MÄLARDALENS HÖGSKOLA
ESKILSTUNA VÄSTERÅS

Model Checking-Based Software Testing for Function Block Diagrams

Licentiate Thesis Proposal

Eduard Paul Enoiu
School of Innovation, Design, and Engineering
eduard.enoiu@mdh.se

November 19, 2013

Main Supervisor: Paul Pettersson
Co-Supervisor: Daniel Sundmark

Abstract

During the last decade, testing with model-checking techniques for software-intensive systems has been developed based on the theory of model-checking. However, the main problem in using model-checking for testing industrial software systems is the potential combinatorial explosion of the state space and its limited application to models used in practice. In this thesis, we improve the current status of testing with model-checking techniques by developing a framework suitable for transforming Function Block Diagrams (FBD), a widely used model in safety-critical software applications, to a formal representation of both its functional and timing behavior. For this, we implement an automatic model–to–model transformation to timed automata. The transformation accurately reflects the data-flow characteristics of the FBD language by constructing a complete behavioral model which assumes a *read-execute-write* program semantics. In addition, we develop a test case generation technique based on model-checking, tailored for structural coverage of FBD programs. We define logic coverage for FBD programs based on the transformed timed automata model. This copes with both functional and timing behavior of an FBD program. This formal definition is necessary for the approach to be applicable to model-checking. We present how a model-checker can be used to generate test cases for covering an FBD program.

The developed techniques have been implemented in a testing tool. To demonstrate the potential applications of our techniques, we present a framework for testing FBD programs and a case study where the tool and its methodology are applied. Based on our experiments, this method is — for the real world models provided by Bombardier Transportation AB — a useful and applicable way of generating test cases.

Chapter 1

Background and Motivation

Within the last decade model-checking has turned out to be a useful technique for generation of test cases from finite-state models [12]. However, the main problem in using model-checking for testing industrial software systems is the potential combinatorial explosion of the state space and its limited application to models used in practice.

Safety-critical and real-time software systems implemented in Programmable Logic Controllers (PLCs) are used in many real-world industrial application domains. One of the programming languages defined by the *International Electrotechnical Commission* (IEC) for PLCs is the *Function Block Diagram* (FBD). Programs developed in FBD are transformed into program code, which is compiled into machine code automatically by using specific engineering tools provided by PLC vendors. The motivation for using FBD as an implementation model comes from the fact that this language is the standard in many industrial software systems, such as rail transport control.

In this thesis, our goal is to help testers automatically develop tests for safety-critical software systems modeled in FBD. One example includes logic coverage which needs to be demonstrated on the developed programs. There has been little research on using logic coverage criteria for FBD programs in an industrial setting. In some cases logic coverage is analyzed at the code level [6]. Even if at the code level, logic coverage is used, it would be difficult to standardize the code generation scheme for different PLC tool vendors in order to map directly the criteria to the original FBD program. Hence, in this model-driven environment it is advantageous to move as much testing activity from code level to FBD program level as possible.

As the first contribution of this thesis, we developed a framework suitable for transforming FBD programs to a formal representation of both its functional and timing behavior. For this, we implement an automatic model–to–model transformation to timed automata, a well known model introduced by Alur and Dill [1]. The choice of timed automata as the target language is motivated primarily by its formal semantics and tool support for simulation and model-checking. Our goal is not to solve all testing issues (e.g., robustness, schedulability, etc.), but to allow the usage of a framework for formal reasoning about testing FBD programs. The transformation accurately reflects the data-flow characteristics of the FBD language by constructing a complete behavioral model which assumes a *read-execute-write* program semantics. The translation method consists of four separate steps. The first three steps involve mapping all the interface elements and the existing timing annotations. The latter step produces a formal behavior for every standard component in the FBD program. These steps are independent of timed automata and thus are generic in the sense that they could also be used when translating an FBD program to another target language. This allowed us to investigate further test case generation techniques based on model checking.

As the second contribution of this thesis, we developed a testing technique based on model-checking, tailored for logic coverage of FBD programs. There have been a number of testing techniques used for defining logic coverage using model-checkers, e.g., [4, 18, 19]. However, these techniques are not directly applicable to FBD programs and semantics. Our main goal with this study was to define logic coverage for FBD programs based on the transformed timed automata model. This copes with both functional and timing behavior of an FBD program. We also found that a formal definition is necessary for the approach to be applicable to model-checking. We show how a model-checker can be used to generate test cases for covering an FBD program.

As the third contribution of this thesis, we developed a testing tool for safety critical applications, described in Function Block Diagram (FBD) language, aimed to support both a model and a search-based approach. We found that FBD programs have many easy to cover structures that allow testing with model checkers to perform surprisingly well. Currently, we are investigating a more elaborate empirical evaluation for the use of testing with model checkers and logic coverage.

Chapter 2

Research Description

The past years have witnessed increasing research within model-based testing. The design of software has a direct impact on the final implementation, with respect to performance and other quality attributes. We argue that there is a need for testing safety-critical software that are originally described in the domain-specific language Function Block Diagram. We propose a model-based approach that integrates model checking, and describe its tool support.

2.1 Thesis Statement

In this section, we present the main goal of the thesis, which is split into three subgoals directly addressed in our work.

Overall Goal. *To provide an approach to software testing for Function Block Diagram models that is useful and applicable in practice.*

As we have described in Chapter 1, the need for testing Function Block Diagrams motivated us to provide a framework for testing of such models using model checking techniques. Thus, we chose it as our overall goal. Since this goal is too abstract to be directly addressed, we have further divide it into three more concrete subgoals.

In order to be able to provide a framework for testing Function Block Diagrams, one needs an expressive and well-defined technique that would support testing functional and timing behavior. Hence we have formulated the first subgoal as follows:

Subgoal 1. *To present a framework for testing tailored to Function Block Diagrams.*

The first subgoal is the basis for the next two subgoals, in that it provides a model-based test generation method tailored for Function Block Diagram programs. The next step is to propose and demonstrate the use of the Uppaal tool for testing, which gives rise to the second subgoal as follows:

Subgoal 2. *To study and apply software testing on industrial systems using an integrated tool.*

In the second subgoal, we develop a testing tool based on the Uppaal model checker. Many benefits emerge from developing an integrated tool, including the ability to automatically generate test cases for real industrial software systems described in Function Block Diagram language.

To support testers and developers when testing Function Block Diagram programs we have formulated the third subgoal as follows:

Subgoal 3. *To investigate how logic coverage can improve testing of Function Block Diagrams and which criteria are suitable for specific characteristics of real models.*

The last subgoal is based on the proposed logic coverage as an useful and applicable criteria to Function Block Diagram models and aims at providing evidence on the effectiveness and efficiency of logic coverage.

2.2 Thesis Contribution

The thesis will include four conference papers.

	Subgoal 1	Subgoal 2	Subgoal 3
Paper A	✓	✓	
Paper B	✓	✓	
Paper C		✓	✓
Paper D			✓

Table 2.1: Contribution of the individual papers to the research subgoals

2.2.1 Paper A

In the first paper, we introduce the framework and by that we address Subgoal 1 and Subgoal 2. In Paper A we propose a translation of FBD programs into timed automata models. We present a test generation approach sing the UPPAAL model-checker in the context of a model-based approach towards unit testing. For the translation of an FBD program into a TA model, a set of rules are presented. On the basis of this model, a model checker has been used for generating consistent test suites.

2.2.2 Paper B

Based on Paper A and aimed at increasing confidence on the results for Subgoal 1 the second paper presents a testing tool for Function Block Diagrams, as well as several specific implications. The tool is aimed at safety critical applications described in Function Block Diagram language, and supports both a model and a search-based approach. In Paper B, and to achieve Subgoal 2, we describe the architecture of the tool, its workflow process, and a case study in which the tool has been applied in a real industrial setting to test a train control management system.

2.2.3 Paper C

As a direct result of the results from Paper A, we address Subgoal 3 in order to improve testing of Function Block Diagrams. We generate tests that cover the structure of Function Block Diagrams. One way of dealing with test generation is to approach it as a model checking problem, such that model checking tools automatically create tests. We start from the framework introduced in Paper A and we show how logic coverage criteria can be formalised and used by a model checker to provide test cases ready to be executed.

From our experiments with a typical program we noticed that for more complicated logic coverage criteria, test cases result in longer traces than for simpler logic coverage criteria. To achieve Subgoal 2 we assess the applicability and scalability of using logic coverage for testing FBD programs with various sizes and specific complexities. For the models used in Paper C, the timer component appears to be significantly affecting the generation time. We modified the program by increasing or decreasing the number of components in the model. We note that the use of timer elements is influencing the handling of larger systems, with an increased cost of generation time and used memory.

2.2.4 Paper D

We conclude this collection of papers with a paper detailing a large case study, as well as a more elaborate empirical evaluation of the use of our framework. To further address Subgoal 3 we measure both efficiency and effectiveness of using logic coverage for Function Block Diagram programs. In Paper D, we empirically evaluate the fault detection effectiveness of logic coverage criteria using mutation analysis. We produce tests satisfying logic coverage criteria and generate mutants automatically for industrial Function Block Diagram models. Based on the results, we compare different logic criteria, and suggest improvements to testing Function Block Diagram.

2.3 Research Methodology

The research is based on both theoretical (Papers A-C) and empirical methodologies (Paper D) including deductive methods and analysis of quantitative data. In Papers A-C deductive research is performed by giving formal descriptions, using prototype implementations, and evaluating the framework on industrial examples.

2.4 Publications Included in the Thesis

This licentiate thesis is presented as a collection of papers. The following papers will be included in the thesis.

2.4.1 Paper A

Model-based Test Suite Generation for Function Block Diagrams using the UPPAAL Model Checker. [10]
Eduard Paul Enoiu, Daniel Sundmark, and Paul Pettersson
Status: Published in the Sixth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pages 158 - 167, ISBN: 978-1-4799-1324-4, 2013, IEEE.

Abstract. *A method for model-based test generation of safety-critical applications using Programmable Logic Controllers and implemented in a programming language such as Function Block Diagram (FBD) is described. It involves the transformation of FBD programs with timed annotations into timed automata models which are used to automatically generate test suites. Specifically we demonstrate how to use model transformation for formalization and model-checking of FBD programs using the UPPAAL tool. Many benefits emerge from this method, including the ability to automatically generate test suites from a formal model in order to ensure compliance to strict quality requirements including unit testing and specific coverage measurements. The approach is experimentally assessed on a train control system in terms of consumed resources.*

The development of the concept was done by me, Daniel Sundmark and Paul Pettersson. I implemented the models, tool implementations, and performed the experiments.

2.4.2 Paper B

MOS: An Integrated Model-based and Search-based Testing Tool for Function Block Diagrams. [7]
Eduard Paul Enoiu, Kivanc Doganay, Markus Bohlin, Daniel Sundmark, Paul Pettersson

Status: Published in the 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE), pages 55 - 60, ISBN: 978-1-4673-6284-9, 2013, IEEE.

Abstract. *In this paper we present a new testing tool for safety critical applications described in Function Block Diagram (FBD) language aimed to support both a model and a search-based approach. Many benefits emerge from this tool, including the ability to automatically generate test suites from an FBD program in order to comply to quality requirements such as component testing and specific coverage measurements. Search-based testing methods are used to generate test data based on executable code rather than the FBD program, alleviating any problems that may arise from the ambiguities that occur while creating FBD programs. Test cases generated by both approaches are executed and used as a way of cross validation. In the current work, we describe the architecture of the tool, its workflow process, and a case study in which the tool has been applied in a real industrial setting to test a train control management system.*

I am the main author and driver of this paper. I implemented the model-based testing part of the tool and performed the experiments.

2.4.3 Paper C

Using Logic Coverage to Improve Testing Function Block Diagrams.[11]
Eduard Paul Enoiu, Daniel Sundmark, Paul Pettersson

Status: Published in Testing Software and Systems, Proceedings of the 25th IFIP WG 6.1 International Conference ICTSS 2013, volume 8254, pages 1 - 16, Lecture Notes in Computer Science,

2013, Springer.
Extension: Invited for a special contribution to the International Journal on Software Tools for Technology Transfer, 2014, Springer.

Abstract. *In model-driven development, testers are often focusing on functional model-level testing, enabling verification of design models against their specifications. In addition, in safety-critical software development, testers are required to show that tests cover the structure of the implementation. Testing cost and time savings could be achieved if the process of deriving test cases for logic coverage is automated and provided test cases are ready to be executed. The logic coverage artifacts, i.e., predicates and clauses, are required for different logic coverage, e.g., MC/DC. One way of dealing with test case generation for ensuring logic coverage is to approach it as a model-checking problem, such that model-checking tools automatically create test cases. We show how logic coverage criteria can be formalized and used by a model-checker to provide test cases for ensuring this coverage on safety-critical software described in the Function Block Diagram programming language. Based on our experiments, this approach, supported by a tool chain, is an applicable and useful way of generating test cases for covering Function Block Diagrams.*

I am the main author of the paper, with my co-authors having academic and industrial advisory role. I implemented the models, the model transformation, and performed the experiments. Elaine Weyuker and Tom Ostrand took part in the discussions and contributed with improving parts of the paper.

2.4.4 Paper D

Software Testing with Model Checkers in Practice: An Industrial Case Study on Logic Coverage
Eduard Paul Enoiu, Adnan Čaušević, Daniel Sundmark, Paul Pettersson, Elaine Weyuker, and Tom Ostrand

Status: To be submitted, January 2014.

Abstract. *Function Block Diagram, one of the PLC programming languages, is a widely used language to implement safety-critical software. We previously proposed logic coverage as useful and applicable to Function Block Diagram models. However important questions remain: How effective is logic coverage in terms of fault detection? In this paper, we empirically evaluate the fault detection effectiveness of logic coverage criteria using mutation analysis. We produce tests satisfying logic coverage criteria and generate mutants automatically for industrial Function Block Diagram models. Based on the results, we compare different logic criteria, and suggest improvements to testing Function Block Diagram.*

2.5 Publications not included in the Thesis

A Methodology for Formal Analysis and Verification of EAST-ADL Models. [15]
Eun-Young Kang, Eduard Paul Enoiu, Raluca Marinescu, Cristina Seceleanu, Pierre Yves Schnobbens, Paul Pettersson
Published in: International Journal of Reliability Engineering and System Safety, 2013, Springer.

ViTAL : A Verification Tool for EAST-ADL Models using UPPAAL PORT. [9]
Eduard Paul Enoiu, Raluca Marinescu, Cristina Seceleanu, Paul Pettersson
Published in: Proceedings of the 17th IEEE International Conference on Engineering of Complex Computer Systems, July 2012, IEEE Computer Society Press.

Extending EAST-ADL for Modeling and Analysis of Systems Resource-Usage. [17]
Raluca Marinescu, Eduard Paul Enoiu
Published in: IEEE 36th Annual Computer Software and Applications Conference Workshops (COMPSACW), July 2012, IEEE Computer Society Press.

A Design Tool for Service-oriented Systems. [8]
E. P. Enoiu, R. Marinescu, A. Causevic, and C. Seceleanu.
Published in: Proceedings of the 9th International Workshop on Formal Engineering approaches to Software Components and Architectures, May 2012, Elsevier, Electronic Notes in Theoretical Computer Science (ENCTS).

A SysML Model for Code Correction and Detection Systems. [23]
Stefan Stancescu, Lavinia Neagoe, Raluca Marinescu, Eduard Paul Enoiu
Published in: Proceedings of the 33rd International Convention on Information and Communication Technology, Electronics and Microelectronics, IEEE Croatia Section, May 2010.

UML Modelling in Design of Error Detection and Correction Circuits. [22]
Stefan Stancescu, Lavinia Neagoe, Raluca Marinescu, Eduard Paul Enoiu
Published in: Proceedings of the 32rd International Convention on Information and Communication Technology, Electronics and Microelectronics, IEEE Croatia Section, May 2009.

Chapter 3

Thesis Outline

The proposed title of this thesis is "Model Checking-Based Software Testing for Function Block Diagrams." The thesis will be written as a collection of three peer-reviewed papers and one in submission paper as presented in the Section 2.4, together with an introduction, a related work chapter and summarising discussions and a conclusion.

Outline of the thesis' sections follows.

Part I: Thesis

1 Introduction

- 1.1 Preliminaries
  - 1.1.2 Model-checking
  - 1.1.1 Testing with Model Checkers
- 1.2 Thesis Overview

2 Research Problems

In this chapter, we will present an overview of the main research goal and then split it between the individual research goals and provide their descriptions.

- 2.1 Problem Description
- 2.2 Research Goals

3 Research Results

Research results are catalogued according to the respective research goals that they are addressing. After presenting the research results, we will reflect on how they have covered the research goals.

4 Related Work

In Related Work, we will present a cross-section of related work relevant to this thesis.

5 Conclusions and Future Work

In this chapter, we will present a list of conclusions from development of this thesis as well as possible future work.

- 5.1 Contributions
- 5.2 Future Research

Part II: Included Papers

Chapter 4

Progress and Time Plan

4.1 Papers

The current status of the papers included in this thesis is:

Paper A is published.

Paper B is published.

Paper C is published.

Paper D is currently in the process of writing.

4.2 Courses

The requirement for licentiate degree is at least 45 ECTS. This requirement is fulfilled, and a summary of the advanced courses is given in Table 4.1 using ECTS points.

Courses	**Credits**	**Status**
Search-based Software Testing	2.5	Completed
Sofware Testing	7.5	Completed
Research Planning	4.5	Completed
Engineering Dependable Software Systems (TU Munchen)	4.5	Completed
Model-driven Engineering	7.5	Completed
Real-Time Systems 2	7.5	Completed
Learning Systems	7.5	Completed
Research Methods in Natural Sciences and Engineering	7.5	Completed
Total Credits	**49**	

Table 4.1: The summary of finished courses

4.3 Writing and compiling the thesis.

With most of the content for this thesis already written, the time plan for the licentiate thesis is to complete the first draft by the end of February and present it on the first opportunity.
In the Table 4.2, we have provided an overview of the upcoming time plan towards the presentation of the thesis.

No.	**Task**	**Start**	**Finish**	**Duration**
1	Thesis Proposal Presentation	November	November	1d
2	Journal Paper Finalization	1 November	Mid December	6w
3	Vacation	Mid December	1 January	1w
4	Writing 1st Draft of the Thesis	7 January	1 February	3w
5	Complete the Thesis	1 February	Mid February	2w
6	Finish Camera Ready Version	Mid February	1 March	2w
7	Licentiate Seminar	End of March	End of March	1d

Table 4.2: Time plan from licentiate proposal to defense

Chapter 5

Related Work

Previous contributions in testing of FBD programs range from a simulation-based approach [20] to verification of the actual FBD program code [3, 14]. The technique in [3] is based on Petri Nets models. In comparison to our work, they are not coping with the internal structure of the PLC logical and timing aspects. It is our opinion that testing FBD programs can be complemented by using a model-checker as presented in this paper. Similar to this work, Rayadurgam and Heimdahl [19] have defined a complete formal framework that can be used for coverage based test-case generation using a model checker. For a detailed overview of testing with model checkers we refer the reader to Fraser et al. [12].
A model checker has been used to find test cases to various criteria and from programs in a variety of formal languages [4, 13]. In addition, Black et al. [2] discuss the problems encountered in using a model-checker for test case generation for full-predicate coverage and explain why logic coverage criteria is not directly applicable for model-checking. Rayadurgam et al. [18] present an alternative method that modifies instead the system model and are obtaining MC/DC adequate test cases using a model-checking approach. Similarly to our work, the system model is annotated and the properties to be checked are expressible as a single test sequence. However, this technique is not coping with the timing behavior of an FBD program as we do and only MC/DC criteria is investigated. We provide an approach to generate test cases for different logic criteria (e.g., PC, CC, and CACC) that are directly applicable to FBD programs.
The idea of using model-checkers for verifying and testing FBD programs is not new [21, 5]. These two approaches use the Uppaal model checker and Uppaal TRON for verification of FBD programs, however they translate their model for functional verification. Soliman et al. [21] provide an automatic transformation to timed automata and their verification methodology is used to check the model against safety requirements. In contrast to the online model-based testing approach used in [5] we generate test suites for offline execution.
Related to this work but outside the PLC testing community, the most notable efforts have been focusing on test coverage for data flow languages. For example, for the Lustre language there are contributions [16] describing an activation condition concept that can be used when data flows from an input edge to an output edge. While this approach studied the effect of structural coverage criteria on the overall program, we study the ability to generate test cases and its effect on the test artifacts, i.e., predicates and clauses, tailored for FBD programs.

Chapter 6

Conclusions

To our knowledge, not much theoretical work and experimental data is available regarding testing for Function Block Diagrams. In our work [10] we have defined a model-based test generation method tailored for Function Block Diagram programs and demonstrated how to use the Uppaal tool for model checking the implementation in order to ensure compliance to quality requirements including unit testing. As a consequence of these results we have developed our own tool [7] to support both a model and search-based testing approach which can include specific coverage measurements. One way of dealing with test case generation for ensuring program coverage is to approach it as a model-checking problem, such that model-checking tools automatically create test cases. We showed [11] how logic coverage criteria can be formalized and used by a model-checker to provide test cases for ensuring this coverage on safety-critical software described in Function Block Diagram language. The testing framework presented in this thesis is based on both our previous work and the related work in this field. It is an attempt to automatically compute tests using a model checker for Function Block Diagrams. We provide evidence for the usage of logic coverage as an improvement to testing of Function Block Diagrams.

Bibliography

[1] R. Alur and D. Dill. Automata for Modeling Real-time Systems. *Automata, languages and programming*, pages 322–335, 1990.

[2] P. Ammann, P. E. Black, and W. Ding. Model Checkers in Software Testing. In *NIST-IR 6777, National Institute of Standards and Technology Report*, 2002.

[3] L. Baresi, M. Mauri, A. Monti, and M. Pezze. PLCTools: Design, Formal Validation, and Code Generation for Programmable Controllers. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, pages 2437–2442. IEEE, 2000.

[4] P. Black. Modeling and Marshaling: Making Tests from Model Checker Counter-examples. In *Proceedings of the 19th Digital Avionics Systems Conference*, volume 1, pages 1B3–1. IEEE, 2000.

[5] L. da Silva, L. de Assis Barbosa, K. Gorgônio, A. Perkusich, and A. Lima. On the Automatic Generation of Timed Automata Models from Function Block Diagrams for Safety Instrumented Systems. In *34th Annual Conference of IEEE Industrial Electronics*, pages 291–296. IEEE, 2008.

[6] K. Doganay, M. Bohlin, and O. Sellin. Search Based Testing of Embedded Systems Implemented in IEC 61131-3: An Industrial Case Study. In *International Conference on Software Testing, Verification and Validation Workshops*. IEEE, March 2013.

[7] E. P. Enoiu, K. Doganay, M. Bohlin, D. Sundmark, and P. Pettersson. MOS: An Integrated Model-based and Search-based Testing Tool for Function Block Diagrams. In *International Conference on Software Engineering Workshops*. IEEE, May 2013.

[8] E. P. Enoiu, R. Marinescu, A. Causevic, and C. Seceleanu. A design tool for service-oriented systems. In *Proceedings of the 9th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA 2012)*. Elsevier, Electronic Notes in Theoretical Computer Science (ENCTS), May 2013.

[9] E. P. Enoiu, R. Marinescu, C. Seceleanu, and P. Pettersson. Vital : A verification tool for east-adl models using uppaal port. In *Proceedings of the 17th IEEE International Conference on Engineering of Complex Computer Systems*. IEEE Computer Society Press, July 2012.

[10] E. P. Enoiu, D. Sundmark, and P. Pettersson. Model-based Test Suite Generation for Function Block Diagrams using the UPPAAL Model Checker. In *International Conference on Software Testing, Verification and Validation Workshops*. IEEE, April 2013.

[11] E. P. Enoiu, D. Sundmark, and P. Pettersson. Using logic coverage to improve testing function block diagrams. In *Proceedings of the International Conference on Testing Software and Systems*. Springer, November 2013.

[12] G. Fraser, F. Wotawa, and P. E. Ammann. Testing with Model Checkers: a Survey. In *Journal on Software Testing, Verification and Reliability*, volume 19, pages 215–261. Wiley Online Library, 2009.

[13] H. S. Hong, I. Lee, O. Sokolsky, and H. Ural. A Temporal Logic-Based Theory of Test Coverage and Generation. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 327–341. Springer, 2002.

[14] E. Jee, S. Kim, S. Cha, and I. Lee. Automated Test Coverage Measurement for Reactor Protection System Software Implemented in Function Block Diagram. In *Journal on Computer Safety, Reliability, and Security*, pages 223–236. Springer, 2010.

[15] E.-Y. Kang, E. P. Enoiu, R. Marinescu, C. Seceleanu, P. Y. Schnobbens, and P. Pettersson. A methodology for formal analysis and verification of east-adl models. Elsevier, July 2013.

[16] A. Lakehal and I. Parissis. Lustructu: A Tool for the Automatic Coverage Assessment of Lustre Programs. In *International Symposium on Software Reliability Engineering*, pages 10–pp. IEEE, 2005.

[17] R. Marinescu and E. P. Enoiu. Extending east-adl for modeling and analysis of system?s resource-usage. In *IEEE 36th Annual Computer Software and Applications Conference Workshops (COMPSACW)*. IEEE Computer Society Press, July 2012.

[18] S. Rayadurgam and M. Heimdahl. Generating MC/DC Adequate Test Sequences Through Model Checking. In *NASA Goddard Software Engineering Workshop Proceedings*, pages 91–96. IEEE, 2003.

[19] S. Rayadurgam and M. P. Heimdahl. Coverage Based Test-Case Generation using Model Checkers. In *International Conference and Workshop on the Engineering of Computer Based Systems*, pages 83–91. IEEE, 2001.

[20] S. Richter and J. Wittig. Verification and Validation Process for Safety IC Systems. In *Nuclear Plant Journal*, volume 21, pages 36–36. EQES, Inc., 2003.

[21] D. Soliman, K. Thramboulidis, and G. Frey. Function Block Diagram to UPPAAL Timed Automata Transformation Based on Formal Models. *Information Control Problems in Manufacturing*, 14(1):1653–1659, 2012.

[22] S. Stancescu, L. Neagoe, R. Marinescu, and E. P. Enoiu. Uml modeling in design of error detection and correction circuits. In *In Proceedings of the 32rd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO09) IEEE Croatia*, May 2009.

[23] S. Stancescu, L. Neagoe, R. Marinescu, and E. P. Enoiu. A sysml model for code correction and detection systems. In *In Proceedings of the 33rd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO10) IEEE Croatia Section,*, May 2010.