Data Flow Model Coverage Analysis: Principles and Practice

Jean-Louis Camus, Carole Haudebourg, Marc Schlickling, Joerg Barrho

Data Flow Model Coverage Analysis: Principles and Practice

Authors: Jean-Louis CAMUS, Carole Haudebourg (Esterel Technologies), Marc Schlickling (Konzept Informationssysteme GmbH), Jörg Barrho (MTU Friedrichshafen GmbH).

Topic Processes, methods and tools

Abstract

Safety critical software requires rigorous processes in order to achieve a high degree of integrity. These processes include so-called "verification of verification". In the case of Model Based Development and Verification, DO-178C/DO-331 requires model coverage analysis. This paper reminds the objectives of model coverage analysis and the difference with structural code coverage analysis. It proposes coverage criteria suited to data flow models. These criteria are a generalization of the functional masking effect and also take into account modularity and in-lining of the model operators. It presents a tool supporting model coverage analysis according to these criteria. It concludes with industry field experience and future extensions.

Keywords Coverage, Model, Data flow, Tool, Qualification, Safety, Embedded, Software, DO-178C/ED-12C, DO-331, DO-330, EN 50128, IEC 61508, IEC 60880, ISO 26262.

1 Regulatory Context

Safety critical software requires a high degree of rigor in its development and verification processes. These processes are regulated by standards such as DO-178C/ED-12C [1] for airborne software, EN 50128 [2] for railway equipment, IEC 61508 [3] for industry, ISO 26262 [4] for automotive, ECSS (in particular Q80, E40) [5] for European space and IEC 60880 [6] for the nuclear industry. These standards are based on process assurance, with a strong emphasis on verification activities: reviews, analyzes, and testing.

These processes include so-called "verification of verification". In the case of Model Based Development and Verification, DO-331 [7] requires model coverage analysis of so-called "design models", i.e. models describing Low-Level Requirements. This paper reminds the objectives of model coverage analysis and the difference with structural code coverage analysis.

1.1 Role of Testing and Coverage Analysis in the Safety Standards

In addition to review and analysis, safety standards require dynamic verification. Simulation can be used as a means for verifying compliance of models to their higher-level requirements and algorithm accuracy. Testing is used for verification of the executable object code.

Safety standards require various types of coverage analysis. For instance DO-178C requires the following categories of coverage analysis:

- High-Level Requirements Coverage analysis (6.4.4.1), with full coverage required by table A-7 objective 3.
- Low-Level Requirements Coverage analysis (6.4.4.1), with full coverage required by table A-7 objective 4. In case of model-based development, the Low-Level Requirements are contained in a model and LLR coverage is model coverage (DO-331 MB 6.4.7).
- Structural [Code] Coverage analysis (6.4.4.2), with full coverage required by table A-7 objectives 5, 6, 7 and 8.

Several aspects of coverage analysis have to be emphasized:

1) Coverage analysis in general (requirements coverage and structural coverage) is not a goal per se, it is a means for verifying that verification activities are thorough and complete ("verification of verification").

2) One should apply requirements-based testing assessed with structural coverage analysis, and not structural testing. As explained by DO-248C [8] FAQ#44, "*Whereas structural testing is the process of exercising software with test scenarios written from the Source Code, DO-178C/DO-278A section 6.4.4.2 explains that structural coverage analysis determines which code structure, including interfaces between components, was not exercised by the requirements-based test procedures…. Since the starting point for developing structural test cases is the code itself, there is no way of finding requirements (high-level, low-level, or derived) not implemented in the code through structural tests. It is a natural tendency to consider outputs of the actual code (which is de facto the reference for structural testing) as the expected results. This bias is avoided when expected outputs of a tested piece of code are determined by analysis of the requirements.* "

1.2 Objectives of Model Coverage Analysis

1.2.1 Objectives of Model Coverage Analysis

As stated in DO-331 6.7, model coverage analysis determines which requirements expressed by the model were not exercised by verification based on the requirements from which the model was developed. Model coverage analysis is a means to assess thoroughness of the model verification activities. Model coverage gaps may reveal:

a) Shortcomings in requirements-based verification cases or procedures
b) Inadequacies or shortcomings in requirements from which the Model was developed
c) Derived requirements expressed by the Model
d) Deactivated functionality expressed by the Model
e) Unintended functionality expressed by a Model

1.2.2 Difference with Structural Coverage Analysis

Model coverage analysis should not be confounded with structural [code] coverage analysis. Model coverage analysis concerns the Low-Level Requirements expressed by a model, and is at a higher level of abstraction than structural coverage analysis. Model coverage analysis concerns the functional aspects of the model regardless of the code that implements this model, which structure is highly dependent on the coding technique. Also, the code coverage criteria that are required by standards and supported by tools are limited to control structures and Boolean expressions: they do not address very important functional aspects such as the definition and use of data.

1.2.3 Is Model Coverage Analysis a New Burden?

Model coverage analysis is not an additional burden introduced by DO-331, it is just the appropriate term for LLR coverage analysis in case of model-based development and verification (MBDV). LLR coverage analysis has always been required by DO-178B, whether one uses traditional development or MBDV. For years, a number of applicants using MBDV used to assess coverage of their LLRs by manual means such as Excel tables or colored pencil on model diagrams, or using specific tools such as pioneering versions of SCADE MTC or Simulink.

2 Challenge of Defining Appropriate Model Coverage Criteria

2.1 The Scade Language and the SCADE Suite Environment

Let us introduce the context, which is the Scade language and its supporting environment. The Scade language [9] is a synchronous data flow language [10] for reactive software. Its formal semantic foundation is the Lustre language [11], extended with state machines and iterators [12]. Scade is fundamentally a **declarative** language, which is not based on the notion of execution. A Scade model is a set of non-ordered **equations** combining flows/variables subject to formal clocks (i.e. conditions under which are defined). It is not a sequence of executable statements and it is the job of the code generator to transform purely declarative sets of equations into imperative statements.

The Scade language supports a free combination of notations that are familiar to control engineers:

- Block diagrams to specify the control algorithms (control laws, filters)
- State machines to specify modes and transitions in an application (e.g., taking off, landing) with parallelism, hierarchy and preemption
- Flowchart-like constructs (called clocked blocks)

The basic building block in Scade is called an operator. An operator is either a pre-defined operator (e.g., +, delay) or a user-defined operator that is built with predefined or user-defined operators. It is thus possible to build structured complex applications with high modularity.

Elemnt	Textual Form	Graphical Form
Formal interface	`node IntegrFwd(U: real ; hidden TimeCycle: real)` `returns (Y: real) ;`	

Local variables declarations	`var`
	`delta : real ;`
	`last_y : real;`	

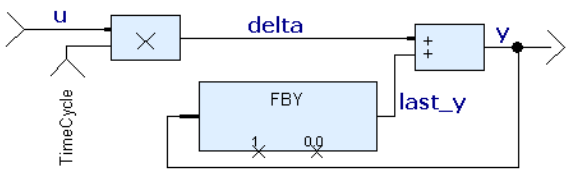
Equations	`delta = u * TimeCycle ;`
	`y = delta + last_y ;`
	`last_y = fby(y , 1 , 0.0) ;`
		

Figure 1 Integrator in Graphical Scade Notation

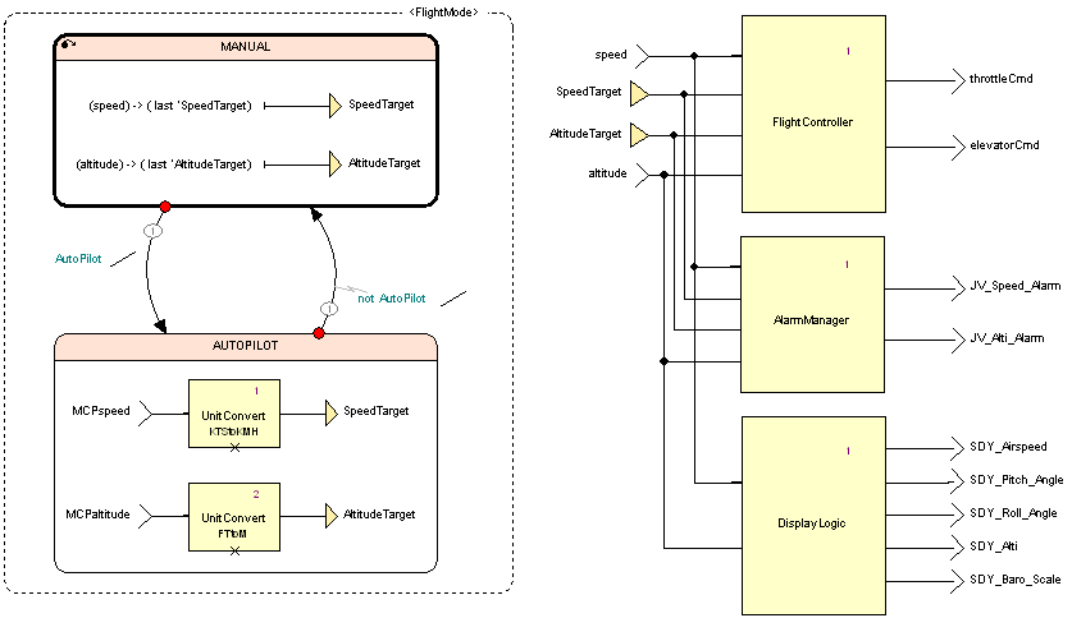


Figure 2: State Machine Combined with Data Flow

SCADE Suite (Safety Critical Application Development Environment) support the development and verification of safety critical software based on the Scade language with:

- Structured graphical editing
- Semantic checks
- Simulation and model coverage analysis
- Formal verification
- Automated code generation qualified for DO-178 level A and IEC 61508/EN 50128 SIL 3-4
- Automated test execution

2.2 Is there a Standard Definition of Model Coverage Criteria?

When writing DO-331, the SC205 SG-4 (subgroup defining the Model Based Development and Verification supplement) has taken into account the fact that there is a large diversity of modeling notations that differ significantly regarding:

- Degree of formality: modeling notations range from non-formal (e.g. UML) to formal (e.g. Scade, SDL).
- They may be based on various concepts and representations such as data flow, state machines, sequence charts.
- They may be based on a synchronous basis (e.g. Scade) and/or an asynchronous basis (e.g. SDL, UML).

And there may also be other notations in the future that are currently unknown.

The committee agreed that it was not possible to impose specific model coverage criteria in DO-331, and the following approach was retained:

1) DO-331 provides general principles for model coverage criteria (e.g. coverage of transitions in state machines) and requires model coverage criteria to be defined in each project's software verification plan.
2) Each applicant project defines in its SVP the model coverage criteria that it will apply.

Thus, it was necessary to define relevant model coverage criteria for the Scade language.

2.3 Are Structural Code Coverage Criteria Suited to Data Flow Model Coverage?

Analysis of classical imperative code has been investigated and applied for decades. This type of analysis is supported by a number of tools. The most commonly used code coverage criteria are Statement Coverage, various forms of Branch Coverage, Decision Coverage (DC), Modified Decision Coverage (MC/DC) [13]. Note that statement coverage and branch coverage address the control structure, whilst DC and MC/DC are actually specific data flow criteria for Boolean expressions (a decision is NOT a branch as recalled by [14].

The first natural approach is to consider traditional structural coverage criteria as candidates for the basis of model coverage criteria. But this approach would not work with Scade for the following reasons:

- It would be highly implementation-dependent
- It would not capture the essence of the functional flows expressed at the model level
- The traditional criteria are based on execution concepts for imperative languages, whilst Scade is a declarative language, not based on the notion of execution (it is the job of the code generator to transform purely declarative sets of equations into imperative statements).

3 Data Flow Coverage Criteria for the Scade Language

3.1 Principles of the Model Coverage Criteria Defined for the Scade Language

The principles of the proposed model coverage criteria are based on two related considerations:

- Good testing practice: an experienced tester develops requirement-based tests checking that the effect of input data values on observed data complies with the requirements. In order to be under appropriate controllability/observability situations, the tester sets the software in conditions where this influence occurs for each concerned data of interest. The proposed model coverage criteria assess that these conditions are met during testing.
- Theoretical basis: a synthesis of *Definition-Use* analysis is provided in [15]. More specifically, [16] proposes coverage criteria for the Lustre language (on which Scade is based). In this paper, we have chosen a definition sharing with it some aspects of path activity. In addition it consistently takes into account masking MC/DC, Arithmetic, Structured data handling, Iterators, Reset, Clocked blocks and State machines.

A data path from one point of the data flow graph to another one is active when its Activation Condition (AC) is true. For instance on Figure 3, B influences Z when C2 is true and C1 is false.

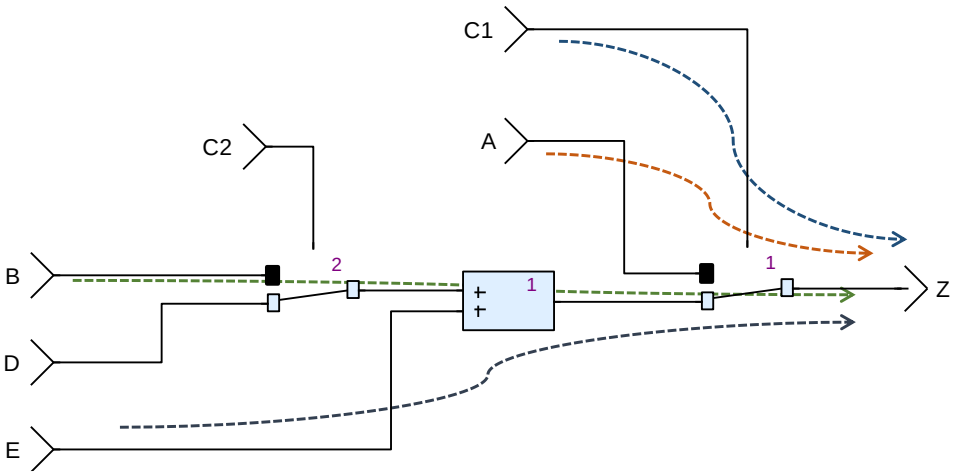


Figure 3: Example 1 Data Flow Activity and Influence

Several coverage criteria all based on this activity concept are defined, as summarized in Table 1.

Table 1: Coverage Criteria Summary

Coverage Criterion	Synopsis
Basic Coverage (BC)	Every element has been on an active path.
Decision Coverage (DC)	Every Boolean expression outcome has toggled while on an active path.
Modified Condition /Decision Coverage (MC/DC)	Every condition of every Boolean expression's has toggled while on an active path (shows independent effect of the conditions).
Control coupling	Every state/transition, clocked block, activate branch has been active
Data coupling	Every input flow to every user operator instance has changed value.
Operator feature	Every characteristic feature (e.g. saturation, reset) has been active.

In order to match industrial practice regarding code coverage analysis, the current approach considers the flows occurring in the current instant (i.e. memories are part of the inputs/outputs).

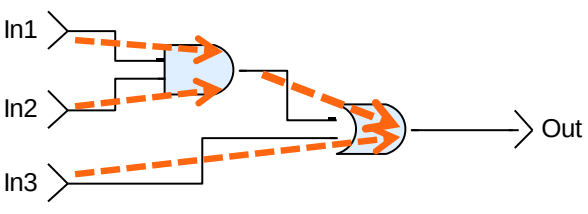
3.2 Application of the Influence Principle to Logic and State Machines

The principles presented above are very general. In the case of Boolean expressions, the activation condition is actually the negation of the masking effect which concerns any level of abstraction (requirements, code, or hardware) as explained by [13]. On example 2, the influence of In1 is masked if its activation condition (In2 and not In3) is not satisfied. Note that our definition, matches masking MC/DC as defined in [17] and [13]. This differs from [16]. Indeed, for X= Y and Z, the AC of the path Y to X is defined as Z in our definition: whereas it is defined as (not Y and Z) in [16].

PATH1 {In1, Out}

PATH2 {In2, Out}

PATH3 {In3, Out}



,

Figure 4: Example 2 Masking and Activation Condition in a Boolean Expression

The masking effect is also taken into account for coverage analysis of higher-level constructs such as state machines, where the influence of data depends on conditions related to the state, flows from/to states and transitions guards and priorities.

3.3 Position with Respect to Structural Coverage

Let us take example 1 to show the difference with structural coverage. The code that implements this model part may typically be implemented in three ways, shown in Table 2 in the form of pseudo-code.

Table 2: Implementation of Example 1

Implementation A	Implementation B	Implementation C

`X = if_block(C2,B,D);`
 	
`Y = plus_block(X, E);`
 	
`Z = if_block(C1,A,Y);`
	`if (C2) {`
	`  X = B;`
	`}`
	`else {`
	`  X = D;`
	`}`
	`Y = X + E;`
	
	`if (C1) {`
	`  Z = A;`
	`}`
	`else {`
	`  Z = Y;`
	`}`
		`if (C1) {`
		`  Z = A;`
		`}`
		`else {`
		`  if (C2) {`
		`  tmp = B;`
		`  }`
		`  else {`
		`  tmp = D;`
		`  }`
		`  Z = tmp + E;`
		`}`

Implementation A can be structurally covered just with just one execution cycle; it is obvious that such a test case does not cover the LLRs expressed in this model and is very poor. Implementation B can be covered just with the following set of test cases for C1/C2= {T/T, F/F}, which does not test the effect of B on Z.

The proposed model coverage criteria (even the most basic one) require at least C1/C2= {T/x, F/F, F/T} (where x stands for don't care) in order to activate the effect of A, B, D and E on Z; more specifically, C1/C2= {F/F,F/T} shows the effect of B on Z . Note that this is also what is required for implementation C.

One can see that even on a very simple example, the proposed model coverage criteria capture the functional activation at model level regardless of the code that will implement it, whereas structural coverage, which is limited to control structures does not address functional flows aspects. This difference is of course amplified when analyzing more complex modeling constructs, such as complex data flow combinations or state machines.

In [18] it is shown that there is a subsumption (a kind of implication) hierarchy between data flow coverage and statement coverage. Intuitively, one understands that in order to activate a Definition-Use data flow, one needs to execute the related imperative statements in the generated C code, whereas execution of statements does not guarantee Definition-Use activation, as shown in the implementations A and B of Table 2. It has been verified in practice for complex models that tests covering the model also cover the code generated from that model, except few systematic cases which are predictable and justifiable. Formal demonstration of implication of code coverage by model coverage require some additional refinement of model coverage criteria (see conclusion of this paper).

3.4 Handling Structured Models

A Scade operator can be considered either as a black box or just as a syntactical/graphical device (it is said to be expanded or in-lined). This choice (which can expressed at model level as well as at code generation time) is a design choice. It is reflected by the code generator both in terms of causality analysis (i.e. any data shall be produced before being consumed) and in the architecture of the generated code: a non-expanded operator is implemented as a C function, whereas an expanded operator is in-lined as part of the function corresponding to the nearest non-expanded parent.

There are a number of possible approaches for handling such hierarchies such as:

a) Fully expand all operators of the model: this ideal analysis is supported by SCADE, and works for small models, but is impractical for large models both regarding the testing effort and machine resources.
b) Expand no operator at all, and consider each of them as pure black boxes. This was supported by historical versions of SCADE.
c) Selectively expand at model level consistently with the expansion selected for code generation and abstract non-expanded operators by considering their interfaces as atomic. This is supported by SCADE R16 and is the most typical use case.
d) The definition provided in [19] proposes a definition that abstracts non-expanded operators with a fine-grained dependency between their inputs/outputs.
e) Abstract specific operators with user-defined criteria. For instance for the integrator shown on Figure 5 , each instance of the integrator can be abstracted from its caller perspective as four objectives to cover: Reset, Normal integration, Low saturation and High Saturation. This analysis, supported by SCADE, is well suited to library operators.

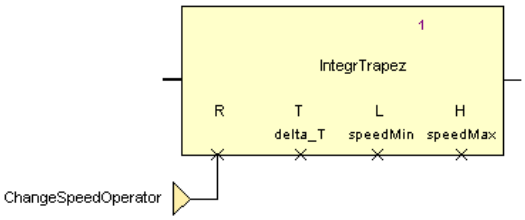


Figure 5: Example 4 Integrator Library Component

In the case of example 3, if node NASA_Example1 is not expanded, there are two Boolean expressions with 3 non-independent conditions each, if it is expanded there is one single expression with 5 conditions.

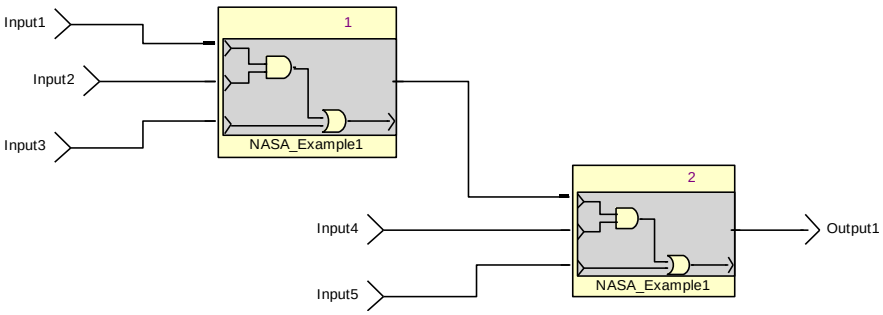


Figure 6: Example 3 Cascaded Operators with Boolean Expressions

4 Tool Support Overview

A toolchain called MTC (for Model Test Coverage) version 6.4.7 supporting of the above described criteria has been developed. Using appropriate observers, it runs test cases and collects model coverage information during simulation. The coverage results can be displayed in graphical form (e.g. Figure 7) or textual form. For applicants wanting to provide direct evidence of structural coverage, there is also a feature supporting direct measurement of structural code coverage of the source code that will be embedded, under execution of the same test cases as those used for model coverage analysis and with the corresponding coverage criteria (DC, MC/DC). The toolset has been qualified according to Tool Qualification Level TQL-5 of DO-330 [20].

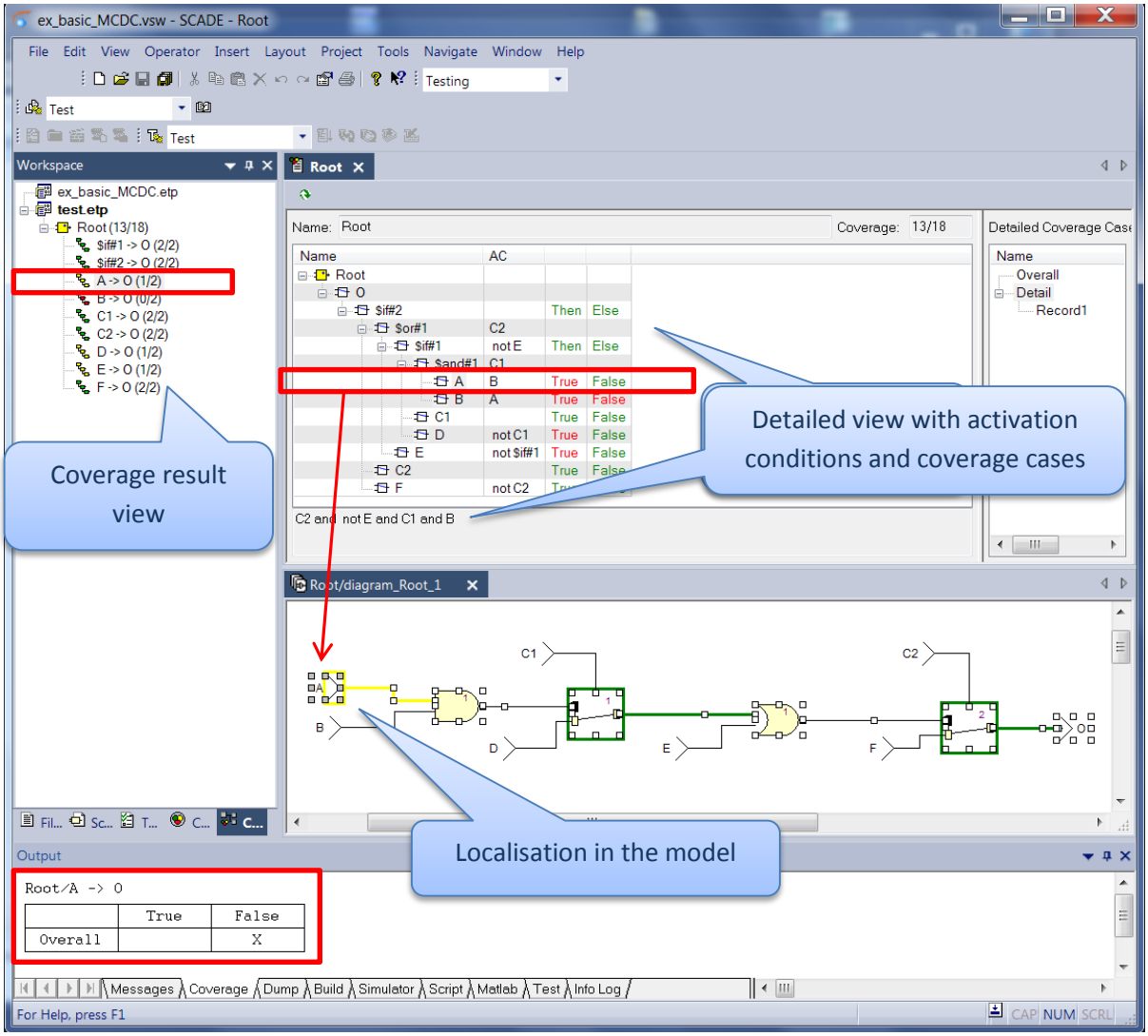


Figure 7: Model Coverage Browser

5 Industrial Return of Experience

Model coverage analysis tools based on simpler criteria have been used since 2005 on many industry projects. The first versions supporting the new criteria described in this paper have been released by mid-2014 and used recently in several industrial projects in avionics and nuclear industry. The current version is MTC 6.4.7a.

5.1 MTU Experience Example

MTU develops emergency diesel engines for nuclear power plants. This function is very critical in case of failure of the electrical network. MTU develops the digital engine control unit for such engines. This unit shall in particular maintain safe state requested by user. The applicable standard are IEC60880 and IEC61508:2010 (SIL 3 for the software).

The previous experience of MTU about coverage concerns:

- Structural coverage with Cantata++
- Scade coverage with earlier versions of MTC:

The new MTC has been in use for more than a year (but not continuously). The main conclusion are the following:

- There is short learning time,
- It allows inspection of missing activation conditions and paths,

- There are potential improvement in the presentation of missing activations, since the reason of missing activations is sometimes hard to understand.

5.2 General Return of Experience

The new MTC has been used on complex airborne software systems (FADEC, flight control sub systems). The main feedback is the following:

- Model coverage analysis supports detection of insufficient testing and dead or deactivated model elements. The detection can be done early in the software lifecycle, which reduces cost and time for software development.
- There is a focus shift from code level to model level both for design and coverage analysis.
- There is a paradigm change from the traditional imperative mindset (statements, branching) to a data flow mindset (definition/use), which is more efficient but requires appropriate user training.
- The tools were able to handle complex software (3000 user operators, 100 000 model coverage points, 50 000 code coverage points). The ability to handle efficiently incremental software changes will be improved. The user interface will be improved for more comfortable navigation in large data sets.
- It is recommended to perform model coverage analysis early in the lifecycle, in order to detect and fix dead and/or non-optimal model elements and take full benefit of model coverage analysis.

6 Conclusion and Future Work

Model coverage analysis was a pioneering approach 10 years ago. It is now explicitly required and/or recognized as an alternate means for checking testing activities completeness in domains such as avionics (DO-331) or automotive (ISO 26262). The combination of the Scade language and of SCADE tools with appropriate model coverage criteria described in this paper allows software engineers to consistently shift their focus from code level to model level. Usage on large industry projects showed that it is an efficient means for detecting insufficient testing and dead or deactivated model elements.

Future work will provide additional benefits. For instance further refinements for addressing numeric aspects would allow performing analysis of singular points. Handling delays such as in [21] would allow assessment of sequential logic. Providing formal evidence of model to code coverage implication would allow application of DO-331 FAQ#11, definitely eliminating need to handle double-check of structural coverage.

7 References

[1] DO-178C, "Software Considerations in Airborne Systems and Equipment Certification DO-178C," RTCA Inc, 2011.

[2] CENELEC, EN 50128 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems, 2011.

[3] IEC, IEC_61508 Functional safety of E/E/PE safety-related systems, IEC, 2010.

[4] ISO, ISO 26262 Road vehicles – Functional safety, IEC, 2012.

[5] ECSS, "Space product assurance Software product assurance, ECSS-Q-ST-80C," 2009.

[6] IEC, IEC 60880 Nuclear power plants Instrumentation and control systems important to safety Software aspects for computer-based systems performing category A functions, IEC, 2006.

[7] DO-331, "Model-Based Development and Verification Supplement to DO-178C and DO-278A DO-331," RTCA Inc., 2011.

[8] DO-248C, "Final report for clarification of DO-178C Software Considerations in Airborne Systems and Equipment Certification DO-248C," RTCA Inc., 2011.

[9] Esterel Technologies, Scade 6.0 Language Reference Manual KCG-SRS-007, 2014.

[10] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic and R. De Simone, "The Synchronous Languages 12 Years Later," *Proceedings of the IEEE,* vol. 91, no. 1, 2003.

[11] N. Halbwachs, P. Caspi, P. Raymond and D. Pilaud, "The synchronous dataflow programming language Lustre," *Proceedings of the IEEE,* vol. 79, no. 9, p. 1305–1320, September 1991.

[12] J.-L. Colaço, B. Pagano and M. Pouzet, "A Conservative Extension of Synchronous Data-flow with State Machines," in *ACM International Conference on Embedded Software (EMSOFT'05)*, 2005.

[13] J. J. Chilenski, "An Investigation of Three Forms of the Modified Condition Decision Coverage (MCDC) Criterion," FAA Tech Center Report DOT/FAA/AR-01/18, 2001.

[14] CAST, "Position Paper CAST-10 What is a "Decision" in Application of Modified Condition/Decision Coverage (MC/DC) and Decision Coverage (DC)?," Certification Authorities Software Team, 2002.

[15] L. Clarke, A. Podgurski, D. J. Richardson and S. J. Zeil, "A Fomal Evaluation of Data Flow Path Selection Criteria," *IEEE Transactions On Software Engineering,* vol. Vol. 15, no. No. 11, November 1989.

[16] A. L. a. I. Parissis, "Structural test coverage criteria for Lustre programs," in *10th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, 2005.

[17] RTCA, "Software Considerations in Airborne Systems and Equipment Certification DO-178C," RTCA Inc, 2011.

[18] P. Frankl and E. Weyuker, "An Applicable Family of Data Flow Testing Criteria," *IEEE Transactions On Software Engineering,* vol. Vol. 14, no. 10, October 1988.

[19] V. Papailiopoulou, A. Rajan2 and I. Parissis, "Structural Test Coverage Criteria for Integration Testing of LUSTRE/SCADE Programs," in *FMICS'11 Proceedings of the 16th international conference on Formal methods for industrial critical systems*, 2011.

[20] DO-330, "Software Tool Qualification Considerations DO-330," RTCA Inc., 2011.

[21] M. Whalen, G. Gay, D. You, M. P. Heimdahl and M. Staats, "Observable Modified Condition/Decision Coverage," in *ICSE 2013*, San Francisco, CA, USA, 2013.