

Combined Simulation and Testing Based on Standard UML Models

Vitali Schneider, Anna Deitsch, Winfried Dulz and Reinhard German

Abstract The development of complex software and embedded systems is usually composed of a series of design, implementation, and testing phases. Challenged by their continuously increasing complexity and high-performance requirements, model-driven development approaches are gaining in popularity. Modeling languages like UML (Unified Modeling Language) cope with the system complexity and also allow for advanced analysis and validation methods. The approach of Test-driven Agile Simulation (TAS) combines novel model-based simulation and testing techniques in order to achieve an improved overall quality during the development process. Thus, the TAS approach enables the simulation of a modeled system and the simulated execution of test cases, such that both system and test models can mutually be validated at early design stages prior to expensive implementation and testing on real hardware. By executing system specifications in a simulation environment, the TAS approach also supports a cheap and agile technique for quantitative assessments and performance estimates to identify system bottlenecks and for system improvements at different abstraction levels. In this chapter we will present the current status of the TAS approach, a software tool realization based on the Eclipse RCP, and a detailed example from the image processing domain illustrating the methodology.

V. Schneider (✉) · A. Deitsch · W. Dulz · R. German
Computer Science 7, Friedrich-Alexander-University of Erlangen-Nürnberg,
Martensstr. 3, D-91058 Erlangen, Germany
e-mail: vitali.schneider@fau.de

A. Deitsch
e-mail: anna.deitsch@fau.de

W. Dulz
e-mail: dulz@cs.fau.de

R. German
e-mail: german@cs.fau.de

1 Introduction

Rapid and efficient development of complex hardware and software systems for telecommunication, automotive, or medical applications needs support from dedicated tool chains and customized modeling environments. Model-driven engineering (MDE) [1] is a promising approach to address the complexity that is inherent in each technical system. MDE is combining two technologies that may help to overcome the complexity hurdle:

- Domain-specific modeling languages (DSML) focus on particular application domains like automotive or telecommunications.
- Transformation and generation mechanisms support the analysis of specific model artifacts in order to generate simulation or source code and alternative model representations. An automated transformation process also ensures the consistency between application development and the assurance of functional and extra-functional requirements like timing aspects, reliability, or performance issues captured by model artifacts.

By providing different abstraction levels and distinct model types, which are standardized by the Object Management Group (OMG) in the Unified Modeling Language (UML) [2], the OMG Model-Driven Architecture (MDA) [3] offers basic concepts and techniques to specify a system independently of the platform that supports it:

1. Starting from a Computation Independent Model (CIM) that specifies the system functionality without showing constructional details,
2. a Platform-Independent Model (PIM) is derived by adding architectural knowledge, at the same time hiding details of the platform used.
3. Finally, a Platform-Specific Model (PSM) arises when all elements and services are added that complete the target platform.

The main advantage of having different views of the same system (see Fig. 1) is to include domain and business experts, as well as software architects and IT

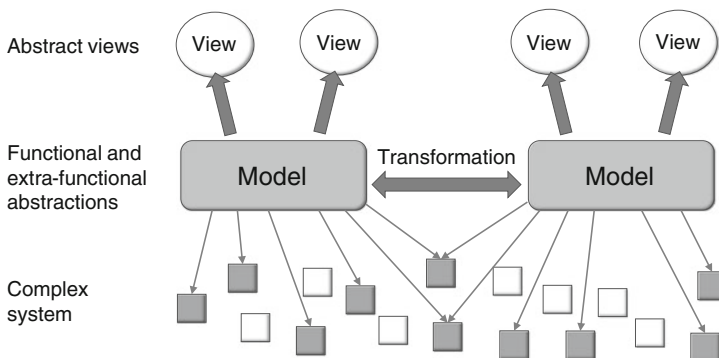


Fig. 1 Relationship between views, models, and system (Reproduced from [1])

specialists in the overall development process, whereby dedicated models focus on different tasks. In an iterative modeling process model transformations support the refinement of the models up to a formal, precise, and detailed specification that will be the basis for implementing and testing the system.

In order to handle specific issues, OMG provides standardized UML profiles that can be combined with normal UML models. For example, the profile for System Modeling Language (SysML) [4] enables modeling of requirements for embedded systems in greater detail. Extra-functional properties, time, and analysis details are expressed by the profile for Modeling and Analysis of Real-time and Embedded systems (MARTE) [5] profile, while UML Testing Profile (UTP) [6] is used to consider testing aspects.

Test-driven Agile Simulation (TAS) [7] intends to improve the overall quality of the development process by combining UML-based engineering, simulation, and testing techniques [8]. The TAS approach assists the transformation of specification models to executable simulation code and uses standardized model-to-text transformation methods. By simulating a given system and running tests on it, TAS provides an agile technique to validate specification models at an early stage of the development process. To express extra-functional requirements and testing details for embedded systems, TAS offers the possibility to integrate model extensions that conform to the SysML, MARTE, and UTP profiles.

This practice is also applied in recent industrial design and development processes, and was evaluated in the European research project COMPLEX [9] for building complex systems. In contrast to the methodology used in COMPLEX, the TAS approach concentrates the verification and validation (V&V) activities on simulation and testing and also involves the UTP profile.

Because UML profiles may handle the same modeling aspect in different ways, model interferences and inconsistencies may appear. For instance, SysML and MARTE provide quite different approaches for the specification of quantitative values, e.g., for time or duration. We therefore showed in a recent paper how to avoid model interferences for test-driven agile simulations based on standardized UML profiles [10].

Avoiding model interferences was also a topic that has been tackled within the EU funded MADES [11] project. Using the MADES language it is possible to restrict SysML and MARTE profiles to a consistent model subset. V&V activities during the development cycle focus on analyzing temporal properties of the components instead of testing. Hence, validation in the context of MADES is mostly time-related.

Since the benefits of standardized graphical languages and simulation-based testing have been recognized, several efforts have been undertaken in that area. In this context, mention can be made of the tool environments Matlab/Simulink¹ or SCADE² that are often used in practice for the development of embedded systems. Both also offer solutions for the integration of UML-/SysML-based modeling techniques within a combined simulation and test environment. However, the tools used

¹<http://mathworks.com/products>.

²<http://www.esterel-technologies.com/products>.

are still very much tied to their own proprietary modeling languages. The notations used in this case primarily support synchronous data flow-oriented paradigms. Thus, both tool environments are rather more suitable for the development of control-oriented systems. In contrast, TAS is seamlessly based on the standardized UML as a more expressive and flexible common modeling language, which can also consider nondeterministic, asynchronous systems.

Furthermore, numerous efforts have been considered so far to develop techniques for deriving quality of service (QoS) properties from UML specifications using diverse analytical techniques. For instance, in [12] the authors propose a framework to transform UML models that are annotated, among others, with stereotypes from the MARTE profile to Stochastic Reward Nets (SRNs). These SRNs are evaluated with the software package SHARPE³ [13] to obtain performability results. In situations where the complexity of the system specification allows the application of analytical techniques, these results will certainly provide a valuable insight in the system under development. In contrast, however, the TAS approach aims on the simulation-based, test-driven development and evaluation of complex systems.

In the following sections, we provide a modeling methodology for the TAS approach by combining standardized UML profiles. We focus on strategies that will avoid model interferences caused by overlapping specification parts that are described by different UML profiles. We also show how tracing of functional and extra-functional requirements can be achieved in the SimTAny tool environment.

2 Test-driven Agile Simulation

In this section, we introduce the concept of TAS. We start with the motivation for the suggested approach, describing its main idea and the basic concept. Then, we outline the most important features covered by TAS.

2.1 *Idea and Concept of TAS*

The development process of complex software and embedded systems usually consists of a series of design, implementation, and testing phases, aligned to some formal process model. Despite a large number of different process models, the development typically starts with requirement definition followed by several specification, programming, and testing steps. Due to a continuously increasing complexity of systems, the approaches based on formal modeling languages like UML are gaining in popularity. On one side, modeling with graphical diagrams helps to deal with the complexity. On the other side, formal specifications enable automated derivation of the implementation code as well as of advanced analysis and validation capabilities.

³<http://sharpe.pratt.duke.edu>.

With our TAS approach, initially introduced in [7], we propagate the combination of model-driven simulation and testing techniques to achieve an improved overall quality during the development process. Thus, our approach enables to derive executable simulations from UML-based specifications of both the system and test models in order to analyze a modeled system and to perform simulated tests on it at early stages of the development process. This approach supports a cheap and agile technique for design error detection as well as for first quantitative assessments and performance estimates. Even prior to expensive implementation and testing on a real system, potential drawbacks and bottlenecks in the system can be identified by the use of simulation. By means of simulation it is also easily possible to investigate and compare alternative designs and solutions at the level of models. Furthermore, the early validation of the specification models helps to reduce development risks. In order to achieve mutual validation of the system and test specifications, we suggest starting with the specification of requirements and then to derive system and test specifications independently and in parallel to each other from these common requirements (see Fig. 2).

Solely based on UML and using its standardized extension profiles, the TAS approach provides for the application of one common modeling language for different design stages like requirements, system, simulation, and test design. Amongst several obvious advantages of having a common modeling language, like simplifying the communication between team members of different disciplines, ability to use only one modeling tool, and cost savings, it also contributes to the quality improvement. Thus, the traceability between relevant model parts can be easily created and examined. Furthermore, in distributed development teams and processes it is particularly important to ensure a joint understanding of definitions and relations across different

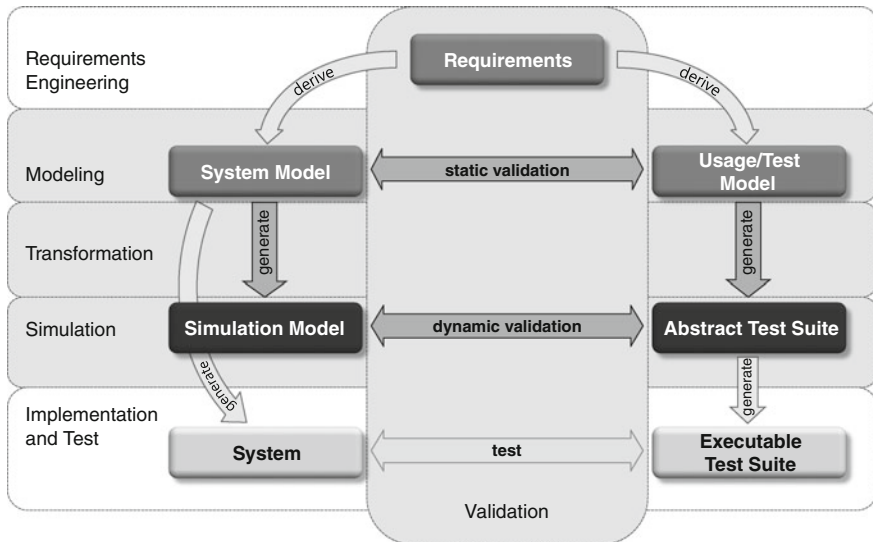


Fig. 2 Concept of test-driven agile simulation (Reproduced from [14])

disciplines. Although specialized tools and languages have been established for each discipline, it is nevertheless possible to use common UML-based specifications as central documents shared during the development process, for instance, by applying integration and transformation solutions as provided by ModelBus.⁴

2.2 Main Features

The TAS approach makes provisions for various aspects of the model-driven development process. Among others, it covers modeling, model-based validation, simulation code generation, and test as well as integrability and extendability for different domains and development environments. In the following, we will shortly introduce some of the main features of TAS (see also Fig. 2).

Modeling

As already mentioned, our approach is widely based on UML as a common modeling language. Due to its general nature, UML is principally suitable for modeling in several development stages, like requirements, system, simulation, and test modeling, which are addressed by the TAS approach. However, for specifications of domain-specific aspects we propose a UML-based modeling methodology, which utilizes several standardized UML extension profiles. As will be shown in detail in Sect. 3, we apply a combined subset of OMG's SysML, MARTE, and UTP profiles. In particular, we use basic elements and diagrams of UML and apply SysML to specify requirements, system blocks, port directions, and traces between requirements and other model elements. A number of stereotypes of several sub-profiles of MARTE are used, for example, to characterize analysis aspects and extra-functional properties, to introduce nondeterminism, and to describe HW/SW allocation. UTP profile is utilized to represent parts of the test model, like test contexts, test components, and test cases. Additionally, our modeling methodology allows for use of the textual action specification language ALF [15] and its library, which provides useful collection types and operations.

However, the combination of different specialized profiles involves special challenges caused primarily by a number of semantic and syntactic overlappings between different profiles. To overcome these challenges, we suggest a strategy of selective combination of proper subsets of profiles presented in our previous work [10].

Verification and Validation

In order to improve the quality of a developed system and to increase the efficiency of the development process itself, it is reasonable to perform verification and validation activities as soon as possible. Our TAS approach enables validation of model-based specifications at very early stages of the design phase. In addition to the known verification and model checking methods, independent formal system and test models derived from the same requirements can serve for their mutual validation.

⁴<http://www.modelbus.org>.

At this point, our approach distinguishes between static and dynamic validation. Static validation can be applied directly on the specification models of both the system and the test models. It consists of examination of constraints, which relates to the static modeling aspects like structure, naming, constraints definitions and traceability to requirements.

On the other side, dynamic validation aims to inspect the dynamic behavior of the modeled system and its corresponding test model. On the level of models the behavior can be validated by executing test cases from the test model on the simulated system model.

Transformation to Simulation Code

Since simulation plays an important role for validation and system's behavior analysis in TAS, our approach assists an automated transformation of the specification models to the executable simulation code. It consists of generation of the simulation code from system models as well as from test models. Thereby, in the latter case we speak about abstract test suites to differentiate the simulated test suites from those derived for real tests on the implemented system.

Referring to the OMG standard for model-to-text transformations MOFM2T [16], a standardized transformation method can be applied to transform structural, behavioral, and analytical elements from UML models to appropriate representations in the desired simulation environment. The definition of such compatible mappings poses the biggest challenge on this stage.

Above all, of course, discrete-event simulation tools are eligible for the purpose of simulation due to the original time-discrete nature of UML and of most computing systems. Thus, the focus of our approach is primarily on supporting the transformation of time-discrete models for discrete-event simulators (see Sect. 4.3). Nevertheless, with some limitations time-continuous aspects and systems may also be represented with SysML and transformed to appropriate simulation tools, as shown for instance in [17].

Simulation

On one side, the simulation code derived from a system model can be utilized for design and performance analysis of the whole system. The generated simulation represents in this case all active components of the system with their reproduced behaviors. Running a simulation of the system one can first investigate its dynamic behavior. Depending on the simulation tool, even interactive or stepwise execution could be possible, which is particularly helpful for debugging. At the end of or even during the simulation, predefined analytical values could be assessed. By modeling different design solutions as specific parameter configurations, simulation tools can provide support for parameter variation and searching for optimal solutions with respect to the specified requirements.

Test

On the other side, abstract test suites generated from a test model serve for the simulated execution of tests. A test suite corresponds at the model level to the UTP's test context, which describes the configuration of a test consisting of a system under test (SuT) with surrounding test components and contains a number of test cases.

Depending on the type of a test, i.e., unit, integration, or system test, the SuT may either be one system component, a set of components, or the whole system, respectively. A test case is thereby usually represented as a sequence of interactions between a SuT and test components.

After the transformation to a simulation code an abstract test suite largely consists of simulated test components, which can create stimuli for the SuT and evaluate its responses by comparing them with the expected results. While the behavior of test components is simulated according to the currently executed test case, the behavior of the SuT is coming from the embedded simulation code generated from the system model. It is the task of each test component to determine its local verdict according to the expected responses from the SuT. The verdict of a test case is then composed of local verdicts of all containing test components. A failed or inconclusive test case first indicates some inconsistencies in the system or test model, or event in the original requirements model and requires a closer inspection of these models.

Analysis

The output provided by the simulation runs helps to assess different design solutions or to predict the performance of a developed system. However, a comprehensive statistical analysis of simulation results is often also needed. In order to facilitate this task, our approach provides support for convenient calculation and visualization of some basic measures.

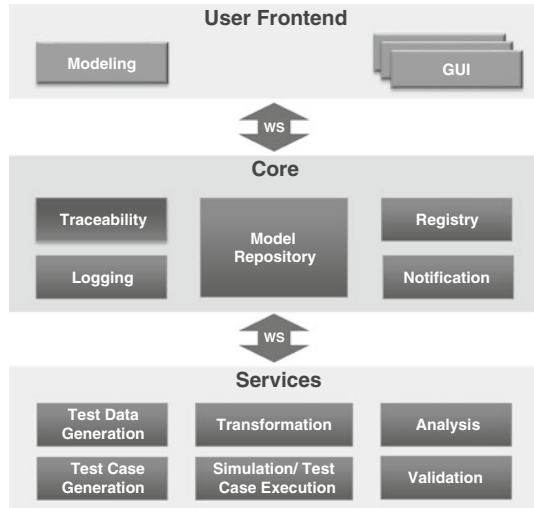
Traceability

Traceability information about relationships between the requirements, the system and test design, and their implementations is quite crucial for the development of complex systems, since the traceability analysis can help to improve the development process enormously. In general, traceability information helps to determine the impact of an element on the other specification parts. Furthermore, traceability analysis can provide coverage and traceability metrics to assist in localizing gaps or inconsistencies in complex specifications.

Using SysML, requirements of the system to be developed can be represented in UML models. They can be either directly defined in the model or imported from specialized requirement management tools. Furthermore, SysML provides special association types to define traceability links between requirements among themselves or between requirements and elements that are determined to realize or to verify requirements.

In order to maintain an overview of the many links and to advance traceability analysis, the TAS approach summarizes traceability information in a dedicated model. Such a traceability model largely includes references (traces) to the relevant elements from the requirements, system, and test model. In addition, this model is enriched with traceability links to the artifacts, like implementation or simulation code, derived from the specification models. Our traceability model allows for clear visualization of the traceability information and for easier navigation across different modeling domains. Furthermore, traceability metrics can be easily calculated with this model and potential gaps like unsatisfied or untested requirements can be easily identified.

Fig. 3 Service-oriented architecture for TAS (Reproduced from [14])



Service-Oriented Architecture

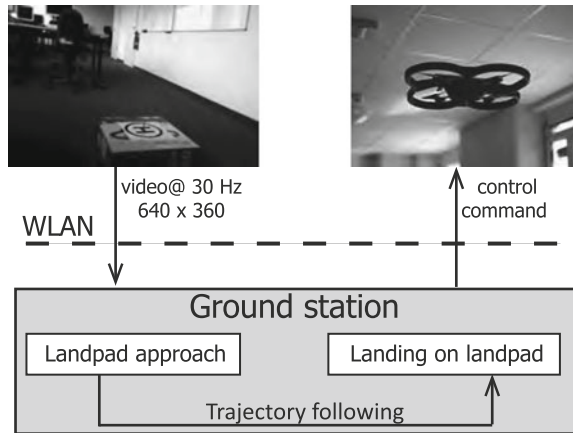
The previously outlined features of the TAS approach require a broad tool support for modeling, model transformations, simulation, and analysis. Of course, it is desirable to have one integrated tool environment on a single workstation. For large distributed processes or due to performance reasons it is, however, often required to source out some designated functionality or rather to make it accessible as services over the network. For instance, the simulation of large, complex models could be very time and resource consuming or could require special system requirements.

Furthermore, to enable integration of the TAS approach into existing development environments, the interoperability and loose coupling of heterogeneous tools via ubiquitous standards is needed. Therefore, in [14] we suggested a service-oriented architecture design [18] for the TAS approach (see also Fig. 3). The core of this design consists of a central repository with several common services, for example, services for registry, notification, or logging. The additional tools or components, provided for TAS, can be either realized as external services or are even part of the user front end. The communication between individual components is realized using open standards and well-defined Web service interfaces.

3 The TAS Modeling Methodology for Image Processing Systems

In this section, we illustrate how our modeling approach can be used for the design of an image processing system. It should be mentioned that we have reported this possible utilization of the TAS approach for the image processing domain in our

Fig. 4 Monocular vision system for the autonomous approach and landing using a low-cost micro aerial vehicle (MAV) system



previous works [19], but we now look at a concrete example. The system, which is shown in Fig. 4, enables an off-the-shelf Parrot AR.Drone 2.0 low-budget quadrotor micro aerial vehicle (MAV) to autonomously detect a typical helicopter landpad, approach it, and land on it. To fly toward the landpad while accurately following a trajectory, monocular simultaneous localization and mapping (SLAM) have been used [20].

The workflow in this application is based on the monocular, stereo, and RGB-D cameras as the main sensors and consists of the following steps: (1) exploitation of the geometric properties of the circular landpad marker and detection of the landpad; (2) determination of the exact flight distance between the quadrotor and the landpad spot; (3) moving toward the landpad by means of monocular simultaneous localization and mapping (SLAM); and (4) landing on the landpad. Development and hardware details related to this application have been illustrated in [20].

The Parrot AR.Drone 2.0 is a low-cost quadrotor with a simple IMU and two monocular cameras. The quadrotor features are 1 GHz, ARM Cortex-A8 processor, 32-bit 800 MHz DSP, and 1 GB of DDR2 RAM 200 MHz. Due to the complexity of the computational tasks, the above-mentioned workflow cannot be performed directly on-board. Therefore, the quadrotor communicates with ground station through wireless LAN. The ground station receives video data, performs the computations, and sends the generated steering commands back. In the example system, there are significant delays in the communication between the quadrotor and the ground station, as all the computations are performed externally. By means of simulation at the level of models, we aim to investigate and to compare alternative designs and solutions for the described system.

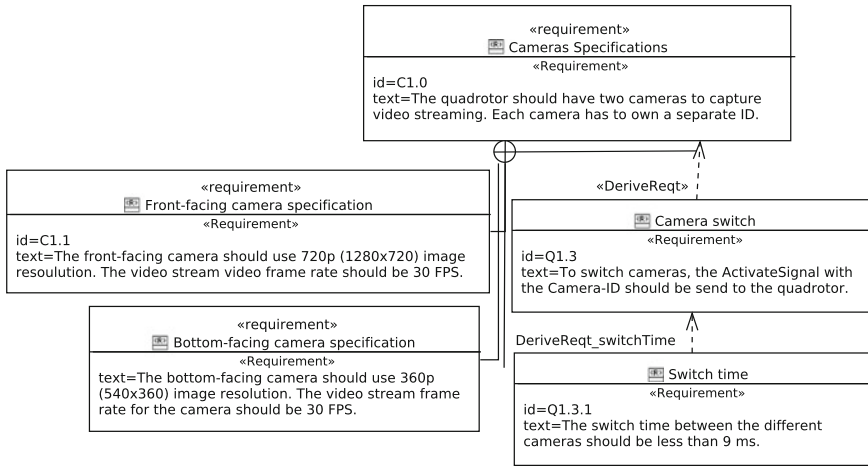


Fig. 5 Simple SysML requirement model for the video stream control systems

3.1 Requirements Modeling

In the scope of TAS, the design specification starts with SysML-based modeling, which involves the initial requirement specification. We have analyzed fundamental requirements on the hardware and software architectures for the example system. The requirements captured in text are represented and clustered directly in the model by means of the SysML requirements and package diagrams. Main requirements, which need to be considered to achieve a safe flying system with the minimum functionality including the video stream control systems, are illustrated in Fig. 5.

The *Camera* requirement includes the description of the quadcopt’s cameras: a HD (1280*720) 30fps front-facing camera and a QVGA (320*240) 60fps bottom-facing camera. During the video streaming, it shall be possible to switch between two cameras in a short time (see Fig. 5).

3.2 Structure Modeling

Based on the specified requirements, the system model as well as the test model can be created independent from each other in order to ensure their utilization for mutual validation (see in Sect. 2.1). The aim of this specification phase is to design the system architecture in terms of functional blocks. The functional and behavior aspects of the block can be expressed using SysML block definition, internal block, and state machine diagrams. Figure 6 shows the main block definition diagram from our example system. In addition to SysML concepts, the diagram will include a set of MARTE concepts in order to specify a context in which the system should be analyzed. We

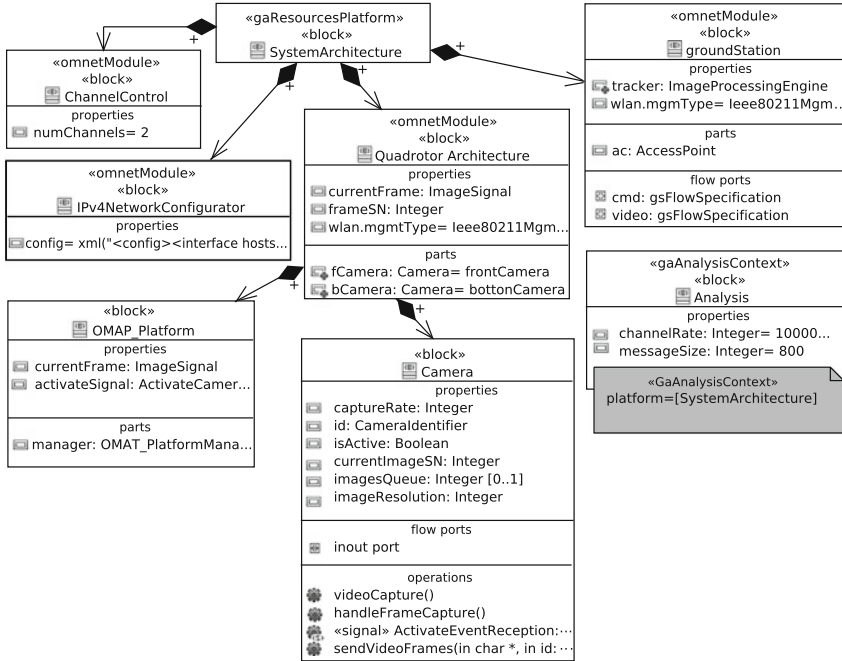


Fig. 6 Example of a description of system components and their relationships using SysML block definition diagram

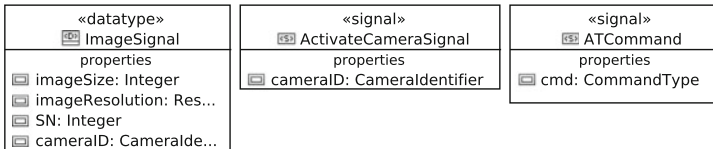


Fig. 7 Example of data type and signal definitions

use the stereotype *GaAnalysisContext* applied to a separate block to specify which of the blocks in the model should be analyzed. The platform attribute points to a *SystemArchitecture* block that has a *GaResourcesPlatfrom* stereotype applied. It represents a logical container for the resources used in the analysis context. All system components nested in this block will be simulated. The *GaAnalysisContext* block can also include input and output parameters for the simulation, which are specified with MARTE stereotype *Var*.

Figure 7 shows the signals in our model. The signal *ATcommand* is used to manage the quadrotor during the flight. *ImageSignal* encapsulates application-level data like images of the video stream for the transmission over the channel. *Activate CameraSignal* is used to switch the camera during the video stream.

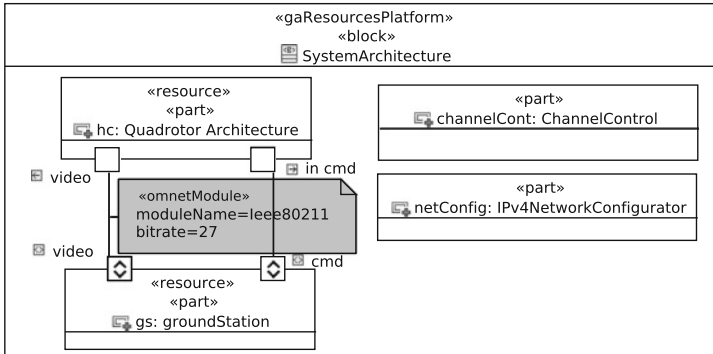


Fig. 8 Composite structure for the system

The SysML internal block diagrams describe the internal structure of a block. We use the SysML internal block diagram to model the system architecture. Figure 8 shows the internal structure of the *SystemArchitecture* block from our model. The system consists of a *Quadrotor Architecture* and a *groundStation*. *ChannelControl* is required for wireless simulation. It represents a system component that keeps track of which nodes are within interference distance of other nodes. *IPv4NetworkConfigurator* component assigns IP addresses and sets up static routing for an IPv4 network. Using the SysML concepts of ports and the connectors, it is possible to express communication links between various components.

3.3 Behavior Modeling

To model system behavior at a high level of abstraction, we primary use UML state diagrams. The state machines representing the image processing algorithm for the detection of the landpad in our model are shown in Figs. 9 and 10.

Figure 9 shows the top-level behavior of the *groundStation* block. As the state name *wait for video stream* indicates, the ground station waits in this state until it receives an *ImageSignal* message. This is the trigger of the only transition leaving from the *wait for video stream* state to the *landpad detection* state. The transition has applied a MARTE stereotype *GaStep* that indicates subsequent occurrences of the *ImageSignal* event which are of interest. The *do* behavior of the *landpad detection* state invokes the *detectionStateMachine* behavior.

Figure 10 shows the ground station behavior for the detection of the landpad. The main transition is triggered by the reception of the *Image* signal. The algorithm consists of the following main steps: (1) edge detection and (2) landpad matching. In the first step, we detect the edges in the image. After that, we group detected edges to a curve and check whether it contains the letter “H” [20], which indicates the landpad. Once the letter is detected, the *NotificationMsg* must be sent out to the quadrotor and the state machine moves into the final state.

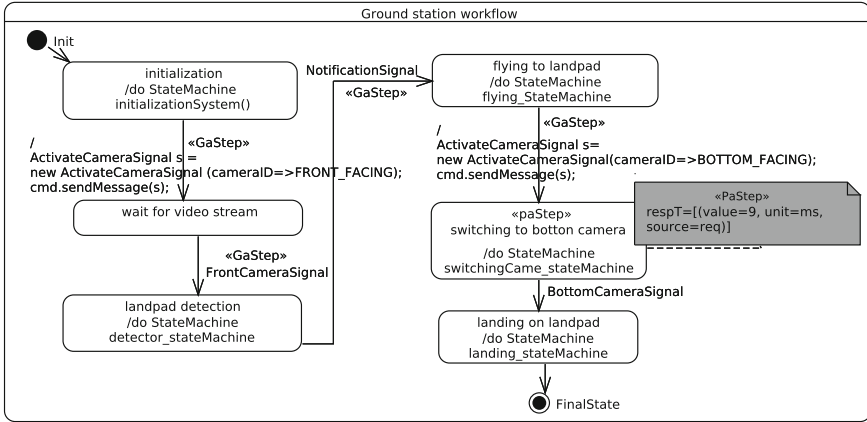


Fig. 9 State machine of the ground station

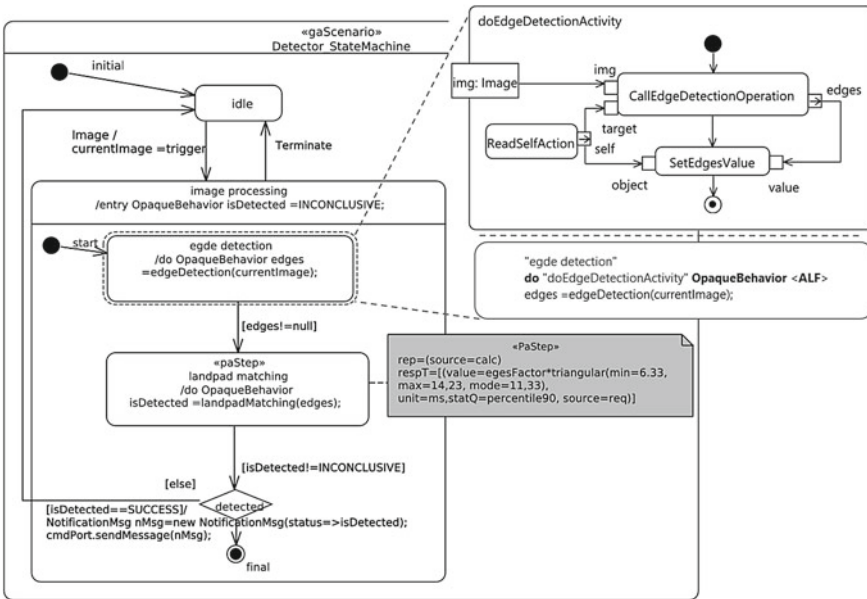


Fig. 10 System behavior building using SysML and ALF syntaxes

As exemplarily shown in Fig. 10 for the *do* activity of the state *edge detection*, we can define detailed behaviors either using standard UML behavior diagrams, for instance, like activity diagrams, or using a more compact high-level action language ALF [15].

In order to express performance attributes of behavior steps, we can apply the MARTE’s *PaStep* stereotype. For example, we specify that the response time of

calling the landpad matching activity is required to match the given distribution. To collect the statistics of interest during the simulation, one just have to apply an additional MARTE expression *source=calc* on the corresponding property value. In our example, we do so for *rep* and *respT* properties of *PaStep* stereotype on *landpad matching* state to determine the number of repetitions and response time. The collected data can then be analyzed for comparison with the expected result by utilizing the analysis capabilities of our framework as it will be shown later in Sect. 4.4.

3.4 Test Modeling

As a counterpart to the system modeling presented in the previous sections, the modeling of test specifications can also be performed in UML. Thereby, quite similar modeling paradigms can be applied as used for system modeling. Based on common requirements a test designer has to specify proper tests for subsequent validation of the system specification and later of its implementation. The purpose of these tests is to determine whether the system satisfies the requirements.

To provide a complete test specification, one has first to define the context of a test identifying the SuT and required test components. Figure 11 shows an example of a test context provided to test the behavior of the quadrotor component of our

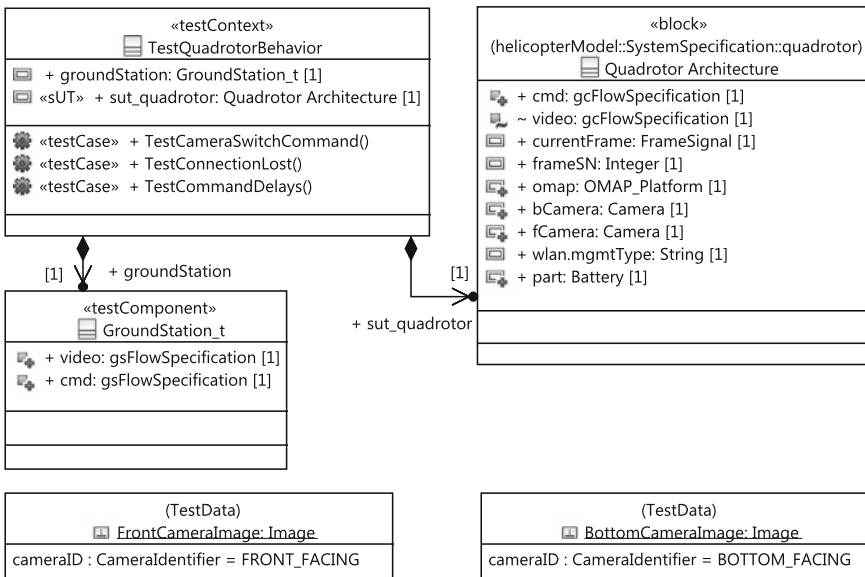


Fig. 11 Example of a test context definition

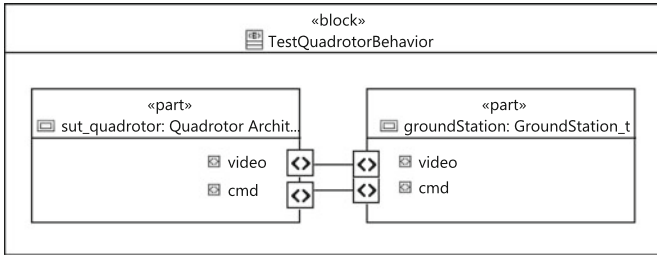


Fig. 12 Internal block diagram of a test context

example system. As already mentioned, we apply stereotypes of the UTP profile in order to declare test relevant aspects.

Thus, using a SysML block definition diagram or UML class diagram, a test context can be modeled as a structured block with the applied *TestContext* stereotype. A test context consists of test components, SuT, and serves at the same time as container for test cases. Whereas test components are simple blocks or classes containing ports for communication and declared in the test model using the stereotype *TestComponent*, SuT represents a block of the system specification model with its own behavior. To identify the SuT in a test context, the property referencing the SuT is marked with the *SuT* stereotype of UTP. The internal structure of the test context, which defines connections between test components and SuT, can be represented by the SysML internal block diagram, as shown in Fig. 12.

The behavior of test components is individually specified for each test case of the owning context. Therefore, using UML sequence diagrams a test case can be

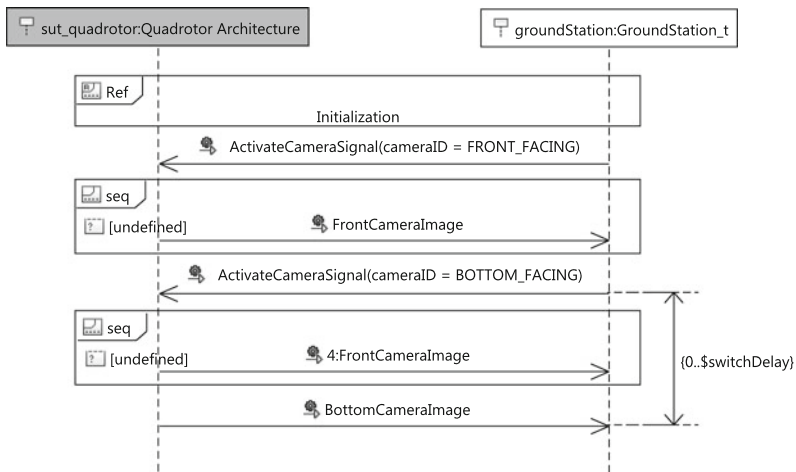


Fig. 13 Test case specification using UML sequence diagram

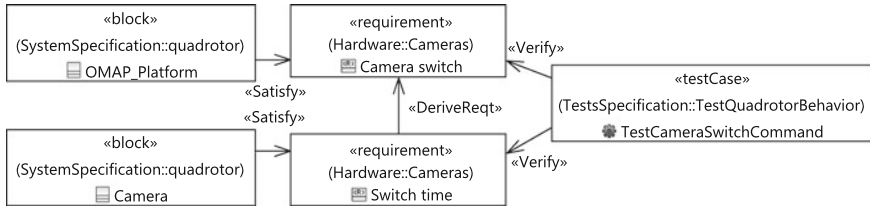


Fig. 14 Modeling of traceability links to requirements from satisfying or verifying elements

defined in every detail. As shown in Fig. 13, a test case is represented as a sequence of messages between the test component *groundStation* and the SuT *sut_quadrotor*. This test case, for instance, checks whether the quadrotor is able to activate its front camera and switch to the bottom camera appropriately after receiving appropriate camera activation command messages from the ground station.

3.5 Traceability Modeling

Although it is not the case in the modeling example presented, in a strict requirement-driven development process, nearly all system components and test cases shall relate to corresponding requirements. To express these relations, SysML provides special association links that can be assigned between requirements and other modeling elements. In Fig. 14 we show an example of traceability links defined for requirements regarding camera switching of the quadrotor. As shown in the figure, elements exist in the system specification, which satisfy the requirements and at least one test case in the test model, which verifies them. Currently, an engineer has to specify and manage most traceability links manually. However, additional tool support for automatic generation of traceability links while deriving model elements from requirements, for instance, is certainly possible and is part of the ongoing work.

4 SimTAny Framework

The main features of the suggested TAS approach are widely supported by the framework SimTAny (formerly introduced in [7] as ‘VeriTAS’) that will be further extended. SimTAny integrates relevant tools with newly developed components in a common environment based on the popular Eclipse RCP⁵ platform. Among others, we utilize a UML modeling tool, a transformation framework, a simulation engine, and an analysis tool (see Fig. 15). In the following, we will depict the main features of our framework and describe their realization in some more details.

⁵http://wiki.eclipse.org/Rich_Client_Platform.

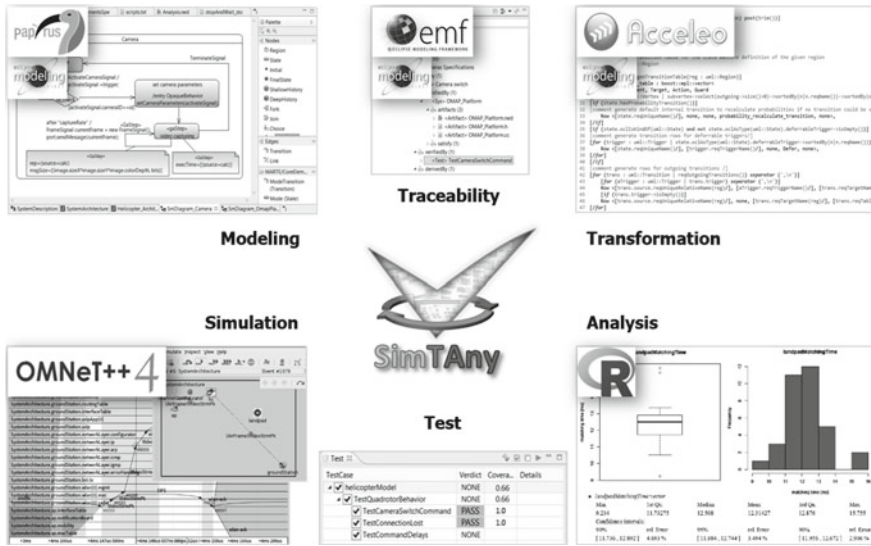


Fig. 15 Overview of tools integrated in the SimTAny framework

4.1 Modeling

In order to support extensive modeling capabilities required for TAS, we apply the open source modeling tool Papyrus,⁶ which is closely integrated with Eclipse. We preferably use Papyrus, since it has been designed to exactly implement the UML specification. All in all, it provides a very advanced modeling support for OMG standards including UML-related SysML and MARTE profiles. Nevertheless, other modeling tools, which can consistently export models into the OMG’ interchange format XMI, can also be used instead of Papyrus.

In order to improve the modeling efficiency, our framework adds some extensions to Papyrus. This primarily concerns the modeling of detailed behaviors and expressions with ALF textual editors. The main reason for supporting ALF’s textual notation is because specifying of a detailed behavior with a higher programming language, in most cases, is much more compact, faster, and intuitive than with standard UML behavior diagrams like state, activity, or sequence diagrams.

As it has been previously illustrated in Fig. 10 in Sect. 3.3, in principle, we can always express detailed activities of a state with activity diagrams, for instance. Because of the excessive complexity and inefficiency of this method, UML also provides the possibility to describe behaviors and expressions with a natural or programming language encapsulated in a so-called *OpaqueBehavior* or *OpaqueExpression* correspondingly. Thus, we apply the standardized action language ALF to directly specify such expressions and behaviors at appropriate places in our models in a more

⁶<http://eclipse.org/papyrus>.

compact and intuitive way. Among others, entry, exit, and do behaviors of a state as well as guard conditions and effects of a transition in state diagrams are predestined for ALF and thus are supported by appropriate text editors in SimTAny.

4.2 *Static Validation and Verification*

The special feature for static validation of system and test models, suggested for the TAS approach, has been realized in our framework by means of the Eclipse EMF validation framework.⁷ In order to achieve static validation and verification of the models, we provide constraints to check, on the one hand, for inconsistencies in each model separately and for compatible relations within the models to each other and to their common requirements, on the other hand. The defects detected in this way by the framework are listed in the Eclipse problems' view. Moreover, affected elements are marked as erroneous in the model editor afterwards. It is further possible to navigate from the problems listed in the view to corresponding elements in the model editor. Although only few simple constraints are currently implemented in SimTAny, the framework can be easily extended by new constraints.

4.3 *Transformation to Simulation Code*

Since the generation of the executable simulation code from UML models is one of the most challenging issues in our approach, a solid methodology with extensive tool support is required to perform this task. That is why we decided to build upon the OMG's standard MOFM2T [16] and its reference implementation, i.e., the Eclipse Acceleo⁸ code generation framework. MOFM2T provides a template-based model-to-text transformation language, where a template is a text containing specially marked areas that have to be generated accessing the elements of the input model. Generally, any kind of text or code for any textual language (C++, Java, Python) can be generated with this method. The framework Acceleo provides a code generation engine along with tools to support the efficient development of code generators. Besides a comprehensive editor with syntax highlighting, error detection, and auto-completion, it assists with a debugger, a profiler, and a traceability API.

With Acceleo we have implemented model-to-text transformation templates for generation of the simulation code (see Fig. 16). In order to demonstrate the feasibility of our approach, we currently generate code that is executable with the simulation engine OMNeT++.⁹ Nevertheless, our transformation module is designed to be

⁷<http://projects.eclipse.org/projects/modeling.emf.validation>.

⁸<http://eclipse.org/acceleo>.

⁹<http://omnetpp.org>.

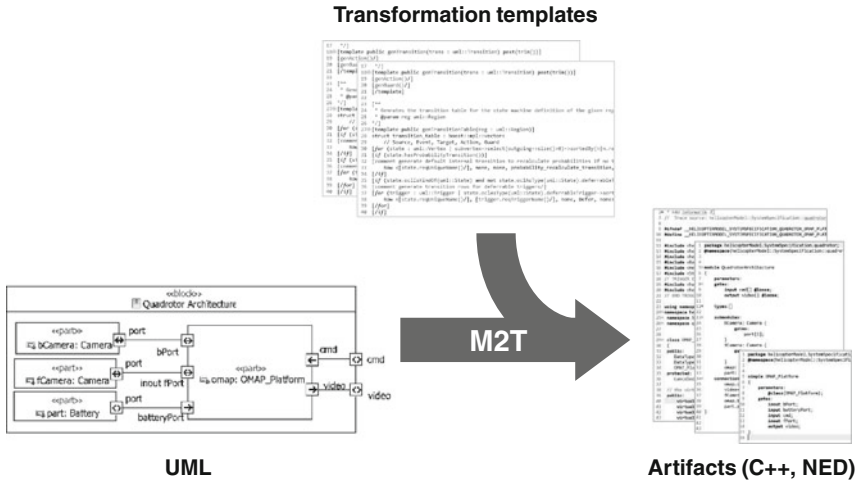


Fig. 16 Template-based model-to-text transformation

extendable for other simulation engines, too. OMNeT++ is an open source discrete-event simulator that is quite popular for simulation of communication networks.

An OMNeT++ simulation project (also called *simulation model*) typically consists of active components (*simple modules*) programmed in C++ that are composed of *compound modules* and *networks* using the OMNeT++’s own NED language. Furthermore, initialization files (ini) are used in OMNeT++ for additional configuration of simulation experiments. Thus, as a result of the model-to-text transformation our framework automatically generates C++, NED, and ini files of the complete simulation model. As shown in Fig. 17 for our use case, the simulation model generated

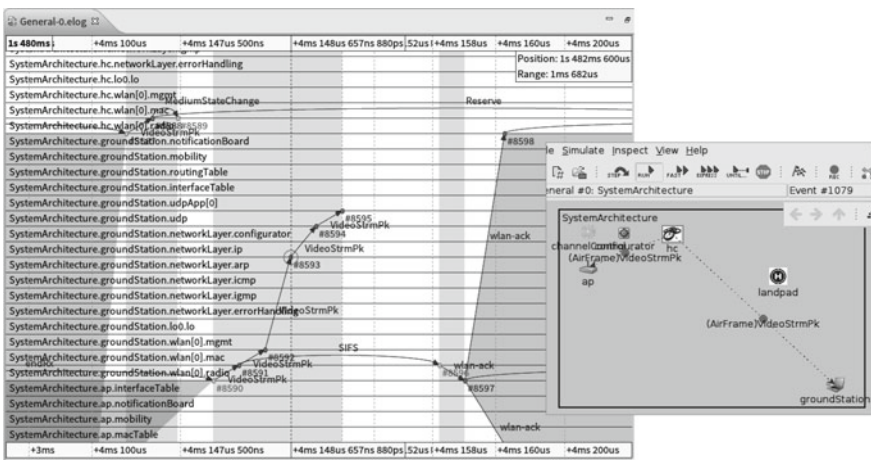


Fig. 17 Simulation with OMNeT++

can then be executed with OMNeT++ to analyze the behavior of the system (during the simulation or afterwards using the event log).

4.4 Analysis

The output data collected during the simulation can be directly analyzed in our framework, in the first instance, without prior external analysis tools being required. Therefore, SimTAny provides a dedicated perspective for analysis where the simulation results can be imported and visualized.

In the background, a very popular environment for statistical computing and graphics, i.e., the R-Project,¹⁰ is applied to generate plots and to calculate statistics of the data. Thus, for instance, the user can obtain an immediate overview about the key statistical measures like the mean, median, deviation, or confidence intervals of a data sample as well as to take a look at its time series, histogram, or box plots. To illustrate this, Fig. 18 shows example plots and statistics generated for landpad matching times observed during the simulation of our quadrotor model.

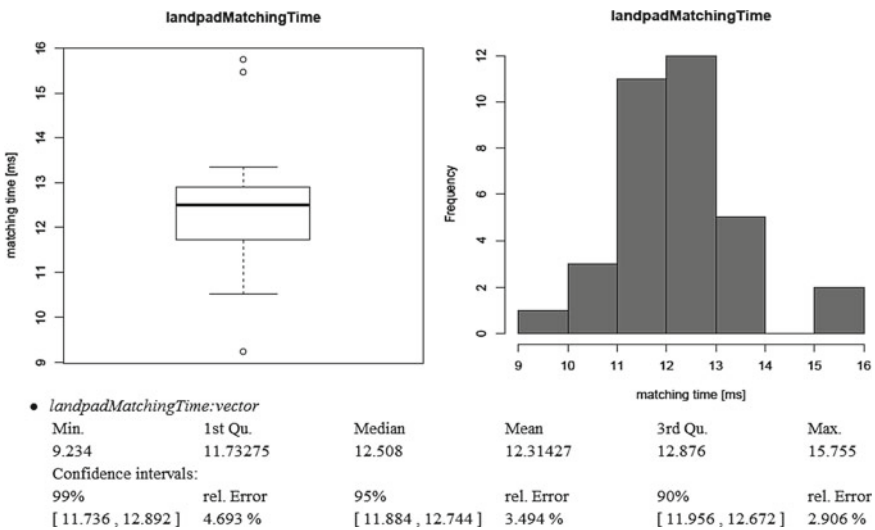
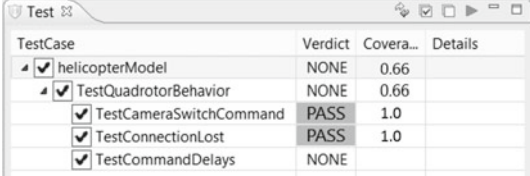


Fig. 18 Analysis of simulation results

¹⁰<http://www.r-project.org>.

Fig. 19 TestView provides an overview of tests contained in a model, control of test case execution, and test verdict



TestCase	Verdict	Covera...	Details
✓ helicopterModel	NONE	0.66	
✓ TestQuadrotorBehavior	NONE	0.66	
✓ TestCameraSwitchCommand	PASS	1.0	
✓ TestConnectionLost	PASS	1.0	
✓ TestCommandDelays	NONE		

4.5 Test

In order to support tests at the level of simulation models as described in Sect. 2, along with the generation of simulation code from the system model our framework provides the generation of executable OMNeT++ simulations for each test case contained in the test model. In Sect. 3.4 we have demonstrated an example for modeling of a test context and a test case. The user can obtain an overview of all test contexts and test cases defined in the model in the dedicated test view (see Fig. 19). Once the model has been transformed to the simulation code, the user can perform the execution of tests from this view. The verdict and eventual error reports of each completed test run are then also accessible in the view.

4.6 Traceability

As already mentioned, our approach provides for a special model aimed to store traceability information. Initially coming from specification models, this traceability information is enriched with the traceability links to artifacts generated during model-to-text transformations. Based on the Eclipse modeling framework EMF,¹¹ the traceability meta model has been developed and integrated in our framework. Instances of this model are created in SimTAny automatically from specification models by performing a model-to-model transformation according to the OMG standard specification (MOF) 2.0 Query/View/Transformation [21] supported by the Eclipse component *QVT Operational*.¹²

Furthermore, using the traceability listeners mechanism provided by Acceleo, SimTAny is able to collect traceability information during the code generation and to add it to the related traceability model instance.

In order to provide a better overview about all available traceability information, SimTAny provides a special traceability view (see Fig. 20). In this view the user can inspect the relationships between elements in both directions starting either from the requirements, from the system or test model elements, or even from the artifacts. The view also provides special filters to show possible deficiencies such as unsatisfied or

¹¹<http://eclipse.org/modeling/emf>.

¹²<http://projects.eclipse.org/projects/modeling.mmt.qvt-oml>.

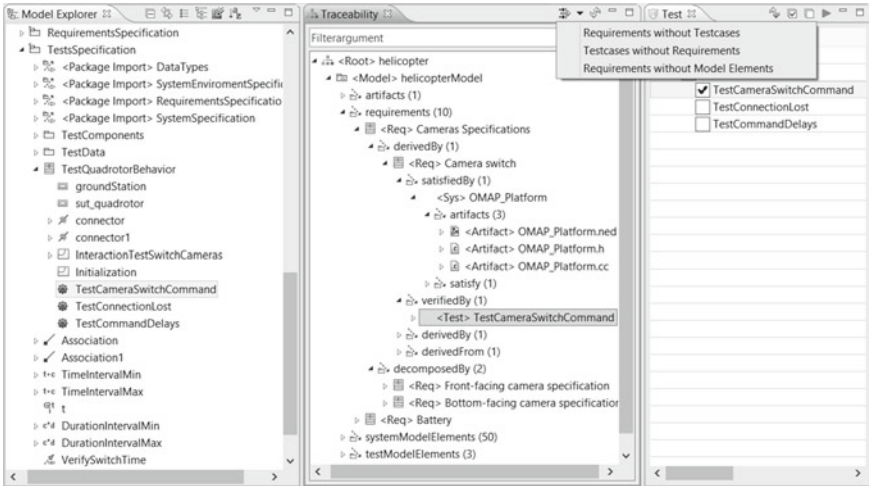


Fig. 20 Traceability view to inspect traceability relationships between different model elements

untested requirements. Thus, for instance, the requirements that do not relate to any test case can be easily detected for additional investigations.

Moreover, as depicted in Fig. 20, SimTAny allows easy switching from the traceability view to other views or editors relevant for further inspection of the selected element. In this manner, a model element occurring in the traceability view can be opened in the model editor, or a test model element can be shown in the corresponding test view.

5 Conclusions

Due to the ever increasing complexity of hardware and software systems, model-driven development methods and tools are gaining popularity as methods to fulfill functional and extra-functional requirements like timing aspects, reliability, or performance issues.

Model-driven engineering technique based on OMG’s UML and the MDA is a promising practice to address the complexity that is inherent in each technical system. The suggested TAS approach improves the overall quality of the development process by combining UML-based engineering, simulation, and testing techniques. TAS also assists the transformation of specification models to executable simulation code and uses MOFM2T, a standardized model-to-text transformation method.

By simulating a given system and running tests on it, TAS provides an agile technique to validate specification models at an early stage of the development process. To express extra-functional requirements and testing details for embedded systems

TAS offers the possibility to integrate model extensions that conform to OMG's SysML, MARTE, and UTP profiles.

In order to integrate the different tasks of the TAS approach we built the domain-independent framework SimTAny, which is based on the Eclipse RCP platform. Amongst other tools, we utilize the UML modeling tool Papyrus, the code generation framework Aceleo, the simulation engine OMNeT++, and the analysis tool R, which provides statistical measures and expressive plots. In addition, the Eclipse modeling framework EMF is used to develop and integrate a traceability meta model in order to store and exchange traceability information between different components.

Last but not least, we have shown how to apply TAS techniques for the development of a concrete image processing system. Here, the main task was the design of a system consisting of a quadrotor and a ground station that exchange image data over a wireless IPv4 channel. Diverse models from different UML profiles are used to specify functional and extra-functional properties of the overall system.

6 Future Work

Our ongoing work consists of proving more detailed hardware and software specifications for image processing systems by further elaboration of the modeling approach presented and by extending the transformation rules for simulation code generation.

Since extensive improvements regarding the UML modeling tools are still required to increase the efficiency of the model-based engineering, one part of our future work will be to make the modeling in the context of the suggested approach more user friendly. Thus, amongst others, to reduce the effort in creating and updating models, dealing with different modeling views, profiles, and a large number of stereotypes, we are developing appropriate extensions within the scope of our tool environment. For the special domain of the modeling of image processing systems, we would like to provide a modeling library, which will contain several predefined elements like typical image operators and data types.

Furthermore, an extensive experiment design framework is being developed to support model-based design and management of simulation experiments. Moreover, in order to allow precise simulation of hardware, we intend to integrate SystemC¹³ in our simulation environment.

References

1. Schmidt DC (2006) Model-driven engineering. IEEE Comput 39(2). <http://www.truststc.org/pubs/30.html>. Accessed Feb 2006
2. Object Management Group (OMG) (2011) UML unified modeling language. <http://omg.org/spec/UML>

¹³<http://www.systemc.org/downloads/standards/systemc>.

3. Object Management Group (OMG) (2003) MDA model driven architecture guide. <http://omg.org/cgi-bin/doc?omg/03-06-01>
4. Object Management Group (OMG) (2012) SysML systems modeling language. <http://omg.org/spec/SysML>
5. Object Management Group (OMG) (2011) UML profile for MARTE modeling and analysis of real-time and embedded systems. <http://omg.org/spec/MARTE>
6. Object Management Group (OMG) UTP: UML testing profile 1.2. <http://omg.org/spec/UTP>
7. Djanatliev A, Dulz W, German R, Schneider V (2011) VeriTAS—a versatile modeling environment for test-driven agile simulation. In: Proceedings of the 2011 winter simulation conference, Phoenix, AZ, USA, Dec 2011, pp 3657–3666
8. Dietrich I, Dressler F, Dulz W, German R (2010) Validating UML simulation models with model-level unit tests. In: Proceedings of the 3rd international ICST conference on simulation tools and techniques (SIMUTools' 10), Malaga, Spain, March 2010
9. Herrera F, Posadas H, Peil P, Villar E, Ferrero F, Valencia R, Palermo G (2014) The COMPLEX methodology for UML/MARTE modeling and design space exploration of embedded systems. *J Syst Archit* 60(1):55–78
10. Schneider V, Yumatova A, Dulz W, German R (2014) How to avoid model interferences for test-driven agile simulation based on standardized UML profiles. In: Proceedings of the symposium on theory of modeling and simulation, Tampa, FL, USA, April 2014, pp 540–545
11. Quadri IR, Indrusiak L, Sadovykh A (2012) MADES: a SysML/MARTE high level methodology for real-time and embedded systems. In: Proceedings of the international conference on embedded realtime software and systems (ERTS2)
12. Khan RH, Machida F, Heegaard PE, Trivedi KS (2012) From UML to SRN: a performability modeling framework considering service components deployment. In: *International journal on advances in networks and services*, vol 5, no c, pp 346–366
13. Trivedi KS, Sahner R (2009) Sharpe at the age of twenty two. *SIGMETRICS Perform Eval Rev* 36(4):52–57
14. Schneider V, German R (2013) Integration of test-driven agile simulation approach in service-oriented tool environment. In: Proceedings of the 46th annual simulation symposium (ANSS'13), San Diego, USA, April 2013
15. Object Management Group (OMG) (2013) Action language for foundational UML (ALF). <http://omg.org/spec/ALF/>
16. Object Management Group (OMG) (2008) MOFM2T MOF model to text transformation language. <http://omg.org/spec/MOFM2T>
17. Johnson TA (2008) Integrating models and simulations of continuous dynamic system behavior into SysML. Master Thesis, Georgia Institute of Technology, Aug 2008
18. Josuttis N (2007) SOA in practice: the art of distributed system design. O'Reilly Media Inc
19. Yumatova A, Schneider V, Dulz W, German R (2014) Test-driven agile simulation for design of image processing system. In: Proceedings of 16th international conference on advances in system testing and validation life cycle (VALID 14), IARIA, Oct 2014
20. Dotenco S, Gallwitz F, Angelopoulou E (2014) Autonomous approach and landing for a low-cost quadrotor using monocular cameras. In: *Computer vision—ECCV, (2014) workshops—Zurich, Switzerland, September 6–7 and 12, 2014, Proceedings, Part I*. Springer International Publishing Switzerland, pp 209–222
21. Object Management Group (OMG) (2011) MOF Query/View/Transformation (QVT). <http://omg.org/spec/QVT>