# Formal Verification of Critical Aerospace Software

Virginie Wiels, Robert Delmas, David Doose, Pierre-Loïc Garoche, J. Cazin, Guy Durrieu

**V. Wiels, R. Delmas,**
**D. Doose, P.-L. Garoche,**
**J. Cazin, G. Durrieu**
(Onera)

E-mail: virginie.wiels@onera.fr

# Formal Verification of Critical Aerospace Software

Embedded software is implementing more and more functions in aerospace, including critical ones. Model Driven Engineering has changed software life cycle development by introducing models in the early steps of software development. Verification and validation is essential, at model and at code levels, and still mostly done by simulation and test. However, formal methods, which are based on the analysis of the program or software model, are being transferred to industry for verification of critical software. This paper presents the context of aerospace software development, a brief overview of formal methods and of the associated industrial practice, and the work done at Onera on formal verification of critical Aerospace software at model and at code levels. This work addresses four themes:

- Specifics of application of formal methods to aerospace;
- Model driven engineering at platform level;
- Cooperation of analysis techniques;
- Automating test using formal methods.

Each of these four themes is a research domain in itself; it is thus impossible in a single paper to provide a detailed state of the art and an exhaustive list of research challenges for each of them. This paper rather aims at giving a broad vision and at presenting work done at Onera in these domains.

## Aerospace context

Aerospace systems have experienced significant and continual evolution over the last 30 years. Digital technologies have become more mature, reliable and efficient and have been introduced inside aircraft for the implementation of more and more functions, including critical ones, as shown in figure 1.

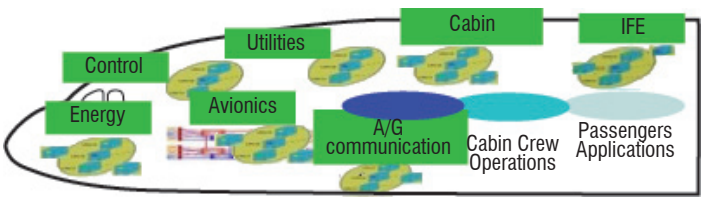


Figure 1 - Embedded functions

Software is essential in the implementation of these functions. For example, the avionics systems in the Airbus A380 include more than 100 millions lines of code. Figure 2 gives an idea of the evolution of the volume of embedded software.

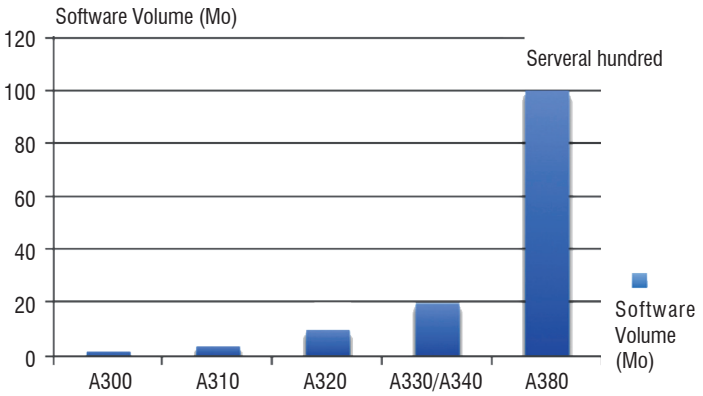


Figure 2 - Evolution of the volume of embedded software

The same thing is happening in the space domain; the number of embedded computers is still much smaller than in aeronautics, but it is growing and will continue to do so. In addition to control functions, satellites and spacecrafts now embed complex management functions (mode management, load, communication, etc). Some functions are critical, such as the one responsible for docking to the space station in the ATV (Automated Transfer Vehicle).

### Software engineering

In this context, special attention must be given to software engineering in order to master software complexity and ensure the correct execution of software executing critical functions. Classically, software development is decomposed into several steps: requirements, design, coding and verification and validation (essentially by test, unit test and integration test), as represented on figure 3.a.

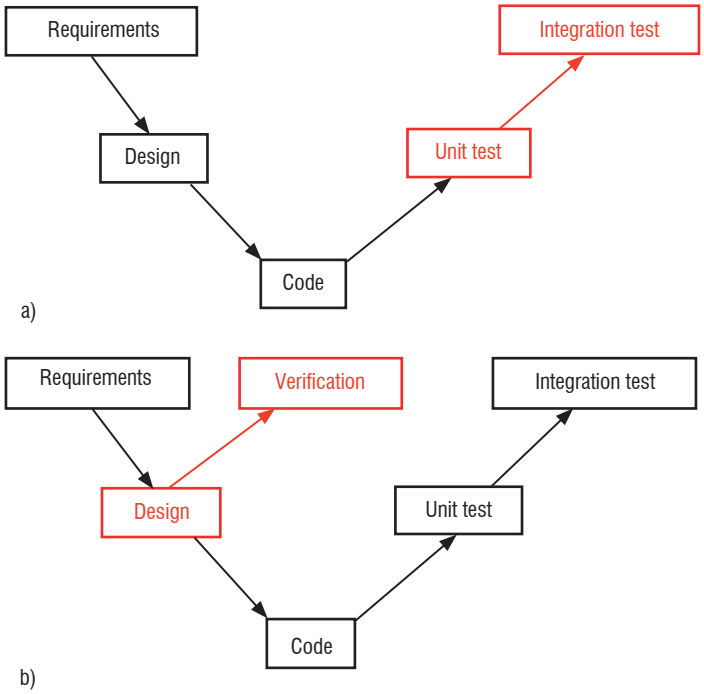


Figure 3 - a) Classical software development cycle / b) MDE software development cycle

Verification and validation is an essential step for critical software and represents more than 50% of software development costs. Another important rule is that the later an error is detected, the more costly it is to correct it [1].

This is one of the main reasons for the advent of Model Driven Engineering (MDE) in software development. In traditional software development, the reference is the code. In MDE, the reference is a model that is developed before the code (and from which the code can in some cases be automatically generated). This model can be simulated and thus verified against requirements. If the code generator is qualified, unit test can be removed and replaced by verification on the model.

### Verification and validation means

Software verification and validation is still mainly achieved by means of simulation and testing. However, these methods are not exhaustive, and still very labor-intensive and costly. They may also reach the limits for specific verification objectives. Alternative techniques exist and are being transferred: formal methods [2]. These methods do not require execution of the software and are based on mathematical analysis of the code or the model. They have the advantage of being automated and exhaustive. Onera has chosen formal methods for the verification of critical aerospace software.

## Formal methods

A formal method is defined by a formal notation together with a formal analysis technique (definition taken from [3]). A notation is formal if it has non-ambiguous, mathematically defined syntax and semantics. The formal analysis technique then allows automated computation of properties of systems modeled using the formal notation.

### Brief overview of existing techniques

The first work on formal methods dates back to the 60's with Hoare logic, to prove the correctness of programs [8], [9]. Many different formal notations and analysis techniques have been defined. Our goal here is not to give a detailed state of the art report, we refer the reader to the formal methods wiki (http://www.formalmethods.wikia.com) which gives many references. Formal notations can be broken down into programming languages, formal modeling languages that are usually dedicated to a specific type of application (such as synchronous languages for reactive systems) and formal notations for the expression of properties (such as temporal logics).

Analysis techniques are typically presented under three categories [3]: model checking, deductive techniques and abstract interpretation, even if the borders between techniques are not as strict as they used to be. Model checking explores all possible behaviors of a formal model to determine whether a specified property is satisfied. Deductive methods involve mathematical arguments, such as mathematical proofs, to establish a specified property of a formal model. Abstract interpretation is a theory for formally constructing conservative representations of the semantics of programming languages.

### State of industrial practice

Formal models are now used in different application domains [4], [6] e.g. the SCADE language is used for the design of command and control systems in aerospace. Automated code generation from models is also common. There are an increasing number of industrial experiments on the application of formal analysis techniques to the verification of software. Model checking is beginning to be used operationally in the railway domain, where correct-by-construction approaches based on B have been used for several years now e.g. for the development of the control software for the Paris metro [7]. Certification credits for the use of formal methods in aeronautics have been obtained by Airbus for the A380 software [13], [14]. Rockwell Collins has been using model checking for the validation of requirements [12], [11]. Early on, NASA was investigating the use of formal methods for the certification of critical systems [5] and has several research teams working on formal verification (Robust Software engineering at Ames, Langley; Formal methods, Laboratory for Reliable Software at JPL).

## Onera work

Onera work in formal verification of software has been around four themes:

- Specifics of application of formal methods to Aerospace (methodological work to integrate these new verification means into industrial processes while taking into account certification constraints);

- Model driven engineering at platform level;
- Cooperation of analysis techniques;
- Test automation using formal methods.

## Specifics of application of formal methods to Aerospace

Certification is a structuring constraint in aeronautics. The certification standard for software is DO-178B/ED-12B. This standard does not prescribe a specific development process but identifies four processes in the software development cycle: the requirement process producing High Level Requirements (HLR) from System Requirements; the Design process producing Low Level Requirements (LLR) and software architecture from HLR; the coding process producing source code from software architecture and LLR; the integration process producing executable object code from source code. DO-178B identifies verification objectives for each of the four processes, as synthesized on figure 4.

Version B of DO-178 was released in 1992 and software development has changed significantly since then. So EUROCAE and RTCA decided to prepare an update of the standard in 2005. DO-178C was released in December 2011 and proposes, in addition to the core document, four technical supplements addressing qualification of tools, object-oriented technologies, model-based design and formal methods. Onera has been active in the writing of the formal methods supplement [3]. This document explains how formal methods can be used for the verification of certified software and proposes adapted verification objectives taking into account the differences with classical verification techniques such as testing. A summary presentation of this supplement can be found in [15].

This work on certification standards is essential to be able to transfer formal methods to industrial practice, as is the work on methods that Onera is doing in collaboration with industrial partners such as Airbus. Having an appropriate formal notation and an efficient analysis technique is necessary but absolutely not sufficient to be able to take advantage of all the benefits of these new verification methods. Methodological work is mandatory and includes the identification of the requirements that can be verified formally, their formalization and the definition of specific verification strategies for the software in question. Onera has been working with Airbus on these aspects at both model and code levels [17], [10], [13].

This methodological work may reveal needs for new techniques, to complement formal analysis. For example, when working on the definition of a method to apply model checking to SCADE models, we noted that a lot of time was spent on the understanding of the counter-examples provided by the model checker. We proposed an automated technique and an associated tool to extract from the counter examples the meaningful information and present it to the user in a comprehensible way [16].

## Model Driven Engineering at platform level

The anticipated spectrum of applications of model driven engineering in aerospace embedded systems design is not restricted to critical flight control software. Indeed, MDE approaches can be used at different conceptual design levels, from aircraft level to platform level down to software components level.

Onera has ongoing projects to support the design of embedded platforms through model driven approaches. In their current state, platform models are dedicated to the study of performance or dysfunctional aspects (or even other non-functional aspects such as electrical consumption or weight). Models help in the formalizing of the different components and architectural concepts around which the execution platform is built, as well as attributes according to which the performances and safety of the platform are assessed. In such models the properties capturing resource allocation requirements and safety related requirements can be formally expressed and automatic tools such as SAT, Pseudo-Boolean or SMT solvers can be used either to verify manual designs or to generate correct-by-construction resource allocations or platform configurations.

In addition, model based specifications can valuably subsume natural language specifications, not only in the design process, but also in the procurement process between contractors and sub-contractors. Models may in the near future become critical assets on which the whole system design process will rely.

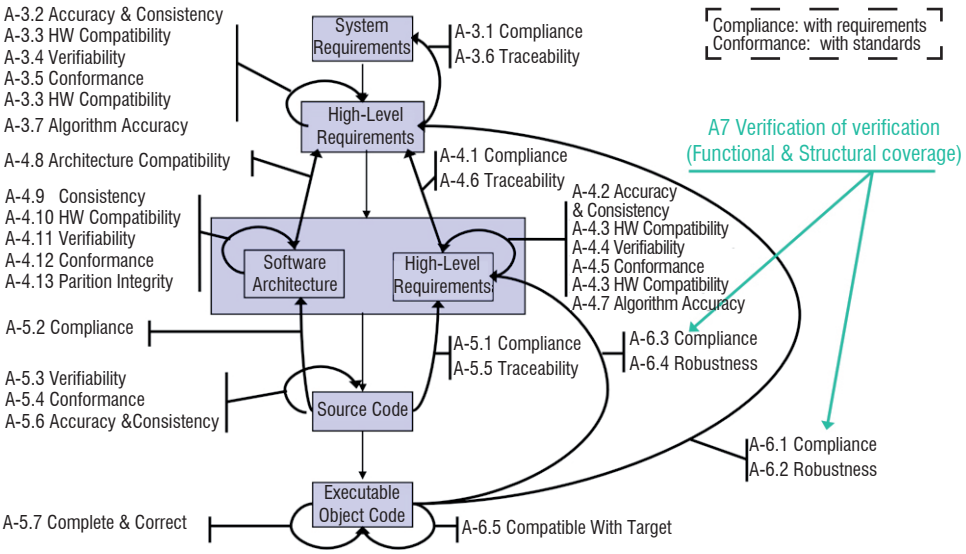


Figure 4 - DO-178B verification objectives.

A remarkable indication of this trend is the Topcased software suite [www.topcased.org], an open-source, Eclipse-based Integrated Development Environment dedicated to embedded systems applications. Onera was involved in the original Topcased project, and is now involved in its successor project OPEES [www.opees.org]. Onera contributions to OPEES consist in providing a theoretical framework, tools and associated methodologies to enable formal verification and automatic model synthesis for users of MDE processes.

### Meta-models and semantics

Meta modeling proceeds by defining class diagrams (e.g. UML, Ecore) and annotating the diagrams with declarative constraints (e.g. OCL) to formalize the invariants of the class diagrams. Such meta-models usually describe the category of acceptable systems or system configurations. Once the meta-model is obtained an instance of the meta-model can be created (which would correspond to a particular system or system configuration). Two questions naturally arise in this modeling process. Firstly, a validation problem: does the meta-model adequately capture all informal requirements without error, ambiguity or redundancy? Secondly, a verification problem: are instances correct with respect to the meta-model and its constraints? The following sections give a quick overview of the state of the art of existing validation and verification techniques, and finally introduce the research objectives and current work at Onera.

### Validation

At the meta-level, generic properties can be investigated, of which the following two are the most important:

- Constraints consistency: the absence of logical contradiction between constraints that would entail an empty set of instances of the meta-model;
- Constraint non-redundancy: there should exist instances which show that each constraint can be satisfied/falsified independently of the others, proving that each constraint adds its own independent semantic contribution to the meta-model.

Tool support is available for such validation tasks: UMLtoCSP [20], UML2Alloy [19], KodKod [18], etc. which all employ constraint satisfaction techniques to generate witnesses of the above properties. Yet, the integration of these tools in major IDEs is sub-par and tools are not easy to use (incompatible data formats, etc).

### Verification

At the instance level, we can make sure an instance is structurally faithful to class diagram constraints (valid types and cardinalities for references). MDE technology allows the creation of instance editors from a DSL specification and creating instances using these editors prevents a lot of basic structural errors; furthermore, any remaining error can be detected using static checkers.

Secondly, we can verify that an instance satisfies all constraints (OCL) of the meta-model. Tools for OCL verification exist (Dresden, Topcased, etc.) and allow the discovery of violations. The performance of these tools is often no better than average (interpretation techniques vs. code generation techniques) and in some contexts there are difficulties in scaling up to real world applications.

### Research challenges

This recent state of the art report shows that a lot is already available for MDE users. Yet, two questions remain open today in MDE: how to assist the creation of large and correct-by-construction instances (think thousands of objects under complex constraints)? How to create instances which make sense from an engineering point of view?

These two questions have given birth to a research work on automatic instance synthesis from a metal-model with optimization of quantitative criteria. An Onera proposal is to use modern and robust combinatorial optimization techniques (Max-SAT, Pseudo-Boolean optimization, SMT solving or CSP solving) to synthesize instances of a meta-model which:

- Satisfy all hard constraints of the meta-model.
- Optimize user provided quantitative criteria.
- Can be used to extend existing (and possibly flawed) instances to valid instances.
- Help the users to explore the design-space by proposing alternative solutions to a single problem.
- Scale up to industrial applications.

### Model transformation for verification and for code generation

Onera is currently working on a generic model transformation framework to import and translate meta-models and instances to a portfolio of constraint satisfaction engines, as shown on figure 5.

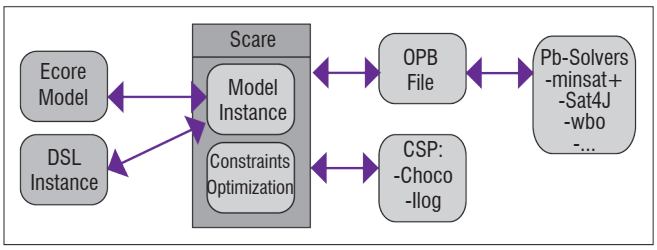


Figure 5 - SCARE framework

For more details about this technology and its possible uses, please consult the following references: [23], [22], [21].

## Cooperation of analysis techniques

While formal techniques are slowly appearing in industrial practice, the ecosystem of methods applicable to a system has yet to become more structured. For a specific description level, either at model level with synchronous data-flow languages (Lustre, Simulink, etc.), or at code level (C, Ada, etc.), a variety of formal analysis tools are available. Each of these tools targets a range of properties and, until now, the relationships between them has been only marginally investigated. A direction of research at Onera is focused on an effective combination of techniques to improve both the efficiency and spectrum of formal verification for embedded systems. Gathering different techniques into an integrated framework means the end-user has access a simpler view of what properties can or cannot be analyzed with respect to the available tool set. This research is being developed through two sub-themes with different partnerships.

### Theoretical combination framework and application at code level

We are doing this work in cooperation with INPT and Airbus. We propose a theoretical framework that gives us a general way to reason about a block of code annotated with a formal specification, typically a Hoare triple {Precondition} Sequence of Instructions {Postcondition}.

Most of the analysis techniques can be expressed either as forward analyses that over-approximate the reachable states of the code, or as backward analyses that under-approximate the co-reachable states of the code starting from a specific property. The formalization of these mechanisms (see figure 6) allows us to demonstrate how to implement the collaboration of techniques.

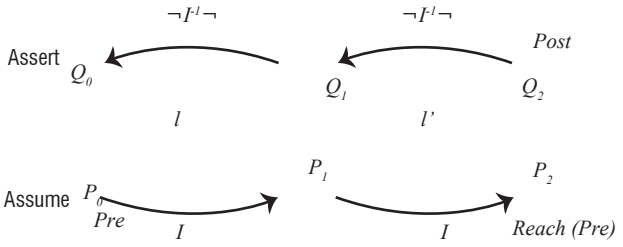


Figure 6 - Forward and backward analyses

Reinforcing the knowledge of an analysis by the intermediate information computed by the other allows a strengthening of the global results.

These ideas have been implemented on C code analysis tools, by combining an abstract interpreter, that over-approximates the collecting semantics (in a forward setting), and a weakest-precondition engine, under-approximating the behavior of the program with respect to the Post property [24].

### Combination of k-induction and abstract interpretation at model level

In cooperation with the University of Iowa and Rockwell Collins, this research focuses on the verification of safety properties on Lustre programs. SAT or SMT [25], [26] based verification approaches such as k-induction [27] give good results on programs with a mostly discrete state space (boolean, bounded integers). However, when numerical computations are involved (real/float computations) the formalization of the property to be proved often needs to be strengthened using auxiliary lemmas to make it inductive with respect to the system's transition relation. When attempted manually the discovery of such lemmas is time consuming and hinders the efficiency and scalability of formal verification. Automating lemma discovery hence appears crucial to allowing end-users to apply formal verification on industrial cases.

Our proposal materializes as a parallel analysis framework for Lustre programs in which each analysis can communicate its own intermediate information to the others: discovered invariants, potential invariants, counter-examples to the induction step of a proof, heuristics about relationship between state variables. The exchanged information is used to drive and tune the analysis of the different tools. Abstract interpretation is used to provide bounds on the state variables and to generalize counter-examples, while k-induction is used to validate any proposed invariants and to discharge the principal proof objective. The main components of the engine and their interactions are presented in figure 7.

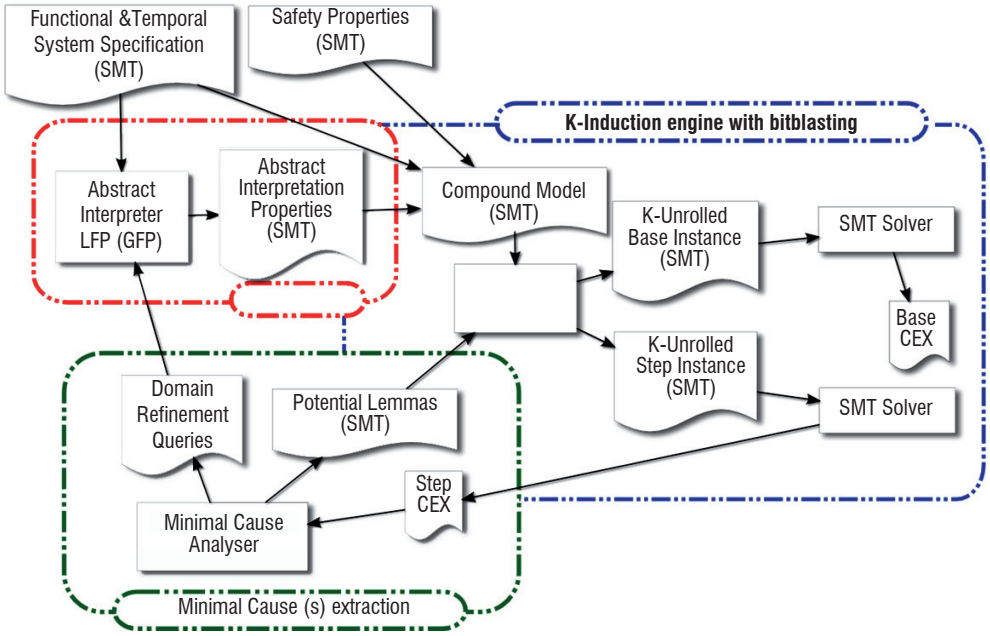


Figure 7 - Cooperation between model checking and abstract interpretation

### Test automation using formal methods

Formal verification will not replace all the tests but can help in automating the test process that is still often very manual. Test is composed of three activities: definition of test scenarios, execution of the scenarios and oracle (did the test fail or pass?)

A lot of work has been done on automated test generation from formal specifications [30], [31]. Test scenarios are generated from a formal model of the software requirement in order to test the correctness of the code with respect to the requirements. Automated test generation can also be used to test a design model with respect to requirements. Onera has worked with Airbus on test generation techniques and methods as well as on the definition of relevant structural coverage criteria for SCADE models [28], [29]. Special attention has to be given to the methods used in order to respect the constraints imposed by DO-178B that forbid the use of structural testing.

Onera has also worked on automation of the oracle for the test of SCADE models. SCADE models are tested against functional requirements (written in natural language) on specific simulation environments. Test scenarios are manually defined by testers, then they are executed on the simulator and finally the tester decides if the test results are correct with respect to the expected results specified in the requirements. The oracle can be done online, or offline by looking at the execution traces of the test. We have proposed a formal language for the expression of test objectives derived from the requirements and an analysis technique to automatically verify that the execution traces are correct with respect to the test objective. It has been applied on Airbus critical avionics software [32].

This work on trace analysis can be seen as a run-time verification approach. Run-time verification is a research field that proposes the use of formal methods in a lightweight fashion in order to verify properties on execution of the software (www.runtime-verification.org). Most of the contributions propose techniques to monitor the program while it is executing and detect violations of the properties; but a posteriori approaches also exist that verify the properties on the program traces [33].

## Future work

Future work in software formal verification at Onera is going to be focused on three themes.

1) We will continue to investigate the cooperation between different verification techniques, in order to improve efficiency of the verification and augment the spectrum of properties that can be addressed by formal methods.

2) We are going to work on the definition of an optimized verification process across the different development levels. This theme is broad and long term, we will conduct research on the following aspects in particular:

- use of invariants from continuous mathematical models to ease verification of software (cooperation with Georgia Tech);
- combination of static and dynamic analyses: test and formal methods will be used in future software verification processes, but research still needs to be done to find smart ways of combining them.

3) We plan to work on incremental techniques to take into account of software evolutions. In Aeronautics, successive versions of software are verified before being embedded in the aircraft, but also during maintenance. Defining efficient techniques for incremental verification would be very interesting in this context ■

**Acronyms**

MDE (Model Driven Engineering)
ATV (Automated Transfer Vehicle)
HLR (High Level Requirements)
LLR (Low Level Requirements)
SAT (Satisfiability)
SMT (Satisfiability Modulo Theories)
OPEES (Open Platform for the Engineering of Embedded Systems)
UML (Unified Modeling Language)
OCL (Object Constraint Language)
IDE (Integrated Development Environment)
CSP (Constraint Satisfaction Problems)

**References**


[1] *Software for Dependable Systems: Sufficient Evidence?* National Research Council (U.S.). Committee on Certifiably Dependable Software Systems, Daniel Jackson, Martyn Thomas, Lynette I. Millett editors, 2007.

[2] M. HINCHEY, J. P. BOWEN, E. VASSEV - *Formal methods*. Philip A. Laplante (ed.), Encyclopedia of Software Engineering, Taylor & Francis, 2010,pp. 308-320.

[3] Formal Methods Supplement to DO-178C and DO-278A. RTCA, 2011

[4] J. RUSHBY - *Automated Formal Methods Enter the Mainstream*. Communications of the Computer Society of India, Vol. 31, No. 2, May 2007, pp. 28-32. Formal Methods Special Theme Issue.

[5] J. RUSHBY - *Formal Methods and their Role in the Certification of Critical Systems*. Technical Report CSL-95-1, March 1995. Also available as NASA Contractor Report 4673, August 1995, and issued as part of the FAA Digital Systems Validation Handbook (the guide for aircraft certification)

[6] Y. A. AMEUR, F. BONIOL, V. WIELS - *Toward a wider use of formal methods for aerospace systems design and verification*. International Journal on Software Tools for Technology Transfer (STTT). Volume 12, issue 1, 2010

[7] P. DEMM, P. DESFORGES, M. MEYNADIER - *An industrial success in formal development.* Proceedings of FM99. LNCS1708

[8] C. A. R. HOARE - *An axiomatic basis for computer programming*. Communications of the ACM, 12(10): 576–580,583 October 1969

[9] R. W. FLOYD - *Assigning meanings to programs*. Proceedings of the American Mathematical Society Symposia on Applied Mathematics. Vol. 19, pp. 19–31. 1967

[10] T. BOCHOT, P. VIRELIZIER, H. WAESELYNCK, V. WIELS - *Model Checking Flight Control Systems: the Airbus Experience*. ICSE companion, 2009

[11] S. MILLER, M. WHALEN, D. COFER - *Software Model Checking Takes off*. Communications of the ACM, Volume 53, N° 2. 2010

[12] S. MILLER, A. TRIBBLE, M. WHALEN, M. HEIMDAHL - *Proving the Shalls: Early Validation of Requirements through Formal Methods*. Software Tools for Technology Transfer, volume 8, number 4. 2006

[13] J. SOUYRIS, V. WIELS, D. DELMAS, H. DELSENY - *Formal Verification of Avionics Software Products*. Formal Methods 2009

[[14]D. DELMAS, J. SOUYRIS - *Astrée: From Research to Industry*. SAS 2007: 437-451.

[15] D. BROWN, H. DELSENY, K. HAYHURST, V. WIELS - *Guidance for Using Formal Methods in a Certification Context.* Proceedings of ERTSS 2010

[16] T. BOCHOT, P. VIRELIZIER, H. WAESELYNCK, V. WIELS - *Paths to Property Violation: a structural approach for analyzing counter-examples*. 12th IEEE High Assurance Systems Engineering Symposium (HASE'10), November 2010.

[17] O. LAURENT, P. MICHEL, V. WIELS - *Using Formal Verification Techniques to Reduce Simulation and Test Effort.* FME 2001: 465-477

[18] E. TORLAK, D. JACKSON - *Kodkod: A relational model finder.* TACAS 2007. http://dx.doi.org/10.1007/978-3-540-71209-1_49.

[19] K. ANASTASAKIS AND B. BORDBAR AND G. GEORG, I. RAY - *UML2Alloy: A Challenging Model Transformation*. 10th International Conference on Model Driven Engineering Languages and Systems 2007. LNCS 4735, Springer.

[20] J. CABOT, R. CLARISO, D. RIERA - *Verification of UML/OCL Class Diagrams using Constraint Programming*. ICSTW '08. IEEE Computer Society.

[21] R. DELMAS, T. POLACSEK, D. DOOSE - *Utilisation de techniques SAT/PseudoBool pour la synthèse de modèles corrects par construction dans le cadre IDM*. In Génie Logiciel, volume 97, Juin 2011

[22] R. DELMAS , T. POLACSEK , D. DOOSE, A. FERNANDES-PIRES - *IDM: Vers une aide à la conception*. Inforsid 2011

[23] R. DELMAS, T. POLACSEK, D. DOOSE - *Supporting Model Based Design*. MEDI 2011. LNCS Springer-Verlag.

[24] Onera, IRIT. Projet CAVALE - *Collaboration d'analyses - Choix techniques et mise en œuvre*. Technical report, janvier 2011

[25] C. BARRETT, R. SEBASTIANI, S. SESHIA, C.TINELLI - *Chapter on Satisfiability Modulo Theories*. A. Biere, H. van Maaren and T. Walsh editors, Handbook on Satisfiability. IOS Press, February 2009

[26] C. BARRETT, A. STUMP, C. TINELLI - *The SMT-LIB Standard: Version 2.0*. Proceedings of the 8th International Workshop on Satisfiability Modulo Theories, 2010.

[27] T. KAHSAI, C. TINELLI - *PKIND: a Parallel k-Induction Based Model Checker*. Proceedings of 10th International Workshop on Parallel and Distributed Methods in verifiCation, Snowbird, USA, 2011. EPTCS

[28] O. LAURENT, C. SEGUIN, V. WIELS - *A Methodology for Automated Test Generation Guided by Functional Coverage Constraints at specification level.* ASE 2006: 285-288

[29] A. LAKEHAL, I. PARISSIS - *Structural Coverage Criteria for Lustre/SCADE Programs.* Software Testing, Verification and Reliability (STVR), Wiley Interscience, 19(2):133-154, 2009

[30] PAUL E. AMMANN, PAUL E. BLACK, W. MAJURSKI - *Using Model Checking to Generate Tests from Specifications*. Proceedings of 2nd IEEE International Conference on Formal Engineering Methods (ICFEM'98), IEEE Computer Society, pages 46-54

[31] M. STAATS, M.W. WHALEN, M. P.E. HEIMDAHL, A. RAJAN - *Coverage Metrics for Requirements-Based Testing: Evaluation of Effectiveness*. NFM 2010

[32] G. DURRIEU, H. WAESELYNCK, V. WIELS. LETO - *A Lustre-based Test Oracle for Airbus critical systems*. Formal Methods for industrial Critical systems (FMICS) - L'Aquila, Italy, September 2008

[33] H. BARRINGER, A. GROCE, K. HAVELUND, M. SMITH. - *Formal Analysis of Log Files*. Journal of Aerospace Computing, Information, and Communication, Volume 7 - Issue 11, 2010

## AUTHORS

**Virginie Wiels** has been a research scientist at Onera since 1998 and her research interest is in the use of formal methods for the verification of critical embedded systems. She is leading the LAPS research unit, focusing on Languages, Architectures and Proofs for embedded Systems. She was research scientist at the NASA IV&V center in Fairmont, USA from 1997 to 1998, working on formal verification of NASA space shuttle embedded software. She graduated from the Ecole Nationale Supérieure d'Electronique, Electrotechnique et d'Informatique in Toulouse (ENSEEIHT). She received her PhD degree in computer science from the Ecole Nationale Supérieure d'Aéronautique et d'Espace (ENSAE) in 1997. She received her Habilitation à Diriger des Recherches (HDR) from the Institut National Polytechnique in Toulouse (INPT) in 2007.

**Rémi Delmas** graduated from ENSMA in 2000 and obtained a Computer Science Ph.D. from ENSAE in 2004. From 2005 to 2009 he worked as an R&D engineer at Prover Technology on the design, implementation and certification of model checking tools for railway control systems. Rémi Delmas joined Onera in late 2009. His current research focuses on the combination of formal verification methods for reactive software, and on the use of combinatorial optimization techniques to automate the design of safe-by-construction embedded architectures.

**Pierre-Loïc Garoche** is a research scientist at Onera. His work is mainly focused on the use of formal methods in critical embedded systems development in a certified context. He graduated from the École Normale Supérieure de Cachan, France, and received his PhD degree in Computer Science from the University of Toulouse, France in 2008.

**Jacques Cazin** has been with INRIA until 1983, then with Onera where he was successively research engineer, head of a research unit focused on the design and validation of computer systems, director of the information processing and modeling department (DTIM). In 2007 he left the department management and returned to research work. For 35 years he conducted research in technical fields as varied as computer languages, operating systems, software engineering, computer security, safety, embedded systems, with a particular and continuous interest in formal approaches and methods. Jacques graduated from the École Nationale Supérieure de l'Aéronautique et de l'Espace (ENSAE) in 1975 and received a Masters in computer science from the same school in 1976.

**David Doose** is a research scientist at Onera. His research interest is mainly in Model Driven Engineering, synthesis of design solutions based on constraint solving and real time analyses of embedded systems. He graduated from the University of Toulouse, and received his PhD degree in Computer Science from the University of Toulouse, France in 2006.

**Guy Durrieu** is a research scientist at Onera. His research interest is mainly in the use of simulation, test and formal methods for the verification of critical systems, in particular from the performance point of view. He graduated from the University of Toulouse, and received his PhD degree in Computer Science from the University of Toulouse, France in 1977