VTT Technical Research Centre of Finland

Transformation of non-standard nuclear I&C logic drawings to formal verification models

Pakonen, Antti; Biswas, Prasun; Papakonstantinou, Nikolaos

Published in:
IECON 2020

DOI:
10.1109/iecon43393.2020.9255176

Published: 18/10/2020

Document Version
Peer reviewed version

Transformation of non-standard nuclear I&C logic drawings to formal verification models

Antti Pakonen, Prasun Biswas, Nikolaos Papakonstantinou
VTT Technical Research Centre of Finland Ltd
Espoo, Finland
name.surname@vtt.fi

Abstract — **Model checking methods have been proven to be a valuable asset for identifying undesired behaviour of safety-critical Instrumentation and Control (I&C) logics. Their application in the nuclear domain has been very successful and has triggered significant interest from the safety community. Creating formal models from the diagrams found on paper or from digital formats without the needed semantics is one bottleneck that hinders the adoption of model checking due to costs in time and may introduce errors. This paper proposes a methodology for the creation of formal models from I&C diagrams drawn in generic modelling tools (lacking specific I&C semantics). The generic I&C logic diagram is transformed into an intermediate UML model that in turn can be transformed to other target formats like IEC 61131 PLCopen XML I&C software or NuSMV formal model code. This methodology is demonstrated with a typical example of a trip signal generator application logic. This application logic is drawn in MS Visio, it is transformed to an I&C model in UML with the needed properties for model checking, then to IEC 61131 PLCopen XML and to an input file for the NuSMV model checker.**

Keywords—I&C, function block diagram, nuclear energy, IEC61131, PLCOpen XML, Model-Based System Engineering

I. INTRODUCTION

Model checking has been proven very useful in verifying the design of instrumentation and control (I&C) system application logics in nuclear power plants [1]. Formal verification tools (such as model checkers) aim at mathematical proof of the of the design's correctness, which means that design issues are revealed in systems that have already subjected to more conventional V&V methods like testing, simulation, or manual review [1].

Even if supported by graphical tools, one of the bottlenecks in I&C system model checking is that the analyst has to manually recreate the function block diagram, even when the original diagrams are available in an electronic format. Attempts have been made for automatic model-to-model translation, but a domain-specific challenge is that the nuclear I&C system vendors rely on closed, non-standard solutions [2]. The source code for the basic blocks is usually not available, only functional descriptions. There is typically no access to the qualified engineering tools used by vendors to design the logics, and the block diagrams are only available (to the plant operator or the regulatory body) as PDF or MS Office document files.

In a broader scope, we are not interested only on point-to-point translation from nuclear I&C design diagrams to a certain model checking tool. Instead, we aim at developing a framework for efficient co-use of different modelling and analysis tools, so that any change made in the I&C system design could immediately facilitate a re-assessment, using a wide range of different modelling and analysis methods. The tools can include, e.g., different types of model checkers, but also tools related to testing, simulation, probabilistic safety assessment (PSA), failure modes and effects analysis (FMEA), or fault tree analyses (FTA). A practical problem therein is that the different tools have a different point of view on the system (and the plant), and therefore use different models.

Model-based Systems Engineering (MSBE) is an approach that can be described as "the formalized application of modelling principles, methods, languages, and tools to the entire lifecycle of large, complex, interdisciplinary, sociotechnical systems" [3]. To facilitate such an approach, we aim at developing a common model of the plant and its systems as a multidisciplinary whole, and enable translations between the common plant model and various tool specific representations. For capturing I&C aspects, we hope to utilise open standards (IEC 61131-3 [4], PLCopen XML [5], UML [6], etc.) to the extent that is feasible, given the nuclear industry specific constraints and requirements.

The scientific contribution of this paper is a workflow for model checking diagrams when the application logic is drawn in generic modelling tools that lack the necessary semantics (only using not strictly enforced drawing conventions).

A reusable I&C logic UML metamodel is introduced for exchanging vendor-specific, non-standard function block diagrams. This generic metamodel has the capability to maintain the allocation to processors (crucial aspect in I&C systems, where redundant processors are used to make the systems fault tolerant [7]), diagram layout & block symbols. This UML model can be used for generating I&C software following the IEC 61131 PLCopen XML standard that can be imported to an I&C software development environment. This intermediate model can also be used to partially generate the formal model code needed (in NuSMV) for model checking of the I&C logic design, so that the necessary vendor-specific features (not directly supported by IEC 61131) are maintained in the transformation.

We demonstrate our approach with an example based on a real design issue detected using model checking in a practical nuclear industry project.

II. PRELIMINARIES

A. Model checking for nuclear I&C systems

Model checking [8] is a formal verification method, where a formal property of a system is verified trough exhaustive exploration of all the reachable states of a system model. If a software tool called a model checker is able to find a model execution path violating a stated property, that path is returned to the user as a counterexample trace. There are differing types of model checkers available, but the system model is usually expressed as a type of Finite State Machine (FSM) and the properties are formalised using temporal logic languages

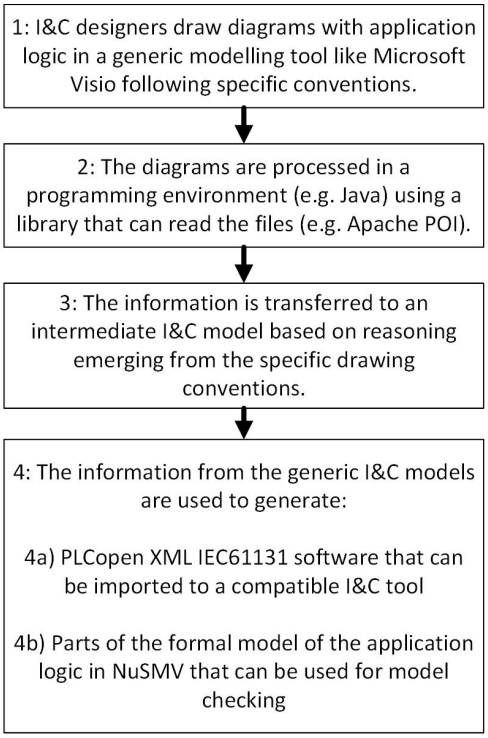

Fig 1. The overview of the methodology proposed in this paper for model checking of I&C diagrams in generic modelling tools. The four steps of the methodology can transform I&C logic in generic drawing tools to an intermediate model that can be used to generate IEC61131 software or partial formal verification models that can be model-checked.

(such as Linear Temporal Logic (LTL) or Computational Tree Logic (CTL) [8]).

Model checking has been successfully used in the nuclear industry. Since 2008, VTT Technical Research Centre of Finland (VTT) has used the method to verify I&C application logic designs in a new-build project (Olkiluoto 3), a project where old analogue systems were replaced with digital technology (Loviisa I&C renewal), and a project in early, construction licence application phase (Hanhikivi-1) [1]. By 2019, 59 confirmed design issues have been identified. Model checking has also been used in nuclear applications in Korea [9], Hungary [10] and at the European Organization for Nuclear Research (CERN) [11].

VTT has also developed a graphical tool called MODCHK [2] for verifying I&C application logics using the NuSMV [12] model checker. MODCHK is used to model function block diagrams, specify properties, generate the necessary input files for NuSMV, and then visualize the counterexamples with an animated model view. Still, modelling the function block diagram is a manual task, even if the original design sheets were available in electronic format.

In studies like [9], [10], [11] it is assumed that the application logic follows the IEC 61131-3 standard. However, nuclear I&C system suppliers such as Framatome (formerly Areva) and Rolls-Royce rely on vendor-specific, non-standard solutions. The source code for the basic blocks is usually unavailable and the vendors only provide a functional description [1]. Although qualified development tools are used for code generation (based on the specified block diagram), common applications like MS Visio are still used in basic design (by Framatome, for TELEPERM XS block diagrams, or by Atomproekt, for Hanhikivi-1 NPP functional diagrams).

B. Representing formal models

There are different model checking tools available. NuSMV, used in our work, is based on synchronous processing of finite states over discrete time. However, there are also tools that can handle continuous time [13], infinite-state transitions [14], or probabilistic automata [15], to name just a few variations.

Due to the diversity of techniques and tools, there is no "standard" language for describing the system model in formal verification (although it could be claimed that AIGER—a low-level language based on And-Inverter Graphs—is an attempt at such a language, being used in the Hardware Model Checking Competition [16] to benchmark different tools). The languages used for property specification also have different variations and extensions.

The input language of NuSMV is described in [17]. VTT's modular and hierarchical approach for modelling function block diagrams is motivated by nuclear industry specific features (particularly signal validity processing and subsystem redundancy), as well as interoperability and tool development needs.

C. Model checking based on IEC61131 and Visio models

As mentioned above, there are studies that deal with model checking of logics based on IEC 61131-3 (e.g., [9], [10], [11]), and also PLCopen. In [18], Safety Function Blocks specified in PLCopen are formalized as library components for the Uppaal [13] model checker, allowing the verification of a safety application. In [19], a NuSMV model is compiled based on a FBD[1] model expressed in PLCopen XML, but the complier only covers the parts of FBD language that are relevant for a relatively simple case study. In the above-mentioned approaches, the common nominator is the assumption that IEC 61131-3 is the generally accepted industry standard, which is not true for the nuclear industry.

Transformation of "active behaviour" specified in UML into an SMV specification was also demonstrated in [20], and XMI file to NuSMV conversion—where XMI is generated from behavioural system model in UML—was demonstrated in [21]. Our work focuses on a more detailed model of the design.

Some tools combining MS Visio and model checking have been proposed. Sequence Chart Studio [22] uses a Visio add-on for modelling of Message Sequence Charts (MSC). Certain types of MSCs can be exported to a parallel distributed model checker called DiVinE. Similarly, M2Code [23] is also a Visio add-on for modelling MSCs, and then verifying the behaviour using Uppaal. VTS [24] is a visual language for verifying real-time properties. The VTS tool uses Visio as a frontend for editing scenarios to produce timed automata for model checkers like Uppaal.

[1] In this paper, we only use the abbreviation FBD when referring to the programming language as specified in the IEC 61131-3 standard. The term "function block diagram" is used in a more general sense.

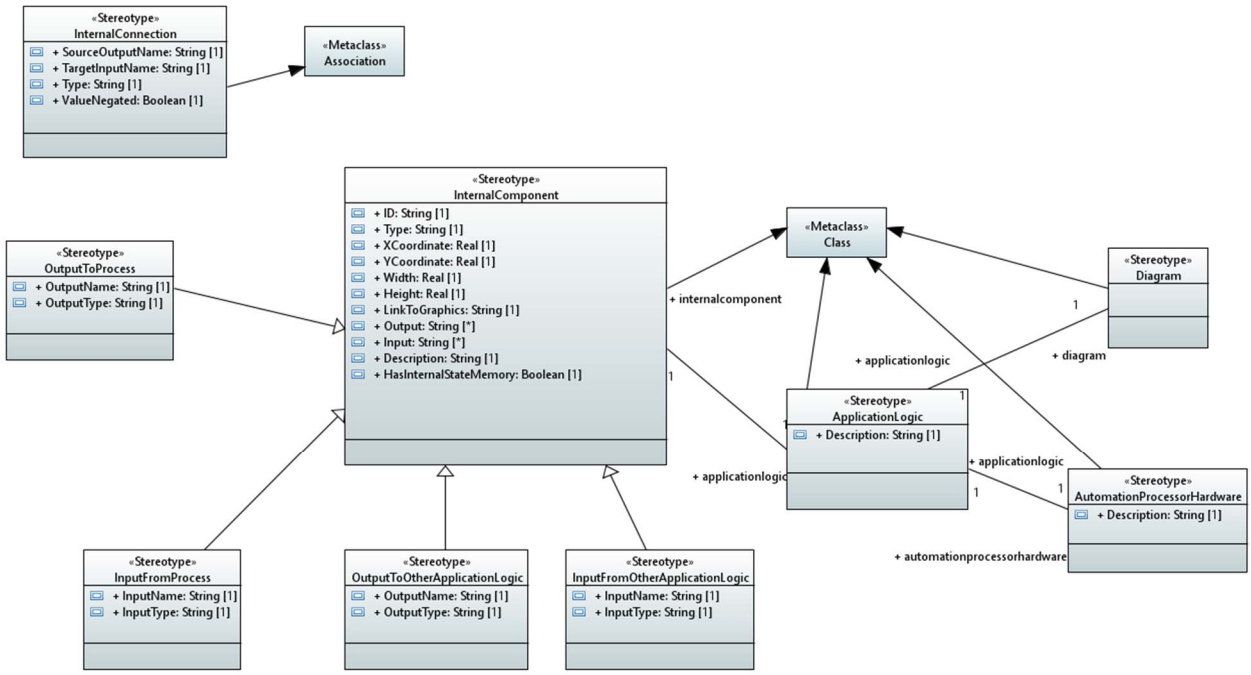

Fig 2. The UML profile (metamodel) for intermediate I&C models that can be used to generate PLCopen XML or NuSMV code.

III. Methodology

We have developed a methodology to assist the model checking of diagrams with application logic drawn in generic modelling tools. An overview of the workflow of this methodology can be found in Fig. 1.

The methodology starts with the assumption that I&C application logic diagrams are drawn using specific drawing and naming conventions in the generic modelling tool, since the tool does not include the needed semantics. The second step is to import the diagrams to a programming environment including a library that can import the files with the drawings. In step 3, an intermediate internal I&C logic model is populated with the generic drawing information, using reasoning rules specific to the conventions used to draw the original diagrams (e.g. in a tool like Visio). An overview of the UML profile describing the metamodel of the intermediate UML I&C model can be found in Fig. 2. A diagram can contain one or more application logics that can be allocated to I&C processor hardware. An application logic can contain generic internal components (like logic components), I/O from/to other application logics, I/O from/to the process and internal connections.

In step 4, the generic I&C logic information stored in the intermediate model can be used to generate IEC61131 PLCopen models (4a). The set of mappings from the intermediate UML I&C model to the IEC 61131 concepts are shown in Fig. 3. An IEC 61131 Program Organizational Unit (POU) is created for every application logic in UML. If the I&C tools library contains the blocks for internal components of the application logic, then they are used, otherwise new POUs will need to be created. The POUs can be implemented using any of the five IEC 61131 languages (Ladder Diagram (LD), Function Block Diagram (FBD), Sequential Function Chart (SFC), Instruction List (IL) or Structured Text (ST)). Practically, FBDs have better visualization, while it is easier to automatically generate ST language code. The resulting I&C project in PLCopen XML can be imported to a compatible I&C tool and if there are no "black box" POUs (internal components that lack implementation details), it can be used for testing the I&C software.

The information from the intermediate UML I&C logic model can be used to partially generate formal I&C models using the mappings presented in Fig. 4 (step 4b). The key information in the original diagram is how the elementary blocks have been connected. This can be represented in NuSMV by invoking instances of elementary MODULE types, and recreating the connections in the parameter list of each instance [2]. What cannot be automatically generated is the information also missing from the source diagram: (1) the processing logic for the elementary, vendor-specific blocks, and (2) the list of formal properties to check. The library of the elementary blocks needs [2] to first be built manually. The properties need to be specified by the analyst performing the verification. Still, this partially generated formal NuSMV model supports the checking of the I&C logic. If weaknesses are found, the I&C designer can re-engineer the logic to improve the functionality/safety of the I&C.

IV. Practical Example

Below, we demonstrate our approach using a real example of a design issue revealed using model checking in practical industry project. The originally verified application logic consisted of 24 function blocks. Here, we have modified the logic to mask the origin, and to include little more than minimum number of blocks needed for reproducing the scenario.

A. Case example: reactor protection system

As an example, we consider a four-redundant reactor protection system. Each of the four divisions has their own sensors, operator panels and actuators, and a processing unit (PU), and a voting unit (VU) (See Fig. 5.). The redundant structure means that the system is able to perform its functions even if any single component in it fails. The VU application

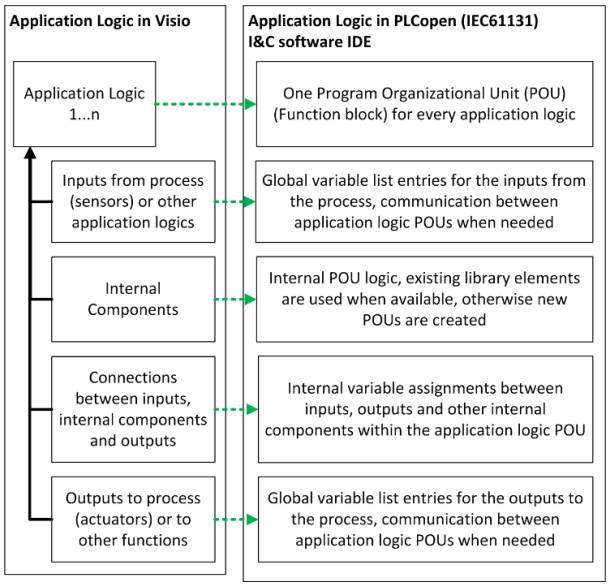

Fig 3. UML to IEC61131 PLCopen mappings. The high-level mappings between the I&C software concepts of the proposed UML metamodel and the concepts of the IEC 61131.

logic for one protection function, drawn in MS Visio (step 1 of the proposed methodology), is shown in Fig. 6.

The intended functionality is a reactor trip based on measurements crossing a safe limit. The four PUs each acquire a signal from a process sensor, perform a limit comparison, and then distribute the results to the four redundant VUs. Each VU then performs a majority vote over the results, and sends the actuation order if at least two inputs are set. When the measurements return to a safe state, and the signals from the PUs are reset, the VU resets the actuation order. The operator also has the option to acknowledge the trip, which will reset the actuation order even if the measurements alone would still justify the order.

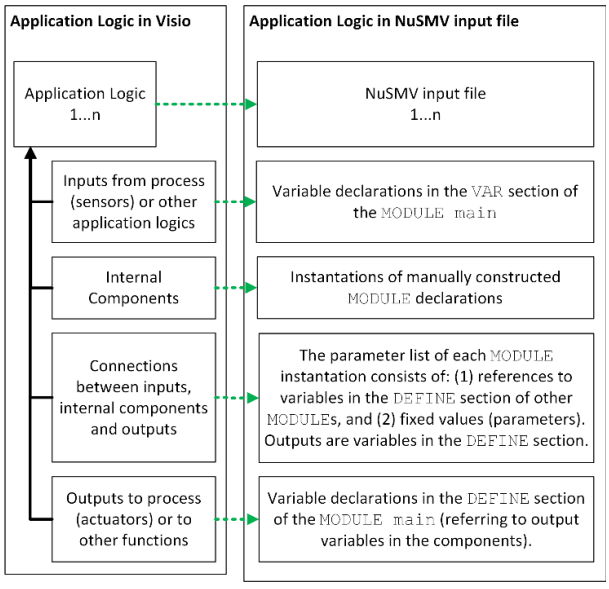

Fig 4. UML to NuSMV mappings. The high-level mappings between the I&C software concepts of the proposed UML metamodel and the concepts of the NuSMV input file.

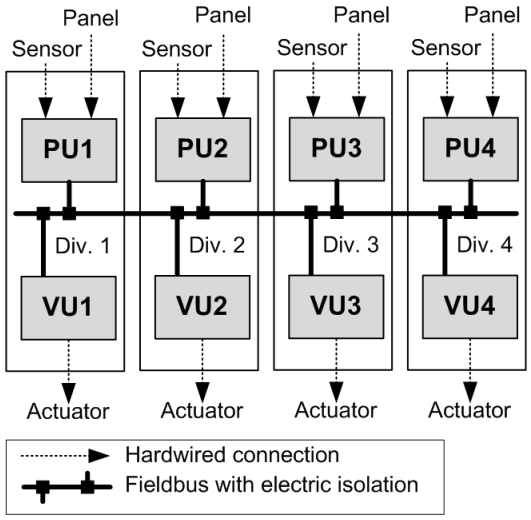

Fig 5. The hardware architecture of a four-redundant protection system.

The R-S1 element in Fig. 6 is a flip-flop latch that stores the manual acknowledgement to memory. The set input dominates over the reset input (if both are active), and the block is initialised to the set side, which will prove significant below. The circle at the R (reset) gate is a negation.

B. Representation in UML

The file with the MS Visio diagram was imported using the Java Apache POI [25] open source library that supports multiple MS Office applications (step 2 of the methodology). This library transfers that information to a source model with the basic generic drawing semantics (e.g. shapes, connections, etc). A set of reasoning rules are needed in step 3 of the methodology to use this information and create an intermediate I&C model that follows the proposed I&C metamodel (see Fig. 2). Based on the case study of this paper (See Fig. 6), a set of reasoning rules was compiled, and is presented in Tables I and II.

After accessing the underlying data of the Visio elements as they are imported by the Apache POI library, thorough analysis is required because a big portion of the data is fragmented and contains irrelevant fields. The challenge is to reason on the data acquired by Apache POI API, filter and classify the diagram elements to categorize and identify the information needed to populate the UML I&C model of the application logic. Most of the required data fields can be attained with API functions but in other cases processing the XML description of some elements is required.

TABLE I. REASONING RULES FOR CREATING I&C UML MODELS FROM VISIO DIAGRAM SOURCE DATA

Visio element	Visio diagram processing strategy
Shape	Retrieve individual Visio shape element and unique shapeID. Get attributes i.e. height, width and coordinates(PinX, PinY) etc. Get contents inside shapes i.e. text description, nested shape(and it's contents) Use available methods in "org. apache. poi. xdgf. usermodel.XDGFShape" with JAVA or write function to process the detailed XML description of elements.

	Example of methods: getShapes(), getPinX(), getPinY(), getHeight(), getWidth etc.
	Store data as class object in suitable data structure for further use
Connection	Retrieve individual connection with shapeID.
	Get attributes i.e. from which shape the connection begins and ends.
	Apply methods in ”org. apache. poi. xdgf. usermodel. XDGFConnection’’ or write function to process the detailed XML description of elements.
	Example of methods: getConnections(), getToShape() getFromShape() etc.
	Store data as class object in suitable data structure for further use.

TABLE II. RETRIEVAL PROCESS FOR I&C MODEL CLASS ATTRIBUTES FROM VISIO

Attributes of I&C model class	Retrieval process
ID: String[1]	Retrieved from XDGFShape with java library methods and assigned value to object
Type: String[1]	Identified with custom function by analysing text and internal data of shape type
XCoordinate: Real[1]	Retrieved from shape data with method “getPinX()”.
YCoordinate: Real[1]	Retrieved from shape data with method “getPinY()”.
Width: Real[1]	Retrieved from shape data with method “getWidth()”.
Height: Real[1]	Retrieved from shape data with method “getHeight()”
Output: String[*]	This represents next element that receives output from current element. Retrieved by analysis of shapeID connected with each “connection” element
Input: String[*]	This shows previous elements that shares a connection as input to current element. Retrieved by analysis of shapeID connected with each “connection” element
Description: String[*]	Identified with custom function by analyzing text inside shape or nested shapes
LinkToGraphics: String[1]	Assigned with custom function based of retrieved graphic data from shapes
HasInternalStateMemory: Boolean[1]	Assigned manually depending on type of element

Some internal components are easy to identify like the logical AND, which is a single rectangle of a particular shape type and contains the text “AND”. However, some other logical parts require more complex steps. The R-S1 flip-flop (latch) in Fig. 6 is such an example. The graphical Visio element for the latch internal component contains 3 adjacent rectangular shapes which are inside a parent rectangular shape of type “group” (in ideal cases, when the drawing notation is followed correctly). Two of the rectangles contain text which is used to identify this shape group as a “latch”. The position of the empty and the black rectangle within the latch shape group determines the dominance of the set/reset function. This is done by analysing the coordinates of the centre of the rectangle and the alignment position. Then to identify the connections to the inputs (which one connects to the

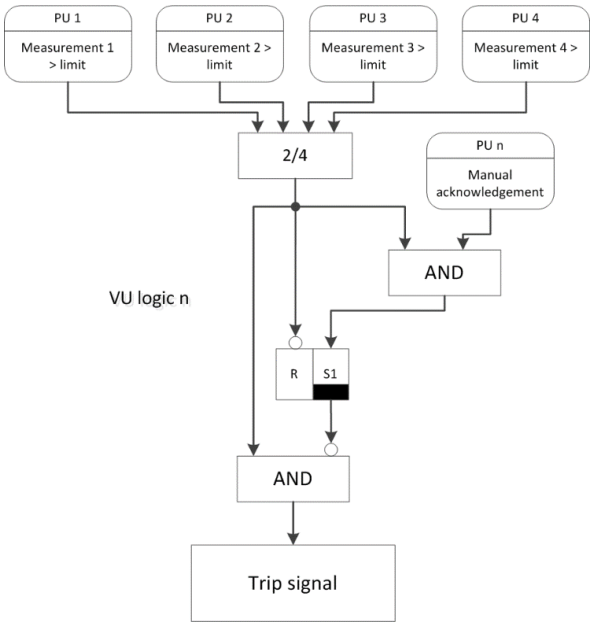

Fig 6. An exemplar application logic (MS Visio diagram) for a voting unit participating in a protection function.

InputReset and which to the InputSet of the latch), the location of the connection related to the “S” or “R” shape is used. The incoming connection from the logical “AND” gate belongs to the “S1” block of latch which has to be identified by finding the closest of the 3 nested shapes inside the parent shape. The second input originates from the voting logic “2/4” through an “inversion” and belongs to the “R” block of the latch. The Apache POI API only brings the shapeID of the inversion shape (represented with small circle). Again, analysis of the coordinates is needed to find the “R”b lock as the nearest shape of the inversion and assign the “InputNegated” property of that input as “true” while the latch object is stored as an InternalComponent class instance. In exceptional cases, these 3 rectangles (“S”, “R” and the black box) can look exactly same visually without being part of a parent shape, which hinders the identification of the latch.

The rules for identifying the internal components of the application logic of the example (like the ones mentioned above, see Tables I and II for a more complete but brief list) are used to populate the Java classes of the UML I&C model. The resulting I&C model for this case study is shown as a UML diagram in Fig. 7 demonstrating the application logic for the Trip signal generator.

C. Representation in PLCopen

In step 4 (4a) of the methodology, the UML I&C model of the diagram will be used as source for the generation of the PLCopen XML file containing the corresponding IEC61131 I&C software code, using the mappings presented in Fig. 3. A PLCopen XML file contains all the information needed to import the software project to an I&C development tool that is compatible with PLCopen XML like the Codesys [26].

Fig. 8 shows the program with the TripSignalGenerator Program Organization Unit (POU) ready to run on the VULogicN processor. It is important to note that in the process of I&C code generation from the intermediate UML model, it is assumed that the logic components have an existing behaviour implementation stored in the default library of the I&C environment (e.g. logic gate definition already exists in

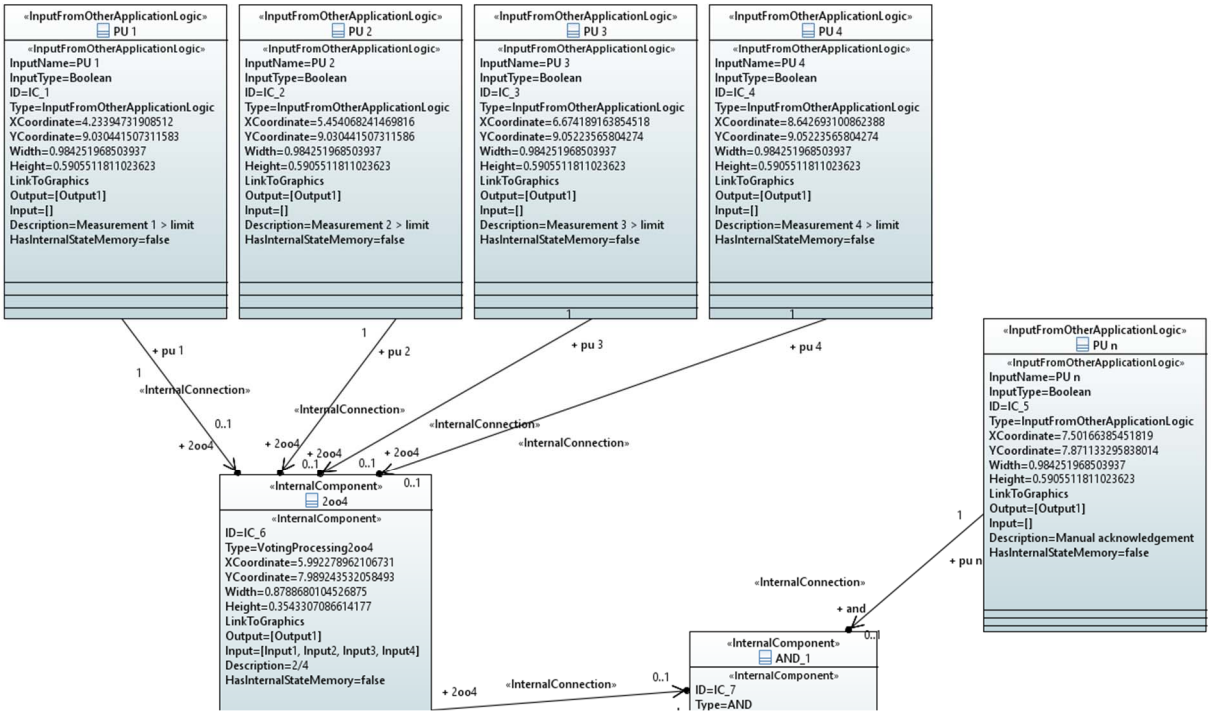

Fig 7. The main UML intermediate model for the Trip Signal Generator application logic (partial).

Codesys) or the behaviour is available in an additional library for more complex components.

D. Verification with NuSMV

Finally, in step 4 (4b), the formal model of the application logic is verified using a model checker.

Listing 1. The auto-generated NuSMV input file (based on manually constructed elementary blocks), with the necessary changes by the analyst highlighted

```
MODULE main()
    VAR

        --# Inputs:
        limit_1: boolean;
        limit_2: boolean;
        limit_3: boolean;
        limit_4: boolean;
        man_ack_1: boolean;

        --# Block instance network:
        vl_1 : _2oo4(limit_1, TRUE, limit_2, TRUE, limit_3, TRUE,
        limit_4, TRUE);
        sr_1 : SRs(and_1.OUT1, TRUE, !vl_1.OUT1, TRUE);
        and_1 : AND2(vl_1.OUT1, TRUE, man_ack_1, TRUE);
        and_2 : AND2(vl_1.OUT1, TRUE, !sr_1.OUT1, TRUE);

    DEFINE

        --# Outputs:
        trip_1 := and_2.OUT1;

        --# Properties:
        LTLSPEC G ((( count(limit_1, limit_2, limit_3, limit_4) >= 2)
        & H !man_ack_1) -> trip_1);

    ASSIGN

--# 2-out-of-four voting block type:
MODULE _2oo4(IN1, IN1_CONNECTED, IN2, IN2_CONNECTED,
IN3, IN3_CONNECTED, IN4, IN4_CONNECTED)
    DEFINE
        OUT1:= (count(IN1,IN2,IN3,IN4) > 1);

--# Logical AND block type:
MODULE AND2(IN1, IN1_CONNECTED, IN2, IN2_CONNECTED)
    DEFINE
        OUT1:=
            case
                !IN1_CONNECTED & !IN2_CONNECTED : FALSE;
                TRUE : ((IN1 | !IN1_CONNECTED) & (IN2 |
                !IN2_CONNECTED));
            esac;

--# Set-reset flip-flop (set priority)
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;
```

A NuSMV script is generated using templates for each elementary block type. For example, for a 2-gate AND block instance declaration, we use the template "AND2({0}, {0a}, {1}, {1b})", where {0} and {1} are placeholders for variables that represent either NuSMV model inputs or outputs from other block instances, and {0a} and {1b} are Boolean values used to pass the information to the block code that the input gate has a wire connected to it. (Here, for readability, we omit the signal validity [2] processing logic. In MODCHK, the signal status is passed as a third Boolean variable added to each input signal.)

The NuSMV input file is created by traversing the intermediate I&C model from the inputs to the outputs, and using the block templates to generate the block instances and connect the signals passing between blocks. The manually specified code for the basic block types is added as a list of MODULE definitions at the end of the file. The NuSMV input

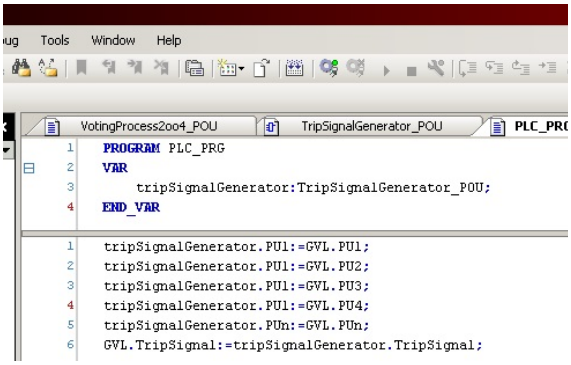

Fig 8. The program with the TripSignalGenerator POU ready to run on the VULogicN processor.

file generated for our example model is shown in Listing 1. They key part is the list of MODULE instances that recreates the block network. Before running NuSMV, the analyst needs to make changes to the file (highlighted in Listing 1), including:

1. Typing of the input variables: binary inputs are typed as Boolean, and analog inputs as integer ranges or enumerations. Even if the input type could be determined based on the block types on the diagram, the analyst would still need to pay attention to the integer value assignment, to avoid state space explosion [2].
2. The formal properties to be checked: for NuSMV, specified using LTL, CTL, or Property Specification Language (PSL).

The formal property in Listing 1 is based on the functional requirement: "If two out of four of the measurements are over the limit, and the operator has not acknowledged the trip, the trip signal shall be set". Using LTL with past operators [27], the property can be formalised as:

G (((count(limit1, limit2, limit3, limit4) ≥ 2) ∧ **H** ¬man_ack) → trip)

For the now completed input file, NuSMV produces a counterexample, where, at the initial state (the I&C system's start-up), the measurements are already over the limit. As a result, the initial state of the R-S1 flip-flop remains set, inhibiting any actuation orders for as long as the actuation criteria are active (In Fig. 9, we show how the counterexample would look if visualised in MODCHK). Obviously, the designer did not consider scenarios where the I&C system is initialised during an accident at the plant.

V. Discussion

Interoperability of different tools would complement formal verification. Model checkers are built to produce counterexample scenarios, and although it is possible to specify a property that will "force" the model checker to "simulate" trough a given sequence of inputs, doing so is very cumbersome. In order to better understand system model behaviour, the analyst performing model checking would benefit from being able to concurrently run the same logic on a dedicated I&C software simulator, instead.

Further, if model checking is performed in open loop (without considering the feedback from the controlled process), the analyst will end up with counterexamples that are

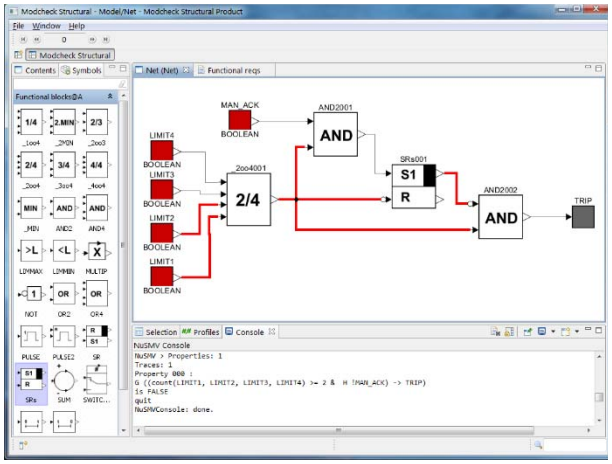

Fig. 9. The initial state of the counterexample animated in MODCHK.

not relevant in the actual context of the I&C system [1]. Simulators could be used to validate the counterexamples in the actual, closed-loop context with the plant included [28]. Of course, a process simulator cannot be forced to run through a scenario that is physically impossible, so using the counterexample as it is might not be feasible. In addition, issues related to, e.g., timing can require specific care to reproduce [28]. For open-loop simulation, the validation will be easier, but the use case might more often be the analyst focusing on some specific set of values (like the acknowledgment signal being set at start-up in our example above), rather than trying to reproduce the original counterexample in full.

MS Visio does not perform the kind of checks that a proper I&C engineering tool would. The diagrams can therefore contain (1) connections that only look connected (but actually float), (2) connection wires drawn in the "wrong" direction (with arrow graphics reversed), or (3) groups of components that look like a single component. What visually appears connected, is then not necessarily connected in the Visio XML file. For transformation, some "broken" connections can be deduced based on graphical "proximity" on the diagram, but it is difficult to make sure that everything is properly captured. Still, for a large and complex block diagram, the analyst performing model checking would certainly benefit from having at least part of the model automatically generated. And, if the analyst does not immediately notice the missing blocks or communication wires, such errors will almost certainly be revealed through counterexamples.

VI. Conclusions

Model checking is a proven method for discovering issues in I&C logic and it is especially important tool for a safety critical domain such as the nuclear energy. There are cases where the I&C logic is drawn is generic tools whose output may be machine readable, but the notation is not standardized. This paper presented a methodology for model checking I&C models that are drawn in generating modelling environments that lack the proper I&C semantics.

The case study of the paper, a Trip signal generator application logic component was successfully transferred from the initial Vision source diagram file to an intermediate UML I&C model and then to IEC 61131 PLCopen XML and NuSMV formats. The use case used the NuSMV version of

the example and demonstrated the discovery of an issue for generating trip signals.

In related work, it is often assumed that the application logic follows the IEC 61131-3 standard. In the nuclear industry, however, safety I&C suppliers use vendor-specific, non-standard function block diagrams. Our translation maintains the vendor-specific features. Although we apply 61131 to promote further interoperability with a range of tools, in formal verification, it is important to make sure that the details—even if they are based on non-standard solutions—are modelled as exactly as possible.

ACKNOWLEDGMENT

This work was funded by the Finnish Research Programme on Nuclear Power Plant Safety 2018-2022 (SAFIR 2022).

REFERENCES

[1] A. Pakonen, T. Tahvonen, M. Hartikainen, and M. Pihlanko, "Practical applications of model checking in the Finnish nuclear industry," in 10th International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies (NPIC & HMIT 2017), San Francisco, California, US, 2017, pp. 1342–1352.

[2] A. Pakonen and K. Björkman, "Model checking as a protective method against spurious actuation of industrial control systems," in 27th European Safety and Reliability Conference (ESREL), Portoroz, Slovenia, 2017, pp. 3189–3196.

[3] A. L. Ramos, J. V. Ferreira, and J. Barceló, "Model-Based Systems Engineering: An Emerging Approach for Modern Systems," in IEEE Transactions on Systems, Man, and Cybernetics, vol. 42, pp. 101–111, 2012. https://doi.org/10.1109/TSMCC.2011.2106495

[4] IEC 61131-3:2013 Programmable controllers - Part 3: Programming languages. URL(https://webstore.iec.ch/publication/4552); International Electrotechnical Commission; 2013 [accessed 16 Jan 2020]

[5] IEC 61131-10:2019 Programmable controllers - Part 10: PLC open XML exchange format. URL(https://webstore.iec.ch/publication/33034); International Electrotechnical Commission; 2013 [accessed 16 Jan 2020]

[6] OMG Unified Modeling Language (OMG UML) specification. URL(http://www.omg.org/spec/UML/); Object Management Group; 2015 [accessed 16 Jan 2020]

[7] I. Buzhinsky and A. Pakonen, "Model-Checking Detailed Fault-Tolerant Nuclear Power Plant Safety Functions," in IEEE Access, Vol 7., pp. 162139–162156,2019. https://doi.org/10.1109/ACCESS.2019.2951938

[8] E. M. Clarke, O. Grumberg, and D. Peled, Model checking, Cambridge, Massachusetts, US: MIT press; 2001. ISBN 978-0-262-03270-4.

[9] J. Yoo, S. Cha, and E. Jee, "Verification of PLC programs written in FBD with VIS," in Nuclear Engineering and Technology, vol. 41, pp. 79–90, 2009.

[10] E. Némethand and T. Bartha, "Formal Verification of Safety Functions by Reinterpretation of Functional Block Based Specifications," in FMICS 2008; Lecture notes in computer science, vol. 5596, F Cofer and A Fantechi, Eds. Springer Berlin, Heidelberg: Springer, 2009, pp. 199–214. https://doi.org/10.1007/978-3-642-03240-0_17.

[11] B. F. Adiego, D. Darvas, E. B. Vinuela, J. C. Tournier, S., Bliudze, J. O. Blech, et al., "Applying model checking to industrial sized PLC programs," in IEEE Transactions on Industrial Informatics, vol. 11, pp. 1400–1410, 2015. https://doi.org/10.1109/TII.2015.2489184

[12] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, et al., "NuSMV Version 2: An Open Source Tool for Symbolic Model Checking," in CAV 2002; Lecture Notes in Computer Science, vol. 2404, E. Brinksma and K. G. Larsen, Eds. Berlin, Heidelberg: Springer: 2002, pp. 359–364. https://doi.org/10.1007/3-540-45657-0_29.

[13] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on UPPAAL," in SFM-RT 2004; Lecture notes in computer science, vol. 3185, M. Bernando and F. Corradini, Eds. Berlin, Heidelberg: Springer: 2004, pp. 200–236. https://doi.org/10.1007/978-3-540-30080-9_7.

[14] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta, "HyComp: An SMT-Based Model Checker for Hybrid Systems," in TACAS 2015; Lecture notes in computer science, vol. 9035, B. Christel and C. Tinelli, Eds. Springer Berlin, Heidelberg: Springer: 2015, pp. 52–67. https://doi.org/10.1007/978-3-662-46681-0_4.

[15] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of Probabilistic Real-time Systems," in CAV 2011; Lecture notes in computer science, vol. 6806, G. Golapahrishan and S. Qadeer, Eds. Berlin, Heidelberg: Springer: 2011. pp. 585–591. https://doi.org/10.1007/978-3-642-22110-1_47

[16] T. Kuismin and K. Heljanko, "Increasing Confidence in Liveness Model Checking Results with Proofs," in 9th International Haifa Verification Conference (HVC 2013), Haifa, Israel, 2013.

[17] R. Cavada, A. Cimatti, C. A., Jocim, G. Keighren, E. Olivetti, M. Pistore, et al., NuSMV 2.6 User Manual. URL(http://nusmv.fbk.eu/NuSMV/userman/v26/nusmv.pdf); 2010 [accessed 27 Nov 2019]

[18] D. Soliman and G. Frey, "Verification and Validation of Safety Applications based on PLCopen Safety Function Blocks using Timed Automata in Uppaal," in IFAC Proceedings, vol 4., pp. 34–39, 2009. https://doi.org/10.3182/20090610-3-IT-4004.00011

[19] J. Li, A. Qeriqi, M. Steffen, and I. C. Yu, "Automatic translation from FBD-PLC-programs to NuSMV for model checking safety-critical control systems," in Norsk Informatikkonferanse (NIK 2016), Berger, Norway, 2016.

[20] M. E. Beato, M. Barrio-Solórzano, C. E. Cuesta, and P. de la Fuente, "UML automatic verification tool with formal methods," in Electronic Notes in Theoretical Computer Science, vol. 127, pp. 3–16, 2005. https://doi.org/10.1016/j.entcs.2004.10.024

[21] C. Rongshang, K. Jinyu, X. Ming, and X. Lei, "An Approach Converting XMI to SMV," in International Journal of Security and Its Applications, vol. 9, pp. 323–330, 2015. https://doi.org/10.14257/ijsia.2015.9.12.30

[22] M. Bezdeka, O. Bouda, L. Korenciak, M. Madzin, and V. Reh'k, "Sequence Chart Studio," in 12th International Conference on Application of Concurrency to System Design, Hamburg, Germany, 2012. https://doi.org/10.1109/ACSD.2012.25

[23] M. R. Cesar, Study of the Applicability of Model-Driven Metodologies for the Design of Autonomic Behaviours. Master's thesis, Aalto University, 2011. http://urn.fi/URN:NBN:fi:aalto-201207022674

[24] A. Alfonso, V. Braberman, N. Kicillof, and A. Olivero, "Visual timed event scenarios," in International Conference on Software Engineering, Edinburgh, UK, 2004. https://doi.org/10.1109/ICSE.2004.1317439

[25] Apache POI - the Java API for Microsoft Documents. URL(https://poi.apache.org/); The Apache Software Foundation; 2019 [accessed 16 Jan 2020]

[26] CODESYS Development System. URL(https://www.codesys.com/); Codesys; 2020 [accessed 16 Jan 2020]

[27] M. Pradella, P. San Pietro, P. Spoletini, and A. Morzenti, "Practical model checking of LTL with past," in International Workshop on Automated Technology for Verification and Analysis (ATVA03), 2003.

[28] S. Patil, V. Vyatkin, and C. Pang, "Counterexample-guided simulation framework for formal verification of flexible automation systems," in 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), Cambridge, UK, 2015, pp. 1192–1197. https://doi.org/10.1109/INDIN.2015.7281905