Integrating MDT in an Industrial Process in the Air Traffic Control Domain

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

*SESM scarl, a Finmeccanica company
Via Circumvallazione Esterna di Napoli, 80014 Napoli, Italy
gcarrozza@sesm.it

†Critiware S.r.l., Incubatore Incipit
Complesso Universitario di M. S. Angelo, Via Cintia, 80126 Napoli, Italy
mauro.faella@critiware.com

‡Dipartimento di Informatica e Sistemistica (DIS)
Università di Napoli Federico II
Via Claudio 21, 80125 Napoli, Italy
(francesco.fucci,roberto.pietrantuono,stefano.russo)@unina.it

***Abstract*—Air Traffic Control (ATC) systems are typical software-intensive mission-critical systems with stringent dependability requirements. The major providers of ATC systems are system integrators that address such requirements at the cost of a very expensive testing effort. They envisage Model Driven Testing (MDT) as a promising approach to reduce this effort while achieving better product quality. Within the context of a public-private partnership for software innovation in the ATC domain, we address the problem of integrating MDT into a software development process based on Model Driven Architecture. Specifically, we propose a solution to the integration of MDT into a V-model, focusing on a parallel MDA-MDT flow in a real industrial software process.**

***Keywords*-MDA; MDT; Testing automation**

I. Introduction

A civil *Air Traffic Control* (ATC) system is a typical software-intensive mission-critical system, that plays a key role in Air Traffic Management (ATM) [1]. It provides facilities and services to ground controllers and pilots for managing safely ground and en-route flight operations, with the aim of preventing collisions, organizing the flow of traffic, and providing support information to operators and pilots. In Europe, for instance, en-route ATC is segmented in several Area Control Centers (ACCs), each responsible for a defined portion of the air space, with ATC systems in ACCs cooperating to guarantee the safety of flights.

The software dependability requirements for this type of large-scale critical systems are very stringent, as failures cannot be admitted because of the impact on consumer dissatisfaction, economical losses and even people's lives. These requirements can be satisfied at the cost of a very thorough development and testing process.

Model Driven Architecture (MDA) [2]–[4] is a software design approach being increasingly envisioned by mission-critical systems providers as a key technique to be incorporated into their software development life-cycle (SDLC), as it promises to be highly beneficial in terms of software quality and of cost reduction. However, the integration of modern model-driven techniques into existing industrial software processes for critical systems poses several problems, which make MDA be still far from being fully mature and commonly adopted in application domains such as ATC.

As testing represents the most expensive part of SDLC for critical systems [5], model-driven techniques are seen with much favor by industries for testing too. Similarly to MDA for design and development, Model Driven Testing (MDT) [6] is a promising approach to cost reduction. Its greatest benefit is claimed to be the ability of reducing the cost of testing by automatically generating and (possibly) executing the test suite from models representing the desired system behavior. Although MDA and MDT are based on similar concepts, they are currently not fully integrated, and they may indeed be applied independently. In the industrial practice models for MDT are often realized manually or by partial reuse of the (MDA) design models (e.g., adding to UML models stereotypes or profiles such as the UML 2 Testing Profile [7]).

A relevant problem for providers in the ATC field is the integration of MDT into existing MDA-based processes in the critical systems domain. We are addressing this problem within the context of a public-private laboratory[1]. In this paper, we propose a solution to the integration of MDT with MDA into a V-model process, which is a typical model in

[1]The "Iniziativa Software" cooperation is a network of public-private laboratories between Finmeccanica companies and Italian universities.

the given application domain[2], focusing on the parallelism of MDA and MDT workflows. The goal is to favor a parallel evolution of development and testing so as to support the automated generation of test artifacts along with system and software artifacts, and early testing of development artifacts.

The integration of MDT and MDA has been addressed recently in the scientific literature in other application domains [8]. Born et *al.* adapted an existing model-driven method for information systems, named *KobrA*, to telecommunication systems, by combining MDA, UTP and TTCN-3 [9]. Tielin et *al.* have integrated the MDT approach in an MDA design in the domain of web applications [10]. Yang et *al.* proposed a method for model transformation from the PIM (specified in UML) to the PIT (in U2TP), and subsequent generation of JUnit test cases from the PIT, showing, again, the application to a web system [11].

The paper is organized as follows. In Section II we give a brief background on MDA and MDT. In Section III we describe what we believe is a relevant challenge for software development companies and system integrators in the ATC domain; we consider this representative of the larger domain of large-scale mission-critical systems. Section IV describes the proposed solution and some technical aspects related to the needed support tools. Section V focuses on the case study. Section VI contains the conclusions.

II. Background

MDA and MDT share the goals of portability, interoperability and reusability [6], [12]. These are achieved by means of *i)* models specification (at various level of abstraction), and *ii)* models transformation. The following models are specified or derived through an iterative approach:

- the *Platform Independent Model (PIM)* defines a high-level *system model* to fulfill *system requirements*; it is specified in a domain-specific modeling language;
- the *Platform Independent Model for software (PIM-Software)* is the high-level model of the system's *software architecture* designed to fulfill *software requirements*. It is specified through a software modeling language in a way independent from the target execution platform and from implementation technologies;
- the *Platform Specific Model (PSM)* is the low-level (technology-dependent) software design model for the specific target platform. The PSM takes into account design choices related to the target execution platform, such as computing architecture, operating systems and middleware technologies; in MDA it can be derived from the PIM through model transformations;
- the *Platform Independent Test model (PIT)* is the high-level model of the test architecture for the system described by the PIM; it can be derived from the PIM;
- the *Platform Independent Test model - Software (PIT-Software* is the high-level software testing model, consisting of the test architecture for the software System Under Test (SUT). It is complemented by *test cases* (TCs), to be automatically transformed into text scripts;
- the *Platform Specific Test model (PST)* is the low-level model of the test architecture for the specific target platform. It is automatically derived from the PIT-Software and TCs through M2M transformations, meta-models and PSM information.

Model transformation is done through specific translation rules and tools. There are two types of transformations: *i)* Model-To-Model (M2M), to convert a model into another model, and *ii)* Model-To-Text (M2T), to convert a model into a textual artifact (e.g. source code or script).

III. Industrial Challenge

The eATMS long-term program is being run by our industrial partner in the public-private laboratory to design a new generation of ATM/ATC systems. eATMS goals include: *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 (SESAR) project[3]. The main eATMS non-functional requirements concern:

- **Dependability**, to provide continuous availability and integrity;
- **Robustness**, to prevent failures in case of anomalous operating conditions;
- **Changeability**, to support long-term evolution and integration/interoperability with other systems, as well as quick response to changes in operating environments;
- **Performance**, to support the air traffic increases in European skies.

To fulfill functional as well as non-functional requirements, the adoption of modern yet mature engineering methods, modeling languages, standards, tools and technologies is envisaged. They include: object- and component-oriented analysis/design (OOA/D, CBSE) methods and programming languages, Service-Oriented Architectures (SOA), Unified Modeling Language (UML), Systems Modeling Language (SysML), Model Driven Architecture, Model Driven Testing, TTCN-3 (Testing and Test Control Notation) [13]. Additionally, there may be constraints on the development tools to be adopted, depending on industrial strategies and investments and on customers requirements.

Most of development processes adopted in the domain of interest are based on the V-Model; we refer to the model shown in Figure 1. We classify the main artifacts produced by the activities in two categories: *i)* design artifacts and *ii)*

[2]A major system integrator in the ATC domain is Selex Sistemi Integrati, a Finmeccanica company, partner of the cited public-private cooperation.

[3]SESAR (www.sesarju.eu) is one of the most ambitious R&D projects ever launched by the European Community.

testing artifacts. They typically comply to the MIL-STD-498 standard [14]. Design artifacts include:

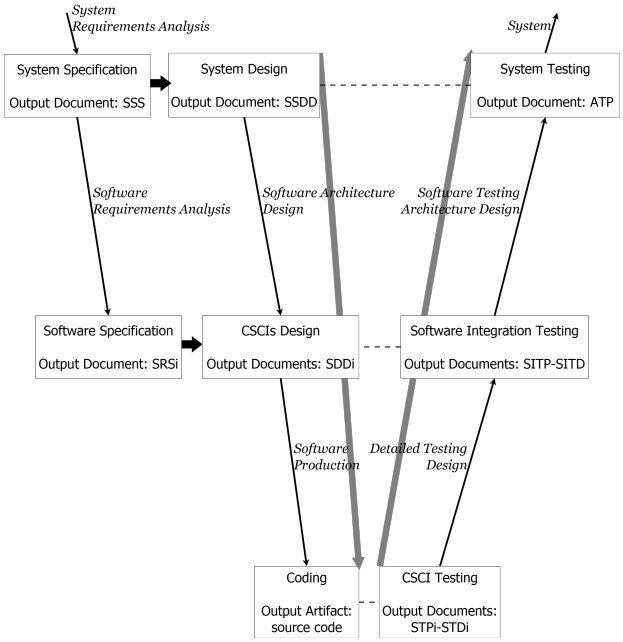

Figure 1: The reference V-Model process

- **System/Subsystems Specification (SSS)**, specifying the system requirements, grouped for a number of coarse-grained functionally identified domain subsystems; this is complemented by the *Interface Requirements Specification* (IRS) describing the system external interfaces and related data model;
- **System/Subsystems Design Description (SSDD)**, with the high-level architecture of the envisaged solution, and the allocation of requirements to its subsystems;
- **Software Requirements Specification (SRS)**, specifying the software requirements for each component of the software system. Deployable components are *Computer Software Configuration Items (CSCI)*, as defined in [14]; an SRS document is produced for each CSCI. Each SRS is complemented by an *Interface Control Document* (ICD) that specifies the CSCI external interfaces and related data model;
- **Software Design Description (SDD)**, describing the internal design of a CSCI, and the allocation of software requirements to its internal subcomponents. An SDD document is produced for each CSCI. Each SDD is accompanied by an *Interface Design Document* (IDD) that specifies the CSCI internal (subcomponents') interfaces and the related data.

Testing artifacts include:

- **Acceptance Test Plan (ATP)**, that specifies the plan and test-cases of the acceptance testing;
- **Software Integration Test Plan (SITP)**, describing the plan for the integration tests;
- **Software Integration Test Description (SITD)**, that specifies the test cases for integration tests;
- **Software Test Plan (STP)**, describing the plan for testing a CSCI;
- **Software Test Description (STD)**, that specifies the test cases for a CSCI.

The flow of MDA artifacts (PIM, PIM-Software and PSM) has to be integrated into the above artifacts and the related workflow. We believe the PIM should be integrated into the SSDD, while the PIM-Software and the PSM into the SDDs. Moreover, the Verification and Validation (V&V) team is challenged to integrate the MDT approach in MDA in order to reach the following goals: *i)* early testing, *ii)* automatic generation of test-cases based on coverage criteria and *iii)* integrating the MDT artifacts in the existent documentation.

IV. Proposed Solution

The proposed process is outlined in Figure 2. It starts with the *system requirements* analysis performed by domain experts, followed by two parallel activities: *i)* the creation of the PIM, and *ii)* the specification of software requirements.

The PIM is described with the following diagram types:

- *Requirements diagram*, specified in SysML;
- *Components diagram*, modeling the relationships among components;
- *Data model*: describing the data managed by the system; these can be categorized into external data (exchanged with external actors), and internal data (exchanged among subsystems);
- *State diagram*, describing the behavior of components in terms of finite state machines.

The PIM is used to generate the PIT. The latter is described in UML Testing Profile (UTP) [7], as it is a standard for the definition and specification of test suites in the given domain. The PIM is also transformed into the PIM-Software using the Software Requirements. The PIM-Software, described in UML, consists of two complementary views:

1) the *static view* describes entities and their structural relationships;
2) the *dynamic view* describes the run-time behavior.

The next step is the generation of PIT-Software and test cases. This task transforms the static view of the PIM-Software into the PIT-Software through M2M transformation. The dynamic view is used to generate the test cases using specific coverage algorithms on the basis of the behavioral diagram. At this point, we are able to generate the PSMs using a M2M transformation. Depending on the selected platform, the right set of M2M translation rules needs to be used. The PST is generated in TTCN-3 notation through M2M transformations. We choose TTCN-3 because

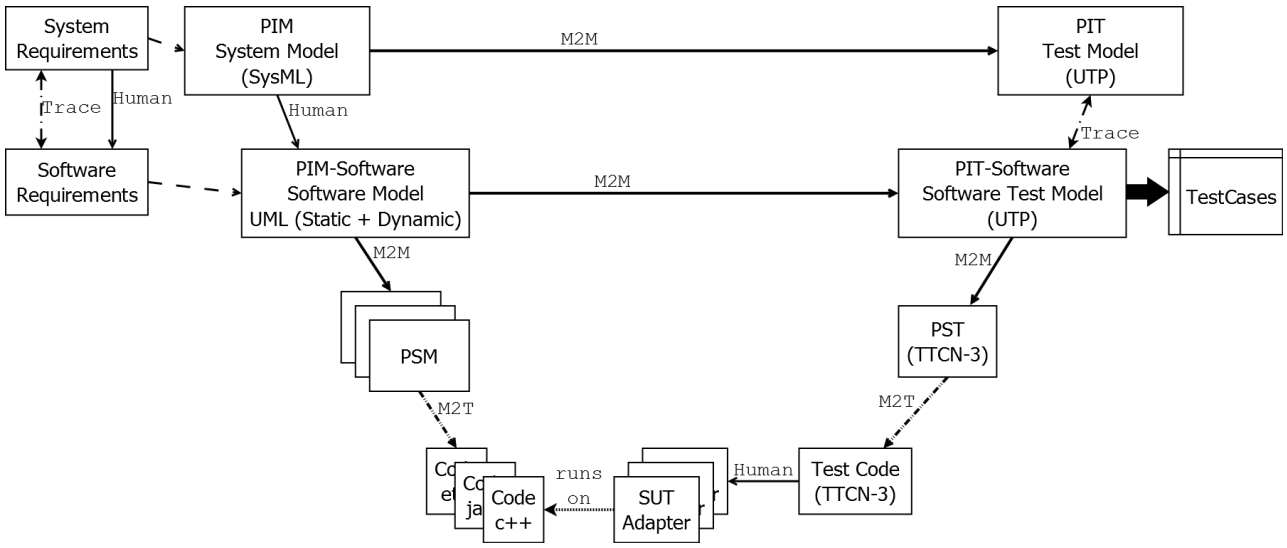

Figure 2: Overview of the proposed process

we can use one PST for different PSMs. The next part of the process concerns with the following procedures:

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

The last step is the execution of the *Test Suite* on the SUT. This is done in a specific TTCN-3 run-time environment by means of the SUT Adapter previously created.

The automatically generated blocks of Figure 2 are:

- the PSMs;
- the code;
- the PIT (using UTP);
- the PIT-Software and test cases (using UTP);
- the PST (using TTCN-3);
- the TTCN-3 scripts.

The blocks that have to be manually created are:

- Software Requirements;
- Software Model;
- SUT Adapter.

The generation of PIT, PIT-Software and TestCase can be done using IBM Rational Rhapsody®. In particular, the PIT is generated automatically in UTP starting from the PIM. PIT-Software and TestCases (Test Model + Test Suite) can be generated using two plugins provided by Rhapsody, respectively Test Conductor and Automatic Test Generator.

V. Case study

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 CSCIs. We describe here the application of the proposed approach to a sub-CSCI of the Controller Working Position, a component of eATMS, named *Data Manager* (DTM). The DTM, which is our SUT, 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. We started from an available PIM that is transformed in PIT through M2M translation rules provided by Test Conductor™, a commercial plugin of IBM Rational Rhapsody® (the design tool required by the customer).

A prototypal PIM-Software is designed with UML on the basis of the SRS. The static view is (partially) shown in Figure 3; it 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 FlightDataStorageManager to do it;
3) **FDOPublisherAdapter:** it manages the services to *publish* new FDOs during a *Publishing Session* and uses the FlightDataStorageManager to do it;
4) **FDOReaderAdapter:** it provides services to read FDOs during a *Reading Session*, using the FlightDataStorageManager to retrieve the requested data;
5) **FDOSessionManager:** it manages sessions for exter-

nal 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.

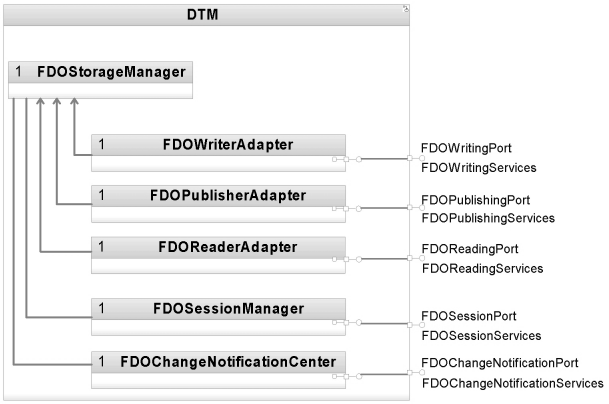

Figure 3: Static view of DTM

The dynamic view (Figure 4) is described with UML state-chart diagrams. 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. In the latter state, recovery activities are performed and the DTM is restarted resuming into the *Idle* state.

The PIT-Software is automatically generated from the static view of DTM, through the translation rules offered by Test Conductor™. The dynamic view is used to generate the test cases with the criterion of covering all states. An example of generated test-case is shown in Figure 5. The next step of the proposed process is to generate the PSM and the PST models. We used the Rhapsody® M2M translation rules to transform the PIM-Software into a PSM where the "Platform Specific" is intended in relationship to the specific implementation language C++. The PST is compliant with TTCN-3 specification.

Finally the system model is transformed into C++ source code and the TestModel is transformed into TTCN-3 scripts. An example of the generated TTCN-3 script is shown in Figure 6. This kind of scripts are executed through Elvior© TestCast that needs a SUT Adapter to use specific APIs provided by the TTCN-3 Execution Environment (EE). The SUT Adapter allows the communication between TTCN-3 scripts and the C++ implementation of the SUT.

VI. Conclusions and Future Work

In the field of large-scale mission-critical systems - systems with tens of thousands of requirements and millions

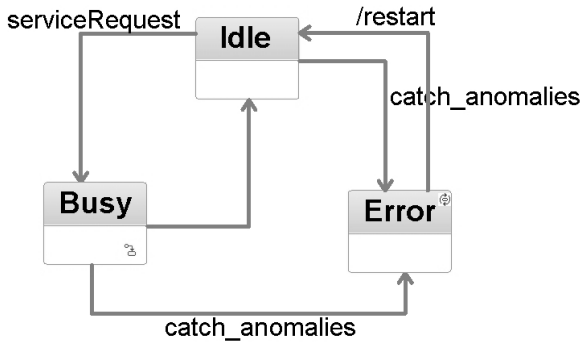

Figure 4: Dynamic view of DTM

of lines of code - the adoption of emerging model-driven software engineering techniques is a felt need, as we witness in the ATC application domain we cope with in a public-private cooperation. The road however is still long. Evidence is required on how these techniques can be seamlessly introduced in well-established industrial processes, on the effectiveness of proper support tools and technologies, and on the impact on quality, cost reduction and time-to-market.

Since testing accounts for a large part of development costs, MDT techniques are of great importance for companies in the domain. The benefit of MDT is typically claimed to reside in the potential for automation, i.e. the automatic generation (and - possibly - execution) of test cases from a model of the SUT. We consider this only a partial key to the successful introduction of MDT in the domain of interest. As testing is a major quality assurance techniques, and quality has to accompany the whole process, we believe the benefit of MD approaches are best achieved if MDT proceeds closely in parallel to MDA design and development, so as to enable early testing of actual design artifacts, to assess their effectiveness in fulfilling functional as well as non functional requirements, and early discovery of design flaws, not only of defects in the code.

To this aim, we have proposed a solution to the integration

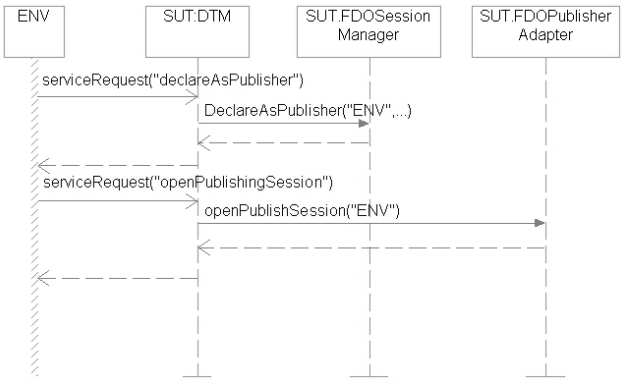

Figure 5: Test case example

```
testcase State_DTM_Idle_to_WritingSession() runs on Tester
      system SUT_adapter
{
  var float oldtimer := 0.0;
  var default default_behaviour_ref;
  var boolean response1;
  var boolean response2;
  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();
}
```

Figure 6: Generated TTCN-3 script example

of MDT into a typical industrial V-model engineering process, favoring parallel progress of design and test, automated generation of tests from actual design models, reuse of test artifacts for different PSMs, and reuse of test scripts templates.

The actual level of automation of model transformations and test cases generation in the domain of our interest needs to be further investigated. We have also seen there is still a poor interoperability among available support tools; yet the precise definition of the workflow in all technical details and the availability of a proper tool-chain is a key factor for the success in industrial settings. Clearly, the proposed solution needs to be fully experimented for a thorough assessment. This is the objective of our future work.

ACKNOWLEDGMENT

This work has been partially supported by MIUR within the PRIN 2008 Project DOTS-LCCI: Dependable Off-The-Shelf based middleware systems for Large-scale Complex Critical Infrastructures, and within the Project PON02_00485_3487758 "SVEVIA"; by Finmeccanica within the Iniziativa Software network of public-private laboratories with CINI. The work of Dr. 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 website. [Online]. Available: http://www.eurocontrol.int/articles/what-air-traffic-management

[2] D. Frankel, *Model Driven Architecture: applying MDA to enterprise computing*. Wiley, 2003.

[3] M. Jiang and Y. Zhihui, "A model-driven approach for dependable software systems," in *Quality Software, 2007. QSIC '07. Seventh International Conference on*, pp. 100–106.

[4] W. Ke, X. Li, Z. Liu, and V. Stolz, "rcos: a formal model-driven engineering method for component-based software," *Frontiers of computer science in Chine*, vol. 6, pp. 17–39, 2012.

[5] M. Pezzé and M. Young, *Software testing and analysis: process, principles, and techniques*. Wiley, 2008.

[6] P. Baker, Z. Dai, J. Grabowski, I. Schieferdecker, and C. Williams, *Model Driven Testing: Using the UML Testing Profile*. Springer, 2010.

[7] *UML Testing Profile (UTP)*, OMG Std., Rev. 1.1, 2012. [Online]. Available: http://www.omg.org/spec/UTP/1.1/PDF

[8] M. Mussa, S. Ouchani, W. A. Sammane, and A. Hamou-Lhadj, "A survey of model-driven testing techniques," *Quality Software, International Conference on*, pp. 167–172, 2009.

[9] M. Born, I. Schieferdecker, H. gerhard Gross, and P. Santos, "Model-driven development and testing - a case study," in *University of Twente*, 2004, pp. 97–104.

[10] Q. Tielin, L. Yafen, and W. Pu, "A model transformation platform design based on model driven architecture," *Intelligent Computation Technology and Automation, International Conference on*, vol. 1, pp. 175–179, 2010.

[11] L. Yang, L. Yafen, and W. Pu, "Design and implementation of automatic generation of test cases based on model driven architecture," in *Information Technology and Computer Science (ITCS), 2010 Second International Conference on*, pp. 344–347.

[12] *MDA Guide Version 1.0.1*, OMG Std., Rev. 1.0.1, 2003. [Online]. Available: http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf

[13] C. Willcock, T. Dei, S. Tobies, S. Keil, F. Engler, and S. Schulz, *An Introduction to TTCN-3*. John Wiley and Sons, 2011.

[14] U. S. D. of Defense, *MIL-STD-498 Overview and Tailoring Guidebook*, 1996. [Online]. Available: http://www.abelia.com/498pdf/498GBOT.PDF