking of I&C system diagrams.

The data types supported by NuSMV are Boolean, integer, enumeration types, word, array and set types [84]. Current NuSMV expression is typed and operands have type constraints. The system is modelled as a finite state machine and the building blocks of the system are a combination of state variables, input variables, and frozen variables. A transition relation describes how many possible stages can be derived from one state whereas, constraints on the valid parts of the implementation of the finite state machine is defined by fairness conditions [84].

3.4 Similar research and development

In research, there has been a successful attempt to produce a formal verification framework to verify UML active behaviour in Symbolic model verifier [85]. The process allowed users to model system behaviour in UML without any prior knowledge of temporal logic for formal model checking where the tool performed automated transformation to SMV specification without user intervention.

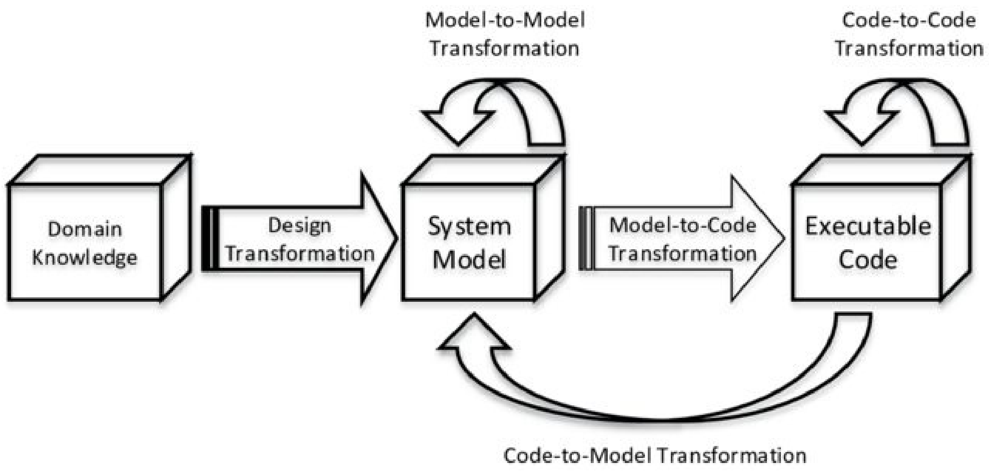

Model-to-Model Transformation
Code-to-Code Transformation
Domain Knowledge
Design Transformation
System Model
Model-to-Code Transformation
Executable Code
Code-to-Model Transformation

Figure 19. *Model to code generation [85]*

The transformation can be done with various tool and strategy. Tools can be developed with general-purpose languages such as C++, java or specific transformation language out of various option, can be used efficiently depending on the model transformation need. ATL framework can be used to generate input language of NuSMV, SPIN from the systems engineering modelling language SysML [86]. Besides, Inter-object communication, system behaviour and structure can be modelled by SysML. There are no widely accepted common platform or input format for a model that would be utilized by a large number of models checking software. In the realm of model-to-model transformation, graph theory played a significant role. Abstract syntax tree, a special case of a graph, is widely used in these transformation technologies as a mean to transform and rewrite graph. The key idea is that, if a state of a computation can be formed as a graph, further the steps can be represented and interchanged to present computation by defining fixed number of transformation rules [87].

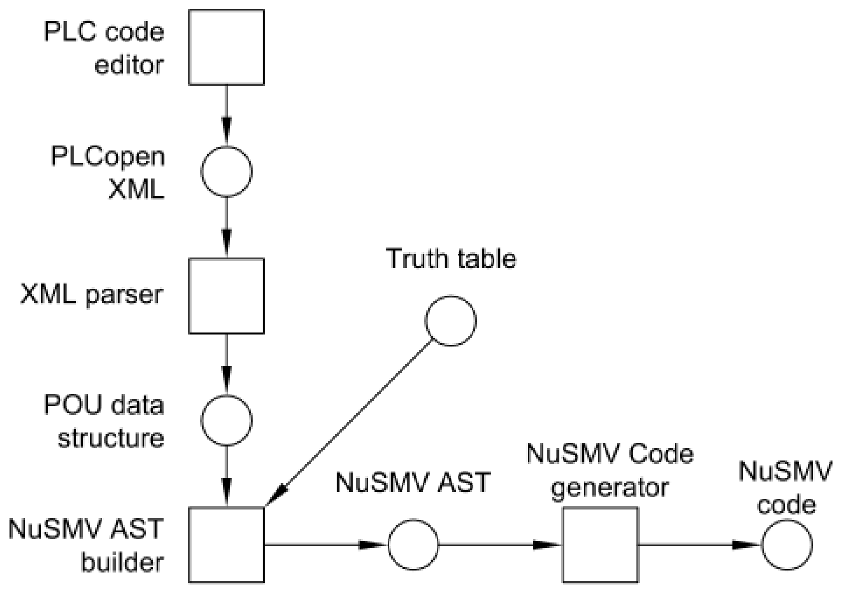


Figure 20. *The architecture of plc to NuSMV MMT programme[82]*

In the University of Oslo, there was successful research in an attempt to generate NuSMV code directly from the PLC code by developing a PLC-NuSMV compiler [82]. This open-source tool supports transformation from PLCOpenXML compliant FBD generated by the open-source tool (Beremiz) in limited aspects. Figure 20 illustrates the architecture of the compiler.

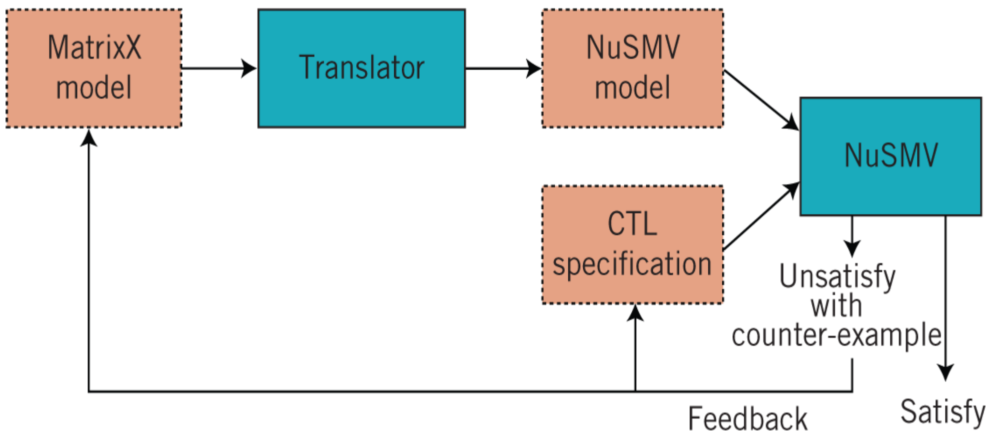


Figure 21. *Automated verification steps for GLOC monitor [51]*

Seonmo et al. [88] presented advanced research (figure 20) to automate formal verification of a model of GLOC monitoring system of aircraft by creating the NuSMV model of a system defined using MATRIXX software.

3.5 Justification: State of the art of this thesis

Literature review of topics relevant to this thesis was very helpful to understand the broader scope of model transformation and formal verification. However, it was understood that no available software can serve the exact need to fulfil the objective of this thesis. In the presented research questions the defined quests are specifically related to NPP safety I&C diagrams designed delivered to the expert analyst as MS VISIO files. C++, C#, VB and Java can access the VISIO diagram in a programming environment but no dedicated library is available to process I&C specific shapes. Complexity arises since the diagrams are not standardized and drawing error exists when the diagrams were manually created. This thesis is a particular attempt to develop methodologies to create models of logical components by performing transformation operation on the VISIO diagrams. The qualified formal verification tools used in VTT is a proprietary graphical verification tool in which the diagram has to be redrawn. Hence, to reduce this hold-up in the process, the modelled components should have data of not only about the "type" of logic blocks but also about the graphical layout of VISIO I&C diagram to support import to the verification tool. In the subsequent step, the modelled logic blocks will be used to graphically demonstrate the correctness of the diagram-to-model translation along with a partial NuSMV script to assist model checking.

These requirements are very specific to the NPP safety I&C diagram in the scope of this research. In the bigger picture, efficient integration of the model with various model checkers, simulation and testing tools is in the realm of development possibilities. This approach can be categorized in endogenous unidirectional transformation.

.

4. RESEARCH METHODOLOGY

Aim of this chapter is to explain the development strategy considered and executed from the beginning of the thesis work. This section focuses on the progression of the development through research questions and steps carry out by the author.

4.1 Methodical approach

The end goal of this thesis is to answer the research question with a clear demonstration of the learned and accrued development technique throughout the whole process. A visual representation of the identified logic block diagrams will assert the process as valid and well-grounded. This requires developing a program that is qualified to demonstrate proof of concept. There are many common software development strategies such as Waterfall model, Spiral model, Prototype methodology etc. But for this thesis implementation of a prototype is the most logical choice. This is a specialized way of development intended to build working sample of the resolution so that functional essence can be validated before building large solution. Prototyping helps to get clear concept of the functional process of the implementation, assists to identify complex functions and minimize the chance of functional failure.

4.2 Identification of viable procedure

This stage involved finding out the right meta-model that encompasses all the necessary properties need to build class objects. The objects are identified from the input diagram and built upon the requirements needed for integration with graphical model checking tool or to create input for formal verification environment for model-checking. The meta-model was finalized by detailed discussions of core requirements with real analysts involved in the project and expert in the field of model-checking of nuclear industry-specific I&C logic for safety verification.

The right procedural steps to develop the model transformation was decided in this stage after studying the abilities of a certain number of tools and approaches. After agreeing upon the meta-model, the next step was figuring out the usage of the tools and the technologies to build up the programming framework.

4.3 Proof of concept

After the initial selection of suitable programming language based upon online research, development phase began with understanding the usage of program libraries and API to import the diagrams in a programming environment. This phase of development attempted to create a small module as proof of concept. A proof of concept is a barebone program built to test the assumption derived from the initial studies of the relevant topics. To-do list for this phase consists of VISIO diagram import to a programming environment, draw a small experimental diagram to test the selected library functions, create a primary class structure of the program to progress to the next stage of developing a prototype. This was an attempt to demonstrate the basic functionality to validate the choice of development tools and certain concepts that aimed to be achieved in later stages.

4.4 Program prototype

Proof of concept was designated for verifying the functionality of some concepts later to be unified into the prototype development. The proof concept did not ensure that the learned technique was readily usable in the development, rather it was a guideline in the first stage of development of a prototype. Prototyping was an attempt to prepare a working model but did not have the robustness to be deployed in real-world usage. Nature of the data is critical for the accuracy of element properties identified by analytics. The data retrieved through API was large in the initial stage, especially if a large diagram was used. The retrieved data in the primary stage was in XML which contained detailed data of each graphical elements in a VISIO file. However, API library in most of the cases had functions to get necessary fields only and if a necessary field was not accessible through API than XML was parsed to collect the data field needed. For this purpose small diagram was built using a combination of simple and complex graphical elements from actual Microsoft VISIO diagrams.

4.5 Scale-up for complex diagram

The prototype developed, to process smaller diagram comprises of fewer complex shapes, provided a snippet of the final version. After getting well versed with the pattern of the diagrams and accustomed to the programming techniques involved with API, more complex VISIO shapes were analysed. Based on the analysis and program classes were formulated. This procedure of gradual scale up not only gave the benefit of accurate development time estimation but also scope for building up the documentation beginning from the primary phase of the development.

4.6 Tools and technologies

This section gives a glimpse of the major software and libraries used in the development.

4.6.1 Microsoft VISIO

Microsoft VISIO is a drawing tool to create a variety of diagram to assist business, product development in various format. VISIO supports diagram format to create flowcharts, floor plan, business process modelling etc.[89]. A data flow diagram can provide a focused approach to technical development. VISIO supports multiple file formats, but for the thesis, the processed diagrams are in ".VSD" format.

4.6.2 Apache.poi

Apache POI is a Java program library developed by Apache Software Foundation to process Microsoft Office documents including VISIO. POI allows to read and create several types of MS Office files. This library can be used extensively with MS Excel files but have limited capability to work on VISIO files. Source diagrams used in the thesis are XDGF (Open Office XML Diagram Format). Apache POI only allows to read XDGF data fields but no writing however, it was enough to create model using POI with Java [90].

4.6.3 NuSMV editor: SciTE

For this thesis **SciTE** text editor was used to experiment with the NuSMV. This is a text editor based on **SCIntilla**, a free set of source code editing component, which is available for both Windows and GTK compatible versions of Linux. SciTE is a very simplistic editor best suited for building test and demonstrate tasks with simple configurations. It has very minimalist environment setup requirement and simple steps to run as a NuSMV editor.

4.6.4 MODCHK

For model checking of I&C system, a capable graphical tool is MODCHK [94][95]. This tool was developed tool by VTT Technical Research Centre of Finland based on NuSMV. MODCHK is suitable for model checking of I&C application logic created as standard or non-standard function block). MODCHK allows to create models for NuSMV input using a graphical interface and to visualize counterexample trace produced by NuSMV. Simantics platform, initially developed by VTT as an open operating system for modelling and simulation, is a major structural part of MODCHK. A user only has to concentrate on creating the model in MODCHK which is translated to NuSMV automatically. "True" values for binary signal are visualized by changing the connecting wire colour into red.

5. IMPLEMENTATION

This chapter discusses the implementation process of the system in details. The Implementation description expresses clearly the coding practice, class design decisions and functions' duty. This chapter is divided into five major parts. In the beginning, a depiction of the system architecture is placed to help to perceive the overall system. Sections onward comprise of detailed discussions about the most important aspects and deciding factor of the program components. The last section explains the implementation of the NuSMV code generator block including relevant logic.

5.1 System structure

The intended solution consists of only machine-readable files and software components without any necessity of hardware apparatus consideration. The system involves three major parts: Microsoft VISIO diagrams, programming environment, NuSMV text editor. Specific to this development process the VISIO diagrams under consideration has two categories; component symbol chart and designed I&C system. To use a particular style of VISIO shape throughout the I&C system design, the component symbol chart meant to be followed as a functional architecture style guide. This guide presented a generic symbol style that encompasses all the "types" of I&C system components used in the actual system diagram along with a short description. Point to be noted that, the symbol chart only served as a visual cue for the system designer but was not used in the system diagrams as exact copies.

The I&C diagrams were essentially an illustrated network of operational elements with connections among them. The elements were drawn on Microsoft VISIO page using a mixture of both default and customized shapes. Simple binary logic (AND, OR), sensor input, manual process input, input from/output to external process unit, various actuators etc. were the building blocks of the network. Directed connection arrows showed the direction of data flow from element to element. Altogether, a network was a representation of Function Block Diagram (FBD).

It is important to know that the term "Function Block Diagram (FBD)" does not imply that these diagrams followed FBD as an IEC 61131-3 programming standard. But these diagrams are similar in representation style of IEC 61131-3 FBD. Hence, from now on the term "FBD" used in a broader sense so that reader can relate to the diagrams with an understanding of IEC 61131-3 FBD.

Once the VISIO diagrams were imported in Java programming environments using Apache.POI library, the next step consists of individual VISIO shape identification. This step was a very important segment of the whole program as the precision of the identification process determine the accuracy of the modelled elements. Later, all model instance followed a metamodel. The metamodel was carefully defined to meet the necessity of carried out development and future expansion of the project, therefore, consisted of all the essential data fields.

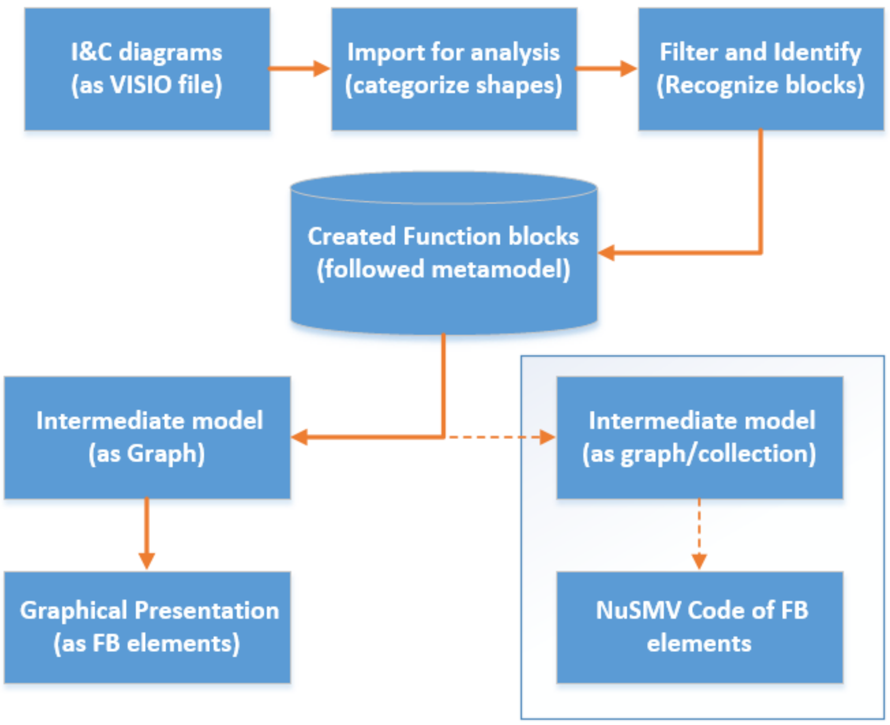

I&C diagrams (as VISIO file)
Import for analysis (categorize shapes)
Filter and Identify (Recognize blocks)
Created Function blocks (followed metamodel)
Intermediate model (as Graph)
Intermediate model (as graph/collection)
Graphical Presentation (as FB elements)
NuSMV Code of FB elements

Figure 22. *The system architecture of the program*

A graph (tree structure) was formed using the identified functional element where each node was a function block and the directed edges were the connector arrow representing data flow. This graph was a very useful data structure for this implementation. The created model and generated graph were used subsequently for two major tasks, data visualization and NuSMV code generation.

FBD visualization in the form of tree structure was crucial to ensure the accuracy of the program. It was easier to single out wrongly identified or unidentified data from the network by visual comparison with the input VISIO diagram. Automatic NuSMV code generation was the next part. It is important to note that the NuSMV code generation is not a standalone process and have several limitations discussed in 6.3.

5.2 Program design considerations

Software and tools used:

- Microsoft VISIO (version)
- Java 1.8.0_241
- Apache NetBeans IDE (version 11.2)
- Apache POI-4.1.1
- GraphStream-1.3
- NuSMV 2.6.0

5.2.1 VISIO diagram import

Given that, the selected tools were appropriate for this development, the programming process began with importing the Microsoft VISIO diagram into NetBeans programming environment.

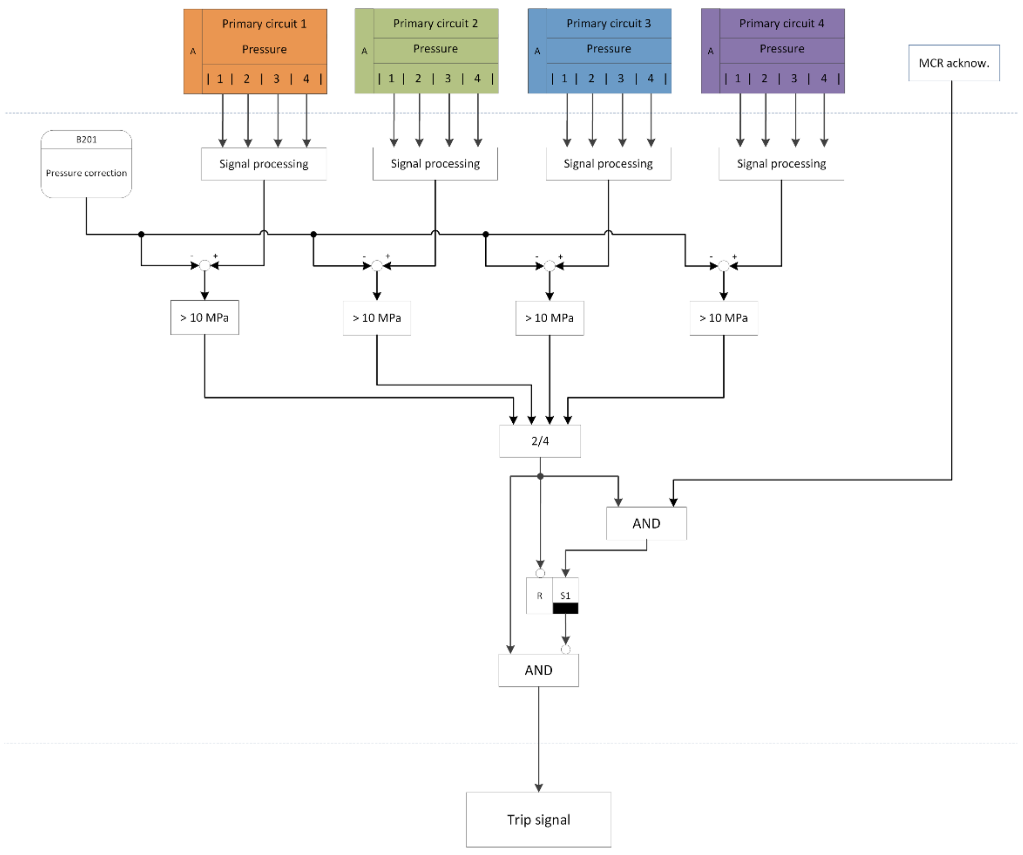


Figure 23. *Example of application logic as a VISIO diagram*

The used version of Apache POI was capable of reading input diagrams as ".vsdx" format, whilst the source VISIO documents were provided in the older version as ".vsd" format. The format was changed to ".vsdx". The input file consisted of multiple pages of diagrams but, initially for the ease of processing only single pages was used as input. POI read individual pages as an instance of "XMLVisioDocument" class which contains every element on the page. In the java program, VisioDataParserICA class imported the diagrams as "xmlVisioDoc" object which contained all the page, shape and connection data. At this point, the diagram import to the programming environment was completed and allowed to proceed to the next phase of deduction, filer and classification.

The following figure 23 is a representation of metamodel. After Identification of a function block such as input_from_process, output_to_process etc. had to have the properties described in the stereotype of a functional component (right side of the figure). From the beginning of the diagram processing, relevant data had to be preserved to conform to the object metamodel.

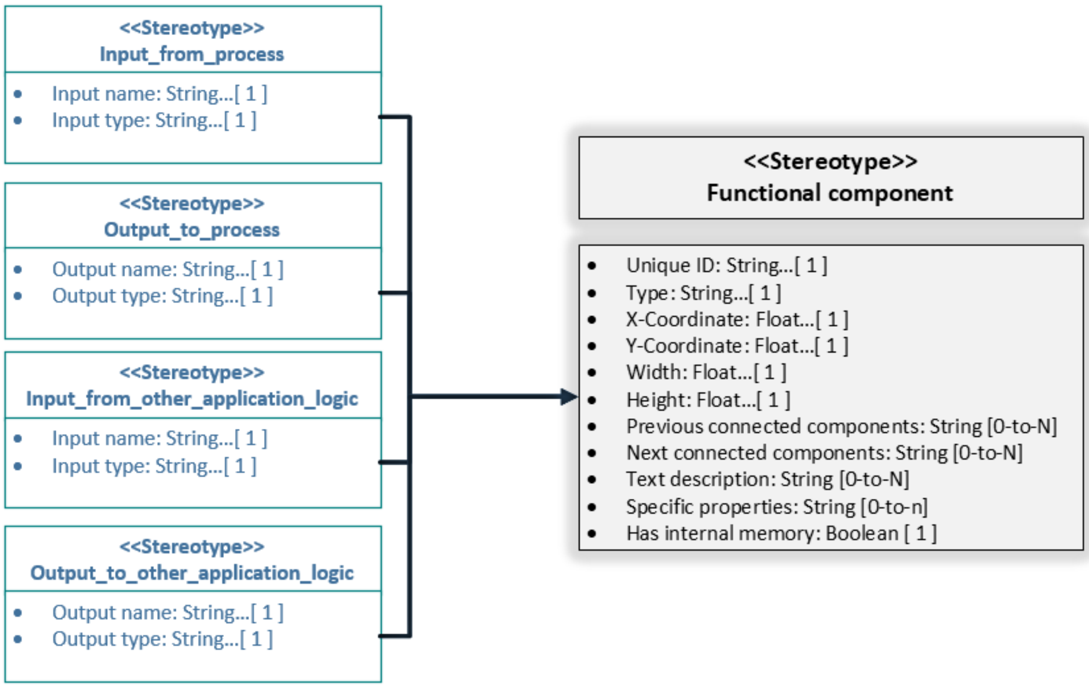

<<Stereotype>>
Input_from_process
Input name: String...[1]
Input type: String...[1]
<<Stereotype>>
Output_to_process
Output name: String...[1]
Output type: String...[1]
<<Stereotype>>
Input_from_other_application_logic
Input name: String...[1]
Input type: String...[1]
<<Stereotype>>
Output_to_other_application_logic
Output name: String...[1]
Output type: String...[1]
<<Stereotype>>
Functional component
Unique ID: String...[1]
Type: String...[1]
X-Coordinate: Float...[1]
Y-Coordinate: Float...[1]
Width: Float...[1]
Height: Float...[1]
Previous connected components: String [0-to-N]
Next connected components: String [0-to-N]
Text description: String [0-to-N]
Specific properties: String [0-to-n]
Has internal memory: Boolean [1]

Figure 24. *Function block properties as Meta-model*

5.2.2 Initial segregation of shapes

The logical analysis began in this stage based on the knowledge gathered from the experiment on the VISIO shapes contained in the provided symbol chart. Every VISIO elements on a page was an XDGFShape. It implied that the connector arrows were also "XDGFShape" even though those were accessible as individual "XDGFConnection" element. The complexity arises from the fact that the non-standard design process involved a variety of customized shape groups that had connector inside those as nested elements, which were used only as a visual design rather than the purpose of symbolizing data flow between two elements. Moreover, some shape groups had connections within them representing data flow, where it was easier to consider that whole group as a unit for identification since those shape groups had very similar properties. So, it was very important to separate the actual connection from the invalid void connections at the primary stage of classification. The following figure 25 is a decision diagram showing initial sorting logic implemented in the program.

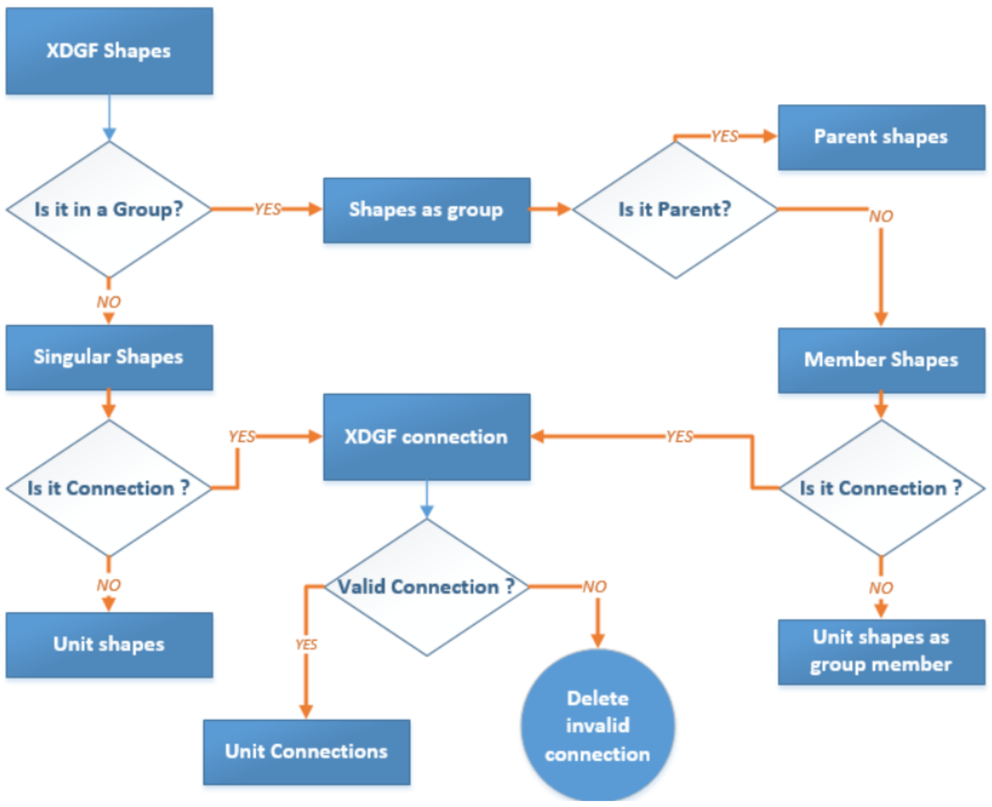


Figure 25. *Decision diagram for initial sorting of shapes*

Every VISIO element had a unique shape ID number which was the only way to trace an element in any stage of the program. XDGFShapes can be a single-unit shape or a part of a group of shapes. A group of shapes can also stay together as a unit, where that unit belongs to another larger shape group as a nested element. All the group members fall under an umbrella shape denoted as “ParentShape” and if the ID is known all the child shapes were retrievable from the shape group. “Set” data structure was particularly useful to ensure unique shape ID had no duplicate while “Map” data structure was used to temporarily store the element’s data as XDGFShape. The reason for this is to make it much easier to search for components using the shape ID.

5.2.3 Intermediate representation

Each VISIO shape had many data fields but, not all of those were useful in this analytical context. Besides, in several cases, a few data fields were needed to be retrieved through parsing XML description (accessed through POI) and the variety of VISIO shapes produces various data fields. It was more convenient to proceed with the analysis if an intermediate representation of the elements existed, where all and only the necessary fields were available. In order to do that an intermediate class(named IRVisioShape) was defined and every single elements was saved as an instance of this class where the essential attributes were filled up using the XDGFShape data and the classification completed in the previous steps. An object of this class had the geometric data of the shapes which were used to create properties such as Point2D, bound rectangle etc.

5.2.4 Repair and build connector

Accuracy of data flow crucially depends on the accuracy of the connector. Unfortunately, in the diagrams, the connections were not readily available to determine the data flow direction when imported to the programming environment, even though it was visually understandable. Therefore, fixing the connecting arrows was a significant step. There were three main issues with the connections: one-to-one connection with unoccupied ends, one-to-many connection with multiple split in between start and end point. And the third type of issue was a combination of previous two mentioned issues and self-ID contained as connected shape. An individual diagram often had connections with all of these issues. Before unifying the broken connections, it was mandatory to fix the loose unoccupied ends of the connections.

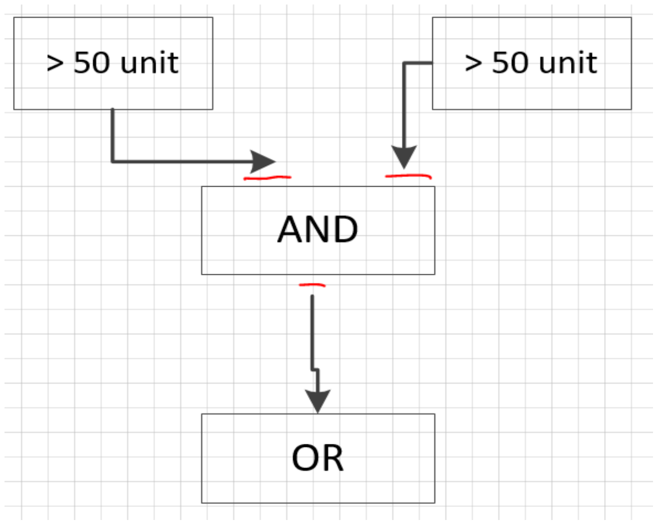


Figure 26. *Unoccupied ends of connections*

Despite being unoccupied, the connection ends were in very close proximity of the intended shapes to be connected. So, the rational repair step was to find the nearest shape (which was not a connection) in the two-dimensional Cartesian space and assign that shape's ID as an attachment ID with the edge of connection. A function was written to dynamically find the nearest element by calculating the minimum distance from the endpoint to the edge of the boundary using the formulated bounded rectangles of each non-connection shapes. This process was repeated If a connection had self-ID as connection edge, and the self-ID was replaced by the ID of nearest found shape.

Broken connection only existed in case one connection began from one shape but led to many output shapes. The bonding bridge between two broken connections was a connector mid-point. This mid-point was the cue to construct an uninterrupted unified connection from broken pieces. After trying several techniques best results seem to occur from using graph (tree data structure) and for these purpose "jgrapht" library functions seemed to suit perfectly. In this approach, each connection was entered as a vertex in a predefined undirected graph. There are three determining factors to consider a broken connector line to be a part of another broken connector line. First, multiple connector line might begin from one common mid-point shape hence sharing the common ID as starting shape. Second, a connector line might end in a mid-point shape from which another connector line begins. And, similar to the second condition, if a connector line begins from a midpoint shape where another connector line ended. If any of these three conditions were met then an edge is added between two connector line represented as a vertex in the graph.

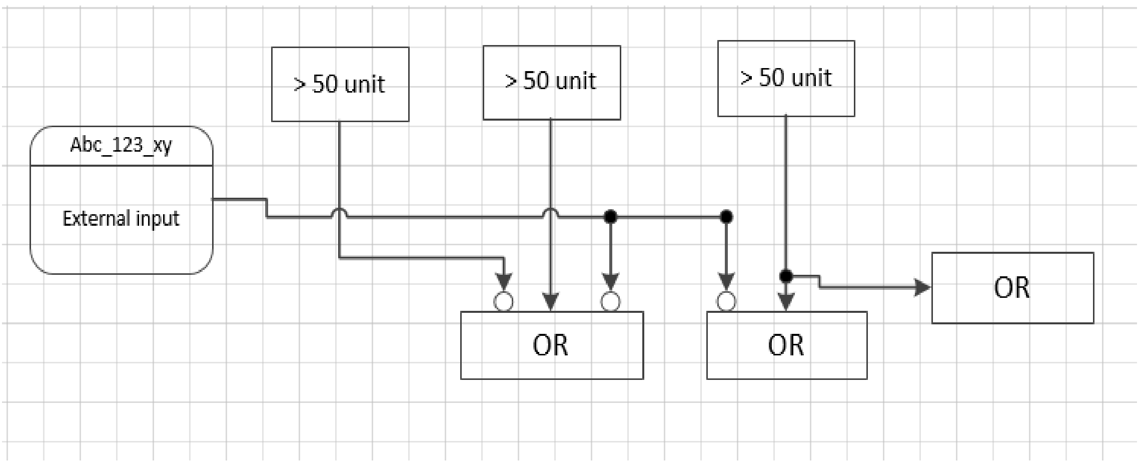


Figure 27. *Broken connecting line with mid-point (black dot)*

In short, a set of connector lines sharing a common midpoint, somewhere down the line to form a chain, belonged in a tree as nodes. After the trees were constructed, the unique connections were built from it. In a tree, there were only one common start point and multiple endpoints at the last layer of each branch. The ID of each broken connector line at the ends was concatenated with common start shape ID to form a unified connector while skipping the intermediate broken lines.

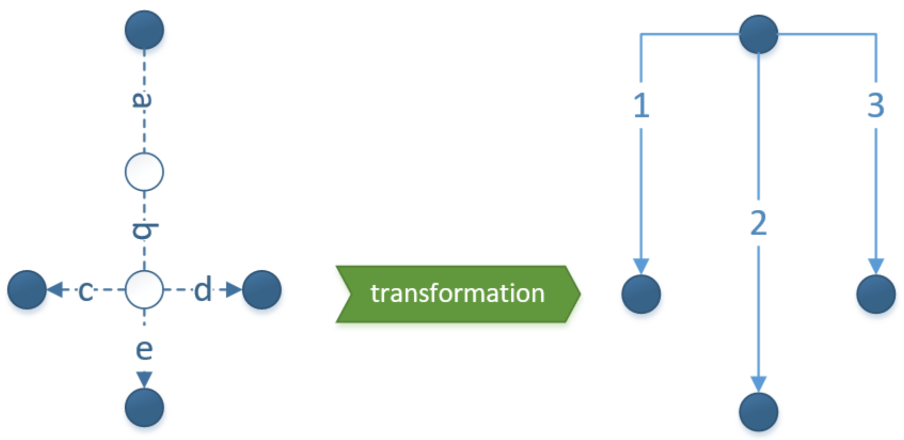


Figure 28. *Fix and build connectors using graph*

After all the connections were repaired and unified those were ready to be assigned to individual shapes. A connection had two important attributes referred to as *begnFromShapeID* and *endToShapeID*. A shape retrieved by beginFromShapeID stored endToShapeID as the next element and shape found with endToShapeID similarly stored beginFromShapeID as the previous element in the grid of the VISIO elements.

5.2.5 Unify and classify shapes: First phase

The shape unification process took place in multiple stages of filtering. As mentioned before the VISIO shapes were not standard, thus made it unreliable to identify the shapes function block element in one go. Single unit shapes (neither a parent group nor a member of a group) consisting simple text/symbol or only having particular shape type with unique descriptions were separated. Rest of the shapes were bundled as various groups.

Unification and classification process of shapes was somewhat analogous to the unification process of connections. An undirected graph was defined where all the shape's ID were added as vertexes. If a shape ID belonged in a parent-child relationship with a group or a member shape, an edge was added between those nodes. This process also unified the groups nested inside another parent group. Finally, the root node of each tree was assigned as umbrella-ID for all the parents & child nodes in a single tree. Single and unified groups are separated from the tree. The shapes were stored as an element of parent shape with all the textual data. Othe attributes of the child shapes were accessed from the previously-stored XDGFShape data.

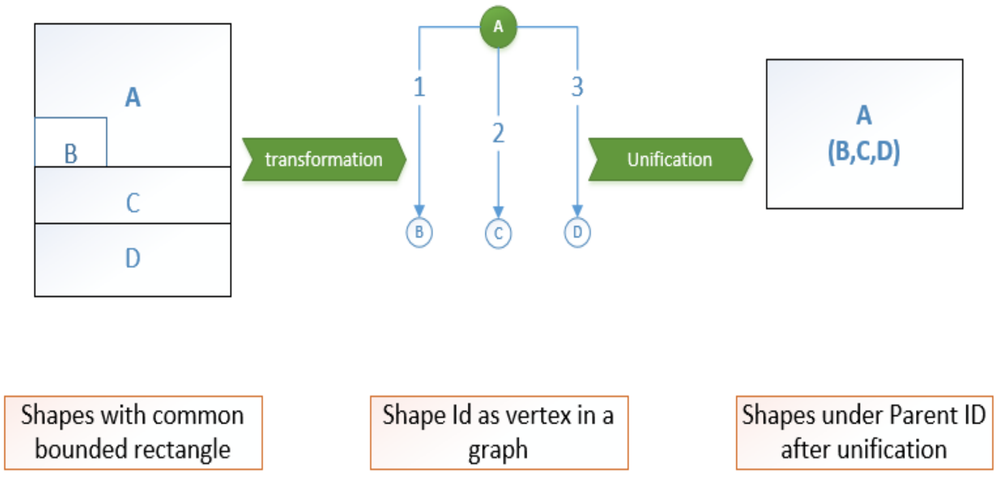

A
B
C
D
transformation
A
1
2
3
B
C
D
Unification
A
(B,C,D)
Shapes with common bounded rectangle
Shape Id as vertex in a graph
Shapes under Parent ID after unification

Figure 29. *Unification steps of a group of shapes*

Roughly all the singular unit shapes were easily identifiable at first, right after the first stage of shape unification was completed. Singular shape identification was conducted by analysing symbol, type of shape and text. Text analysis was done either to find a straightforward string (AND, OR etc.) or to discover underlying patterns. Pattern analysis was carried out with the help of "Regular expression". Regular expression is a very powerful technique, supported by all major general-purpose programming languages, to identify patterns within strings with a diverse combination of text and symbol.

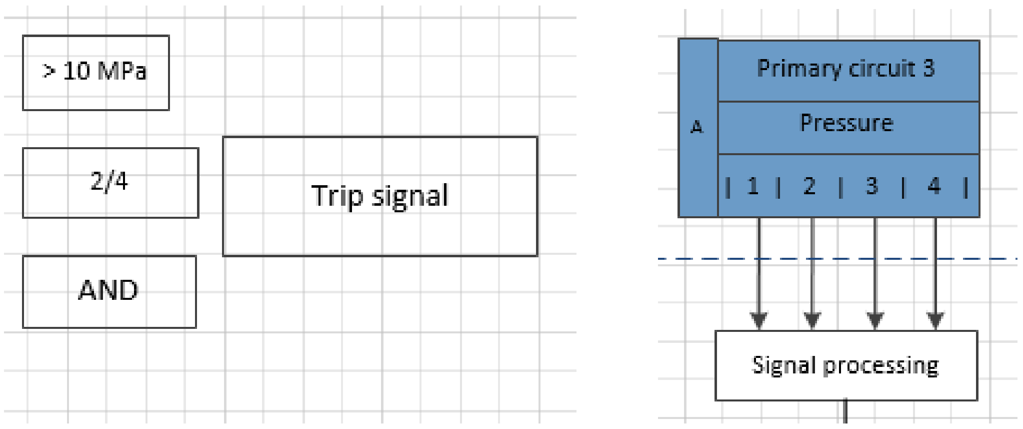


Figure 30. *Example of single shapes and group of shapes*

Regular expression was also used to discover patterns from the text inside the group shapes. Table 1 contains a list of a few important expressions with their function. If a singular shape/shape group was undisputedly identified as FBD component and if Its input/output(I/O) connections were independent of any relationship with yet unidentified shapes, those were transformed into object following requirements from the metamodel. Rest of the shapes were left to be processed in the next phase.

Regular Expression	Identification task
([=][nx]*[\\d]+[\\w]+)	Value_Of_Delay
([<>]([-]?)[\\d]*[,°]?([\\d]+?[\\w]+)?[\\w%/]+)	Limit_checker
([\\d][/][\\d])	Voting_logic
([0-9a-zA-Z]*[_]{1}[0-9a-zA-Z]*)	External_safety_function
([|][\\d]{1}[|][\\d]*[|][\\d]*[|][\\d]*[|])	Binary/Analog_input_process
([0-9]*[°][c][/][h])	Cooling_gradiant

Table 1: Regular expression for identification

5.2.6 Unify and classify shapes: Second phase

This phase of analysis dealt only with the unprocessed shapes from the previous phase. A fewer number of shapes resulted in less cluttered space which increased the accuracy of operation on shapes. Three major tasks were carried out in this part: adjustment of dependent shapes, creation of virtual connection, and creation of virtual clusters of shapes. At the end of this phase most of the shapes were identified but there remained still unrecognized shapes.

There were a few shapes that did not have a standalone utility but were integral part of the system to instruct operation of other function blocks. Figure 31 is such an example of shapes containing sign (+,-) placed in very close proximity of the summation element represented with circular shapes. A correctly identified summation element possesses two incoming data connection but the relative position of the signs determine whether an incoming shape's data would have addition or subtraction operation. The signs were located at a close distance from the endpoint of a connection but not connected. The connection could come to the summation element from right, left or top direction hence, signs were also located in various position. In some cases, a sign could have equal distance from both endpoints of incoming connections. Also, a sign could be in an unfavourable position, for example, a plus (+) sign could be closer to an incoming connection where in fact the minus (-) sign should have been the closer one. To resolve these types of scenario always relative distances of both signs were compared against distance from both of the endpoints to apply the operation accurately.

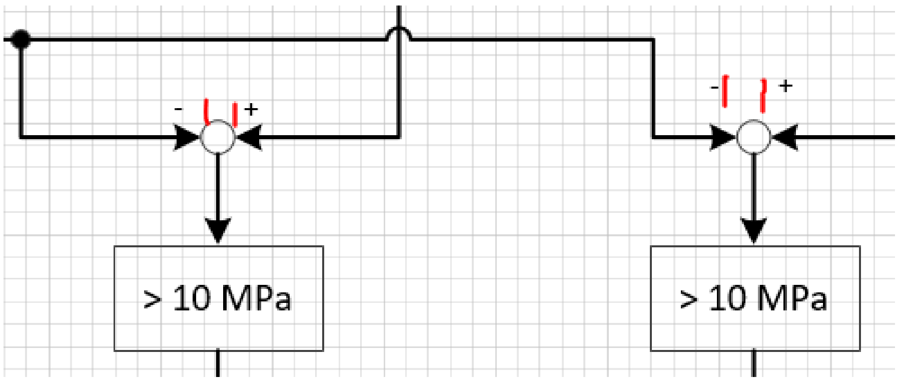


Figure 31. *summation element with disconnected signs(+/-)*

In several scenarios, there were occurrences of a direct connection between two logical components in an actual FBD network but was completely absent in the visual representation in VISIO. To resolve such type of issues, a virtual connection was assigned to the component instances in the form of ID of previous/next shape found by various geometric shape rationale. Figure 32 shows such an example where a connection must exist between Logical “AND” and inversion element (represented by circular shape). The flip-flop in the same figure would have a connection from the inversion element but in this case, it required more assessment to determine whether a connection would go into R or S block inside the flip-flop. It was important to remember that a flip-flop was a shape group, so the connection should be assigned in such a way that connection can be tracked to individual inner shape (i.e. R and S).

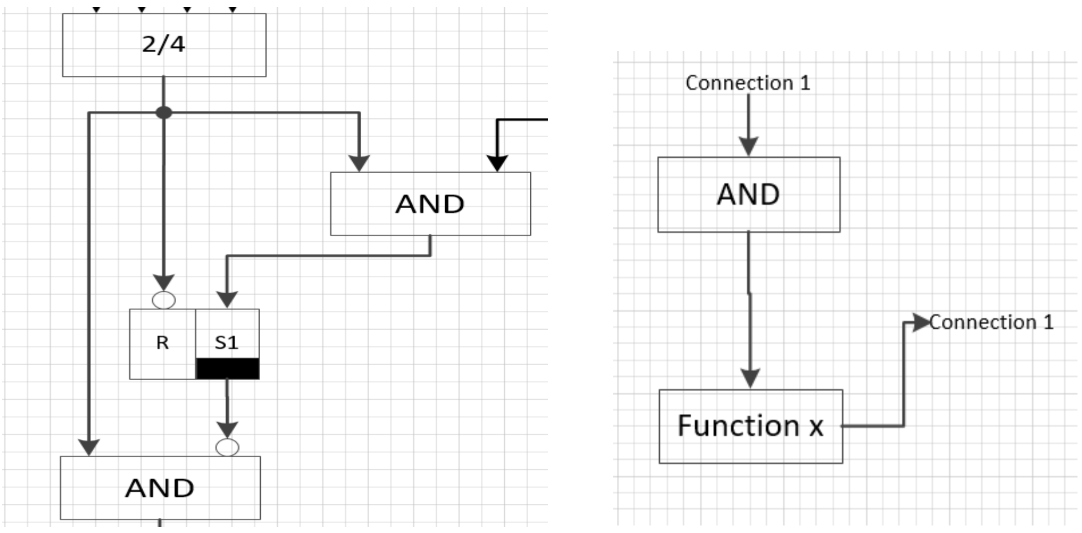


Figure 32. *Two examples where virtual connections required*

The diagrams had many of occurrences where it was visually comprehensible that output from point “A” goes to another point “A” as input but the actual connection was missing. Here “A” represents a connection but drawing the full connection was omitted for the sake of a cleaner overall view. Therefore virtual connection needed to be assigned to maintain the data flow within functional elements.

There were still a few shapes left unidentified at this point and attempts were taken to find any identifiable pattern within them if existed. For example, In shape groups the program searched for text pattern, the number of nested shapes, type and count of individual inner shapes, bound rectangle size etc., to find a reasonable pattern distinguishable from other shapes. All the unified and identified shapes as FBD elements were transferred into object instances and were added in a graph as nodes. The unidentified objects were also stored in that graph to maintain consistency during the visualization.

5.3 Component visualization

The function blocks identified and stored following the metamodel in the previous stages were eligible to be deployed for creating a visualization. However, if a unified shape was not identified still it had most of the properties as metamodel demands for, except for the "ShapeType" attribute of the element, hence were eligible to be a part of visualization. Moreover, logic block visualization carried more importance because it helped to spot identification error by visual means and portray the FBD element network as a graph to ensure that the layout was preserved from the original VISIO diagrams. For this visualization, GraphStream library was chosen. This java library supported many different types of graphs, from which the built-in "SingleGraph" was best suited for visualizing the formed tree structure of components. All the elements were added to the SingleGraph(from GraphStream) with the properties such as type, element ID, Cartesian co-ordinate in a 2D plane in the layout. A directed edge was added between components if a connection existed between two elements indicating dataflow among them. For the unidentified shapes, red colour was used in conjunction with original shape ID, which in turn, made the search of unidentified shapes easier.

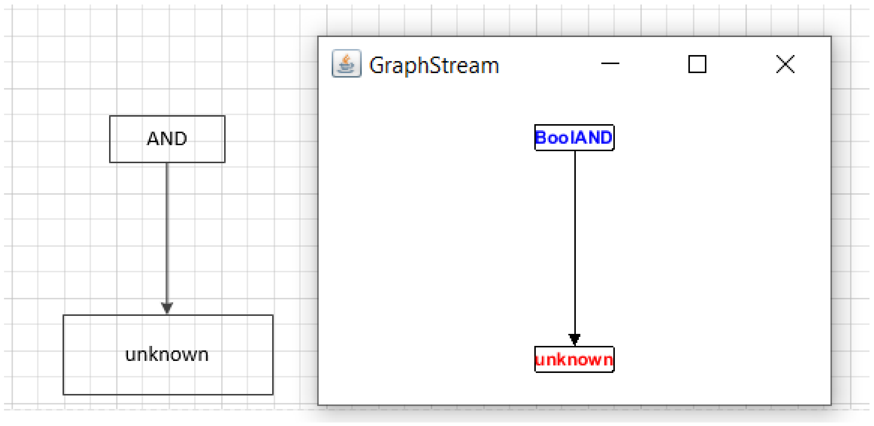


Figure 33. *Visualization of classified objects*

5.4 NuSMV script creation

The FBD elements can be now used to generate NuSMV code. This task required four key operations on the constructed model of function blocks.

In the first phase, the FBD elements needed to be classified into three types of elements. First, Input variables, which had no incoming connection and but had an only outgoing connection to other elements. Second, Output Variable, which had an incoming connection from other elements but did not have an outgoing connection. Third, the rest of the elements in between input and output variables in the network of elements, which were referred to here as middle variables. One way to check if the elements were connected was to create a graph with the elements and search for an existing path between the output and each input variable. If no path exists to a variable then that variables were kept aside from using to create NuSMV script.

Following the NuSMV syntax, declaration of each type of FBD element was stored. This declaration strings had placeholder(s) which were changed programmatically with the type of input it had received. For example: An AND function with two input connections had a declaration can be “and2 = AND2 ({0}, TRUE, {1}, TRUE)". Values in placeholder {0} and {1} were assigned by the program. The Status (signal validity) is omitted.

Each type of function such as, AND, voting logic, had a declaration in NuSMV which determined how a function executed. These blocks of codes were called a MODULE. Even though both of visitor-based approach and template-based approach were applicable but for the experimental implementation only template-based approach was applied. The NuSMV code elements modelling the elementary block processing logic were manually specified by a VTT expert.

Now all the classified FBD elements as input, output and middle variables were required to be iterated to generate the script. At the first part of the script, the input variables were placed followed respectively by middle variables and output variable. The next segment after the variable declaration was LTLSPEC which was supposed to consist of requirements. The analyst could use CTL or PSL as well. Model-checking determines if these requirements were satisfied by a model. The segment of LTLSPEC was left empty to be written later by editing the script. The last segment contained the declaration of function MODULEs. Those were already declared and stored. By adding the required MODULEs this part of the completed.

On top of adding the LTLSPEC, it is required to thoroughly examine to ensure all variables and modules are declared properly in the script and make the necessary correction.

6. RESULT

In this chapter result of the developed program is represented along with the limitation of this implementation.

6.1 Graphical representation of FBD

In order to identify the VISIO shapes as function block, the prerequisites are unification and classification of the shapes. The connection from shape to shape has to be generated from the original diagram by eliminating errors in the drawing. The mentioned requirements are completed successfully and following figure34 graphically represented the final shape (with their unique ID) and established connections using graph network powered by GraphStream.

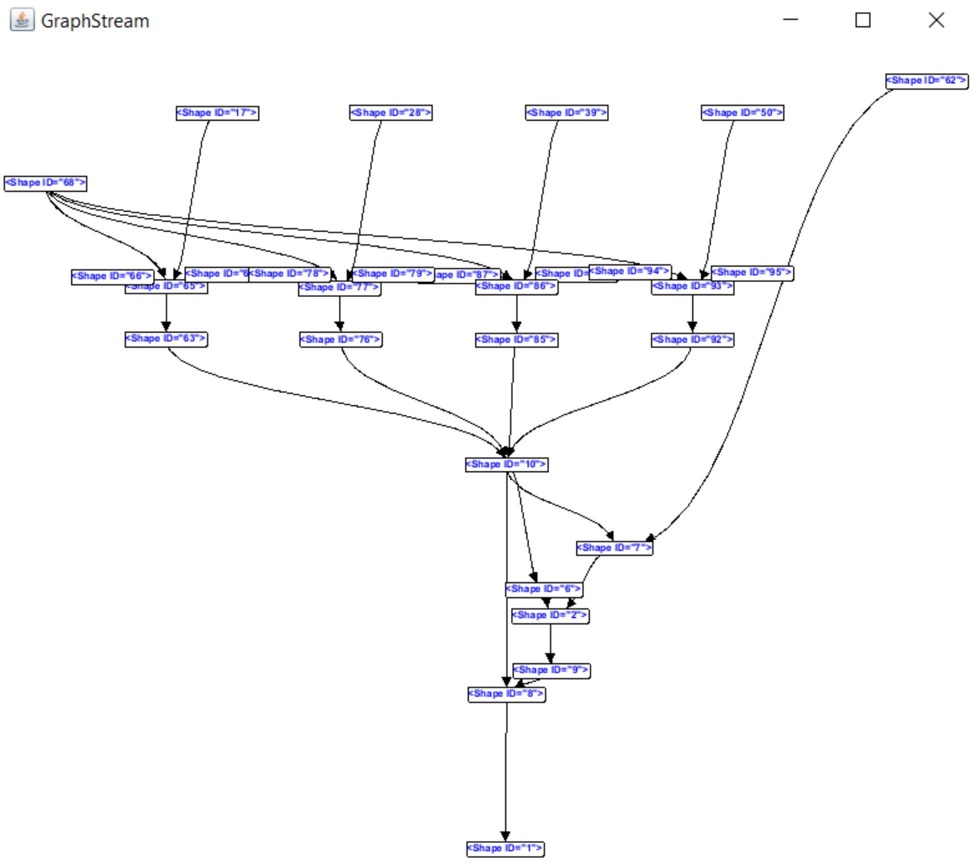


Figure 34. *Illustration of unified shapes and connections*

Finally, unified shapes were put to use for identification. The identification process was completed in multiple stages as described in the implementation chapter. Following diagram is a graphical representation of the function recognized by analysing pattern in the shapes.

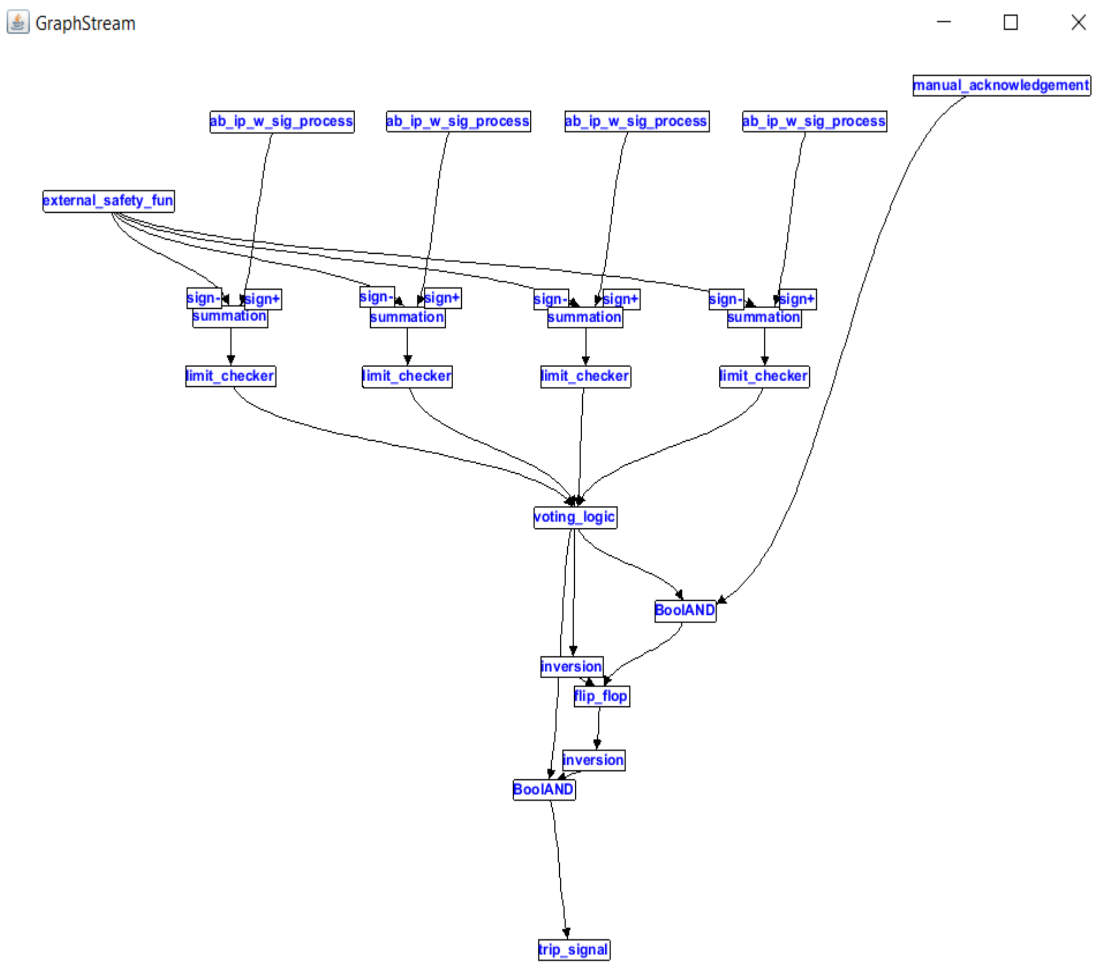


Figure 35. *Illustration of identified shapes as function blocks*

This result shows that the graphical layout in Cartesian space is preserved from the original diagram (figure 23). Even though the diagrams show the type of the function other data attributes are also preserved after analysis.

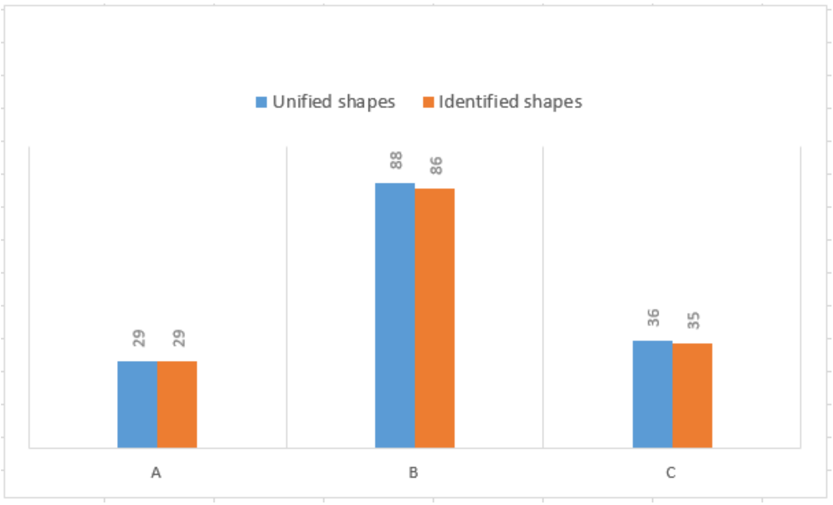
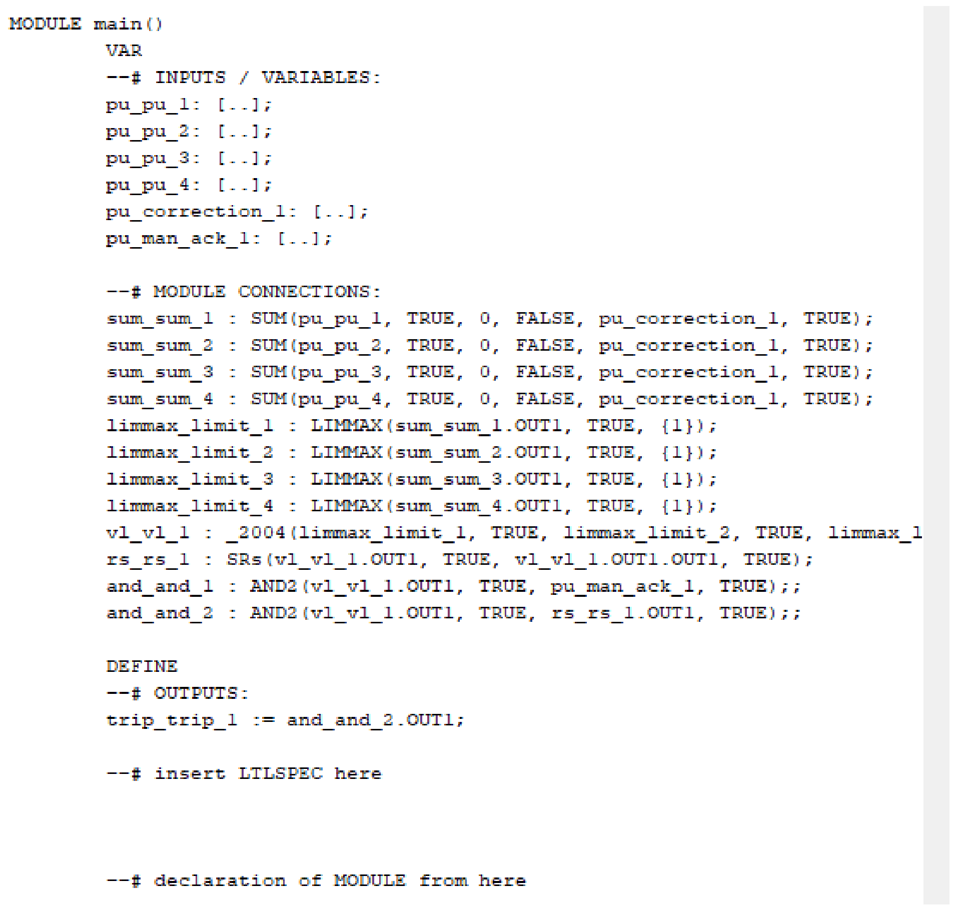


Figure 36. *Comparison: number of shapes after unification and identification*

For a better understanding of the accuracy of the program, figurexxx presents a bar chart that shows the number of shapes after unification vs total identified shapes using three reference diagram. It has been observed that there is no direct correlation of identification accuracy with the size of the diagram and shape count. Regardless of size, diagrams were processed with high accuracy where accuracy seems to be dependent on drawing style and complexity of shapes.

6.2 NuSMV script

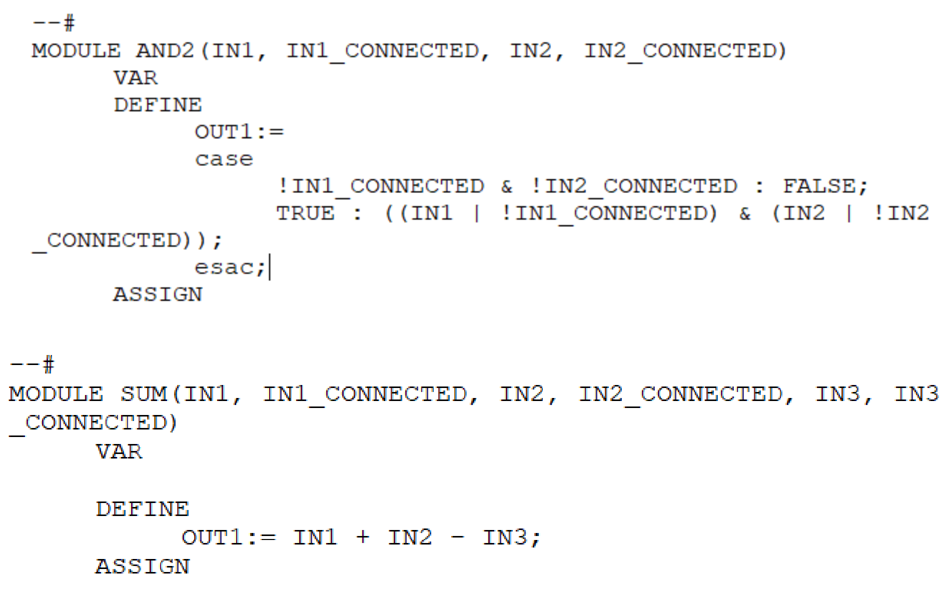
The following Figure is a generated script from the constructed FBD elements.



```
MODULE main()
        VAR
        --# INPUTS / VARIABLES:
        pu_pu_1: [..];
        pu_pu_2: [..];
        pu_pu_3: [..];
        pu_pu_4: [..];
        pu_correction_1: [..];
        pu_man_ack_1: [..];

        --# MODULE CONNECTIONS:
        sum_sum_1 : SUM(pu_pu_1, TRUE, 0, FALSE, pu_correction_1, TRUE);
        sum_sum_2 : SUM(pu_pu_2, TRUE, 0, FALSE, pu_correction_1, TRUE);
        sum_sum_3 : SUM(pu_pu_3, TRUE, 0, FALSE, pu_correction_1, TRUE);
        sum_sum_4 : SUM(pu_pu_4, TRUE, 0, FALSE, pu_correction_1, TRUE);
        limmax_limit_1 : LIMMAX(sum_sum_1.OUT1, TRUE, {1});
        limmax_limit_2 : LIMMAX(sum_sum_2.OUT1, TRUE, {1});
        limmax_limit_3 : LIMMAX(sum_sum_3.OUT1, TRUE, {1});
        limmax_limit_4 : LIMMAX(sum_sum_4.OUT1, TRUE, {1});
        vl_vl_1 : _2004(limmax_limit_1, TRUE, limmax_limit_2, TRUE, limmax_l
        rs_rs_1 : SRs(vl_vl_1.OUT1, TRUE, vl_vl_1.OUT1.OUT1, TRUE);
        and_and_1 : AND2(vl_vl_1.OUT1, TRUE, pu_man_ack_1, TRUE);;
        and_and_2 : AND2(vl_vl_1.OUT1, TRUE, rs_rs_1.OUT1, TRUE);;

        DEFINE
        --# OUTPUTS:
        trip_trip_1 := and_and_2.OUT1;

        --# insert LTLSPEC here



        --# declaration of MODULE from here
```

Figure 37. *Created NuSMV variables*

While creating this script the data type for the variables (under “VAR”) is left empty. If the data type (e.g. BOOLEAN) is known before, it can be added programmatically. Several variables require specific values, hence can be added later by editing the script. At the bottom of the segment presented in Figure 37 followed by several MODULEs. Figure 38 shows SUM and AND2 specified by VTT expert.



```
--#
MODULE AND2(IN1, IN1_CONNECTED, IN2, IN2_CONNECTED)
      VAR
      DEFINE
            OUT1:=
            case
                  !IN1_CONNECTED & !IN2_CONNECTED : FALSE;
                  TRUE : ((IN1 | !IN1_CONNECTED) & (IN2 | !IN2
_CONNECTED));
            esac;
      ASSIGN

--#
MODULE SUM(IN1, IN1_CONNECTED, IN2, IN2_CONNECTED, IN3, IN3
_CONNECTED)
      VAR

      DEFINE
            OUT1:= IN1 + IN2 - IN3;
      ASSIGN
```

Figure 38. *AND2 and SUM module in NuSMV script*

6.3 Discussion: Outcomes and Limitations

The development of the program for the thesis requires a fine blend of learnings to answer the research questions. The process was also helpful to discover several limitations. However, from the limitations also new possibilities of future development arise. The findings of the research questions are pointed below.

- Importing the VISIO diagrams was easy, although some shapes had null data. This issue was eventually resolved in the program.
- It was possible to define a flexible meta-model where the creation of function block elements could conform to that. However, there was no opportunity to check the robustness practically by integrating with other tools and frameworks.
- Creation of model was completed and for the most part identification of shapes was performed. There is room for improvement to level up the accuracy to 100%.
- Graphic representation was successful. In future, this visualization can be improved to add more capabilities. For example: Displaying detailed properties of shapes on click from the graph would be a nice to have feature.
- Creation of NuSMV script is a way to demonstrate that the processed shape data from VISIO diagram I&C can be utilized. However, the script does not have requirements in LTLSPEC which is required to be added manually.

To summarize the key achievement, it was a successful effort to use the VISIO I&C designs and technical know-how on the development process was achieved.

7. CONCLUSION AND FUTURE RESEARCH

This chapter concludes the thesis with a summary of the implementation completed so far. Several possible development scope is discussed as a future research opportunity.

7.1 Conclusion

This research journey began with the observation that the verification of nuclear power plant-specific I&C system involves several steps of design data transfer within design and verification tools. The cumbersome manual transfer process is a bottleneck in overall verification progression. To automate the process and ease the complication of manual model recreation process, a transitional stage based on suitable metamodel was determined to be a viable solution. Intermediate FBD model, in particular, aimed at solving several hindrances posed by the model-to-model transformation of non-standard nuclear I&C diagrams. This thesis is a research effort to mitigate this limitation, aimed at assisting model checking whilst heavily focused on I&C diagrams transformation from Microsoft VISIO. Abstract overview of the necessary concepts was provided in the theoretical background chapter as a precursor to the state-of-the-art. State-of-the-art elaborated on the concepts and discussed use cases related to these topics of this thesis. Research methodology contains development approaches and strategies. All the important steps and crucial deciding factors are detailed in the implementation chapter to illustrate the decision behind programming architecture and class design decision. An evaluation of performance pertaining to the accuracy of the model transformation is portrayed in the result chapter, which also holds the limitations of the current development efforts. The main achievement of this thesis is the experience and technical know-how of transforming a non-standard nuclear I&C diagram..

7.2 Future Development Scope

The generated function block component network is based upon a metamodel suitable for expressing captured properties and extending to integrate with other tools. A plausible direction of future research is integration with MODCHK by importing the processed diagram if suitable import capability is available. Another scope could be the modification of the visuals in VISIO diagram based on the result generated by NuSMV.

REFERENCES

[1] “nuclear energy explained risk-or-opportunity” [Online]. Available: https://www.eia.gov/totalenergy/data/annual/. [Accessed: 12-Dec-2019]

[2] “Instrumentation and Control engineering is for perfectionist” [Online] Available: https://www.electronicsforu.com/resources/instrumentation-control-engineering-perfectionists. [Accessed: 12-Dec-2019]

[3] Jae-Gu Song, Jung-Woon Lee, Park Gee-Yong, “An analysis of technical security control requirements for digital I&C systems in nuclear power plants”, Nuclear Engineering and Technology, 2013

[4] L.Finkelstein, “Measurement and instrumentation science-An analytical review”, Measurement, Volume 14, Issue 1, Pages 3-14, 1994

[5] JJ Di Steffano, AR Stubberud, IJ Williams. Schaums, “Feedback and control systems" - outline series, McGraw-Hill, 1967

[6] NASA, “INSTRUMENTATION AND CONTROL QUALIFICATION STANDARD REFERENCE GUIDE”, Instrumentation and Control Qualification Standard, DOE-STD-1162-2013, June 2013, https://www.energy.gov/sites/prod/files/2013/10/f4/QSR-InstrumentationControl.pdf

[7] Jonathan Love, “Process Automation Handbook”, Springer, 2007.

[8] Bellman, Richard E, “Adaptive Control Processes: A Guided Tour”. Princeton University Press, 1961

[9] Henry, M, OY Bushuev, and OL Ibryaeva, “Prism Signal Processing for Sensor Condition Monitoring”, IEEE, 2017

[10] Ibrahim F. Taha Alzaidi, “Adaptive iterative learning control for a linear system with unknown parameters”, University of Turkish Aeronautical Association Institute of Science and Technology.

[11] Instrument Society of America, “Process Instrumentation Terminology”, 1979

[12] Robert G. Jenkins, Anupam Sanyal, Alan Williams, ”Fuel and Energy Abstracts”, Volume 46, Issue 5, Pages 303-304, 2005

[13] OVERVIEW OF INSTRUMENTATION AND CONTROL SYSTEMS FOR NUCLEAR POWER PLANTS, IAEA Nuclear Energy Series, No. NP-t-3.12

[14] ”TUV Rheinland”, [Online] Available: https://www.tuv.com/world/en/testing-and-evaluating-of-i-c-systems-in-npps.html. [Accessed: 05-Jan-2020]

[15] A. Selwin, Mich Priyadharson, M.S. Saravanan, N. Gomathi, S. Vinson Joshua and A. Mutharasan, “Energy-efficient flow and level control in a hydropower plant using fuzzy logic”, Journal of Computer Science, 2014

[16] Cimatti, Alessandro & Clarke, Edmund & Giunchiglia, Fausto & Roveri, Marco. “NUSMV: a new symbolic model checker”, 2000.

[17] "Verification of automation software by model checking", VTT Technical Research Centre Of Finland, [Online]. Available: www.simulation-store.com/node/52, [Accessed: 04-April-2020]

[18] "Automation Basics Programmable Logic Controller", [Online], Available: https://www.isa.org/standards-publications/isa-publications/intech-magazine/2010/december/automation-basics-programmable-logic-controllers-hardware-software-architecture/ . [Accessed: 21-Mar-2020]

[19] "Code Components of IEC 61131-10 PLC open XML exchange format", [Online] Available: https://plcopen.org/sites/default/files/downloads/plcopen_xml_exchange.pdf, [Accessed: 21-Mar-2020]

[20] B. Vogel-Heuser, S. Braun, M. Obermeier, K. S., and K. Schweizer, "Usability evaluation on teaching and applying model-driven object oriented approaches for PLC software", American Control Conference (ACC), 2012.

[21] Amal Khalil, Juergen Dingel, in Advances in Computers, Optimizing the Symbolic Execution of Evolving Rhapsody Statecharts, Elsevier, 2018

[22] "guana, V: running acceleo and ATL transformation Programmatically". University of Alberta(2016) [Online]. Available: http://victorguana.blogspot.com/2016/05/running-acceleo-and-atl-transformations.html][system analysis and modelling .[Accessed: 11-Mar-2020]

[23] Nafiseh Kahani, Mojtaba Bagherzadeh, James R. Cordy, Juergen Dingel & Daniel Varró, "Survey and classification of model transformation tools", Software & Systems Modeling volume 18, 2019

[24] Rothenberg, J., Widman, L., Loparo, K., Nielsen, N, "The nature of modeling", 1989

[25] Brambilla, M., Cabot, J., Wimmer, M, "Model-driven software engineering in practice". Morgan & Claypool Publishers, 2012

[26] Jakumeit, E., Buchwald, S., Wagelaar, D., Dan, L., Hegedüs, Á., Herrmannsdörfer, M., Horn, T., Kalnina, E., Krause, C., Lano, K., Lepper, M., Rensink, A., Rose, L., Wätzoldt, S., Mazanek, S, "A survey and comparison of transformation tools based on the transformation tool contest", Elsevier, 2014

[27] Jean-Louis Boulanger, in Certifiable Software Applications 3, 1st edition, Chapter 1, 2018

[28] "Query/views/transformation language (QVT)" [Online]. Available: http://www.omg.org/spec/QVT. Accessed 16 Feb 2018 .[Accessed 03-jan-2020]

[29] Andries, M., Engels, G., Habel, A., Hoffmann, B., Kreowski, H.J., Kuske, S., Plump, D., Schürr, A., Taentzer, G. "Graph transformation for specification and programming". Science of Computer Programming, 1999

[30] Uhl, A, "Model-driven development in the enterprise", IEEE Software, 2008

[31] Lúcio, L., Amrani, M., Dingel, J., Lambers, L., Salay, R., Selim, G., Syriani, E., Wimmer, M, "Model transformation intents and their properties". Software and Systems Modeling, 2014

[32] IEEE standard computer dictionary: A compilation of IEEE standard computer glossaries, IEEE, 1990.

[33] Fred´ eric Jouault, Ivan Kurtev, “On the interoperability of model-to-model transformation languages”, Elsevier, 2007

[34] Tom Mens,Pieter Van Gorp, ”A taxonomy of model transformation”, Elsevier, 2006

[35] Amine Benelallam, Abel Gomez, Massimo Tisi, Jordi Cabot, “Distributed model-to-model transformation with ATL on MapReduce”, Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering, 2015

[36] Alexander Egyed, Andreas Demuth, Achraf Ghabi, Roberto Lopez-Herrejon, Patrick Mäder, Alexander Nöhrer, and Alexander Reder, Fine-Tuning Model Transformation: Change Propagation in Context of Consistency, Completeness, and Human Guidance, ICMT, 2011.

[37] Stahl, Thomas & Völter, Markus, Model-Driven Software Development, John Wiley & Sons, 2006.

[38] Advances in Computers. Levi Lúcio, ... Yves Le Traon, in Advances in Computers, 2014

[39] Amal Khalil, Juergen Dingel, “Optimizing the Symbolic Execution of Evolving Rhapsody Statecharts”, Advances in Computers, 2018

[40] Kevin Lano, Shekoufeh Kolahdouz-Rahimi, “Model Transformation Specification and Design”, Advances in Computers, 2012

[41] Mohamed Arezki Mellal, “Soft Computing Methods for System Dependability”, IGI Global, 2019

[42] Gabriel Barbier, Frédéric Madiot, A Model-Driven Platform to Support Real Legacy Modernization Use Cases, Information Systems Transformation, 2010.

[43] R Lutz, “Analyzing software requirements errors in safety-critical embedded systems”, IEEE International Symposium on Requirements Engineering, January 1993.

[44] Sobel, Ann & Clarkson, Michael, “Formal Methods Application: An Empirical Tale of Software Development”. Software Engineering, IEEE Transactions, 2002

[45] Russinoff, David M, “A Mechanically Checked Proof of Correctness of the AMD K5 Floating Point Square Root Microcode.” *Formal Methods in System Design,* 1999

[46] Anthony Hall, “Using Formal Methods to Develop an ATC Information System “, IEEE Software,1996

[47] Chen, Zhe & Gu, Yi & Huang, Zhiqiu & Zheng, Jun & Liu, Chang & Liu, Ziyi, “Model checking aircraft controller software: A case study, Software: Practice and Experience, 2013

[48] Jae-Gu Song, Jung-Woon Lee*, Cheol-Kwon Lee, Kee-Choon, “A cyber security

risk assessment for the design of I&C systems in nuclear power plants", 2012.

[49] Yu Lu, Zhaoguang Peng, Alice A. Miller, Tingdi Zhao, Christopher W. Johnson, How reliable is satellite navigation for aviation? Checking availability properties with probabilistic verification, Elsevier, 2015

[50] Quer, S. Model checking evaluation of airplane landing trajectories. Int J Softw Tools Technol Transfer, 2014

[51] S. Kim, W. Nam, H. Kil and M. Park, "Formal Verification of a Gravity-Induced Loss-of-Consciousness Monitoring System for Aircraft", *Computing in Science & Engineering*, 2012

[52] Bolton, Matthew L ; Bass, Ellen J, "Using Model Checking to Explore Checklist-Guided Pilot Behavior" ,The International Journal of Aviation Psychology, Vol.22(4),2012

[53] Buzhinsky, Igor & Pakonen, Antti, "Model-checking detailed fault-tolerant nu-clear power plant safety functions". IEEE, 2019.

[54] Cerny, Eduard & Dudani, Surrendra & Havlicek, John & Korchemny, Dmitry, "The Power of Assertions in SystemVerilog", Springer, 2011

[55] Thomas Osterland, Thomas Rose, "Model checking smart contracts for Ethereum, Pervasive and Mobile Computing", Volume 63, 2020,

[56] Yeung, W.L, "Formal verification of negotiation protocols for multi-agent manu-facturing systems. International Journal of Production Research", Wiley, 2013

[57] Zahir Tari; Peter Bertok; Kazi Sakib, "Model checking, Verification of Communi-cation Protocols in Web Services: Model-Checking Service Compositions", Pub-lished by Wiley, 2013

[58] Solovyev, Alexey & Hales, Thomas, "Formal Verification of Nonlinear Inequali-ties with Taylor Interval Approximations", NASA Formal Methods Symposium, 2013

[59] V´ıctor Braberman1, Diego Garbervestky1, Nicol´as Kicillof2, Daniel Monte-verde, Alfredo Olivero3 ,"Speeding Up Model Checking of Timed-Models by Combining Scenario Specialization and Live Component Analysis",

[60] Cabodi, Gianpiero & Camurati, P. & Palena, Marco & Pasini, Paolo & Ven-draminetto, Danilo, "Interpolation-Based Learning as a Mean to Speed-Up Bounded Model Checking (Short Paper)", Springer, 2017

[61] Barnat, Jiri & Brim, Lubos & Ceska, Milan & Lamr, Tomas. "CUDA accelerated LTL model checking". Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS,.2009.

[62] "What is Formal Methods",[Online] Available: https://shemesh.larc.nasa.gov/fm/fm-what.html [Accessed: 15-Apr-2020]

[63] Hubert Kaeslin, "Top-Down Digital VLSI Design", Morgan Kaufmann, 2014

[64] Cook, Byron & Kroening, Daniel & Sharygina, Natasha. "Accurate Theorem Proving for Program Verification", Journal on Software Tools for Technology

Transfer, 2011

[65] Filliâtre, Jean-christophe, “Deductive software verification”, International Journal on Software Tools for Technology Transfer, Heidelberg Vol. 13, Iss. 5, 2011

[66] Asok ray, Rogelio Luck, “An introduction to Sensor Signal Validation in Redundant Measurement Systems”. IEEE, 1991

[67] Pakonen, A., & Björkman, K. “Model checking as a protective method against spurious actuation of industrial control systems", CRC Press, 2017

[68] Maidl M, “A Unifying Model Checking Approach for Safety Properties of Parameterized Systems” Springer, 2001

[69] Armin Biere, Marijn Heule, Hans van Maaren and Toby Walsh, Handbook of Satisfiability (Eds.), IOS Press, 2009

[70] Marc Frappier Benoît Fraikin Romain Chossart Raphaël Chane-Yack-Fa Mohammed Ouenzar, “Comparison of Model Checking Tools for Information Systems”, International Conference on Formal Engineering Methods, 2010

[71] J. Lahtinen, J. Valkonen, K. Bjorkman , J. Frits, I. Niemela, K. Heljanko. “Model checking of safety critical software in the nuclear engineering domain”, Reliability Engineering and System Safety, 2012

[72] Gerard O'Regan, “Concise Guide to Formal Methods Theory, Fundamentals and Industry Applications”, Springer, 2017

[73] Niloofar Razavi, Razieh Behjati, Hamideh Sabouri, Ehsan Khamespanah, Amin Shali, and Marjan Sirjani. “Sysfier: Actor-based formal verification of SystemC, ACM Transactions on Embedded Computing Systems, 2010

[74] Huth, Michael & Ryan, Mark. (1999). Logic in Computer Science: Modelling and Reasoning About Systems. Logic in Computer Science: Modelling and Reasoning about Systems, Cambridge University Press, 2004

[75] "Linear-time Temporal Logic". Archived from the original on 2017-04-30. Retrieved 2012-03-19.

[76] Diekert, Volker. " First-order Definable Languages ", University of Stuttgart, 2008

[77] Kamp, Hans, “Tense Logic and the Theory of Linear Order”, University of California Los Angeles, 1968

[78] Baier, Christel; Katoen, Joost-Pieter, Principles of Model Checking (Representation and Mind Series). The MIT Press, 2008

[79] E. Allen Emerson, Joseph Y. Halpern: "Sometimes and not never revisited: on branching versus linear time”, International Conference on Concurrency Theory, 1998

[80] Dov M. Gabbay; A. Kurucz; F. Wolter; M. Zakharyaschev, “Many-dimensional modal logics: theory and applications”, Elsevier, 2003

[81] “Introduction to Formal Verification”, [Online] Available:https://ptolemy.berkeley.edu/projects/embedded/research/vis/doc/VisUser/vis_user/node4.html , [Accessed:15-Apr-2020]

[82] Jingyue Li, Altin Qeriqi, Martin Steffen, Ingrid Chieh Yu, “Automatic Translation of FBD-PLC-programs to NuSMV for Model Checking Safety-Critical Control Systems”, NIK-2016 conference, 2016

[83] “NuSMV User Manual” [Online] Available:http://nusmv.fbk.eu/NuSMV/userman/v26/nusmv.pdf. [Accessed: 21-Feb-2020]

[84] Roberto Cavada, Alessandro Cimatti, Charles Arthur Jochim, Gavin Keighren, Emanuele Olivetti, Marco Pistore, Marco Roveri and Andrei Tchaltsev, “NuSMV 2.5 User Manual”, 2010

[85] Gutiérrez, Maria Encarnación & Barrio, Manuel & Cuesta, Carlos E. & Fuente, Pablo, “UML Automatic Verification Tool with Formal Methods”, Electronic Notes in Theoretical Computer Science, 2005

[86] 33 Kölbl M., Leue S., Singh H, “From SysML to Model Checkers via Model Transformation”, Model Checking Software. SPIN 2018. Lecture Notes in Computer Science, 2018

[87] Jérôme Kunegis, Andreas Lommatzsch, “Learning spectral graph transformations for link prediction”, ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning, 2009

[88] S. Kim, W. Nam, H. Kil and M. Park, "Formal Verification of a Gravity-Induced Loss-of-Consciousness Monitoring System for Aircraft," *Computing in Science & Engineering*, 2014.

[89] “Comparison between Lucidchart and Microsoft Visio [Online], Available: ”https://www.lucidchart.com/pages/what-is-microsoft-visio[Accessed: 11-Nov-2020]

[90] “Apache.Poi Documentation”, [Online], Available: “http://poi.apache.org/components/diagram/”, [Accessed: 10-Nov-2020]

[91] István Majzik, András Pataricza, Andrea Bondavalli, “Stochastic Dependability Analysis of System Architecture Based on UML Models”, DBLP, Conference: Architecting Dependable Systems [the book is a result of the ICSE 2002 Workshop on Software Architectures for Dependable Systems, 2002

[92] Alireza Rouhi, “Presenting a process for generating a pattern language verifier”, DOI: 10.13140/RG.2.2.16991.97446, 2017

[93] Uribe T.E., “Combinations of Model Checking and Theorem Proving”, Frontiers of Combining Systems. (FroCoS). Lecture Notes in Computer Science, vol 1794. Springer, 2000

[94] Cepin, M. (Ed.), Bris, R. (Ed.). (2017). Safety and Reliability. Theory and Applications. London: CRC Press, https://doi.org/10.1201/9781315210469

[95] A. Pakonen, T. Mätäsniemi, J. Lahtinen and T. Karhela, "A toolset for model checking of PLC software," 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA), Cagliari, 2013, pp. 1-6.

APPENDIX A: A SAMPLE NUSMV DESCRIPTION

```
--# Description:
MODULE main()
     VAR
          --# INPUTS / VARIABLES:
          PRESSURE_1: 0..20;
          PRESSURE_4: 0..20;
          CORRECTION: 0..5;
          PRESSURE_3: 0..20;
          MCR_ACK: boolean;
          PRESSURE_2: 0..20;

          --# MODULE CONNECTIONS:
          SRs002 : SRs(AND2002.OUT1, TRUE, !_2oo4002.OUT1,
TRUE);
          LIMMAX004 : LIMMAX(SUM004.OUT1, TRUE, 10);
          SUM001 : SUM(PRESSURE_4, TRUE, 0, FALSE, CORRECTION,
TRUE);
          LIMMAX002 : LIMMAX(SUM002.OUT1, TRUE, 10);
          SUM004 : SUM(PRESSURE_1, TRUE, 0, FALSE, CORRECTION,
TRUE);
          LIMMAX003 : LIMMAX(SUM003.OUT1, TRUE, 10);
          AND2001 : AND2(!SRs002.OUT1, TRUE, _2oo4002.OUT1,
TRUE);
          AND2002 : AND2(MCR_ACK, TRUE, _2oo4002.OUT1, TRUE);
          LIMMAX001 : LIMMAX(SUM001.OUT1, TRUE, 10);
          _2oo4002 : _2oo4(LIMMAX001.OUT1, TRUE, LIMMAX002.OUT1,
TRUE, LIMMAX003.OUT1, TRUE, LIMMAX004.OUT1, TRUE);
          SUM002 : SUM(PRESSURE_3, TRUE, 0, FALSE, CORRECTION,
TRUE);
          SUM003 : SUM(PRESSURE_2, TRUE, 0, FALSE, CORRECTION,
TRUE);

     DEFINE
          --# EXTERNALS:
          TRIP := AND2001.OUT1;

          --# OUTPUTS:

          --# REQS:
          --# If 2 out the 4 measurements are over the limit,
and the operator has not acknowledged the
--# trip, the trip signal shall be set.
LTLSPEC G ((( count(((PRESSURE_1 - CORRECTION) > 10),
                          ((PRESSURE_2 - CORRECTION) > 10),
                          ((PRESSURE_3 - CORRECTION) > 10),
                          ((PRESSURE_4 - CORRECTION) > 10)) >= 2)
& H !MCR_ACK) -> TRIP); --# FALSE

--# If 2 out of 4 measurements are still over the limit, but the
operator has acknowledged the trip,
-- the trip signal shall be reset.
LTLSPEC G (TRIP & X MCR_ACK -> X !TRIP); --# TRUE
```

```
--# If 2 out of 4 measurements fall below the limit, the trip
signal shall be reset.
LTLSPEC G ((count(((PRESSURE_1 - CORRECTION) > 10),
                  ((PRESSURE_2 - CORRECTION) > 10),
                  ((PRESSURE_3 - CORRECTION) > 10),
                  ((PRESSURE_4 - CORRECTION) > 10)) < 2)
-> !TRIP); --# TRUE



     ASSIGN
          --# INTERNAL STATE:


--# 2OO4 (2 out of 4)
--#
--# SEARCH / SAFIR2022 public example
--# Author(s): Antti Pakonen
--# Version 21.11.2019
--#
--# Validity processing ignored
--#
MODULE _2oo4(IN1, IN1_CONNECTED, IN2, IN2_CONNECTED, IN3,
IN3_CONNECTED, IN4, IN4_CONNECTED)
     VAR
     DEFINE
          OUT1:= (count(IN1,IN2,IN3,IN4) > 1);
     ASSIGN

--# LIMMAX (Maximum limit value)
--#
--# SEARCH / SAFIR2022 public example
--# Author(s): Antti Pakonen
--# Version 21.11.2019
--#
--# Validity processing ignored
--#
MODULE LIMMAX(IN1, IN1_CONNECTED, MaxValue)
     VAR

     DEFINE
          OUT1:= (IN1 > MaxValue);
     ASSIGN


--# AND2 (2-gate AND)
--#
--# SEARCH / SAFIR2022 public example
--# Author(s): Antti Pakonen
--# Version 9.3.2020
--#
--# Validity processing ignored
--#
MODULE AND2(IN1, IN1_CONNECTED, IN2, IN2_CONNECTED)
     VAR
     DEFINE
          OUT1:=
          case
               !IN1_CONNECTED & !IN2_CONNECTED : FALSE;
               TRUE : ((IN1 | !IN1_CONNECTED) & (IN2 | !IN2_CON-
NECTED));
          esac;
     ASSIGN


--# SUM (Summation)
--#
--# SEARCH / SAFIR2022 public example
--# Author(s): Antti Pakonen
--# Version 9.3.2020
--#
--# Validity processing ignored
--#
MODULE SUM(IN1, IN1_CONNECTED, IN2, IN2_CONNECTED, IN3, IN3_CON-
NECTED)
     VAR

     DEFINE
          OUT1:= IN1 + IN2 - IN3;
     ASSIGN


--# SET-RESET (set initialisation)
--#
--# SEARCH / SAFIR2022 public example
--# Author(s): Antti Pakonen
--# Version 21.11.2019
--#
--# Validity processing ignored
--#
MODULE SRs(SET, SET_CONNECTED, RESET, RESET_CONNECTED)
     VAR
          mem : boolean;
     DEFINE
          OUT1:=
          case
               SET : TRUE;
               RESET : FALSE;
               TRUE : mem;
          esac;
          OUT2:= !OUT1;
     ASSIGN
          init(mem) := TRUE;
          next(mem) := OUT1;
```