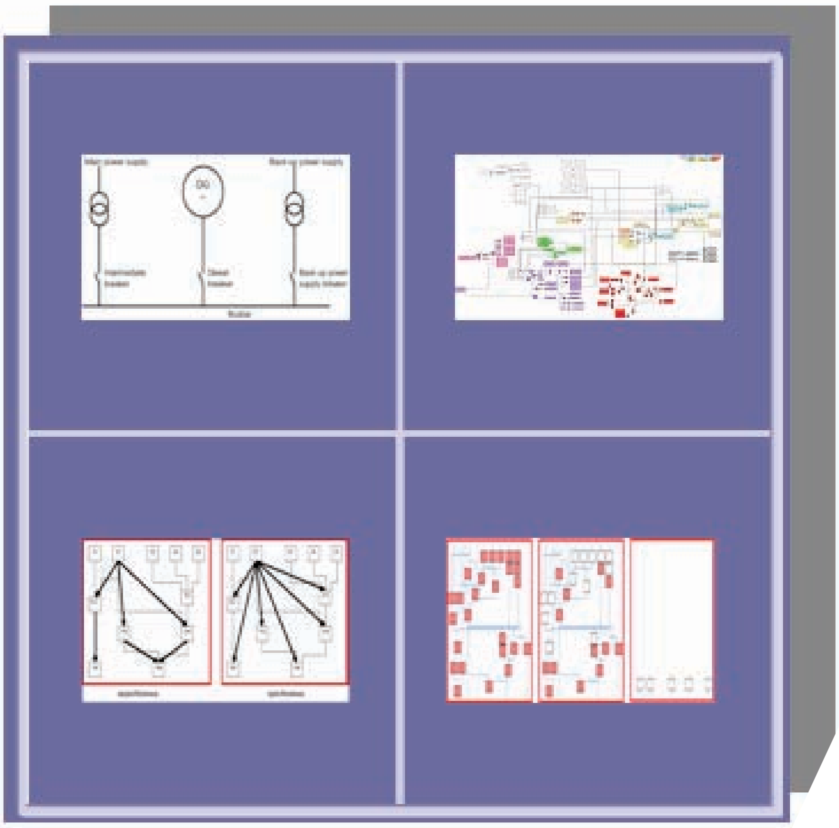
VTT WORKING PAPERS 156



Jussi Lahtinen, Kim Björkman, Janne Valkonen, Juho Frits & Ilkka Niemelä

Analysis of an emergency diesel generator control system by compositional model checking

MODSAFE 2010 work report

ISBN 978-951-38-7497-1 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 1000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 4374

VTT, Bergsmansvägen 5, PB 1000, 02044 VTT
tel. växel 020 722 111, fax 020 722 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O. Box 1000, FI-02044 VTT, Finland
phone internat. +358 20 722 111, fax +358 20 722 4374

Series title, number and report code of publication

VTT Working Papers 156
VTT-WORK-156

Author(s) Jussi Lahtinen, Kim Björkman, Janne Valkonen, Juho Frits & Ilkka Niemelä		
Title **Analysis of an emergency diesel generator control system by compositional model checking** **MODSAFE 2010 work report**		

Abstract

Digital instrumentation and control (I&C) systems containing programmable logic controllers are challenging to verify. They enable complicated control functions and the state spaces (number of distinct values of inputs, outputs and internal memory) of the designs easily become too large for comprehensive manual inspection.

Model checking is a formal method that can be used for verifying that systems have been correctly designed. A number of efficient model checking systems are available which provide analysis tools that are able to determine automatically whether a given state machine model satisfies the desired safety properties.

The practical case analysed in this research project is called an ”emergency diesel generator control system” and its purpose is to provide reserve power to critical devices and computers that must be available without interruption. This report describes 1) the development of a compositional approach for checking the models in large system designs, 2) the development of a modular model checking approach for modelling function block diagrams with the Uppaal model checker and 3) the experience of utilising the new modelling approaches in practice.

ISBN 978-951-38-7497-1 (URL: http://www.vtt.fi/publications/index.jsp)		
Series title and ISSN VTT Working Papers 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)		Project number 41251
Date December 2010	Language English	Pages 35 p.
Name of project Model-based safety evaluation of automation systems	Commissioned by	
Keywords Model checking, automation system, I&C, NuSMV, Uppaal, system safety, emergency diesel, MODSAFE, SAFIR	Publisher VTT Technical Research Centre of Finland P.O. Box 1000, FI-02044 VTT, Finland Phone internat. +358 20 722 4520 Fax +358 20 722 4374	

Preface

This report has been prepared as part of the research project Model-based Safety Evaluation of Automation Systems (MODSAFE), which is part of the Finnish Research Programme on Nuclear Power Plant Safety 2007–2010 (SAFIR2010). The goals of the project were to develop methods for model-based safety evaluation, apply the methods in realistic case studies, evaluate the suitability of formal model checking methods for Nuclear Power Plant (NPP) automation analysis and develop recommendations for the practical application of the methods. This report describes the development of a compositional model checking approach for analysing large system designs and summarises a case study where the approach was utilised.

We wish to express our gratitude to the representatives of the organisations who provided us with the case examples and all those who have given their valuable input in the meetings and discussions during the project.

Espoo, December 2010

Authors

Contents

1. Introduction

Verification of digital instrumentation and control (I&C) systems is challenging because programmable logic controllers enable complicated control functions and the state spaces (number of distinct values of inputs, outputs and internal memory) of the designs easily become too large for comprehensive manual inspection. Design verification is a key task in the design flow, because it can eliminate tricky design errors which are hard to detect later in the development process and are very expensive to repair, often leading to a major redesign and reimplementation cycle. Typically, verification and validation (V&V) activities rely heavily on subjective evaluation, which covers only a limited part of the possible behaviours of the system, and therefore more rigorous formal methods are required. Such formal methods have been studied (see [28] for an overview, for example) but they are not yet widely used.

Model checking [14] is a formal method that can be used for verifying the correctness of system designs. Before the Model-based Safety Evaluation of Automation Systems (MODSAFE) project, it was not previously applied in the safety evaluation of nuclear power plant (NPP) automation systems (at least in Finland), but internationally it has been used in verifying the correct behaviour of e.g. hardware and microprocessor designs, data communications protocols and operating system device drivers. A number of efficient model checking systems are available which provide analysis tools that are able to automatically determine whether a given state machine model satisfies given specifications. Model checking can also handle delays and other time-related operations, which are crucial in safety I&C systems and challenging to design and verify.

The objective of the MODSAFE project was to evaluate and develop methods based on formal model checking and apply them to the safety analysis of NPP safety automation (I&C). The purpose was to compile a group of methods and tools that can support the practical safety evaluation work and benefit utilities, regulators and vendors. The main tasks of the first two project years were to review the state of the art of employing formal methods and models for safety evaluation of industrial and nuclear safety systems [28], to develop basic methodology for applying model checking to safety evaluation and to study the feasibility of the approach [17, 29, 30, 31]. The objective of the third and fourth project year was to develop an approach that was more flexible and suitable for analysing larger models [6, 7, 32, 18, 8].

This report summarises the experiences gained during project year 2010 while developing a compositional approach for model checking in large system designs, developing a modular model checking approach for modelling function block diagrams with Uppaal and utilising the approaches for analysing a case study concerning the control system of an emergency diesel generator

The rest of the report is structured as follows: Section 2 provides background information on model checking methodology. Section 3 describes the emergency diesel generator control system and a selection of its main requirements. Section 4 introduces some compositional system verification approaches and explains the compositional technique developed in the MODSAFE project. The emergency diesel control system was analysed using the NuSMV and Uppaal model checkers. Section 5 describes how the NuSMV and Uppaal models were constructed and their special features. Finally, Section 6 sums up the results and findings concerning the emergency diesel generator case, and Section 7 concludes the report.

2. Model checking

Model checking [14, 12, 25] is a computer-aided verification method developed to formally verify the correct functioning of a system design model by examining all of its possible behaviours. The models used in model checking are quite similar to those used in simulation, as basically the model must describe the behaviour of the system design for all sequences of inputs. However, unlike simulation, model checkers examine the behaviour of the system design with all input sequences and compare it with the system specification. In model checking, at least in principle, the analysis can be fully automated with computer-aided tools. The specification is expressed in a suitable language, temporal logics being a prime example, describing the permitted behaviours of a system. Given a model and a specification as inputs, a model checking algorithm determines whether the system has violated its specification. If none of the behaviours of the system violate the given specification, the (model of the) system is correct. Otherwise, the model checker will automatically give a counter-example execution of the system demonstrating how the specification has been violated.

The MODSAFE project has been using two model checkers: NuSMV [11, 22], which was originally designed for hardware model checking, and UPPAAL [27], which supports model checking of timed automata. NuSMV is a state-of-the-art symbolic model checker that supports synchronous state machine models where the real-time behaviour must be modelled with discrete time steps using explicit counter variables that are incremented at a common clock frequency. NuSMV supports model checking using both Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) [14], making it quite flexible in expressing design specifications. The model checking algorithms employed in this work are based on symbolically representing and exploring the state space of the system by using Binary Decision Diagrams (BDDs) [6, 9, 20]. In addition, SAT-based (Propositional Satisfiability) bounded model checking [4] is also supported by NuSMV [5] for finding bugs in larger designs. The sophisticated model checking techniques used by NuSMV can handle non-determinism induced by free input variables well, but modelling real-time aspects can be more challenging due to the inherently discrete time nature of the synchronous state machine model employed by NuSMV.

UPPAAL is a model checking tool for timed systems based on modelling the system as a network of timed automata that communicate through message channels and shared variables. The timed automata have a finite control structure and real-valued clocks [1], making the modelling of timers fairly straightforward. Networks of timed automata can express the real-time behaviour of the system in continuous time and still be automatically analysed. This is feasible because all the possible behaviours of the system can be captured using a finite graph where different clock valuations with the same behaviour, intuitively, are grouped into a finite number of equivalence classes called regions [1]. The model checking algorithms use symbolic methods to compactly represent the clock valuations associated with each state of the system quite efficiently in terms of memory. The model checking algorithms employed inside UPPAAL [2, 19] are able to check a subset of the temporal logic TCTL (Timed Computation Tree Logic) [2] by explicit state model checking that explicitly traverses the finite graph induced by the behaviour of the system. The main strength of UPPAAL is in analysing the complex timing behaviour of a system. However, it is not well suited to systems with a very high amount of non-determinism as induced by, e.g., reading a large number of input variables (sensor readings) provided by the environment because each combination of inputs is explicitly explored by the employed model checking algorithms.

3. Description of the EDG system

3.1 Emergency diesel generator control

In this case study the behaviour of an emergency diesel generator system was examined. The case study is based on high-level design documentation that does not take into account the redundant implementations of the system and related voting logic that will be realised in the final system. This work focuses on the control logic of the diesel generator system that is represented as function block diagrams.

The purpose of the emergency diesel system is to provide reserve power, as it is essential that electricity is always available to the maintained system. In case of a black out or a power failure in the main power supply, the diesel generators can be quickly turned on to keep the necessary devices available.

The inputs of the control logic include voltage and frequency measurements, operator commands, check-back signals and measurements of the conditions of the diesel generator. The outputs of the logic are control signals for the diesel generator, the breakers, cooling systems and load protection signals for several pumps and other devices, for which power can be supplied by the diesel generator.

In addition to the function block diagrams, parts of the system environment, i.e. the expected diesel functionality, the busbar and the related breakers are also included in the model.

Figure 1 illustrates the high level architecture of the electrical connections. Besides the main power supply and the diesel generator, there is another additional diverse back-up power supply.

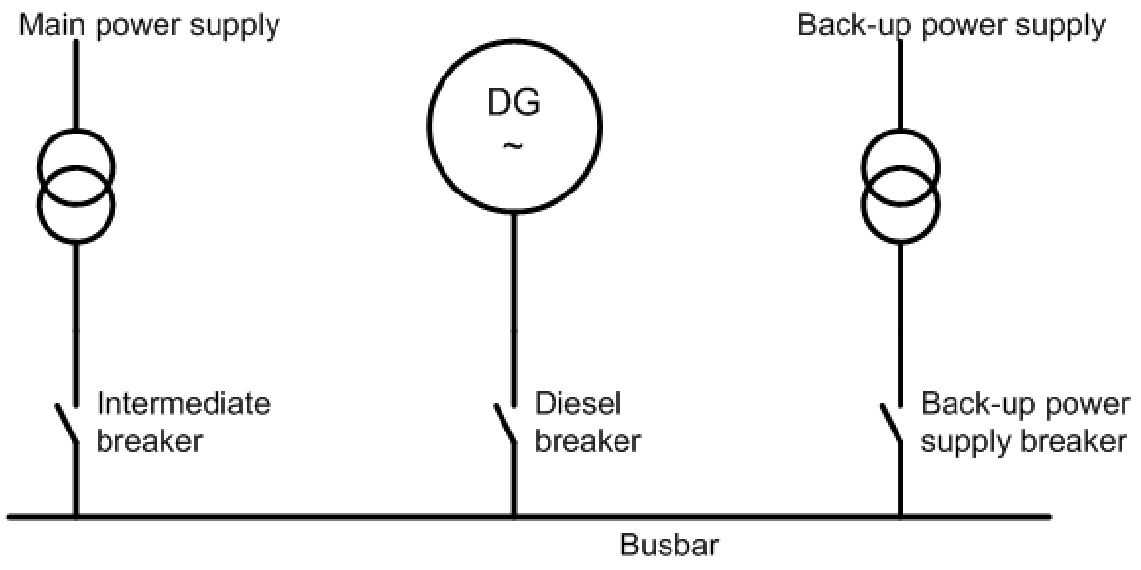


Figure 1. High level architecture of the power supply.

There is a large amount of logic related to the diesel generator control and part of it is shown in Figure 2. The system analysed here covers 10 different control functions, including functions related to the activation of the diesel generators, operation of the diesel generators, protection of the diesel generators, and voltage and frequency regulation. In addition, some functions are diverse implementations of other functions.

The functions of the diesel generator system are intentionally decentralised and, thus, the diverse functions especially are typically not implemented on the same computer equipment. As a result, it is presumed that two functions communicating with each other via a bus will not be in full synchronisation. The signals sent from one computer are received only after a few clock cycles by the other computer. Additionally, it is possible that the computers' clock frequencies differ marginally so that the same time delay in the logic results in different actual time lapse.

Because of the multiple diverse systems, priority logic is also involved in the system. Typically, all breakers, pumps and valves are controlled by a separate module that prioritises the control signals related to that device. In this case study, only the priority logic related to the three breakers (see Figure 1) is considered.

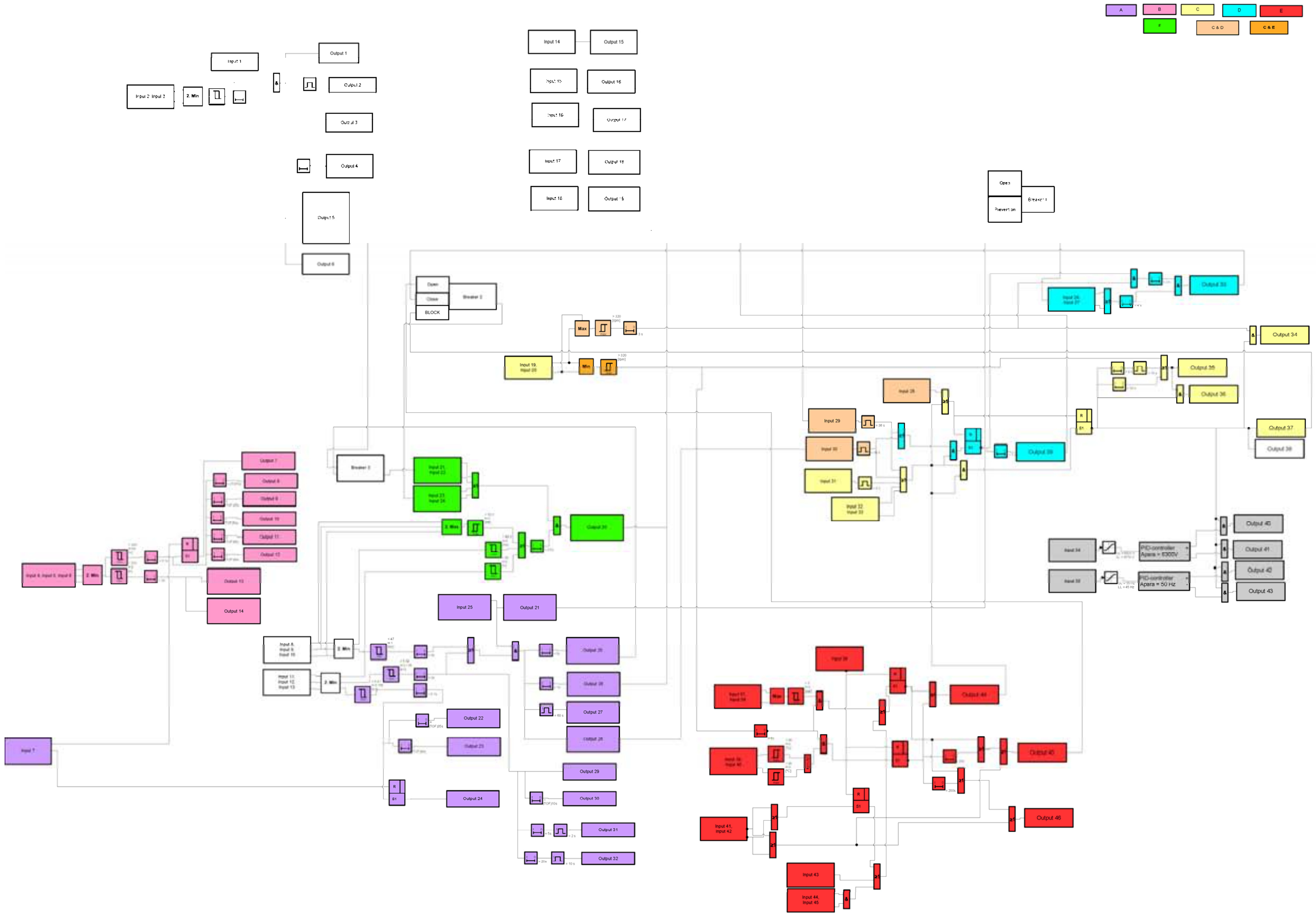


Figure 2. Part of the logic of the emergency diesel system.

3.2 About the desired system properties

The emergency diesel system provides reserve power for critical power plant devices in case of power failures. There are a few constraints that need to be taken into account when the diesels are used. Some of the general system requirements are:

- If there is a reason to start the diesels, they will be started, and they will eventually feed power to the busbar.
- When the diesels are started, a specified starting sequence is followed.
- Loads are connected to the diesels according to a specified loading sequence.
- The diesels take a few seconds to reach their operating speed. No loads should be connected to the generators during this time.
- The connections to the busbar are controlled by several breakers as illustrated in Figure 1. The breakers should be operated in a safe fashion.
- The control of the diesels is realised by several diverse systems. The prioritisation of the different systems' signals must be correct.
- There should be no race conditions.

Based on some general system requirements, a list of 17 detailed requirements was created. These requirements were formalised and used as input when the model of the system was verified. The detailed requirements and temporal specifications are not covered in this report.

4. Compositional verification methods for large systems

Many systems are so large that conventional model checking methods cannot be applied because of the state space explosion. In such systems, it is often possible to break down the specification into several properties describing the behaviour of individual modules of that system. Checking these local properties is usually more feasible, and if the conjunction of the local properties implies the original specification, it is possible to deduce that the entire system satisfies the specification as well. These compositional verification techniques [3, 24, 13] require that the system is composed of interconnected modules.

Several compositional verification approaches exist. These techniques include compositional minimisation, assume-guarantee reasoning (including circular reasoning techniques), partitioned transition relations and lazy parallel composition.

In the compositional minimisation technique [13] the system is abstracted using reduced versions of some of the system's modules. The reduced modules, or "interface modules", are abstracted away from their inner functionality, so that only signals visible to other modules are implemented. Interface modules can reduce the state space of the model significantly.

Another compositional verification technique is the assume-guarantee reasoning technique [26, 23]. In this technique, an assumption is made of the environment of a module. The assumption can be verified separately. When the assumption is known to hold, we can check if a specification is true in an individual module under this assumption. If the specification is true in the individual module, it is also true in the whole system.

Traditional assume-guarantee reasoning requires non-trivial human input. However, it is also possible to avoid this by using a learning algorithm to create assumptions of the model automatically. The learning algorithm creates assumptions of the environment model, and iteratively improves these assumptions based on the model checker's output (see e.g. [16]).

Some verification techniques are based on circular reasoning [21]. In this approach, each module is verified to behave correctly if its environment behaves correctly. For instance, correctness of module A is assumed when module B is verified and vice versa. The circularity of the reasoning can be resolved using induction over time.

The state space explosion in large models can also be diminished by special-purpose model checking algorithms. Traditional model checking algorithms calculate a parallel composition of the components and create a global model that depicts the whole system. The number of states in the global model is exponential to the size of the component models. Techniques such as partitioned transition relations [10] or lazy parallel composition [3] examine the transition relations of different components in the model separately, which can reduce the state space.

The compositional model checking approach we have chosen supports the existing modelling methods of function block diagram-based designs. The idea behind the method is that usually not every part of the logic is needed to verify properties in a system. The approach is based on compositional minimisation on two abstraction levels and program slicing, but does not prevent additionally using assume-guarantee reasoning. The reasoning used in the approach requires that only properties stating that “no undesired behaviour occurs” (safety properties) are examined.

Our existing modelling technique already partitions the design in suitable modules defined by the function block diagrams. These modules are further divided into a set of function blocks.

In this approach, function block diagrams are modelled as follows. Each module (function block diagram) has a set of inputs and a set of outputs. A module’s output values are calculated by a set of function blocks that are instantiated from a function block library of these components.

Compositional minimisation can be applied on these modules in several different abstraction levels; see Figure 3. The modules can be reduced into completely over-approximated interface modules or semi-interface modules containing parts of the original logic of the modules. Either way, compositional verification requires a systematic way of reducing the behaviour of the modules into interface modules.

In our models, the full-interface modules are easy to construct. A full-interface module contains no function blocks, and the outputs of the module are defined as free variables. The inputs of a full-interface module are left intact because of technical issues allowing compatibility with the model, but the inputs have no influence on the outputs. Definition of the output variables as free variables is a complete over-approximation of the module, i.e. no restrictions on the behaviour of the module are set. Full-interface versions of modules can be constructed in parallel with the normal model construction.

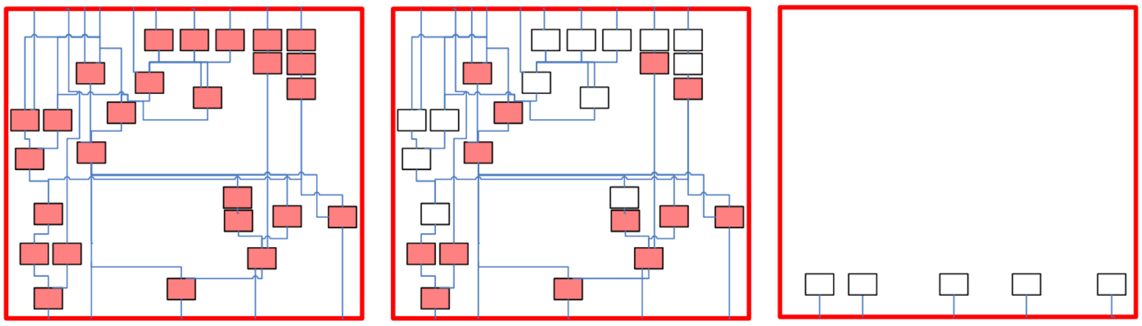


Figure 3. Abstraction of function block diagrams on different levels. A red box indicates that the function block is a non-abstracted version. A white box indicates that an interface function block is used instead. The image on the left represents a non-abstracted module (function block diagram). The image on the right is a full-interface module without inner functionality. The middle image represents a semi-interface module, in which some function blocks are left intact, and some are over-approximated by interface function blocks.

Semi-interface modules are partially over-approximated modules, in which some function blocks are non-abstracted, and some are approximated by “interface function blocks”. Interface function blocks are dummy function blocks, the outputs of which are defined as free variables. Some interface function blocks cannot be completely over-approximated in this way, because the ranges of the output signals of these blocks might not be known. These function blocks are abstracted on a case-by-case basis. For easy utilisation, an interface function block library can be created.

The semi-interface module is created as follows. The parameters of the abstraction are a set of module outputs and the depth *n* of the abstraction. According to these values, a program slicing method is used to choose the function blocks that can be reached by *n* steps from the defined outputs and may have influence on the given outputs. This set of function blocks remains non-abstracted. All other function blocks are transformed into interface function blocks. Semi-interface modules can be created on-the-fly using a small computer script when needed.

The combination of modularity and the possibility to use interface modules enables effortless selection of the model configuration, i.e. which modules are non-abstracted and which modules are interface/semi-interface modules. In our previous modelling techniques, abstracting away from some functionality of a large system requires manual effort. This work is reduced to selecting an abstraction level for each module. The model can then be generated using a simple computer script.

Different model configurations are shown in Figure 4, Figure 5 and Figure 6. The model in these figures consists of five modules. Each of these modules has a set of function blocks. The function blocks are depicted as boxes, where red boxes stand for non-abstracted function blocks and white boxes stand for interface function blocks. Figure 4 illustrates a model configuration in which the modules are non-abstracted. Figure 5 illustrates a model configuration in which one module is non-abstracted, and four modules are full-interface modules. Figure 6 represents a configuration of one non-abstracted module, one semi-interface module and three full-interface modules.

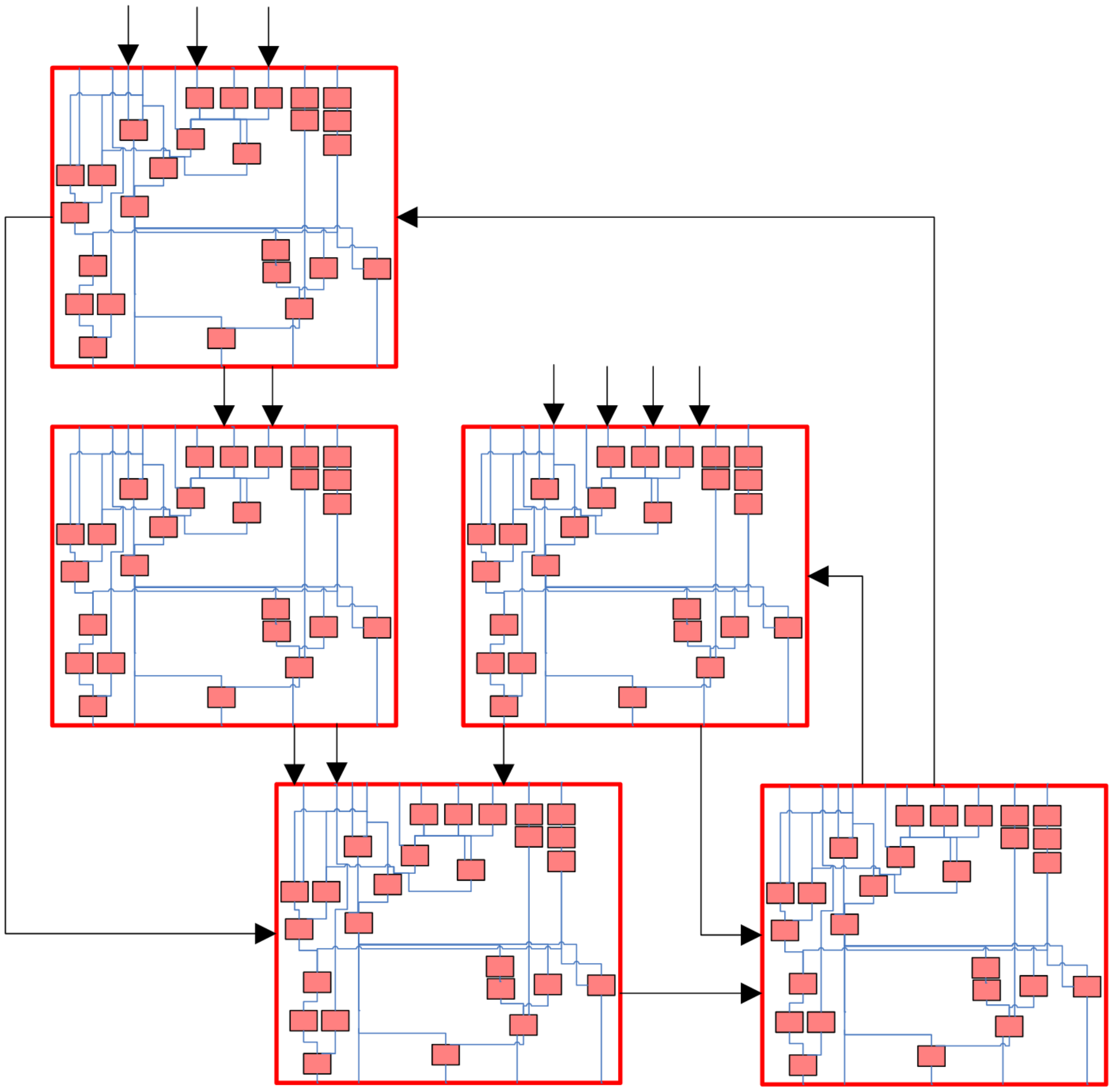


Figure 4. A model configuration, in which all the modules are non-abstracted.

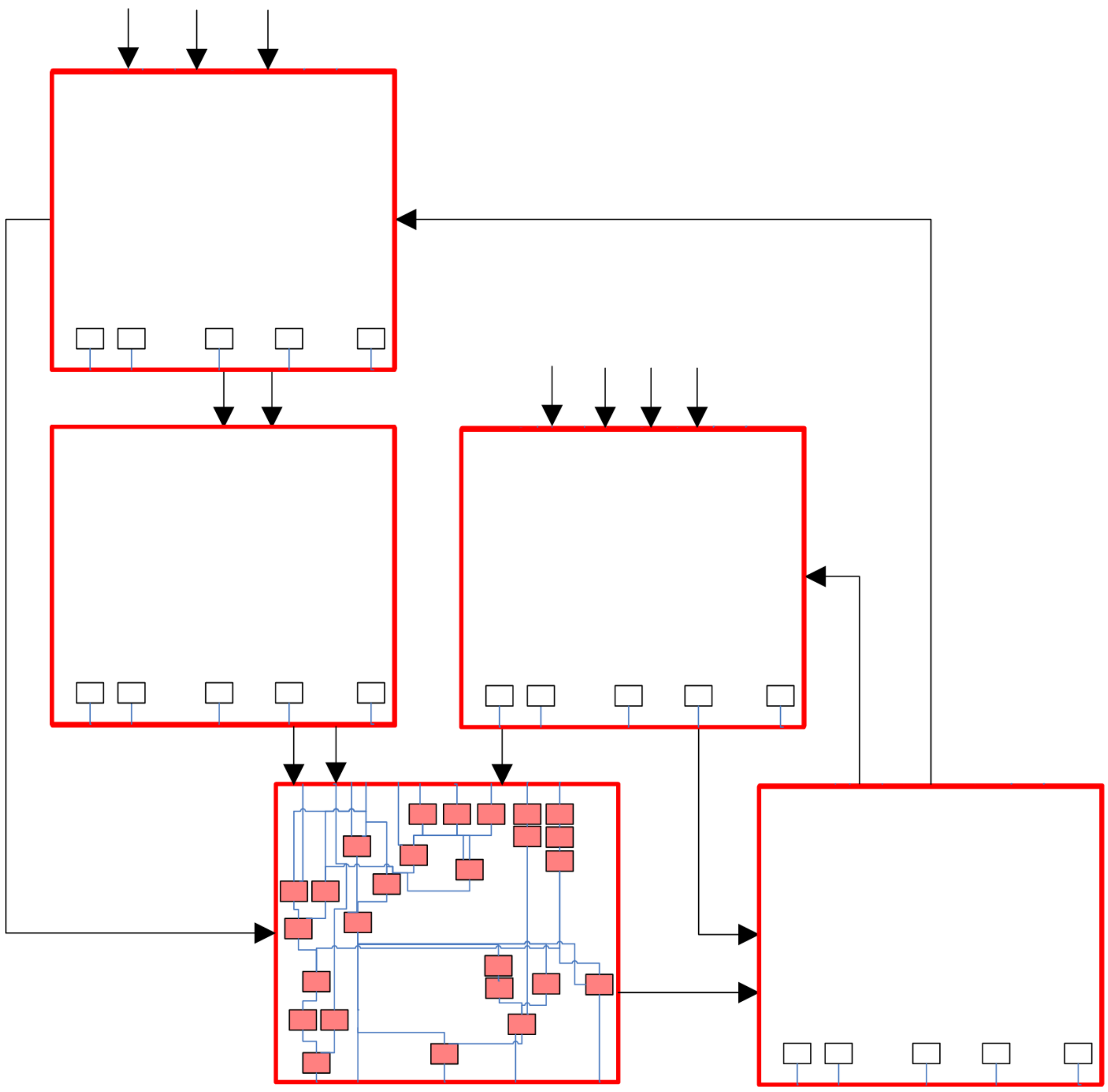


Figure 5. A model configuration, in which one module is non-abstracted, and four modules are full-interface modules.

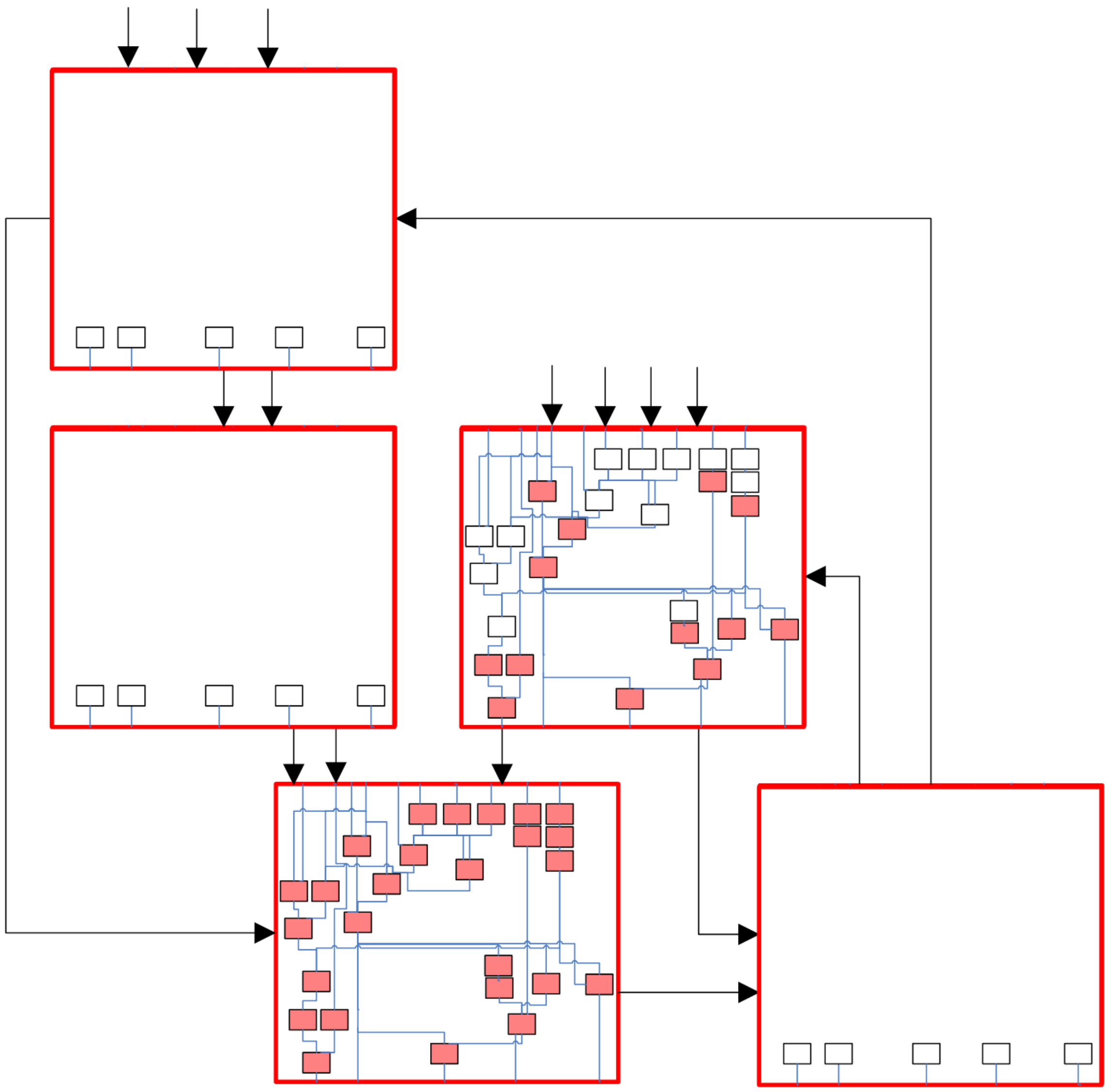


Figure 6. A model configuration of one non-abstracted module, one semi-interface module and three full-interface modules.

As the configurations can be easily compiled, the remaining questions are: 1) How are the abstract configurations used to deduce whether a property is true in the accurate non-approximated model? 2) How can the correct configuration of modules be found, that both allows a property to be verified and is computationally manageable?

The first question can be answered when the examined system property is a safety property, i.e. the property states that undesired behaviour never occurs. In case of safety automation systems, the temporal properties that need to be verified are typically safety properties.

If a safety property is true in a model configuration, in which some of the modules are replaced by interface modules, the same property is also true in the accurate non-abstracted model. Interface modules are over-approximations, i.e. they have more behaviour than the regular non-abstracted modules. If a model configuration that uses these interface modules cannot produce undesired behaviour (violate the checked property), then the accurate model also cannot violate the property.

If a safety property is false in a model configuration containing interface modules, it should be determined whether the violation of the property is feasible in the non-abstracted model, or if the violation is caused by the interface modules. This check can be performed manually, but could also be done automatically using the counter-example given by the model checker.

If the property is violated because of the behaviour of the interface modules, a new configuration of interface modules and non-abstracted modules should be selected for examination.

Regarding the second question, it is our vision that an automatic method for the selection of model configurations (similar to [15]) can be created, and thus the verification of large systems can in many cases be automated based on an iterative algorithm. These subjects are left open for future research.

5. Modelling of the system

The emergency diesel generator control system was modelled with two model checkers: NuSMV and Uppaal. The tools have different modelling methodology and thus the models are not completely identical. The strengths of the tools were taken into consideration in the modelling process. Our existing function block diagram modelling techniques could be used with NuSMV effectively. Therefore, with NuSMV we tried to focus on how to manage and check properties of large models using compositional verification. With Uppaal the focus was more on the creation of a methodology for the model checking of function block diagrams and laying the foundation for the methodology of asynchronous function block models.

5.1 NuSMV modelling

The objective of NuSMV modelling was to study the compositional verification of large models. Substantial parts of this work are the techniques for interface module construction and a technique for easy selection of model configurations that consist of non-abstracted and interface modules.

The NuSMV model consists of nine modules representing the ten system functions (two functions were merged in the model), a function block library and four modules representing the environment of the system. In addition, interface modules were created for each module (13 modules). It is also possible to create semi-interface modules, in which the output values of some function blocks can change freely. For this purpose an interface function block library was also created. Details of interface and semi-interface modules can be found in Section 4.

The NuSMV model has 2200 lines of code, including tests and comments. This does not include code for interface modules (680 lines) or the interface function block library (230 lines).

The NuSMV model assumes that the whole system is one synchronous unit, in which a signal travels through all modules during one clock cycle. The assumption is, however, false because the modules are typically implemented on separate decentralised processors. Using this assumption, the model checking is simpler, but some faults may remain undetected. In NuSMV, it is also assumed that all modules use the same clock, i.e. clock drift or similar phenomena are not taken into account. The clock cycle used in the NuSMV can be changed. The clock cycle values used in this case study were 0.1 s and 1.0 s.

5.1.1 The environment model

The environment model includes the expected diesel functionality as it is turned on, the relevant busbar and three circuit breakers. The busbar is modelled as a separate module and it is powered if a breaker connected to an operational power supply is closed.

The three breakers are modelled as separate modules. A breaker can be open or closed based on its inputs. The priority logic of the signals controlling the breakers is also modelled.

The diesel generator model contains the functionality of the generator as it is turned on. The model includes the control inputs of the generator, the time dependent behaviour from start-up to full power operation and the outputs of the generator, such as the speed of the diesel generator and information of produced voltage level.

5.2 Uppaal modelling

A modular model checking approach was developed for modelling function block diagrams with Uppaal. In addition to modularity, one objective of the Uppaal modelling was to study asynchronous features of the system and how Uppaal is suited to investigating asynchronous systems.

The Uppaal models are different from NuSMV models: there is no clock cycle, but the system changes state whenever some of the inputs changes or some timeout happens. Two versions of the model were created in Uppaal. One version is synchronous as in NuSMV and the other is completely asynchronous, taking into account all possible signal propagation orders.

Corresponding to the function block libraries used in the NuSMV modelling, component libraries for both modelling approaches (synchronous and asynchronous) were created. In the component library there is one parameterisable automaton template for each function block. The models are created by instantiating the templates. The automata communicate with each other through synchronisation channels and with shared variables. Shared variables are used to keep track of the current value of each signal.

The component libraries make modelling new function block diagrams easier. The models can also be created in some other ways, by combining several function blocks into one automaton, for example. With those techniques a somewhat smaller state space can be obtained, but the modelling is more difficult and time-consuming because the automata cannot be reused. Modular models are also easier to read and understand. In addition, making modifications to a modular model is easy if a new revision of the system needs to be modelled.

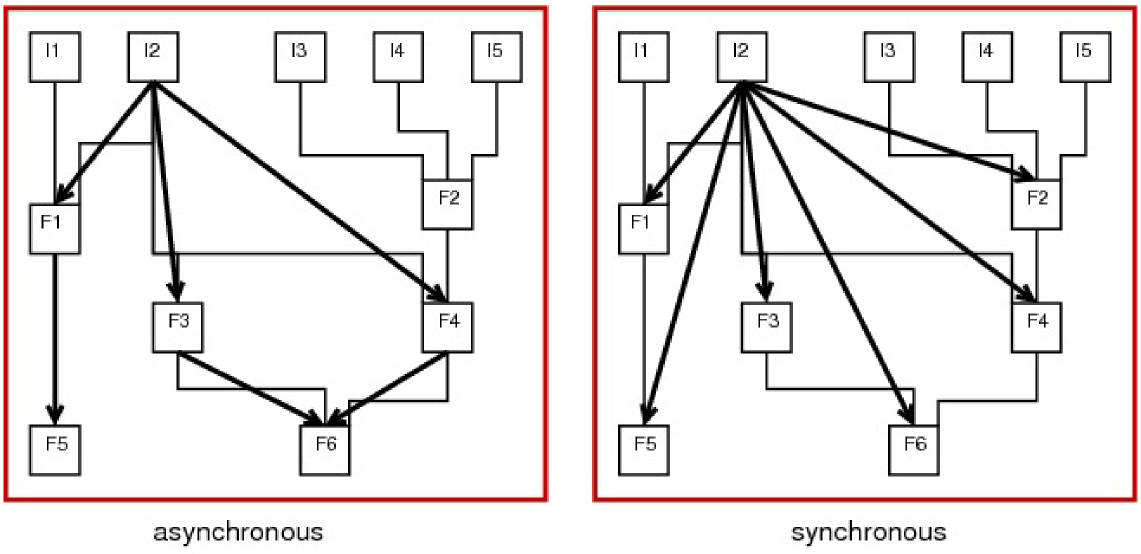


Figure 7. Signal propagation by synchronisation in asynchronous and synchronous Uppaal models.

5.2.1 Asynchronous model

The asynchronous model takes into account all the different signal propagation sequences. This makes it possible to find bugs related to distributed systems. In the case of asynchronous models the model checker investigates all the possible interleavings of the signals.

In asynchronous models there is one synchronisation channel for each signal in the logic diagram. The signals are relayed between the function blocks one block at a time. When some signal changes, synchronisation is sent to all the blocks receiving the signal as input. When synchronisation about a signal change is received, globally shared signal values are used as guards to determine which transitions are enabled. The receiving blocks then send new synchronisations if their outputs change. In this way the signal changes propagate through the whole logic diagram.

The asynchronous model does not make any assumptions concerning the order in which the function blocks are updated and thus asynchronous modelling allows all possible orders of events to be examined. As a result, the asynchronous model may even contain behaviour that the original system cannot reproduce.

In the left side image of Figure 7 is an example of asynchronous signal passing. In the top row are five automata generating input signals. The other squares depict automata modelling function blocks. The arrows are synchronisations sent from one automaton to one or more receiver automata. In the figure, input I2 changes its state and synchronisation is sent to function blocks F1, F3 and F4. If I2's change causes the outputs of some of those function blocks to change, they send new synchronisation to the following function blocks. The order of evaluation of the function blocks F3 and F4 is not specified but they can be evaluated in any order. In this way the model checker investigates both orders of arrival of the two input signals of F6.

5.2.2 Synchronous model

The behaviour of the synchronous model is more like that of the NuSMV model. All the function blocks are updated simultaneously as a signal change occurs. In synchronous models there is only one synchronisation channel, which is used to globally announce signal changes. For example, if some input changes, all the automata modelling the function blocks change their state in one global state transition.

On the right side image of Figure 7 a synchronisation of the change of input I2 is sent to all the automata modelling the function blocks. The signals are propagated in a predetermined order and, for example, function block F6 calculates its output after both F3 and F4 have been updated.

6. Results and findings

This section presents the results of model checking: a comparison of the two model checking methods and descriptions of some of the unwanted system behaviour that was found using model checking. The compositional verification techniques described in Section 4 were not used during the comparison of the methods, but for analysis of the system to find the NuSMV counter-examples (Section 6.3).

6.1 Utilisation of compositional model checking methods

The non-abstracted NuSMV model has such a large state space that properties cannot be checked on the non-abstracted model using a practical amount of time or memory. The examined properties in this case study were verified using the compositional model checking technique described in Section 4, which reduces the state space and the required resources. In Uppaal, similar restricted models were constructed manually so that the model checking was feasible.

Based on the experiences of using the compositional technique in this case study, only one or two modules are usually needed for the verification of a single property. Some properties, however, require the inclusion of several modules. The selection of these relevant modules is not always straightforward, and heuristics for module selection are still needed. When several modules are required for verification, semi-interface modules can be used to further avoid state space explosion.

6.2 Comparison of model checking performance

The performance of the two model checking tools and associated modelling methods was compared. The methodologies of the two tools are quite different. Nonetheless, for the comparison, models for five different module configurations were constructed in a way that minimises these differences. For example, the NuSMV models were manually edited to replace interface modules with a set of free global variables, which better corresponds to Uppaal models. The checked property (no deadlocks) was also chosen so that tool differences had little effect. The deadlock property is such that the whole state space is considered in both tools.

The model configurations used in the comparison consisted of only one or two modules. In other words, the compared models cover only a small part of the diesel generator control system’s functions.

A timeout limit of one hour was used in the comparison. The memory usage was limited to 4 GB by the PC used in the comparison. The model checking runs were performed on a PC with Intel Core 2 Quad Q9550 processor and 8 GB of RAM. The operating system used was Ubuntu 9.10. For model checking, NuSMV version 2.5.0 and Uppaal version 4.0.11 were used.

Table 1 illustrates NuSMV performance for each module configuration. Using the 1 s clock cycle, all configurations could be checked well within the limits of time and memory. Two of the configurations using 0.1 s clock cycles could also be checked, while three 0.1 s clock cycle configurations could not be checked within one hour.

Table 1. NuSMV results of comparison using 0.1 s and 1.0 s clock cycles.

	0,1 s		**1 s**	
Model	**Time**	**Mem (MB)**	**Time**	**Mem (MB)**
Module 1	> 1 h	≥ 190	3 min	69
Module 2	41 s	41	17 s	20
Module 3	> 1 h	≥ 20	2 min	46
Modules 3, 4	> 1 h	≥ 24	17 min	68
Module 4	10 s	16	4 s	10

Table 2 illustrates Uppaal performance for both (asynchronous and synchronous) modelling methods. One asynchronous module configuration was checked within resource limits. Other asynchronous configurations and all synchronous configurations were either timed out or consumed too much memory.

Table 2. Uppaal (including fault bits) results of comparison for the synchronous and the asynchronous models.

	Async		**Sync**	
Model	**Time**	**Mem (MB)**	**Time**	**Mem (MB)**
Module 1	> 1 h	≥ 1100	–	> 4 GB
Module 2	24 s	28	–	> 4 GB
Module 3	–	> 4 GB	–	> 4 GB
Modules 3, 4	–	> 4 GB	–	> 4 GB
Module 4	–	> 4 GB	> 1 h	≥ 140

Simpler versions of the Uppaal models were created in order to obtain more useful results. Unlike in the NuSMV models, the fault signals that are carried with every signal were not included in these Uppaal models. Table 3 illustrates these results. Two of the asynchronous configurations and two of the synchronous configurations could be checked, while other configurations exceeded the limits set for time and memory.

Table 3. Uppaal (without fault bits) results of comparison for the synchronous and the asynchronous models.

	Async		Sync	
Model	**Time**	**Mem (MB)**	**Time**	**Mem (MB)**
Module 1	> 1 h	≥ 270	–	> 4 GB
Module 2	< 1 s	4	3 s	5
Module 3	> 1 h	≥ 920	> 1 h	≥ 400
Modules 3, 4	> 1 h	≥ 2600	> 1 h	≥ 2500
Module 4	24 s	65	20 s	5

6.3 Findings

Model checking of the system properties resulted in several counter-examples that were analysed. Analysis of these counter-examples led to the discovery of a few system requirements that were violated. Many of these violations could be explained by the generality of the design documentation, i.e. the level of detail used in the design documentation did not fully include signal status handling. Other findings were related to the timing issues of the logic. The reasons behind these remaining findings can be divided into three categories: 1) two signals changing values at the same clock cycle causes unplanned operation, 2) two consecutive operational sequences interfere with each other and 3) asynchronous operation between modules results in a property violation. The findings related to asynchronous operation could only be detected using Uppaal. Other findings can be found using NuSMV. In what follows, two finding types are examined in more detail.

6.3.1 Findings related to overlapping of sequences

In two cases a design fault was found, in which a control sequence of the diesel is disrupted and restarted in a rapid manner. This results in unwanted behaviour since the first sequence has not ended properly before the second sequence starts.

Part of the logic causing an overlapping sequence is illustrated in Figure 8. The logic consists of a set-reset flip-flop, two TON timer blocks (8 s, 30 s), a time pulse function block (10 s), AND-block and an OR-block. The logic intends to carry out a starting sequence of alternating signals given to a device. The sequence is specified so that first the ON-signal is given for 8 seconds, and then the OFF-signal is given for 10 seconds. After the OFF-signal, the ON-signal is given again for 12 seconds.

The intended sequence may be interrupted by the Reset signal but the interrupt should occur in a safe way. In particular, it is expected that the ON-signal is not given continuously for long periods of time, since this might be harmful to the device. When this property of the system was examined by model checking, a counter-example was found that shows how the ON-signal can be continuously set up to a maximum of 22 seconds. This unwanted behaviour occurs when the starting sequence is reset and quickly restarted just after 8 seconds after the first Start signal. This way the time pulse block is not reset, which interferes with the newly restarted system behaviour. In particular, the time pulse will not be re-initiated because the time pulse function block does not detect the rising edge from the 8 s TON-block.

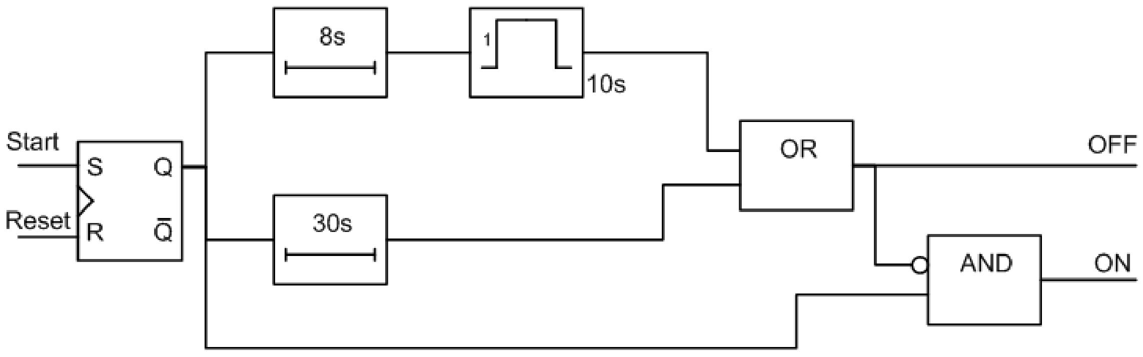


Figure 8. Part of the logic related to overlapping sequences.

6.3.2 Findings related to the asynchronous behaviour of the system

The asynchronous Uppaal models revealed some unwanted behaviour related to the distribution of the systems. As mentioned in Section 3.1, the different logic diagrams are executed in different computers and the computers communicate with each other through a data bus. If there are any communication delays, the signals might not propagate from one logic diagram to another within one clock cycle. On the other hand, if the processors use their own clocks without synchronisation, clock skew can cause the execution cycles to overlap in such a way that signals do not propagate from one diagram to another within one clock cycle. This may cause some unexpected behaviour. With the asynchronous Uppaal models some properties were found to be violated in case of signals not propagating within one clock cycle.

For example, take a property that a valve may not be controlled to open and close simultaneously. The valve is controlled to open in one logic diagram and when the output signal to open the valve is set, a signal preventing the closing of the valve is relayed to another diagram running in another processor. Because of signal delays, the signal does not relay to the other diagram within the same clock cycle and the outputs are set simultaneously.

With the asynchronous Uppaal models, undesired functionality relating to distributed operation was found in four of the checked properties. In all of these cases a signal is relayed to another logic diagram. The functionality can be prevented by not running those logics in separate processors but this is not always feasible. In that case the logic needs to be designed so that all asynchronous operations are taken into account.

7. Conclusions

An emergency diesel control system was analysed using two model checkers: NuSMV and Uppaal. The work done with NuSMV tool focused on the development of a compositional model checking method for large modular systems, while Uppaal was used to study the systems in more detail, i.e. modelling also the asynchronous communication between components.

A method of compositional model checking was adopted and successfully utilised in a real industrial case. The method allows the model checking of large systems that otherwise could not be examined as quickly and smoothly. The method is based on compositional minimisation, in which modules can be easily replaced by abstracted interface modules.

The compositional verification technique used here significantly reduces the manual work required. The technique can probably be further automated to ease the analysis of large systems. The objective of the technique is to automatically find a suitable configuration of modules that is computationally feasible but at the same time describes the system to be analysed with enough details to enable verification of selected properties. The responses of the model checking tool could be used for selecting such a configuration but that question is left for future projects.

However, the benefit of the method is dependent on the checked temporal property. Only safety properties can be examined with the current methodology. Also, all properties cannot be checked with the method because some properties are dependent on a large portion of the logic and, thus, verifying the property requires the inclusion of too many modules. The methodology brings real added value to the model checking tool box but it is clear that additional research and development is still required.

Uppaal modelling shows that function block diagrams can be modelled using timed automata. Additionally, an asynchronous modelling method for function block diagrams was created for Uppaal that depicts the system in more detail. This technique makes it possible to find faults that cannot be found using the synchronous NuSMV modelling technique. However, model checking function block diagrams with Uppaal is currently inefficient. More effective real-time based model checking techniques are needed.

The current NuSMV modelling technique is based on an assumption of synchronous behaviour between different functions that are implemented on separate computers. Following the technique developed in Uppaal, a similar technique could also be implemented with NuSMV on top of our current methods.

NuSMV and Uppaal have different strengths. It would be quite beneficial if both tools could be used in the compositional verification of a single property. Systematic methodology for this does not yet exist.

Future work includes the development and improvement of compositional model checking methods for large systems and modelling of systems with more precision so that the asynchronous phenomena in particular could be taken into consideration.

References

1. Alur, R. & Dill, D. L. “A theory of timed automata”. Theoretical Computer Science, 126(2), 1994, pp. 183–235.

2. Alur, R., Courcoubetis, C. & Dill, D. “Model-checking for real-time systems”. In: Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science, 1990. Pp. 414–425.

3. Berezin, S., Campos, S. V. A. & Clarke, E. M. “Compositional Reasoning in Model Checking”. In: Revised Lectures from the International Symposium on Compositionality: The Significant Difference (COMPOS’97). Roever, W. P. de, Langmaack, H. & Pnueli, A. (Eds.). Springer-Verlag, London, UK, 1997. Pp. 81–102.

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

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

6. Björkman, K., Frits, J., Valkonen, J., Lahtinen, J., Heljanko, K., Niemelä, I. & Hämäläinen, J.J. 2009. Verification of safety logic designs by model checking. In: Proceedings of the Sixth American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies NPIC&HMIT 2009, Knoxville, Tennessee, April 2009.

7. Björkman, K., Frits, J., Valkonen, J., Heljanko, K. & Niemelä, I. 2009. Model-based analysis of a stepwise shutdown logic. VTT Working Papers 115, VTT Technical Research Centre of Finland, Espoo. 36 p. + app. 4 p. http://www.vtt.fi/inf/pdf/workingpapers/2009/W115.pdf.

8. Björkman, K., Valkonen, J. & Ranta, J. Verification of Automated Changeover Switching Unit by Model Checking. In: Proceedings of the 7th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human-Machine Interface Technologies (NPIC&HMIT 2010), November 7–11, 2010, Las Vegas, Nevada. Pp. 1719–1728.

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

10. Burch, J. R., Clarke, E. M. & Long, D. E. “Symbolic Model Checking with Partitioned Transition Relations”. North-Holland, 1991, pp. 49–58,

11. Cavada, R., Cimatti, A., Jochim, C. A., Keighren, G., Olivetti, E., Pistore, M., Roveri M. & Tchaltsev, A. 2010. “NuSMV 2.5 User Manual”. FBK-irst.

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

13. Clarke, E., Long, D. & McMillan, K. “Compositional model checking”. In: Proceedings of the Fourth Annual Symposium on Logic in computer science, IEEE Press, Piscataway, NJ, USA, 1989. Pp. 353–362.

14. Clarke, E. M., Grumberg, O. & Peled, D. A. 1999. “Model Checking”. The MIT Press.

15. Clarke, E. M. “SAT-Based Counterexample Guided Abstraction Refinement”. In: Proceedings of the 9th International SPIN Workshop on Model Checking of Software. Bosnacki, D. & Leue, S. (Eds.). Springer-Verlag, London, UK, 2002.

16. Cobleigh, J. M., Giannakopoulou, D. & Păsăreanu, C. S. “Learning assumptions for compositional verification”. In: Proceedings of the 9th international conference on tools and algorithms for the construction and analysis of systems (TACAS'03). Garavel, H. & Hatcliff, J. (Eds.). Springer-Verlag, Berlin, Heidelberg, 2003. Pp. 331–346.

17. Lahtinen, J. 2008. Model checking timed safety instrumented systems. Vol. 3. Espoo: Helsinki University of Technology. TKK reports in information and computer science. ISBN 978-951-22-9445-9. http://lib.tkk.fi/Reports/2008/isbn9789512294459.pdf.

18. Lahtinen, J., Valkonen, J., Björkman, K., Frits, J. & Niemelä, I. “Model checking methodology for supporting safety critical software development and verification”. In: European Safety and Reliability Conference, ESREL2010. Rhodes, Greece, 5–9 Sept. 2010. Reliability, Risk and Safety – Back to the Future. Ale, Papazoglou & Zio (Eds). European Safety and Reliability Association, ESRA. London (2010). Pp. 2056–2063.

19. Larsen, K. G., Pettersson, P. & Yi, W. “UPPAAL in a nutshell”. International Journal on Software Tools for Technology Transfer, 1(1–2), 1997, pp. 134–152.

20. McMillan, K. L. 1993. “Symbolic Model Checking”, Kluwer Academic Publ.

21. McMillan, K. L. “Circular Compositional Reasoning about Liveness”. In: Proceedings of the 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME’99), Laurence Pierre and Thomas Kropf (Eds.). Springer-Verlag, London, UK, 1999. Pp. 342–345.

22. NuSMV Model Checker v.2.4.3, 2008. http://nusmv.irst.itc.it/.

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

24. Peng, H. & Tahar, S. “Survey on compositional verification”. Technical report, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada, November 1998.

25. Quielle, J. & Sifakis, J. 1981. “Specification and verification of concurrent systems in CESAR”. In: Proceedings of the 5th International Symposium on Programming. Pp. 337–350.

26. Rushby, J. “Formal Verification of McMillan’s Compositional Assume-Guarantee Rule”. Technical Report, University of Minnesota, Minneapolis, 2001.

27. UPPAAL integrated tool environment v. 4.0.6, 2009. http://www.uppaal.com/.

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

29. Valkonen, J., Pettersson, V., Björkman, K., Holmberg, J.-E., Koskimies, M., Heljanko, K. & Niemelä, I. 2008. Model-Based Analysis of an Arc Protection and an Emergency Cooling System. VTT Working Papers 93, VTT, Espoo. 13 p. + app. 38 p. http://www.vtt.fi/inf/pdf/workingpapers/2008/W93.pdf.

30. Valkonen, J., Koskimies, M., Pettersson, V., Heljanko, K., Holmberg, J.-E., Niemelä, I. & Hämäläinen, J. J. 2008. Formal Verification of Safety I&C System Designs: Two Nuclear Power Plant Related Applications, Enlarged Halden Programme Group Meeting. Proc. Man – Technology-Organisation Session. Loen, Norway, 18–23 May.

31. Valkonen, J., Koskimies, M., Björkman, K., Heljanko, K., Niemelä, I. & Hämäläinen, J. J. 2009. Formal verification of safety automation logic designs. In Automaatio XVIII 2009 Seminaari.

32. Valkonen, J., Björkman, K., Frits, J. & Niemelä, I. “Model Checking Methodology for Verification of Safety Logics”. In: Proceedings of the 6th International Conference on Safety of Industrial Automated Systems (SIAS 2010) Tampere, June 14–15, 2010.

ISBN 978-951-38-7497-1 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)