Engineering Air Traffic Control systems with a Model-driven approach

Gabriella Carrozza * Mauro Faella †, Francesco Fucci‡, Roberto Pietrantuono‡, Stefano Russo‡§

*SESM scarl, a Finmeccanica company, Via Circumvallazione Esterna di Napoli, 80014 Giugliano, Italy
†Critiware S.r.l., Incubatore Incipit, Complesso Universitario Monte Sant'Angelo, Via Cinthia, 80126, Naples, Italy
‡Dipartimento di Informatica e Sistemistica, Universitá di Napoli Federico II, Via Claudio 21, 80125, Naples, Italy.
§Laboratorio CINI-ITEM "Carlo Savy", Complesso Universitario Monte Sant'Angelo, Via Cinthia, 80126, Naples, Italy.

***Abstract*—Testing software in Air Traffic Control (ATC) systems costs much more than building them. This is basically true in every domain producing software-intensive critical systems. Software engineers strive to find methodological and process-level solutions to balance these costs, and to better distribute verification efforts along all the development phases.**

There is considerable interest in applying model-driven approaches in the critical systems engineering field. Kept promises and failed expectations of model-driven engineering are still debated today; we report our experience in trying to take the *model-driven* best achievements and, at the same time, to fill its lacks in the considered industrial context.

***Index Terms*—Software Process Models, Lifecycle, Model-based Design, Model Driven Testing**

I. Building ATC systems today

One of the fundamental pillars of Air Traffic Management (ATM) is **Air Traffic Control** (ATC). ATC systems are software-intensive critical systems which take care of assuring that aircrafts are safely separated in the sky as they fly, and at the airports where they land and take off [1]. An ATC system is in charge of managing all ground and en-route flight operations, with the aim of preventing collisions and organizing the flow of traffic.

To build software for ATC systems, the most consolidated development process model is by far the V-Model. Its key benefit is the accounting of verification and validation at early stages, as soon as requirements are elicited, which allows having development and V&V activities conducted in parallel flows. In a V-Model, criteria for testing are defined on the basis of what is actually intended to produce, and not a posteriori, i.e., on what has been already produced. The model is also trusted because of the net separation of phases, supported by entry and exit criteria, and the precise definition/separation of role and responsibilities. This helps in embedding know-how more in the process than in people's mind; the final products' quality is more prone to be independent of individual skills.

However, pressures from the market require more time- and cost-effective ways to produce and assess software. And when talking about effectiveness in software production, the prime suspect is *testing*, especially for critical systems. Starting thinking about testing as requirements are available, as V-model constrains, is certainly necessary, but it seems no longer sufficient at all. Testing and on-site maintenance cost deriving from quality issues are still a relevant concern for manufacturers and system integrators.

The main source of cost is in the left side of the "V", where early verification is still not well supported by methodologies and tools. But for better cost-quality balance, improvements are required not only from the testing perspective. Design- and process-level reasoning is a key issue to optimize testing efforts. Resources required in terms of personnel and skills, poor communication within the team, and minor involvement of end-user are V-Model deficiencies that heavily reflect on quality and cost management.

Solutions from the literature present valuable and promising advancements; yet, there is still lack of practical experience in industrial contexts, which are the real enabler proofs in critical domains. This makes it tough to tailor toy examples to real-scale complex systems.

In the context of a public-private collaboration between the University of Napoli and the Finmeccanica companies "Selex Sistemi Integrati" and SESM, we are jointly looking for process-level solutions to impress a sharp yet reliable change in the way of building high-quality software for ATC systems.

The current V-Model process we refer to comes as in Figure 1 (with labelled artefacts compliant with the MIL-STD-498 standard [2]). We are challenged with improving the cost-quality trade-off without impacting the current well-proven practices in such a process. This translates into finding a solution able of: *i)* detecting more specification and design errors earlier; *ii)* fighting inconsistencies among artefacts; *iii)* keeping the main flow of the V-model (with role and responsibilities) unaltered; *iv)* scaling with respect to systems complexity.

II. Looking at Model-driven approach

Given these requirements, a model-driven approach seems attractive for us. We look, as specific paradigm, at Model-driven Architecture (MDA) [3]. Besides the claimed advantages in terms of interoperability, portability and reusability, we are interested in these key features:

- manual activity in repetitive error-prone tasks are minimized;
- redundant descriptions, at different stages, of the software behaviour are avoided by automatic transformations; this minimizes inconsistencies;
- early verification and validation of design artefacts are aided by tools and favoured by modelling notation and rules;

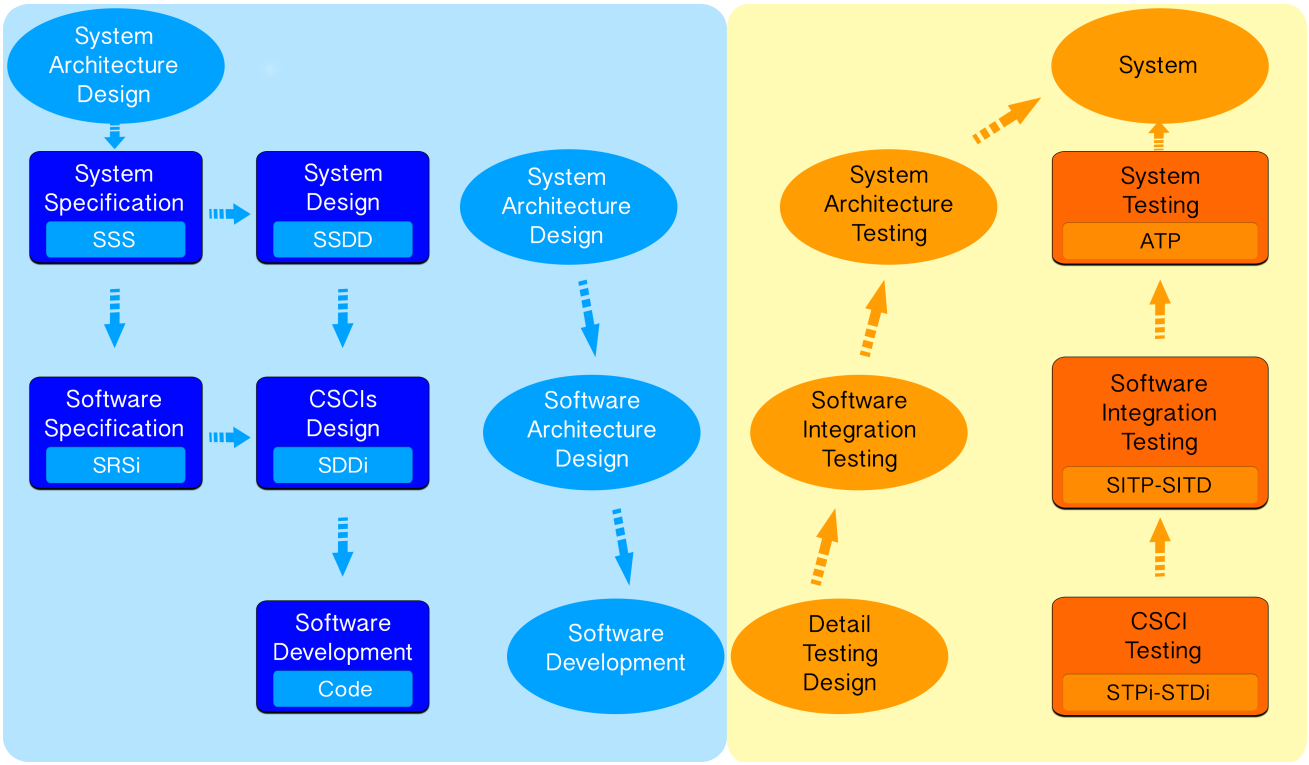


Fig. 1: The reference V-Model process

- design-oriented flow helps to optimize testing effort;
- code can be generated automatically, which is definitely the most striking feature;
- maintenance cost is also reduced, since the effort of introducing a change at upper level can be minimized by automatic transformations *model to model* and *model to code*;
- compared to pure text, models are less prone to misinterpretation. They dramatically reduce the possibility of misunderstandings on artefacts between different teams, as well as between teams and stakeholders, at every stage.

We distinguish two major benefits in a possible integration of MDA into our V-Model: *i)* a direct testing and maintenance cost reduction by early defect detection, since the idea of V-model of verifying correctness and consistency at each stage would be much enforced; *ii)* a further cost reduction coming from the possibility of generating code automatically, of favouring reuse, and of easing updates and maintenance actions during operation.

These benefits are not in contradiction with V-model; rather they look as a natural improvement and refinement of its pillars. So, integrating MDA into our V-Model is the next step we decide to take. But, again, MDA alone does not suffice, and what it cannot cover requires integration.

III. Embedding MDA/MDT into a V-Model

Incorporating a model-driven way of thinking in a full development cycle cannot be accomplished by simply placing MDA steps in the design/coding phase. If we want real benefits, we must handle: how to deal with phases not covered by MDA, and how existing well-proven activities will interact with MDA ones. This is of paramount importance to avoid bottlenecks and to not cancel potentialities of Model-driven.

In ATC systems engineering, an important need is to optimize testing activity. MDA primarily focuses on the "development" side. Verification is basically supported only as cross-checking of design artefacts consistency, but it is mostly neglected. Model-driven Testing (MDT) is the key; it shifts MDA concepts into testing [4]. Nevertheless, these two practices are not fully integrated and people do not see them under the same umbrella during everyday work. As MDA, MDT proposes Platform-Independent and Platform-Specific models as well, named PIT and PST, where T stand for "Test". And, exactly like MDA does, MDT can potentially reduce testing cost by deriving test cases automatically from these models [4], [5].

Today, the few companies investing on MDT do not usually manage the whole process automatically; they create models manually or partially reusing (MDA) design models (e.g., adding stereotypes or profiles to UML models, such as the UML 2 Testing Profile [6]). We present our solution to let MDA and MDT flow in parallel along the entire process; to this aim, we realize Model to Model (M2M) transformations to generate PIT and PIT-Software automatically from design models (the design models are referred to as Platform-Independent models, PIM, and PIM-Software, and Platform-Specific model, PSM [3]). Figure 2 shows the implemented links between MDA and MDT in the proposed solution.

System requirements analysis performed by domain experts is the opening step of the process. Then, two activities run in parallel: *i)* the creation of the PIM, and *ii)* the specification of

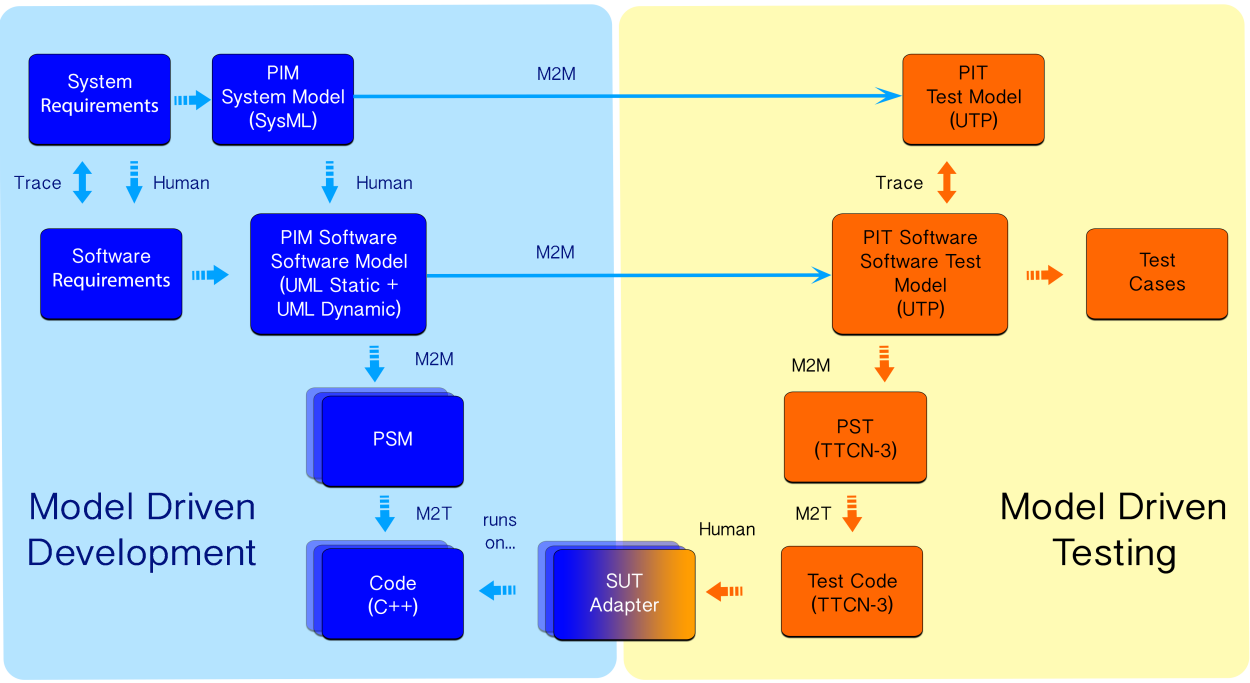


Fig. 2: Overview of the proposed process

software requirements.

Both the PIM and PIM-Software have two complementary views:

- the *static view* describes entities and their structural relationships;
- the *dynamic view* describes the run-time behaviour.

The PIM at system level is described by SysML diagrams (e.g., *requirement diagram*, *block diagram*, *state machine diagram*), and is transformed into the PIM-Software using software requirements. The PIM-Software is described in UML2 and primarily considers, among others, the following diagram types: *component diagram*, modelling the relationships among components; *state machine diagram*, describing the behaviour of components in terms of finite state machines; *data model*, describing the data managed by the system; these can be external data (exchanged with external actors), and internal data (exchanged among subsystems).

The static view from the PIM and PIM-Software is used to generate the PIT and PIT-Software through M2M transformation. These are described in UML Testing Profile (UTP) [6], as it is a standard for the definition and specification of test suites in the given domain. The dynamic view is used to generate the actual test cases by model-based coverage criteria (e.g., algorithms for specific coverage criteria of behavioural diagrams, such as state/transition coverage).

Then, on the left side, from the PIM-Software we generate the PSMs by using the right set of M2M transformation rules depending on the selected platform. On the right side, the PST is generated in TTCN-3 [7] notation through a further M2M transformation. We choose TTCN-3 so as we can use one PST for different PSMs.

The last part of the process concerns with:

1) the M2T transformations of PSMs into source code and of PST into TTCN-3 scripts;
2) the manual creation of SUT (Software Under Test) Adapters, one for each specific implementation, which is a piece of software used to translate TTCN-3 scripts into messages sent to the SUT.

Finally, the *Test Suite* is executed on the SUT. This is done in a specific TTCN-3 run-time environment, by means of the SUT Adapter. All the artefacts, but *software requirements*, *software models*, and *SUT Adapter*, are generated automatically.

At this point in time, we went to the market for getting support tools that are already there, more or less ready to use (e.g., *IBM Rational Rhapsody*®). However, we could not find a complete tool-chain able to support us throughout the whole integration and generation process. Despite several tools exist able to cover many steps of the process, they are very hard to be integrated either because produced from different vendors or due to the different hw/sw platforms they target. We have been accomplishing the difficult task of building such a tool-chain for several months, due to the huge benefit it would bring to the design, development, and verification teams.

However, besides these technicalities, the trickiest part is to make MDA/MDT flow good enough for building critical software. Indeed, there are still several deficiencies in the MDA/MDT paradigm. For coping with them, we rely on the very well proven V-Model activities, to be linked with MDA/MDT ones.

- An inherent problem of MDA/MDT is surprisingly again about testing, where the major benefits were expected. Indeed, testing automation is the most substantial contribution of MDT inasmuch as testing model contains the static and dynamic view as well; but *test automation, and*

MDT more generally, does not necessarily imply test-suite cost-effectiveness. Through the above-mentioned model-based coverage criteria, MDT automatically creates a lot of *functional test cases* from the test model in turn derived from design model; that is fine for testing what the system is expected to do against what specified at design stage. There are some problems in this: *i)* in large-scale complex systems, as the ones we deal with, exercising all the produced test cases is by far non-practicable; *ii)* in critical systems, conformance to certification standards, and the consequent best practices taken for quality assurance, provide already a certain degree of confidence on *functional* behaviour: *the missing link is fulfilment of non-functional requirements*. Standards require quality assurance evidences; other than a coverage level of functional behaviour, RAMS, robustness, and, more generally, dependability requirements satisfaction must be demonstrated.

To tackle these issues, we decide to rely on MDA/MDT to cope with functional test case production. Then, we faced non-functional testing by means of consolidated RAMS analysis steps at each stage.

In particular, as for the former point, to prevent the number of functional test cases from exploding we need adequacy criteria and test case selection techniques: we are exploring solutions to pursue high coverage at lower cost. Along with the implementation of several coverage criteria for test suites generated from state machines, we are now focused on similarity-based test case selection techniques [8].

As for the latter point, we use RAMS analysis for identifying the most critical software components to which allocate the greatest effort, in terms of time and budget, for non-functional test cases. According to this, we generate test cases to prove software robustness, and/or to run stress and performance tests. Although UTP provides some support to this task, you can exploit a much lower degree of automation compared to functional test cases generation.

- MDA does not cover the uppermost part of the V, that is from requirements to high-level design. Certification standards of interest (e.g., DO178B/DO178BC/DO248) deem *requirement management* as a crucial activity of the lifecycle. Even if MDA provides great support facilities in designing and checking conformance to requirements, it can be even improved by an integrated MDA/MDT approach. First, having executable design models allows exercising them against requirements. Second, looking at the generated test models helps in identifying discrepancies between the corresponding design model and requirements.

  However, this is still not enough to cover everything is needed in practice, especially from the certification perspective. Requirement completeness, correctness, and traceability (among requirements at different level of abstraction, e.g., user- and system-level, as well as between requirement and design) is still to be verified by static manual analysis (e.g., inspection, checklist, walkthroughs/design reviews) and requirements engineering techniques [9]. For their validation, the V-Model encompasses the creation of acceptance tests, which give us feedback about user's needs and about what is really worth to prove in terms of system performance as well.
- A further concern is about the integration with OTS (Off-the-Shelf) components and/or legacy code; this is a common way of developing the large systems for ATC, for which MDA/MDT has a limited support.

  The defined flow supports only the test cases creation for OTS components (with some documentation support) at unit level and for their interaction with others in the architecture. This simplifies one task; but, the rest of OTS integration cycle, namely OTS searching, interface matching, adaptation, and integration strategy, must be managed separately in our V-Model.

IV. Example

We report an example of the model instantiation, developed in the context of the mentioned industry-university partnership. The industrial partner is currently running the eATMS long-term program aimed at designing a new generation of ATM/ATC systems. eATMS goals are: *i)* optimizing system deployment and maintenance, *ii)* achieving the performance required to manage the traffic increase, and *iii)* converging towards interoperability with other European ATM systems as required by the Single European Sky ATM Research project [10].

The ATC system subject of our case study is designed with a component-based approach. It has tens of thousands of requirements and it consists of many interacting deployable components, known as CSCIs (Computer Software Configuration Items). We report the application of the proposed approach to a sub-CSCI of the eATMS *Controller Working Position* component, named *Data Manager* (DTM). The DTM is our SUT; it is responsible for:

- managing the transition of Flight Data Objects (FDOs) from external source to the graphical user interface; they are composed by flights and air traffic data (e.g. weather information, altitude and coordinates of the flight);
- converting data in different standard format and storing them into a database;
- offering publishing/subscribing services for the FDOs.

DTM has about *seventy* requirements and it is meant to be used by other components.

For DTM development, we implemented MDA/MDT in the V-Model of Figure 1. We started from an available PIM that is transformed in PIT through M2M translation rules provided by *Test Conductor*™, a commercial plug-in of *IBM Rational Rhapsody*®.

A PIM-Software is designed with UML2 on the basis of software requirements specification. The high-level architecture (i.e., the static view) consists of 6 components:

1) **FDOStorageManager:** it manages the format conversion and the persistent storage of FDOs in a database;
2) **FDOWriterAdapter:** it manages the services to *modify* the FDOs during a *Writing Session* and uses the *FDOStorageManager* to do it;

3) **FDOPublisherAdapter:** it manages the services to *publish* new FDOs during a *Publishing Session* and uses the *FDOStorageManager* to do it;
4) **FDOReaderAdapter:** it provides services to read FDOs during a *Reading Session*, using the *FDOStorageManager* to retrieve the requested data;
5) **FDOSessionManager:** it manages sessions for external components to manipulate FDOs. There are three kinds of sessions, for writing, publishing and reading;
6) **FDOChangeNotificationCenter:** the DTM follows the publish/subscriber paradigm; the component has the role of *message broker*: it requests the *FDOStorageManager* to store the FDOs posted by *publishers* and notifies the *subscribers* about FDOs changes.

The dynamic view is described by UML2 statechart diagrams, manually verified against software requirements. At high level, the DTM component starts in an *Idle* state, waiting for a request of service that activates the transition in the *Busy* state. When the requested service is carried out without anomalies, it comes back into the *Idle* state, otherwise it transits into the *Error* state. From this, when recovery activities are performed, the DTM is restarted resuming to *Idle*.

The PIT-Software is automatically generated from the static view of DTM, by Test Conductor™ transformation rules. The dynamic view is used to generate the test cases with the criterion of covering all the states. A very simple example of generated test-case at this high level is shown in Figure 3; as we go at lower level, test cases become more and more complex.

Then, on the left side of the "V", we used the *Rhapsody*® translation rules to transform the PIM-Software into a PSM where the "Platform Specific" is intended in relationship to the specific implementation language, C++, and then to C++ source code. On the right side, we used the *ConformiQ*™ tool for getting to TTCN-3 scripts from test models. An example of the generated TTCN-3 script is in Figure 4. Finally, this kind of scripts are executed through *Elvior*© *TestCast*, which uses a SUT Adapter to specific APIs provided by the TTCN-3 Execution Environment (EE). The SUT Adapter, that we implemented in Java, cares about the communication between TTCN-3 scripts and the C++ implementation of the SUT.

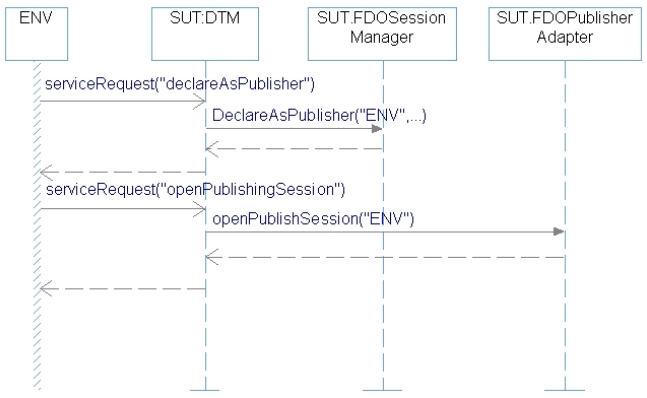


Fig. 3: Test case example

All the produced artefacts by this flow are included in the

```
testcase State_DTM_Idle_to_WritingSession() runs on Tester
     system SUT_adapter
{
  var float oldtimer := 0.0;
  var default default_behaviour_ref;
  start_test_case();
  default_behaviour_ref := activate(testerDefaultBehaviour
       ());
  send_ServiceRequest_to_input(DeclareAsPublisherTemplate1)
       ;
  oldtimer := 0.0;
  timeoutTimer.start(10.0 - oldtimer);
  alt
  {
    [] timeoutTimer.timeout {}
  }
  timeoutTimer.stop;
  send_ServiceRequest_to_input(
       OpenPublishingSessionTemplate1);
  setverdict(pass);
  deactivate(default_behaviour_ref);
  end_test_case();
}
```

Fig. 4: Generated TTCN-3 script example

documents foreseen by our V-Model for each design/testing phase.

V. Insights

In the middle of the story, we are now able to summarize what we see as the enablers for integrating V-Model and Model-driven development to get the most of both:

- Model-driven flow is essential on *both the sides of the "V"*; it allows for parallel evolution of artefacts, and favours cross-checking between corresponding activities at any given level of abstraction. This corroborates V-Model principles and favours transfer of knowledge and communication among teams.
- Procedures for integrating Model-driven development into customized processes are crucial and can bring significant benefits, as also remarked in [11]. However, one should neglect neither the start-up effort needed to set them up, nor the fact that they may need to be tailored for different systems.
- A very important role is played by tools. There is still poor interoperability among available tools; a one-for-all tool-chain does not exist yet. For instance, the *Rational Rhapsody*® tool and the related plugins (*Test Conductor*™ and *Automatic Test Generator*) cover a relevant slice of the depicted model, but part of the right side of the V (i.e., transformation into TTCN-3, to TTCN-3 test scripts, and then the TTCN-3 execution environment) should be implemented with others (examples for these tasks are *ConformiQ*™ tool-chain, *Elvior*© TestCast). We definitely believe it is worth building it, if we want to let Model-driven really penetrate into the reference industrial domain. Open source integrated alternatives (e.g., based on languages/tools in the Eclipse environment, such as *ATL transformation language* and *Acceleo*) would also be widely desirable.

A last consideration is on the opportunity of radical changes in this area. Besides social, cultural, or economic hurdles [12], we believe that industry and academia are still not ready. The former is, not without reasons, firmly anchored to consolidated processes and practices, which work well even if dated and not in line with modern technologies and paradigms. The latter, instead, misses real world application scenarios to make research real and practically assess methodologies and approaches [13]. We believe that concrete experiences in industrial settings are the missing link. In this domain, only more industrial examples would convince people, more than any theoretical evidence, that certain changes are possible and that are worth to be considered for improving the quality of delivered critical, large scale, software systems.

ACKNOWLEDGEMENTS

This work has been supported by MIUR under Project PON02_00485_3487758 "SVEVIA" of the public-private laboratory "COSMIC" (PON02_00669) and by Finmeccanica under the "Iniziativa Software" project. The work of Dr. Pietrantuono and Fucci is supported by the project Embedded Systems in Critical Domains (CUP B25B09000100007) within the framework of POR Campania FSE 2007-2013.

REFERENCES

[1] EUROCONTROL European Organisation for the Safety of Air Navigation, "Eurocontrol website", http://www.eurocontrol.int/articles/what-air-traffic-management
[2] U. S. D. of Defense, *MIL-STD-498 Overview and Tailoring Guidebook*, 1996.
[3] J. Mukerji and J. Miller, "MDA Guide Version 1.0.1", 2003; http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf
[4] P. Baker, Z. Dai, J. Grabowski, I. Schieferdecker, and C. Williams, *Model Driven Testing: Using the UML Testing Profile*, Springer, 2010.
[5] M. Mussa, S. Ouchani, W. Al Sammane, and A. Hamou-Lhadj, "A Survey of Model-Driven Testing Techniques". Proc. of the *International Conference on Quality Software*, IEEE Computer Society, 2009, 167-172
[6] OMG Std., *UML Testing Profile (UTP)*, OMG, 2012.
[7] C. Willcock, T. Deiß, S. Tobies, S. Keil, F. Engler, and S. Schulz, *An Introduction to TTCN-3*, John Wiley and Sons, 2011.
[8] H. Hemmati, A. Arcuri and L. Briand, "Achieving Scalable Model-Based Testing Through Test Case Diversity", *ACM Transactions on Software Engineering and Methodology*, to appear in vol. 22, no.1, 2013.
[9] E. Hull, K. Jackson and J. Dick, *Requirements Engineering*, Springer, 2010
[10] SESAR, "SESAR Joint Undertaking", http://www.sesarju.eu
[11] J. Hutchinson, M. Rouncefield, and J. Whittle,"Model-driven engineering practices in industry", Proc. of the *International Conference on Software Engineering*, IEEE Computer Society, 2011, 633-642.
[12] B. Selic, "What will it take? A view on adoption of model-based methods in practice", *Software & Systems Modeling*, Springer, vol. 11, no. 4, October 2012, pp. 513-526.
[13] L. Briand, "Embracing the Engineering Side of Software Engineering", *IEEE Software*, vol.29, no.4, 2012, pp. 96

Carrozza.jpg
Carrozza.jpg

Gabriella Carrozza, Ph.D., is currently leading the V&V team at SESM. Her main research interests are in dependability evaluation and assessment of complex software systems, as well as on the V&V of large critical systems. She is managing two projects in the SESAR research programme, aimed at developing novel ATC and Airport systems supervision to improve overall software quality and reliability. Since 2008, when she held her Ph.D. in Computer and Automation Engineering at the Federico II University of Naples, she has been serving as reviewer and PC member of several conferences and journals in the dependable systems research community.
Contact her at: gcarrozza@sesm.it

Faella.jpg Faella.jpg

Mauro Faella is a R&D Software Engineer at Critiware S.r.l.. He is also consultant at SESM. His research interests include the model-driven approaches and practices in testing activities of critical systems. He has an MS in computer engineering from the Federico II University of Naples, Italy.
Contact him at mauro.faella@critiware.com

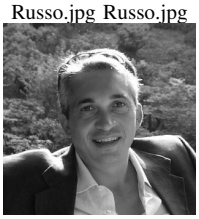
Fucci.jpg Fucci.jpg

Francesco Fucci is a PhD student in Computer and Automation Engineering at the Federico II University of Naples. He received his M.Sc at the same university in 2011. He was also consultant at SESM, where he worked in the V&V team for projects in the field of Air Traffic Management.
Contact him at: francesco.fucci@unina.it

Pietrantuono.png
Pietrantuono.png

Roberto Pietrantuono, PhD, IEEE Member, is currently a post-doc at Federico II University of Naples. He collaborates with several companies of the Finmeccanica group, in the field of critical software system development. His research interests are in the area of software engineering, particularly in the software verification of critical systems, software testing, and software reliability. He received his PhD degree (2009) in Computer and Automation Engineering from the Federico II University of Naples, Italy.
Contact him at: roberto.pietrantuono@unina.it

Russo.jpg Russo.jpg



Stefano Russo is Professor and Deputy Head at the Department of Computer and Systems Engineering, Federico II University of Naples. He is Chairman of the Curriculum in Computer Engineering, and Director of the "C. Savy" Laboratory of the National Inter-Universities Consortium for Informatics (CINI). His research interests are in the areas of distributed software engineering, middleware technologies, and dependable software systems. He coordinates the national project DOTS-LCCI on software dependability for critical infrastructures. He has (co-)authored more than 120 scientific papers.

Contact him at: stefano.russo@unina.it