

Model Driven Development of Certifiable Software: A Best Practice for Safety-Critical Applications

**Moving beyond Model Driven Design of Systems to Formal Development
of Software as a Best Practice for Certification in Mil/Aero, Industrial,
Transportation, and Medical Applications**

Jerry Krasner, Ph.D., MBA

December 2008

Embedded Market Forecasters

American Technology International, Inc.

Embedded Market Forecasters
Research and Consulting
for Embedded Products,
Markets and Channels



About EMF: www.embeddedforecast.com 508-881-1850

EMF is the premier market intelligence and advisory firm in the embedded technology industry. Embedded technology refers to the ubiquitous class of products which use some type of processor as a controller. These products include guided missiles, radars, and avionics as well as robots, automobiles, telecom gear, and medical electronics.

Embedded Market Forecasters (EMF) is the market research division of American Technology International, Inc. EMF clients range from startups to Global 100 companies worldwide. Founded by Dr. Jerry Krasner, a recognized authority on electronics markets, product development and channel distribution, EMF is headquartered in Framingham, Mass.

About the author:

Jerry Krasner, Ph.D., MBA is Vice President of Embedded Market Forecasters and its parent company, American Technology International. A recognized authority with over 30 years of embedded industry experience, Dr. Krasner was formerly Chairman of Biomedical Engineering at Boston University, and Chairman of Electrical and Computer Engineering at Wentworth Institute of Technology and Bunker Hill Community College. In addition to his academic appointments, Dr. Krasner served as President of Biocybernetics, Inc. and CLINCO, Inc., Executive Vice President of Plasmedics, Inc. and Clinical Development Corporation, and Director of Medical Sciences for the Carnegie-Mellon Institute of Research. Earlier, he was Senior Engineer at the MIT Instrumentation Laboratory. Dr. Krasner earned BSEE and MSEE degrees from Washington University, a Ph.D. in Medical Physiology / Biophysics from Boston University and an MBA from Nichols College. He is a visiting professor at the Universidad de Las Palmas (Spain) where he is recognized for his work in neurosciences and computer technology.

Copyright 2008 by Embedded Market Forecasters, a division of American Technology International, Inc, 1257 Worcester Road #500, Framingham, MA 01701. All rights reserved. No part of this document covered by copyright hereon may be reproduced or copied without expressed permission. Every effort has been made to provide accurate data. To the best of the editor's knowledge, data is reliable and complete, but no warranty is made for this.

Overview

As an “old timer” I can assure today’s developers that they have it good. The plethora of development tools and development processes available today stagger the imagination – and make the tools that I had available to me many years ago seem prehistoric. To be honest, designs have become orders of magnitude more complex since I developed software.

What is surprising to us (maybe it shouldn’t be so surprising) is that developers are so busy that many are unaware of the available design concepts and processes that might shorten their development cycle and produce a better and more reliable product.

From the annual detailed EMF surveys of embedded developers one can examine the tools that developers use, the applications that they use them in, whether they are “new to the world” designs or upgrades from prior designs, the time it takes to start a project and ship the product, and how close to one’s final design compares with the pre-design expectation for performance, systems functionality, and features and schedule. Project managers can look at the tools market and processors to be used and see what fellow engineers have experienced.

Using data filtering, we can examine each of these data in context with any or all of the others, so we can see what processors or programming languages are used for aerospace, consumer devices, medical applications, etc. Also, we can look at design outcomes where formal methods were employed and compare these to other methods.

From year-over-year surveys, EMF is able to determine what practices developers are using (successfully or not) for ten different vertical applications (e.g., automotive, telecom, medical, etc).

These data can be used to define different application markets that can create independent documentation to satisfy user needs or established requirements.

The Federal Drug Administration’s Center for Devices and Radiological Health (CDRH), for example, is responsible for regulating firms who manufacture, repackage, relabel, and/or import medical devices sold in the United States. The FDA in their 510k pre-market notification process (required before a product can be sold in the U.S.) stipulates such criteria as proving that the product is “Safe and Effective” yet it provides no certification guidelines or requirements for software content.

So, a developer or vendor can produce a product that meets the 510k requirement without using software best practices processes, or can use OSe or code that has been exposed to rigid testing by certifying agencies. Such software testing is required by the FAA, by the military and by the government for use in mission and safety-critical applications for which a software failure would be catastrophic (bringing down an airliner for example). If a vendor is going to produce a medical product, wouldn’t it make sense to develop it in a manner such that failures would be kept to an acceptable minimum? EMF data can be used to determine design outcomes for those that use certified OSe compared with other OSe.

So why don't medical device manufacturers incorporate such technologies in their designs and processes? The same can be said for other market verticals such as aerospace, military, automotive, and transportation, applications..

While some attention has been given to OSEs that meet DO-178B level "A" certification (or ARINC 653, among others), that meet MILS or Common Criteria, much has been overlooked by developers. Such pre-design considerations as code preservation, code reuse and documentation maintenance that have become paramount for software and systems developers – are available with Model Driven Development tools like IBM/Telelogic's Rhapsody and Esterel Technologies' SCADE Suite, which also enables software redeployment when underlying hardware has changed.

The fact that software can be certified using formal processes and tools has been largely overlooked in the development processes and the shipment of products to these markets.

Think of being able not only to use established OSEs and tools shown to be most useful in the design and support of products shipped, but to also assure the customer (as well as any regulating agency) that the software meets the stringent requirements set forth with certification, i.e., legal recognition by the certification authority that the software product complies with the requirements. Such certification comprises the activity of technically checking the software product, and the formal recognition of compliance with the applicable requirements. In particular, certification of a software product involves the process of assessing the design of a product to ensure that it complies with a set of standards applicable to that type of product so as to demonstrate an acceptable level of safety and/or security.

Standards and Requirements That Are Driving the New Embedded Marketplace

As the critical software and tools marketplace seeks a path to new and more profitable markets the strategies of the past are being cast aside as vendors seek to use standardization to their advantage rather than to pursue past based on non-compatibility.

Keeping the customers in mind (developers and OEMs) a vendor can benefit from the fact that the use of software standards can reduce costs and enhance the design process:

- by certifying the software product using formal methods
- by increasing requirements, design and code reusability, project to project, system to system
- by reducing the learning curve for developers
- by enabling swift redeployment of developers, project to project
- by enabling the use of third-party products that are integrated and tested to work in that environment (e.g., Eclipse)

The embedded software and tools marketplace became largely commoditized. RTOS and IDE vendors were competing in a virtual zero sum game. What the embedded industry needed was new and expanding markets – particularly those that dwarf the established ones.

UML has enabled embedded vendors to address new and non-traditional markets – markets that are opening and expanding at a rapid rate. These markets are creating new opportunities for smaller vendors as well. One way that the embedded market is expanding is to move into new or larger markets, such as the enterprise and the military. But another way to expand the embedded marketplace is to address the problems of creating safe and secure device software, not merely addressing where to place that software.

Niche market opportunities are a characteristic of expanding markets as engineering-depleted larger companies/OEMs need smaller and faster responding partners in order to deliver superior products, on-time, and within allowable cost structures.

There are three items that characterize the new embedded marketplace:

- Interoperability
- Guaranteed software quality
- Enhanced safety and security

By providing any or all of these, any vendor can significantly broaden their market opportunities.

Interoperability

As software complexity continues to increase and applications are distributed across a wide array of utilizations, it is essential to be able to interoperate software designs – many of which have been developed using different RTOSes. This need is amplified by the requirement of forward and backward compatibility in order to upgrade systems without major software rewriting and to reuse existing applications in newer upgrades.

One example is the U.S. Navy. Consider the findings of a conference sponsored by the Carnegie-Mellon Software Engineering Institute.

The next generation of Navy ships will have a 30 to 50 year life expectancy. The Navy discovered that the cost of on-board personnel (sailors) is extremely costly and the expense forecast on current staffing levels would double the cost of the ship itself. With an all-volunteer military there is also the uncertainty of maintaining staffing levels. Hence there is a major incentive to reduce the number of sailors required on ships by automating processes with highly reliable and secure computing.

Central to this strategy is the ability to create application software that can be used and upgraded across a large number of installations. The mission can be summarized as follows:

- Determine how small a crew is needed to effectively operate all systems.
- Determine how uniformity can be implemented across all ships and systems enabling a greater uniformity and expedience in training.
- Software interfaces need to be uniform across all operating systems (e.g., POSIX).
- As much as possible, existing software should be reusable for newer and upgraded systems.

- Rewriting of software should be held to a minimum.
- Certified software products can be maintained.

Guaranteed Software Quality

Requirements for software quality are not limited to one or a few verticals, but are appropriate for many.

For example, Aerospace and Defense manufacturers must optimize their device software development to provide next-generation applications with more interoperability, safety, security and connectivity. Central to this need is the ability to qualify both the performance of developed software and the development process that created the software in the first place. This “best practice” assurance is central to defense acquisitions and to avionics programs across the board.

For U.S. coalition and allied forces to counter current and future threats successfully, they must operate worldwide with speed, agility and flexibility. They must have access to accurate, current and timely information, and the capability to share this information securely. In order to meet these requirements aerospace and defense manufacturers are faced with the following:

- Integrating devices to the Department of Defense's (DoD) Global Information Grid as a key element of future combat power. To support the continued growth and advancement of Net-Centric Warfare capabilities, the DoD requires that all devices will be IPv6 compliant by 2008. The NSA has accelerated the requirement that all devices will be compliant.
- A demand for mission-critical systems with secure connections that meets higher evaluation assurance levels (EAL 6-7) and DO-178B safety certification level A.
- Adhering to open standards like POSIX and ARINC 653, to facilitate device interoperability.

The ability to deploy certified software products is consistent with this effort.

If you are the Joint Commission of Accreditation of Health Organization (JCAHO – pronounced Jay-co) and are ultimately responsible for setting software and patient information security requirements (as will be required under the Health Information Protection and Accountability Act – HIPAA) it should be a no-brainer to look at the security and software performance requirements set forth by the government for defense and avionics applications.

A similar case can be made for financial data transfer and acquisition applications that are now dictated by GLB legislation. Enterprise applications that involve company confidential information, e.g., CRM, inventory, pricing schedules, and customer data files, face similar concerns that may be addressed through application of the government's solutions for defense and avionics requirements.

Enhanced Embedded Safety and Security

Embedded systems are responsible for the availability and functionality of many critical systems, from factory automation to gas pipeline monitors to networking equipment. Unfortunately, the critical importance of embedded systems is seldom matched with a strong, comprehensive security infrastructure. Some of the critical security issues presented by modern embedded systems are:

- Diverse network-connected embedded systems use combinations of custom and COTS software, the details of which are typically known only to the vendor of each embedded device, making vulnerability assessment, risk analysis and patch management difficult.
- Many embedded protocol implementations derive from older versions of open-source software like OpenSSL and the BSD TCP/IP stack, resulting in vulnerabilities to known attacks, which have since been patched in the main software distributions.
- Many other protocol implementations are built entirely from scratch and have not benefited from years of public analysis and repeated attack, resulting in unproven protocol implementations that may be vulnerable to attack. Formal methods that insure that code is certifiable can be of value.
- Even when vulnerabilities are identified, patches must be developed for each device or device family by the vendor, requiring tight collaboration between embedded software developers and the OEM's building devices based on the developers' software.
- Most network-aware embedded devices lack sufficient management and auditing functionality thereby making centralized configuration and monitoring difficult and costly and severely limiting the data available for attack-pattern detection and after-attack forensic analysis.
- Embedded systems are not always considered an IT responsibility and thus often fall outside IT control resulting in lax policy enforcement, minimal configuration management and auditing, distorted risk analyses and little or no integration with enterprise security tools.

Remediation of these issues will require the following: a concerted effort among commercial and custom embedded software developers, OEM's building embedded systems, vendors selling such, and customers purchasing and implementing products based on network-aware embedded software. Until information security becomes a strategic technology for embedded systems developers, their products will continue to be characterized by complacency and vulnerability.

However embedded vendors and OEMs alike do not have the time to be complacent regarding security. The U.S. government has mandated that all equipment (including embedded designs) that provide connectivity for any level/type of connectivity, utilize FIPS 140-2 certified modules for all purchases by the U.S. government and for all procurements by OEMs that have government contracts. The NSA has decreed that there are **NO** exceptions to the requirement. Device software developers could well increase the reliability of their software by complying with this and other security standards.

Formal Methods for Certifying Software Products and how it differs from Non-Formal or Semi-Formal Methods

We have seen that on the OS and API side, standards and common interfaces are already wide spread and well accepted in the market. Software interfaces (such as POSIX, ARINC653) and communication protocols are standardized or at least industry standards are established.

The situation is quite different for core application software development itself, beyond platforms and middleware, where it comes to the real meat of intellectual property, where vendors and developers can differentiate themselves by offering new functionality, innovative algorithms and add real value, the situation is quite different.

Model-Driven Development is seen as a way to bring common best practices to this field. We have seen in the past how model-based exploration of the main system algorithms can be used by system designers and control engineers for the development of system requirements that contains greater details than text-based specifications. This allows for simulation of control laws together with a model of the physical environment long before any physical prototypes and systems are available. Going further down, we have also seen how the system engineers can use Model-Driven Design tools such as UML or SysML to build an architectural design of the system.

Model-based methods and related tools can be classified into three basic categories.

- Non-formal: The method and related models are either based on natural language, or is only defined through its implementation in a tool and it is open to arbitrary new symbols
- Semi-formal: The syntax is defined based on a mathematical definition, but the semantics are defined either through natural language or its implementation in a tool
- Formal: A fixed language with mathematically defined syntax and semantics.

Based on this definition it is clear, that only a formal method will ensure that the specification will be complete, precise and consistent.

Now consider the fact that the method and tools used for algorithm exploration, such as Simulink™, are in general **non-formal**, the reason being that control engineers need more flexibility than formality when performing this task. Also consider that the methods for creating architectures, such as UML, can be classified as **semi-formal**.

When it comes to developing **critical software**, our point of view is that **formal** methods and tools will then provide the necessary rigor for developing and certifying the software product in an efficient way in the various application domains that we have described above.

In the end, we obtain the classification presented in Figure 1 below for specifying and developing a critical system:

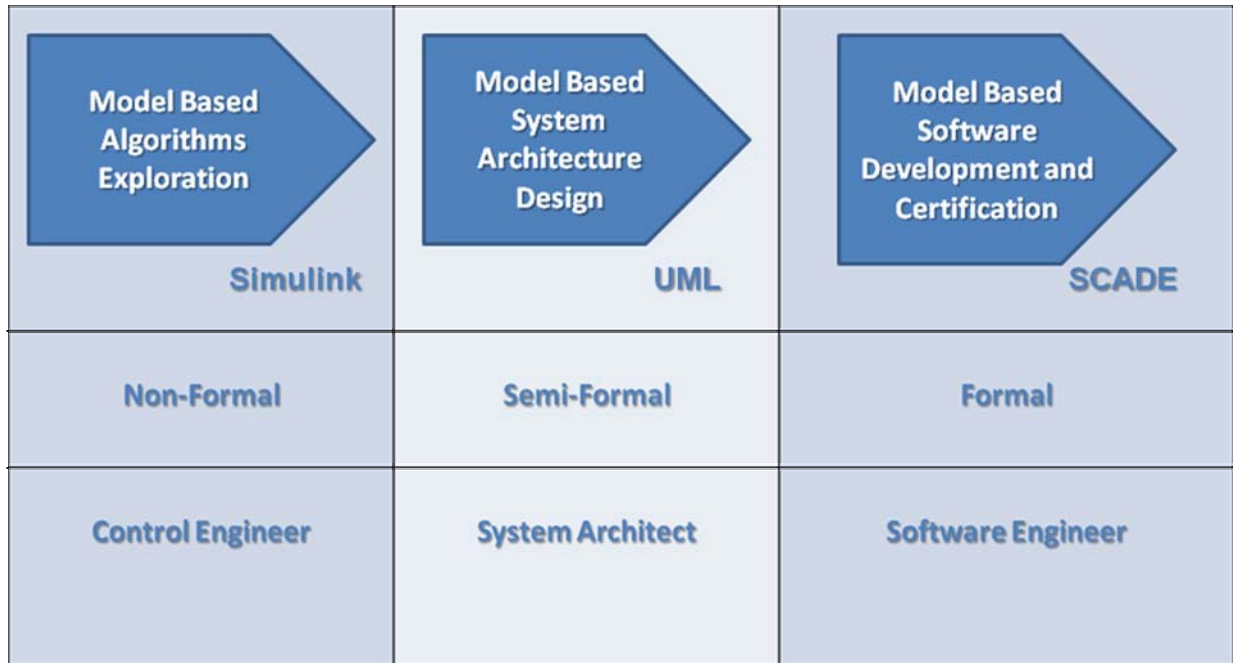


Figure 1: Model-Driven Methods and Tools for Developing Critical Systems

Let us now continue by stating in the Table 1 the main challenges that have to be faced when **developing safety or mission-critical software**:

<p>1. Mastering complexity and scaling</p>	<p>Certified critical software may be very large and complex in terms of architecture and functionality. It can be distributed over many computing nodes, control thousands of elements and have complex interdependencies.</p>
<p>2. Avoiding multiple descriptions of the software</p>	<p>Processes that are mandated by the certification standards divide software development into several phases according to which the related documents are created (requirements specification, software architecture specification, module design, and source code). At each step, it is important to avoid as much as possible rewriting of the software descriptions. This rewriting would not only be expensive, it would also be error-prone. And therefore there is a major risk of inconsistencies between the different descriptions, leading to a significant effort in verification.</p>
<p>3. Fighting ambiguity and lack of accuracy of specifications</p>	<p>Requirements and design specifications are traditionally written in some textual-based language, often complemented by non-formal or semi-formal graphical descriptions, such as UML. It is an everyday experience that textual-based languages are subject to interpretation. Their inherent ambiguity can lead to different interpretations, depending on the reader. This is especially true for the dynamic behavior of the applications. How to interpret several parallel sentences containing “before X” or “after Y”?</p>
<p>4. Avoiding manual coding</p>	<p>Coding is the last transformation in a traditional development process. It takes as input the design specifications. The programmer generally has a limited understanding of the system requirements specification, which makes him</p>

	vulnerable to ambiguities in the specification. He produces source code, which is difficult, if not impossible, to understand by the author of the initial systems requirements. In this updates approach, the combined risk of interpretation error and coding error is so high that a major part of the lifecycle's verification effort is consumed by code testing.
5. Finding specification and design errors as early as possible	Many specification and design errors are only detected during software integration testing. One reason is that the requirements and design specifications are often ambiguous and subject to interpretation. The other reason is that it is particularly difficult for a human reader to understand the dynamic behavior of a specification described in a non-formal or semi-formal notation. In such a traditional process, the first time one can truly exercise the software is during integration. This is very late in the process. The cost of fixing an error is very much higher than if it has been detected during the specification phase.
6. Lowering the complexity of	There are many sources of changes in the software, ranging from bug fixing to function improvement or the introduction of new functions. When something has to be changed, all products of the software life cycle have to be updated consistently, and all verification activities must be performed accordingly.
7. Improving verification efficiency	The level of verification for certified safety-critical software is much higher than for other non-safety-critical software. For level A DO-178B avionics software, the overall verification cost may account for up to 80% of total costs. Verification is also a bottleneck for project completion. So, clearly, any change in speed and/or cost of verification has a major impact on the project time and budget.
8. Providing an efficient way to store Intellectual Property (IP)	A significant part of the systems suppliers' know-how resides in software. It is therefore important to provide tools and methods to efficiently store and access IP relative to these safety-critical systems. Such IP vaults should typically contain: textual requirements, software models, source code, tests cases, and certification artifacts.

Table 1: the Challenges of Developing Safety and Mission-Critical Software

Let us show how formal methods and tools, such as Esterel Technologies SCAD Suite™, which is one of the few tool suites based on formal methods that is well-established in the market, provide an answer to the challenges that we have listed above.

The Esterel SCAD Suite is based on formal methods and tools and it relies on the **“DESIGN-VERIFY-GENERATE”** paradigm, which has been proven in industrial applications to be much more efficient than the traditional “DESIGN-CODE-VERIFY” paradigm. This is shown in Figure 2.

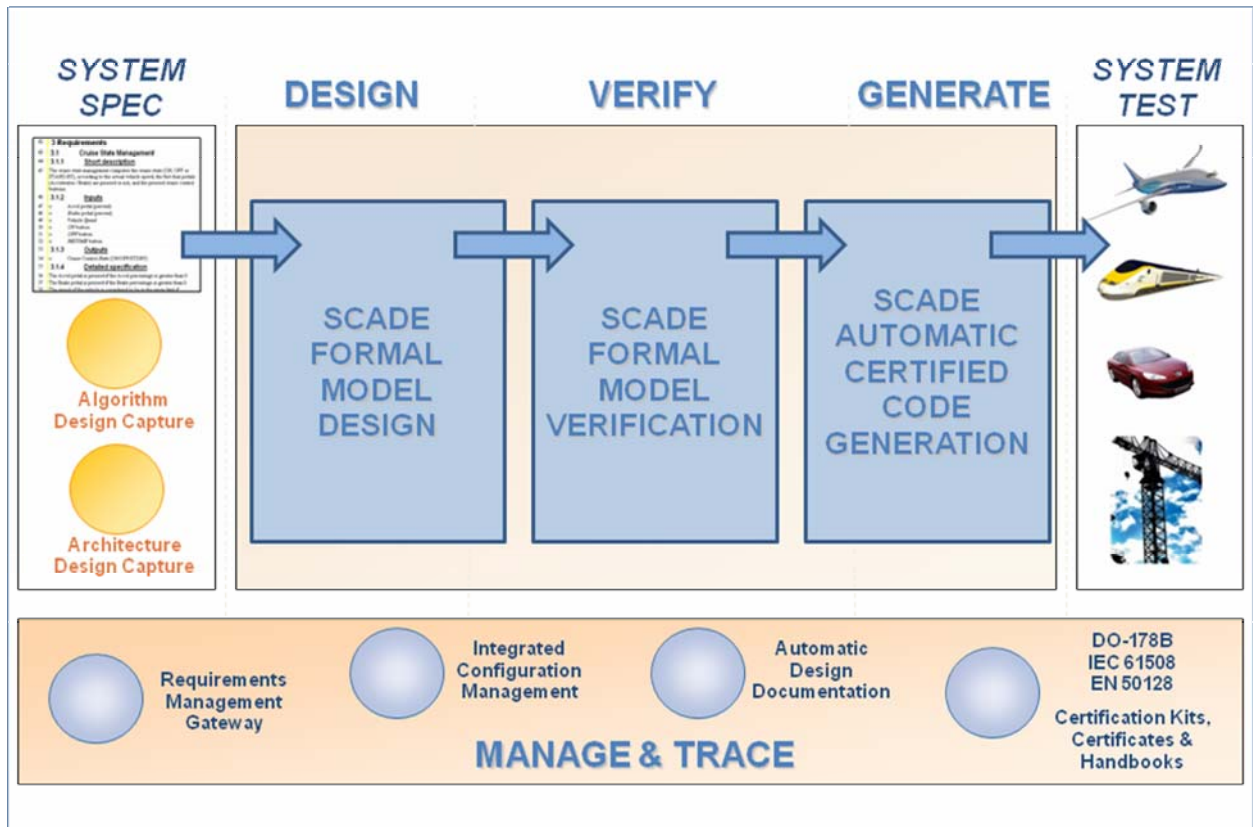


Figure 2: the DESIGN- VERIFY- GENERATE Model-Based Development Approach for Certifying Software

The main idea behind this lifecycle is to front-load all verification activities at model level, doing these very early and efficiently in the project lifecycle.

Design

The first step is modeling with SCADA Suite in the formal Scade® language. The inputs to this process are the system requirements. This may be English text managed in a COTS Requirements Management tool or it can simply be a Word document. Furthermore, other inputs to the SCADA Suite design paradigm can include the control algorithms, as we discussed earlier, and the system architecture that may be expressed in UML or SysML, two notations that are particularly efficient at this stage.

Verify

The second step comprises the verification activities:

- *Simulation on the basis of the Scade model.* Such a simulation is driven by test cases that are based on system requirements and produces results that are guaranteed to be an exact representation of the behavior that is going to be

observed on the target. The fundamental reason why this is possible is the complete formality of the underlying Scade notation.

- *Coverage analysis of the Scade model.* Certification usually requires that the software contains the required functions and no more than the required function. i.e. the software should not contain any unintended functionality. Model coverage analysis is an efficient way to achieve this objective.
- *Formal verification of safety properties.* The fact that Scade is a formal notation enables the execution of formal verification tools that can establish that a safety property is satisfied or not (e.g. the landing gear is not going up while the aircraft is on the ground), without testing, by an exhaustive examination of the state space of the application.

Generate

Finally, the source code is automatically and entirely generated by the certified code generator. This is only possible because of the inherent formality and rigor of the formal Scade model. Because the code generator produces certified code (e.g. DO-178B at level A) the source code is correct with respect to its specification, i.e. the Scade model. Therefore, there is no need for verification activities such as code reviews or code testing. The application is then integrated on target and can undergo system tests. The fact that the execution model of a SCADE generated application is very simple and very little dependant on the target guarantees a high degree of interoperability, as this was listed earlier as an important objective.

Moreover, a set of project management and traceability services are provided by the SCADE model-based development environment (requirements management, configuration management, production of documentation, and certification evidence for such standards as DO-178B for aeronautics, EN 50128 for railway, and IEC 61508 for industry).

In Table 2, let us now assess what the use of formal methods and tools has achieved with regard to the challenges we had listed previously in Table 1:

1. Mastering complexity and scaling	Formal models can be used in the context of very large and complex systems, generating in the end potentially millions of lines of source code in real world applications.
2. Avoiding multiple descriptions of the software	The initial software requirements can be described as a mix of natural language and semi-formal notation such as UML. Then, a formal model is constructed and it is at the center of all activities in the software development lifecycle (verification, code generation, certification etc).
3. Fighting ambiguity and lack of accuracy of specifications	Being formal, Scade provides a notation that is used to describe non-ambiguous, complete and accurate software specifications.
4. Avoiding manual coding	Thanks to the SCADE Suite certified code generator, source code can entirely and automatically be generated from a model.
5. Finding specification and design errors as early as possible	Verification and validation activities are mostly performed at the level of the formal model. They can therefore start in a very early stage of the project.

6. Lowering the complexity of updates	When a bug is detected, it can be fixed in the formal model, which is re-verified as needed, and code may be re-generated automatically, with no further verification activities.
7. Improving verification efficiency	Errors can be detected at a very early stage of the project, thus cutting a lot of the verification cost. Moreover, certification of the code generator eliminates the need for verifying that the source code agrees with the formal model.
8. Providing an efficient way to store Intellectual Property (IP)	A formal model is an ideal vehicle to store systems suppliers' know-how in the form of re-usable libraries of formal models that are independent from their implementation since source code can be generated for any target. Together with the models, this IP vault would typically contain requirements, verification evidence, and more generally all certification artifacts.

Table 2: the Benefits of Formal Methods for the Development of Safety and Mission-Critical Software

Conclusion

Mathematical modeling theories and techniques have been developed over the last 20 years in the context of avionics control and have matured into formal methods and tools for developing critical software. For example, the Airbus 380's fly-by-wire software is automatically generated from such Scade models, and has been a showcase for the European prowess in establishing the right mathematical principles and models for requirements capture, and the transformations required to bridge the gap between system requirements models and certified software development.

We have shown in this paper the efficiency and value of the design-verify-generate paradigm. It is now time these methods that have been used with tremendous success primarily in the European Mil/Aero domain, gain further momentum in other regions and industries developing safety-critical applications. Clearly, to remain competitive in the global economy, companies will have to move from manual informal and semi-formal methodologies and tools to proven certified model-based development environments.