Model-driven Framework for Requirement Traceability

Nader Kesserwan[1], Jameela Al-Jaroodi[2]
School of Engineering, Mathematics, and Science (SEMS)
Robert Morris University
Pittsburgh, USA

***Abstract*—In software development, requirements traceability is often mandated. It is important to apply to support various software development activities like result evaluation, regression testing and coverage analysis. Model-Driven Testing is one approach to provide a way to verify and validate requirements. However, it has many challenges in test generation in addition to the creation and maintenance of traceability information across test-related artifacts. This paper presents a model-based methodology for requirements traceability that relies on leveraging model transformation traceability techniques to achieve compliance with DO-178C standard as defined in the software verification process. This paper also demonstrates and evaluates the proposed methodology using avionics case studies focusing on the functional aspects of the requirements specified with the UCM (Use Case Maps) modeling language.**

***Keywords*—Requirements; traceability; model transformation; do-178c; model-driven testing; traceability scheme**

I. Introduction

The largest part of traceability research so far has been done in the last two decades by the requirements engineering community [1]. Traceability, known as the ability to describe and follow the life of software artifacts [2], has become more important and traceability topics are being researched in many other areas of software development. One example is model-driven development where some components of the software development process are executed automatically using model transformations [3]. Model-driven development provides the foundation for the use of models as primary artefacts throughout the software development phases [4]. The variety of different models produced in the model-driven process pose challenges to requirements traceability and assessment. This diversity of artifacts results in an intricate relationship between requirements and the various models. The model-based testing (MBT) is a technique where test cases are generated from models [5]. MBT needs the ability to relate the "abstract values of the specification to the concrete values of the implementation" [6]. The relationships between artifacts play an important role to support automation of testing activities and it has been recognized for some time [7]. Relationships between behavioral models and test cases and between test cases and test results support better capabilities to measure coverage, evaluate results and perform selective regression testing. As a result, creating and maintaining explicit relationships among test-related artifacts is a main challenge to the automated support of these activities.

In this paper, model transformation techniques are used to create traceability links among MBT artifacts during the test generation process. The approach extends previous testing methodology presented in [8] that generates tests based on behavioral models. This paper's contribution is building a traceability model to support the creation and persistence of such relationships among heterogeneous models representing various testing artifacts. Moreover, this work enables the support for traceability visualization, model-based coverage analysis and result evaluation. The case study used in this work is an industrial product, flight management system (FMS), to evaluate the correctness of the approach that ensures all the generated test cases determine correctly the behavior of the FMS and are traceable to requirements.

The rest of this paper is organized as follows. Section 2 offers background information on traceability and model-based approaches in requirements and testing. A discussion of some related work about model transformation, model-based test generation, and traceability applied to automated testing approaches is presented in Section 3. Section 4 presents and describes the proposed approach, which is followed by Section 5 where two case studies are used to demonstrate the applicability and the evaluation of the approach. Section 6 offers a discussion of relevant approaches and draws future work guidelines, while Section 7 concludes the paper.

II. Background

In the domain of requirements engineering, the term traceability is usually defined as the ability to follow the traces (or, in short, to trace) to and from requirements. Two common definitions of requirements traceability are given by Pinheiro [9] as the ability to define, capture, and follow the traces left by requirements on other elements of the software development environment and the traces left by those elements on requirements; and by Gotel and Finkelstein as the ability to describe and follow the life of a requirement in forward and backward directions (i.e., from inception, through specification and development, to subsequent deployment, in addition to on-going refinement and iterations in any of the phases).

The Radio Technical Commission for Aeronautics updated the guidance document DO-178C [10] "Software Considerations in Airborne Systems and Equipment Certification" to address the safety concerns in new technologies such as model-based and object-oriented technologies. The document defines objectives and design

assurance levels for assuring the quality of the software and for an airborne system to perform its intended function with a level of confidence in safety that complies with airworthiness requirements.

The software verification process in DO-178C defines an activity to verify that the system requirements assigned to software have been developed into high-level requirements that meet those system requirements. In order to support this verification, trace data should be generated that show a link between each single system-level requirement and its propagation to test cases. The relationship between a high-level requirement and a test case is bidirectional allowing to trace in forward and backward directions.

Model-driven testing approach, based on transformation rules, uses a model-transformation technique to map a source model to a target one [11]. Model composition approaches automate the composition of heterogeneous models by relying on matching/merging operators [12]. Model-driven approaches move the focus in development from the third-generation programming language coding to more abstract models. This aims to increase productivity and reduce time to market by enabling the use of development concepts closer to the problem domain than those programming languages offer. The main challenge of model-driven development is transforming the high-level models to platform-specific models such that tools can use them for code generation [13]. It is possible to use models horizontally to describe different system aspects; however, they are useful for vertical representation to refine abstractions from the higher to the lower levels, where at the lowest level models use mechanisms based on implementation technology. Significant efforts are needed to work with multiple interrelated models to ensure their overall consistency. Furthermore, using these models can significantly reduce the burden of several other activities like reverse engineering, view generation, application of patterns, and refactoring through automation that is facilitated by the models. Such activities are usually performed as automated processes using one or more source models as input and producing one or more target models, while following a well-defined set of transformation rules. This process is referred to as model transformation.

The guidance document DO-178A [14] introduced at the beginning of 1985 a new technique that supports test coverage and traceability between requirements and tests. This technique, known as requirements-based testing, has been applied in the testing of complex software systems and demonstrated that the systems meet the requirements.

There are several modeling languages to express system requirements as scenarios and numerous languages that can be used to write test scripts. This paper refers to three different notations to capture functional requirements, describes the software description as test specification, and implements and executes scripts against the system under test (SUT). The key points are: (1) system behavioral requirements are formalized and modeled into scenarios representing the same requirements in an alternate Use Case Map (UCM) representation [15], [16], [17], and [18]; the UCM scenarios can be grouped by functionality into sets, for ease of comprehension and maintenance; (2) those UCM models are transformed to abstract test cases using the Test Description Language (TDL) [19], [20] , this process can be viewed as stepwise refinement and model transformation; (3) the obtained TDL abstract scenarios are used as the basis to generate executable test cases in Testing and Test Control Notation (TTCN-3) language. TTCN-3 [21] is a standard language for test specification that is widespread and well-established.

III. Related Work

It is important to establish and maintain relationships among software artifacts because these relationships are useful for many different software engineering activities like software change impact analysis and software validation, verification and testing processes. For instance, the traces can be used to keep models consistent and to identify pairs of related artifacts. These pairs can then be verified and validated against each other. A commonality between MBT and traceability is essential to manage the relationships among different artifacts. Relationship management should assist conception, persistence, and preservation of meaningful relationships across software artifacts in addition to assisting in the destruction of relationships.

Automated MBT approaches exploit two types of relationships: (1) implicit relationships embedded in the tool's algorithms and models, and (2) explicit relationships created and made explicit either automatically by the tool, or manually by the users.

Some approaches as in [22], [23] and [24] use implicit relationships to support test generation, execution and evaluation; while others like in [25] use implicit relationships to support regression testing. Further approaches use explicit relationships to support test generation [26], test execution and evaluation [27], or coverage analysis.

Naslavsky et al. [28] use one kind of behavioral UML model for test generation. A control-flow representation is used along with domain analysis of the parameters of the sequence diagram.

Basanieri et al. [29] use a tool (COW_SUITE) that loads UML models to create explicit relationships as edges in hierarchical trees among them.

Anquetil et al. [30] addressed some of the challenges in developing software product lines in two steps; (1) develop a model-driven framework to identify traceability of variability and (2) specify a metamodel for recording the traceability links.

In [31], the authors integrated a model-driven approach that exploits traceability relationships between monitoring data and architectural model to derive recommended refactoring solutions for the system performance improvement.

Bünder et al. introduce a domain-specific language called Traceability Analysis Language [32] to create and maintain relations of all artifacts that specify, implement, test, or document a software system. The relations are recorded in a

traceability information model and later aggregated to support software development and project management activities with a real-time overview of the state of development.

In [33], the authors adopt the tool (AGEDIS) that uses user-created explicit relationships to execute and evaluate the test scripts. The created relationships map abstract stimuli to method invocations and abstract observations to value checking. In addition, this tool expresses relationships between abstract test suites and test trace results during test execution. Manual coverage analysis is supported via the visualization of the test traces and the abstract test suite that generated them.

In [34], the (AsmL) tool uses user-generated explicit relationships to execute and evaluate abstract test scripts. The use of relationships in the AsmL tool supports the parallel execution of the model and its implementation by relating them and comparing their states.

An approach presented by Abbors et al. [35] provides requirements traceability across an MBT process and the tools used. Additional earlier research addressed using requirement-based testing to support traceability between the requirements and the related testing cases.

Arnold et al. propose a scenario-driven approach [36] (supporting both functional requirements and non-functional requirements) that helps create the traceability between generated and executed test cases, and the executions of an implementation under test.

Furthermore, a model-driven approach combining the strengths of both scenario-based and state-based modeling styles is described in [37]. The tool proposed enables tracing from requirements to testing and from testing to requirements in a round-trip engineering approach.

Pfaller et al. suggest [38] using varying levels of abstraction in development to derive test cases and link them to the corresponding user requirements.

Another approach suggested by Boulanger and Dao [39], where requirement engineering is performed in different phases of the V-model to enable requirements validation and traceability.

Felderer et al., however, focus on model–driven testing of service-oriented systems in a test–driven way [40]. They suggest that the Telling TestStories tool can support traceability among all types of modeling and system artifacts. Marelly et al. discuss linking requirements and testing through the extension of sequence charts with symbolic instances and symbolic variables [41].

IV. Traceability Approach

This work builds on some of the techniques described earlier to create the traceability approach of MBT artifacts. The Ecore trace model is integrated into Eclipse Modeling Framework (EMF) and it is independent of the models it connects. The traceability approach in Fig. 1 [42] shows how system requirements, represented in an abstract model, are propagated through model transformation to more refined models. Furthermore, the traceability approach shows how the relationships among the generated models are created and recorded in a trace model. The first step in the approach is to represent the functional requirements of a system. The use of the modeling tool jUCMNav [43] help describe the system requirements as scenario models in UCM notation. In step 2, the behavioral models, described in step 1, are flattened to scenario definitions using the path traversal algorithm in the jUCMNav tool. Each flattened scenario is transformed, based on transformation rules, to test description in TDL. During the transformation process, the traceability information between the two models (UCM and TDL) are explicitly defined as a trace model. Lastly, test cases generation starts in step 3; it uses the transformed TDL test description models and data model (additional information) to generate the TTCN-3 test cases. Once more, during the process of generating test cases, the traceability information between TDL and TTCN-3 artifacts are explicitly defined and made persistent based on and guided by a traceability scheme.

The key points of the traceability approach are: (1) natural language requirements are described as scenario models in UCM; (2) the UCM models are transformed to test scenario in TDL; and (3) the resulting TDL test scenarios are used along with data model, detailing test data, to generate test cases in TTCN-3. Since the UCM models emphasize behavior and abstract from concrete data, this work focuses on developing a metamodel to support the test data. The developed data model is based on test requirements consisting of three metamodel elements: (1) the UCM responsibilities for message exchange, (2) A set of typed TDL data, and (3) a detailed TTCN-3 data with concrete value. During model transformation, traceability information is defined explicitly into a trace model (tracemodel.ecore). In the following subsections, an example is used to show how relationships among the testing artifacts are created and captured in the trace model during model transformation. The applicability and the evaluation of the approach is demonstrated via case studies in Section 5.

A. Scenarios in UCM Metamodel

The user requirement notation standard suggested UCM notation to capture the functional requirements of a system in terms of visual use case. This latter represents the behavior of a system as a casual scenario composed of responsibilities that can be attached to abstract components. The scenario models, as shown in Fig. 1 (step 1), represent the functional requirements of a system. The UCM models help design and understand systems. The UCM models could be used as a base to derive the test specification cases which in their turn used to develop the test cases.

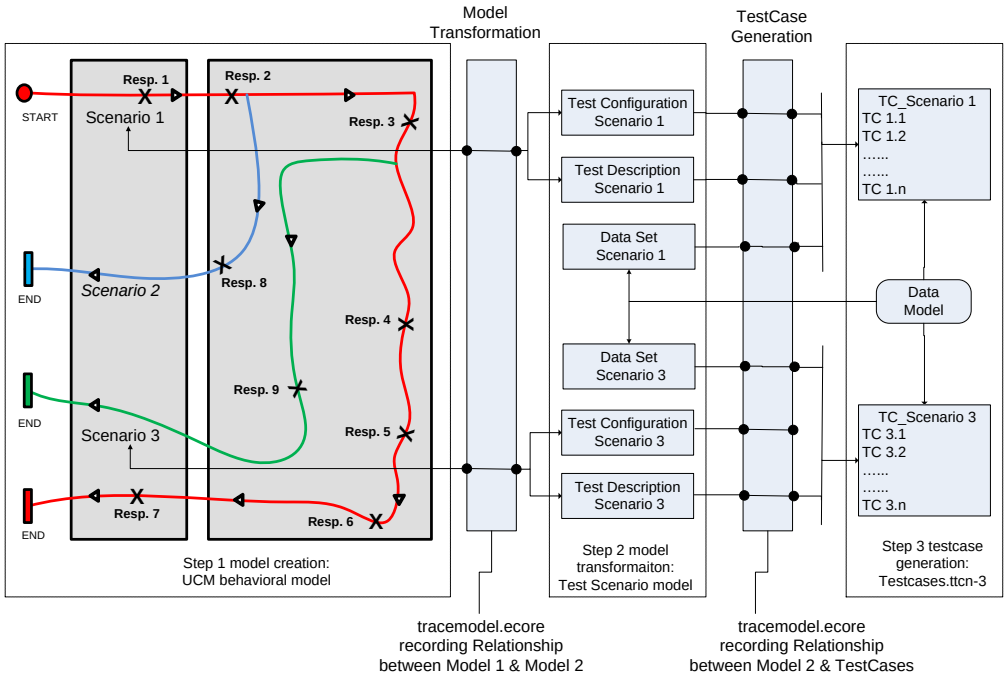

Fig. 1. Traceability Approach Overview.

B. Test Scenarios in TDL Metamodel

The European Telecommunications Standards Institute proposed TDL [44] as a standardized scenario-based approach to specify software test cases as scenarios. TDL is a new standard developed for specifying "formally defined Test Descriptions used for test automation. It offers a high level of abstraction for specifying scenarios beyond programming or scripting languages. TDL can also be used to represent tests generated from other sources like simulators, test case generators, or earlier runs' logs". As described in [45], TDL is a general formal language for representing Test Descriptions which are used mainly for communication between stakeholders as the basis for implementing concrete tests. The TDL design is centered on three separate concepts: (1) the metamodel principle that expresses its abstract syntax; (2) concrete syntax, which is user defined for different application domains; and (3) the TDL semantics that can be found in metamodel elements.

Our approach's main goal is to discover relationships between testing artifacts to support requirement coverage and test evaluation. The model-based test scenario method will support scenario derivation from the UCM behavioral models, and link the relationships from the behavioral model to the test cases. TDL metamodel is used to support the description of scenarios. An instance of TDL metamodel can describe the essential elements of a test scenario such as messages, behavior, actions, interacting components, etc. The TDL test description metamodel, shown in Fig. 2, describes test description based on the exchanged communications between an SUT and a tester.

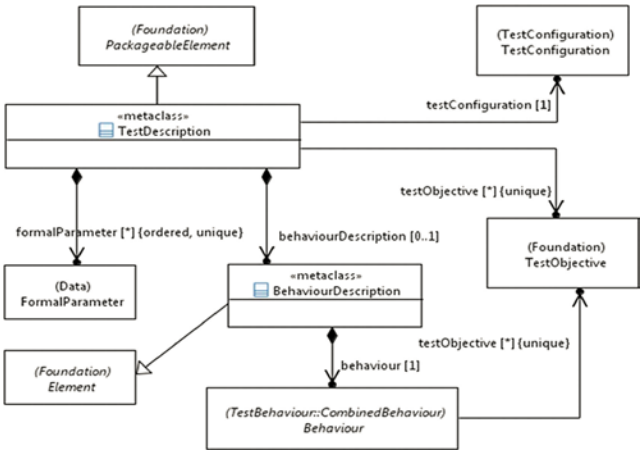

Fig. 2. TDL Test Description Metamodel.

C. Linking UCM Scenarios to TDL Specification

The UCM scenario model shown in Fig. 3 describes the Internet's Domain Name System (DNS) example that verifies whether a DNS server can correctly map a host name to its equivalent IP address.

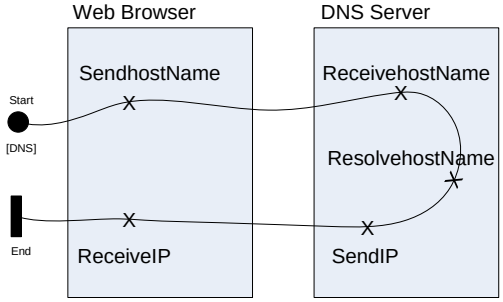

Fig. 3. DNS Scenario Model.

The DNS scenario model has one map contains: a Causal path represented by a wiggly line, two rectangular boxes that represent components Web Browser (Tester) and DNS Server (SUT) and four responsibilities bound to components along the path, and one scenario. The responsibilities elements in UCM are abstract and can represent actions or tasks to be performed by the components. The components themselves are also abstract and can represent software entities (objects, processes, network entities, etc.) as well as non-software entities (e.g. users, actors, processors).

As depicted in Fig. 4, a process (ATC Builder) has been developed to transform the UCM scenario model and data model (additional information) into an abstract test case expressed as a valid TDL.

The outcome of this process is a TDL specification composed of four elements; (1) Data Set, (2) Test Objective, (3) Test Configuration, and (4) Test Description. The DNS scenario model shown in Fig. 3 is transformed into a TDL specification as depicted in Fig. 5.

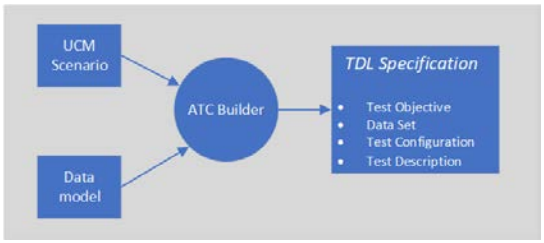

Fig. 4. The Process to Build a TDL Test Specification.

The Component objects, Web browser and DNS server objects, in DNS are transformed into Test configuration items including for example Component Instances, Gate Instance, and Connection. Component Instances can be a part of a Tester or a part of an SUT. Component Instances are connected via the Gate Instance for the exchange of information. The responsibility objects in the DNS scenario model are transformed to Test Description elements such as Action Reference and Interaction. The action to be performed on the Component Instance has an attribute to identify the latter. The gates are used to exchange abstract information which is referenced by the Interaction elements in TDL. This Interaction element could be seen as an exchanged message between source and target.

D. Linking TDL Scenarios to TTCN-3 Test Cases

The UCM scenarios are used as a base to derive the TDL elements. However, the transformed TDL test specification is an abstraction that cannot be executed on SUT. The TDL elements such as Data Instances and Interactions lack concrete details about how to communicate with the SUT. In order for a test case to be executable, it should contain detailed test data and interface specifications. The test inputs for the test cases were developed in a data model during the test analysis and design process. In a UCM scenario, the responsibility object represents an interaction or an action to perform. Therefore, the interaction messages are developed from those responsibilities of nature stimulus/response, mapped into TDL Data Instances, and in turn are developed into a TTCN-3 Template as shown in Table I.

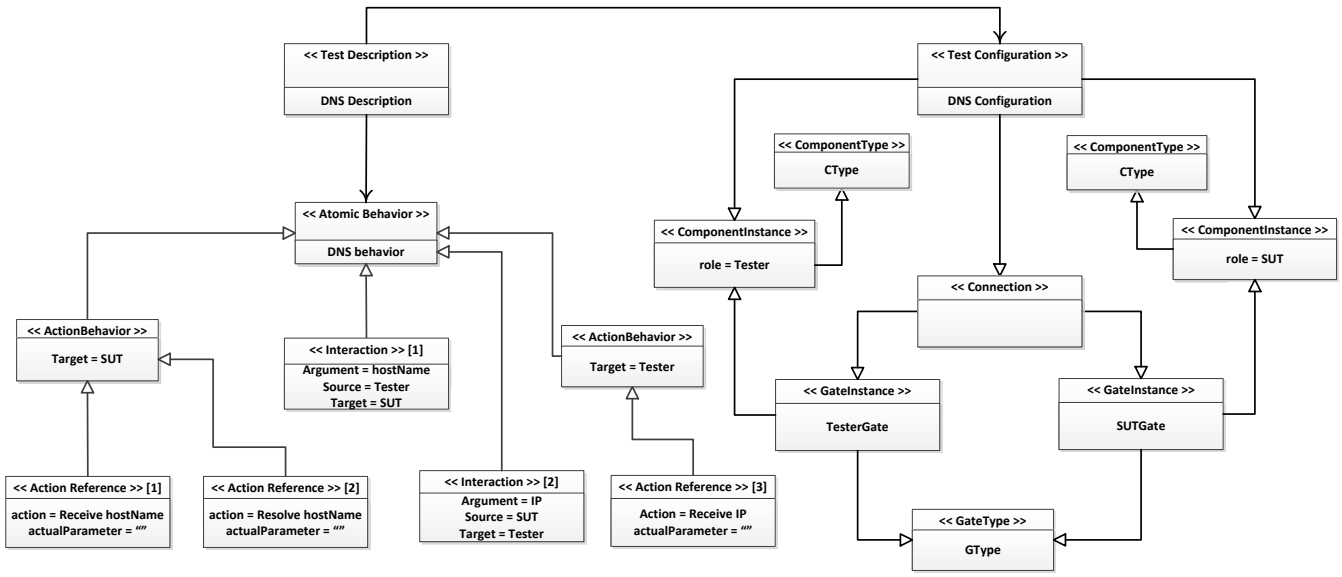

Fig. 5. TDL Metamodel for Test Specification Model.

TABLE I. REFINEMENT OF TEST DATA FROM ABSTRACTION TO CONCRETE [42]

Test Data Input/ Output	**Abstract Data in UCM**	**Data Instance in TDL**	**Data template In TTCN-3**
Stimulus	***SendhostName***	**instance SendhostName**	**Template String** SendhostName :="myHostName"
Response	***ReceiveIP***	**instance ReceiveIP**	**Template String** Receiveip:= "192.124.35.5"
Stimulus	***SendhostName***	**instance SendhostName**	**Template String** SendhostName :="myHostName"
Response	***ReceiveIP***	**instance ReceiveIP**	**Template String** Receiveip:= "192.124.35.5"

Based on data specifications, this work included developing a data model composed of different test data abstraction:

- Stimulus/response: a subset of abstract test data requirements characterized as input and output messages expressed as responsibility objects in UCM.
- Test data instances: the abstract subset of test data requirements is developed to Data Instances and Data Sets in TDL.
- Test data template: using the TTCN-3 templates that define the concrete data, the Data Sets are finally developed and detailed.

The generation of TTCN-3 test cases from the TDL test specification and the data model becomes feasible after applying the transformation rules between the two languages. Transformation rules are defined between TDL and TTCN-3 metamodels resulting in four TTCN-3 modules that together constitute an executable test case: (1) the Configuration module which usually contains several linked test components with unique communication ports, (2) the Description module that consists of behavioral program statements specifying the dynamic behavior of the test components, (3) the Oracle module that contains the expected responses, and (4) the Input module that contains test input data to be transmitted over the communication port. The modules (3) & (4) are derived from the Data Sets and data model. Each requirement to be tested in the data model has an input domain that is subdivided into a set of templates (partitions) and used as a concrete test data. This type of structure will create dependency relationships between a requirement and the relevant test case data. This will help improve regression testing as mentioned in [46]. Since the model transformation starts with flattening the scenario model into scenario definitions, a scenario coverage strategy is applied. Each flattened scenario is transformed to a test scenario and enriched with test data to derive the test cases. This way, straightforward relationships are established between the scenario and the test cases.

E. Traceability Metamodel

In the context of model-driven development, traceability schemes are usually explicitly expressed in metamodels, which are also usually linked to models specifying model transformations. Currently there is no single standardized traceability metamodel. The traces among testing artifacts can be produced on-line, where case traces are stored automatically by a tool as a by-product of the development activity. It can also be done off-line, where traces are recorded (automatically or manually) after the actual development activity has ended. The approach proposed earlier uses a trace metamodel inspired from Jouault et al. [47] that supports traceability. This work's contribution is externalizing and maintaining the relationships between the test-artifact models (i.e. the UCM scenario models, Test scenario models and Test cases models) and recording them in the new trace model. The relationships are recorded semi automatically in the trace model to support various activities like results evaluation, regression testing and coverage analysis. The traceability metamodel to hold the relationships among testing artifacts is defined in UML class relationship diagram as shown in Fig. 6. A class relationship diagram describes the types of objects in the model and selected relationship among them. The relationships can be of type (1) 'Generalization' that relates a specific classifier to a more general classifier. Generalization is denoted by an arrow with an unfilled, triangle head. (2) 'Association' that denotes responsibilities and are shown as lines connecting classes. (3) 'Dependency' where a class A depends on another class B. Dependency is indicated by a dashed line ending at a navigability arrow head. (4) 'Aggregation' can be read as "is part of" or, in the opposite direction as "has a". Aggregation is denoted by an arrowhead drawn as an unfilled diamond. (5) "Composition" implies that the "lifetime" of the parts is bound to the lifetime of the whole. Composition is denoted by an arrowhead drawn as a filled-in diamond.

F. Traceability Scheme

The first step of model creation constructs the UCM model with integrated features (path traversal algorithm) capable of exporting scenario models that conform to the EMF metamodel, Ecore, and implementation of the UCM notations. The implementation of the second step, model transformation, is based on the "UCM scenario to test scenario" model transformation. To support traceability, the transformation tool is extended in this work to create traces that relate the model elements between UCM scenarios and TDL specification. Guided by the traceability scheme defined in Table II, the produced traces in the traceability model called "tracemodel.ecore" were recorded. Implementation of the third step, test case generation and traceability information, takes place when the transformed TDL specifications and the data model developed earlier are ready. These traces were again recorded as a product of the transformation, with the guidance of the traceability scheme.

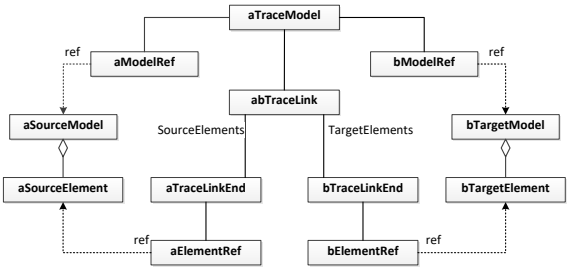

Fig. 6. Traceability Model (Kesserwan Dissertation [42]).

TABLE II. TRACEABILITY SCHEME

Testing artifacts/ Traces	**What information to record**	**Constraints**	**Source**
UCM Scenario	Component, Interaction, Action Reference		Scenario Definition
TDL Test Specification	Test Configuration, Test Description, Gate, Interaction, Action Reference, Data Instance, and Data Set	No duplication in Gate	Connected components
			Set of Interaction
		No duplication in Data Set	Action reference
			Component Interaction
			Data model
TTCN-3 Testcase	Port, Record, Record field, Send, Receive Template, and Function	No duplication in Port	Gate
			Interaction
			Data Set
			Data Instance
			Action reference

V. APPROACH APPLICABILITY AND EVALUATION

The application and the evaluation of the traceability framework have been demonstrated by conducting two case studies from the avionics industry. The first case study is called the landing gear system [48], used to demonstrate the applicability of the approach, where the second one is the FMS used for the evaluation.

A. Test Cases and Trace Model Generation

The description of the landing gear behaviour is captured in UCM scenarios and explained in the following. The goal of the landing gear in an aircraft is to provide support during taxi, take-off and landing. Before landing, the landing order of an airplane is: unlock the landing gear doors, extend the gears and lock the landing gear doors. Fig. 7 depicts a successful deployment of extending sequence scenario [DeploymentSucceeded], and two unsuccessful deployment scenarios; [DeploymentFailed] and [NormalModeFailed].

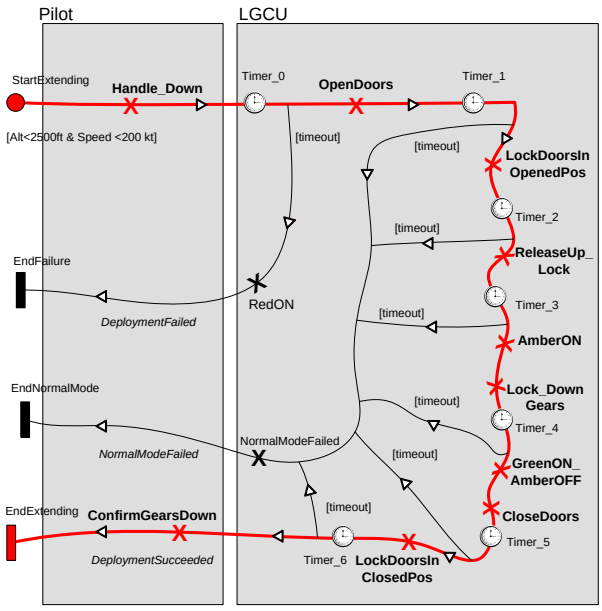

Fig. 7. Visual UCM Scenario Describing the Extending Sequence Case.

The creation of the UCM model was described as step 1 of the approach (Fig. 1). The next step is to transform the UCM model into a TDL test specification, and create the traceability information. The test data for the successful scenario [DeploymentSucceeded] is shown in Table III.

The graphical representation of the transformed model, composed of test description and test configuration elements, is depicted in Fig. 8. Traceability information for the test configuration is depicted in Fig. 9, while part of the traceability information for the test description is depicted in Fig. 10.

In Fig. 9, the traceability model is named TraceUCMModel2TDLModel. It relates models UCMScenarioModel and TDLTestScenarios. It has one trace link named DSScenarioTraceLink that relates the UCMDSScenario in the UCMScenarioModel to the TDLDSTTestSpecification in the TDLTestScenarios. DSSScenarioTraceLink has many children; the figure shows the link DSTestConfigurationTraceLink, which relates the component Instances (Pilot and LGCU) in the UCMDSScenario to the gate instances (Tester and SUT) in the TDLDSTestSpecification.

In Fig. 10, the trace link DSSScenarioTraceLink has another child DSTestDescriptionTraceLink, which relates the interactions and action references in the UCMDSScenario to the interactions and action references in the TDLDSTestSpecification. The figure shows one “Interaction” and one “Action Reference”.

The last step in the approach is the generation of test cases and the creation of the traceability information in the TDL test model and the generated test cases. Information from the data model in Table II, from the trace model in Fig. 10 and from the test specification model in Fig. 8 is used to complete the step. The data model is developed from the testing requirement and represents the input space for the scenario model [DeploymentSucceeded] under transformation. The instances in the data model are grouped into two sets; stimulus (Tester) and response (SUT) to build the TDL Data Sets elements. Each Data Set is mapped to records and variables elements in TTCN-3 using the transformation rules between the two languages. In Fig. 11, the trace link DSSScenarioTraceLink has a child DSTestDataModuleTraceLink, which relates the Data Set, Data Instance and Interaction in the TDLDSTestSpecification to the Record, Record field and Send in the TC_DS_[seq]. The figure shows one “Data Set”, one “Instance” and one Interaction. The TDL test scenario [DeploymentSucceeded] is transformed into a test case in TTCN-3. The approach defined in [8] applies structural transformation where each TDL element is transformed into a number of TTCN-3 modules. Based on transformation rules, the resulting test case is composed of three types of modules: (1) a Test Configuration module, (2) a Test Description module, (3) and a Data module. The TTCN-3 data module is refined with test input and expected output when this data becomes available. A new test case is added “TC_DS_01” to the test suite “TTCN-3_DC_TestSuite” for each new pair of test input and expected output found in the Data model in Table II.

TABLE III. THE DEVELOPMENT OF TEST DATA FOR [DEPLOYMENTSUCCEEDED] SCENARIO [42]

Test Data Requirement	UCM responsibility Stimulus/Response	TDL Data Instance	TTCN-3 Template
Send stimulus when handle is pushed down	***Handle_Down***	**instance Handle_Down**	**Template String** Handle_Down_Type
Receive a response when locking doors in opened position	***LockDoorsInOpenedPos***	**instance LockDoorsInOpenedPos**	**Template String** LockDoorsInOpenedPos_Type
Receive a response when Gear is in transition	***AmberON***	**instance AmberON**	**Template String** AmberON_Type
Receive a response when locking Gears in down position	***GreenON_AmberOFF***	**instance GreenON_AmberOFF**	**Template String** GreenON_AmberOFF_Type
Receive a response when locking doors in closed position	***LockDoorsInClosedPos***	**instance LockDoorsInClosedPos**	**Template String** LockDoorsInClosedPos_Type

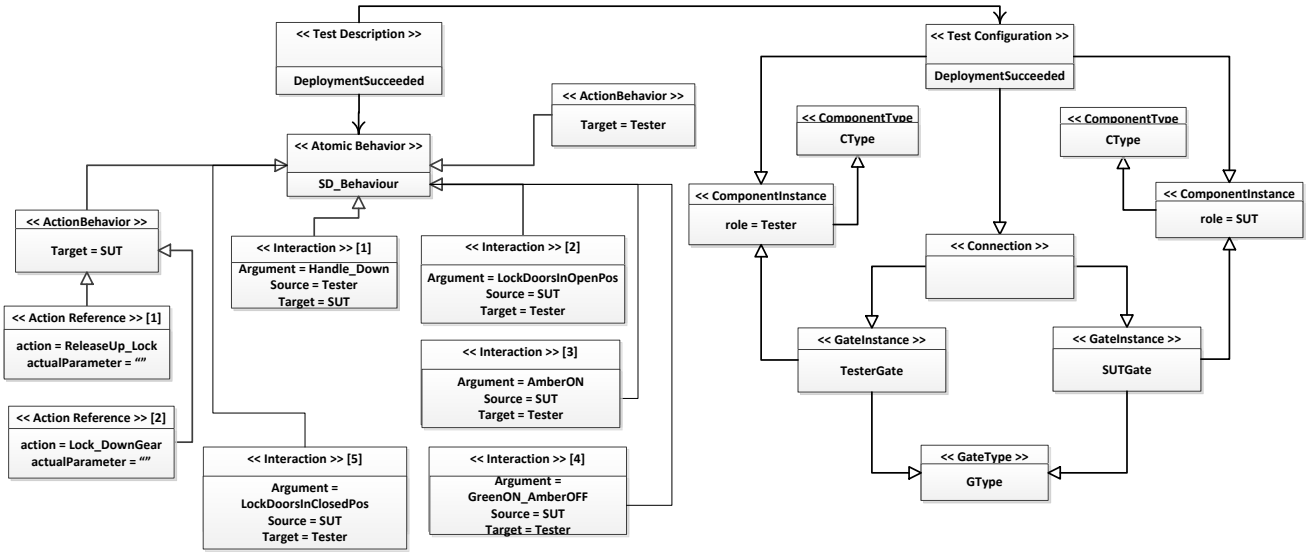

Fig. 8. Test Specification Model for [DeploymentSucceeded] Scenario (Kesserwan Dissertation [42]).

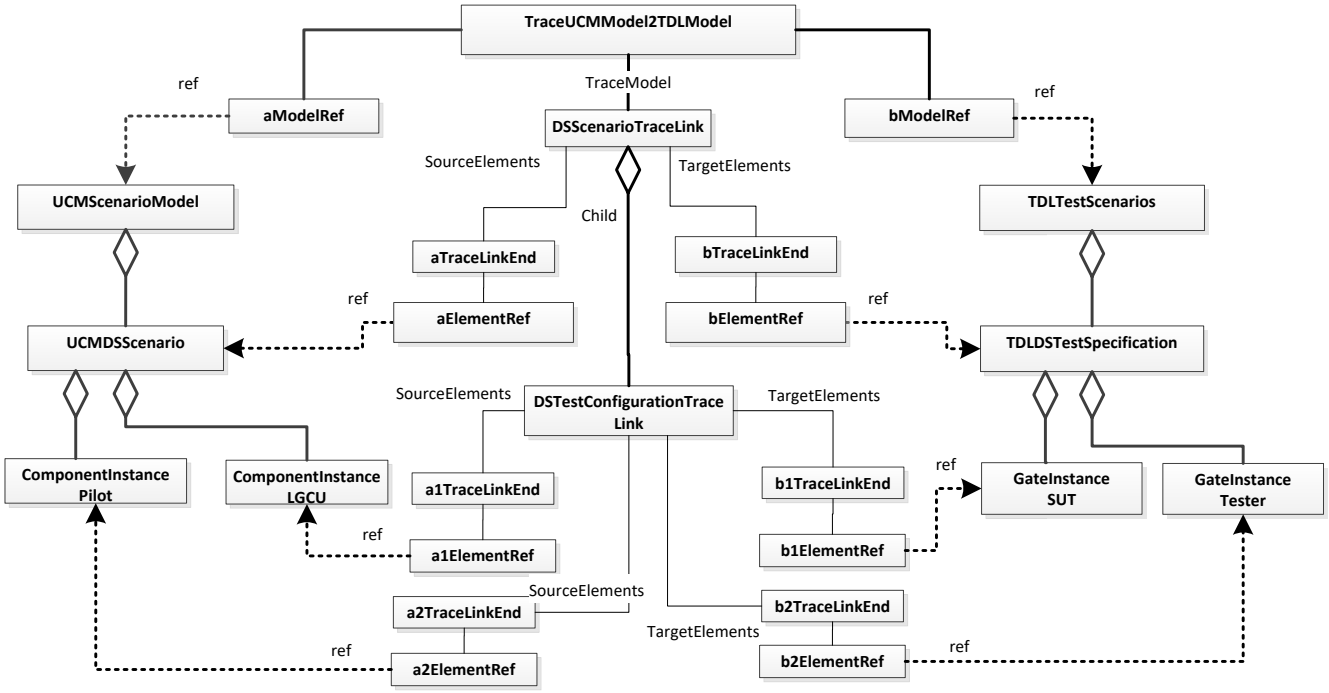

Fig. 9. Traceability Model shows Traceability Links between the UCM and TDL Models for [*DeploymentSucceeded*] Scenario (Kesserwan Dissertation [42]).

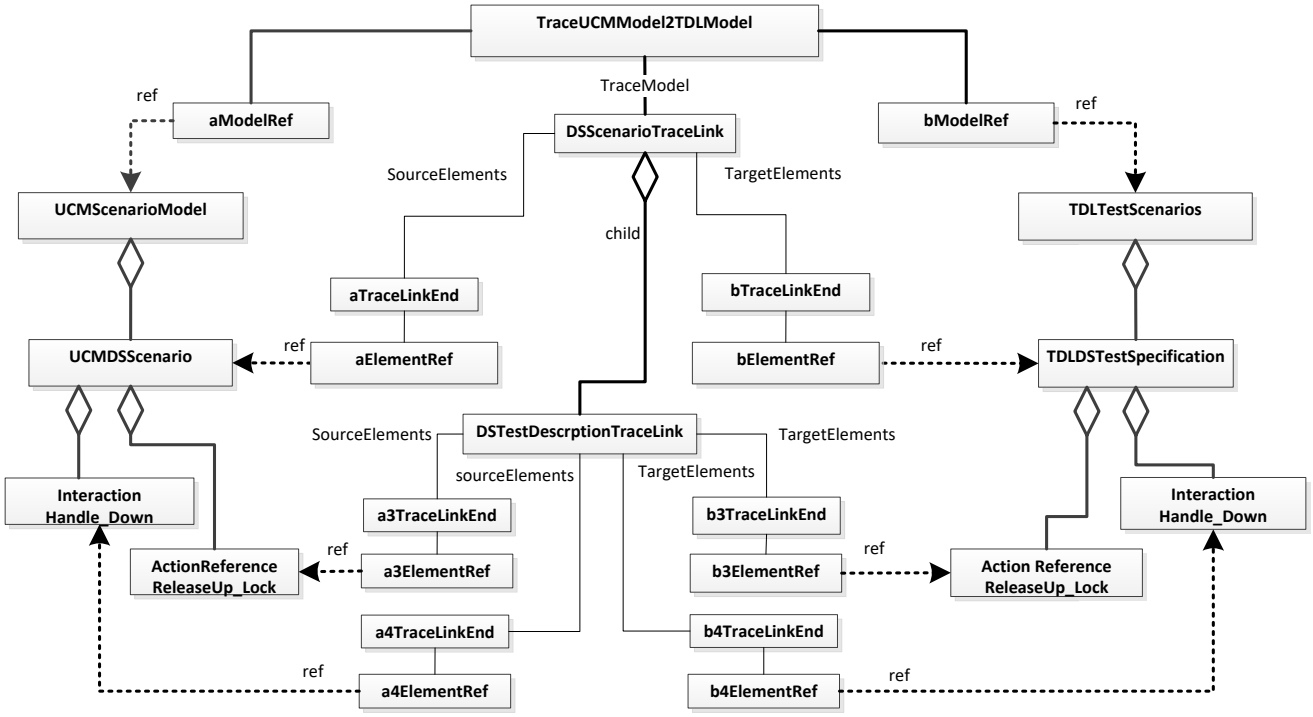

Fig. 10. A Small Part of Traceability Links between the Two Models for [DeploymentSucceeded] Scenario (Kesserwan Dissertation [42]).

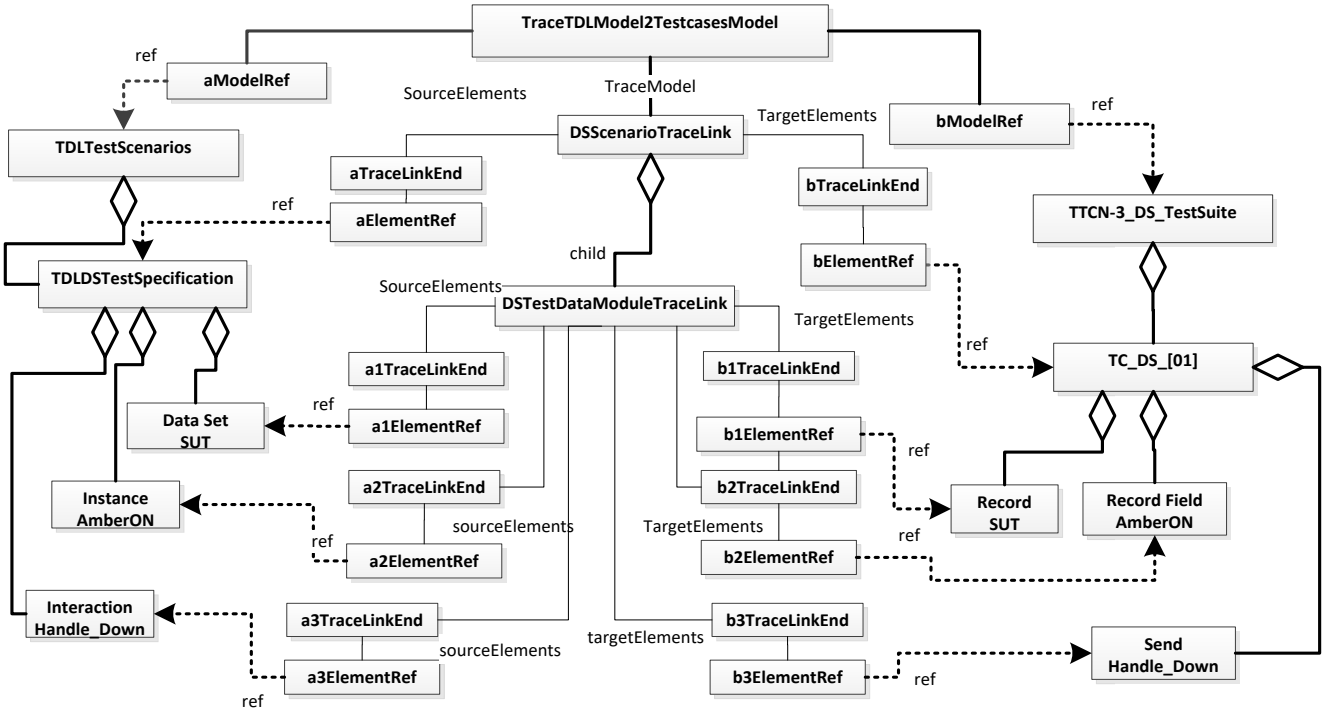

Fig. 11. Traceability Information between TDL and TTCN-3 (Kesserwan Dissertation [42]).

B. Traceability Links and Alignment with Test Result

To evaluate the extended testing methodology in this work, the experiment method described in [8] is reused to generate the test case. The new obtained result is a trace model (tracemodel.ecore) which relates UCM scenario models to TTCN-3 test cases grouped in test suites. Each test case, generated with a unique identifier, is a sequence of actions and interactions with defined input parameter values and output parameter values. The execution of the test case results in the assignment of a test verdict; pass or fail. In the trace model, the links between requirements and test cases may have several possible cardinalities:

- One-to-one: one requirement is tested exactly by one test case and this test case tests only this requirement.
- One-to-many: one requirement is tested by several test cases and these test cases participate to test only this requirement.
- Many-to-many: one requirement is tested by several test cases, which are used to test several requirements.

Fig. 12 shows the relationships between the testing artifacts for the [DeploymentSucceeded] scenario. The traceability link DSScenarioTraceLink[1] relates the model UCMDSScenario to the model TDLDSTestSpecification which is related to several test cases via the traceability link DSScenarioTraceLink[2]. The generated test cases are children of the test suite TTCN-3_DS_TestSuite.

The trace model takes a significant importance in the test generation process. On one hand, it provides a clear meaning for each generated test case: the tested requirement(s) gives the purpose of the associated test case(s). It is a kind of rationale for the generated test suite. On the other hand, the trace model exhibits clearly which requirements are actually tested (and how), and which requirements are not tested. For the not tested requirements, this suggests completing the test suite to obtain full functional coverage. During test execution of the test case, the traceability links in the trace model help to identify the related requirements when it fails. Similarly, when the test case passes, they certify that the related requirements were implemented and tested.

C. Requirement Coverage and Compliance with DO-178C

The trace model helped analyze the generated TDL test description from UCM models to check if the test cases cover the requirements. The trace model showed full coverage between UCM scenarios and their developed TDL specifications. The trace model realized complete requirement and scenario coverage. For each path in the UCM model, there is a TDL test scenario linked to it and the number of links in the trace model equals the number of scenarios found in the UCM model.

Furthermore, the trace model helped analyze the generated TDL test description to check if they are actually traceable to the original software requirements (UCM elements). The trace model meets the traceability objective as defined by DO-178C standard where an association between a requirement and its related items is necessary. The trace model contains links between the UCM models and the TDL test scenarios which in turn are traced to the generated test cases in TTCN-3. Therefore, compliance with DO-178C is achieved.

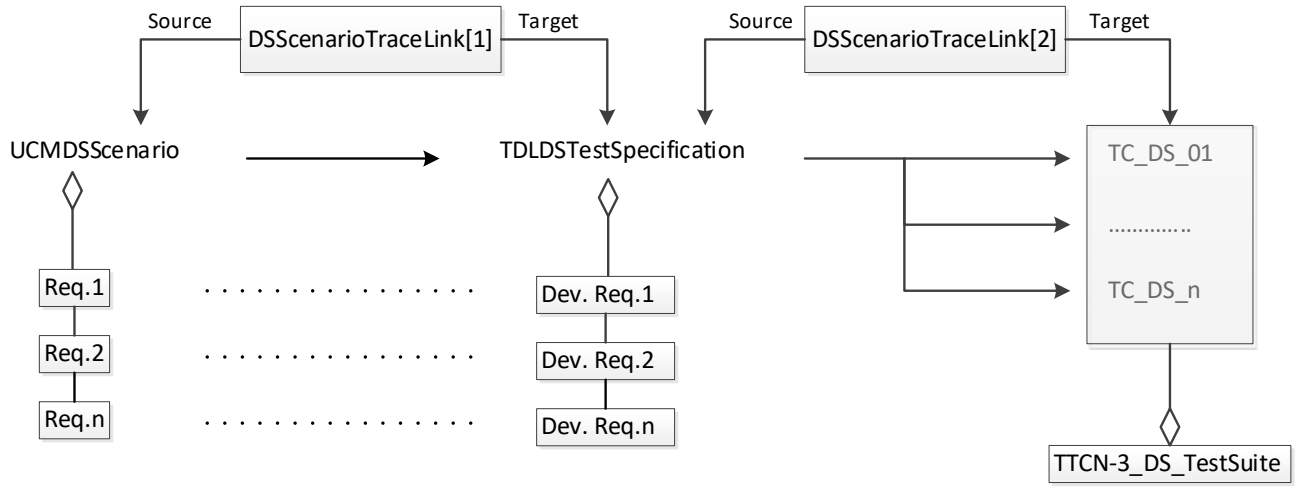

Fig. 12. Traceability Links among Testing Models for [DeploymentSucceeded] Scenario.

VI. Discussion and Future Work

Similar to the approaches discussed in the related work section (Section 3), this paper proposes to create traceability links among testing artifacts. However, this work differs from them as the proposed method extends the model-driven testing methodology to create explicit relationships in a trace model among testing artifacts. The approach creates UCM behavioral models and relates them to test cases via abstract test models during model transformation where n-ary links among models could be visualized. This is an important factor in visualizing relationships among models because it is almost impossible to represent more than one link in a two-dimensional traceability matrix in an understandable way. Moreover, the number of relationships in traceability matrices is high and fixed. The trace model records a small number of relationships from model to a testcase to enable the support for model-based coverage analysis, visualizing traceability and result evaluation.

Another important difference is creating a semiautomatic process for trace recording. This reduces some of the repetitive and time consuming tasks testers need to do to generate these traceability connections. Most models discussed require manual recording. This also distinguishes this work from the earlier research in this specific topic as it extends the scope and capabilities of the model developed and improves its processes.

This work is the start of research efforts to offer more effective ways to ensure traceability and create better pathways for validation. Following this contribution, future work will focus on enhancing the model to provide additional traceability aspects and addressing some of the current limitations. More research into enhancing the traceability process such that it could use additional sources (other than UCM) to provide access to non-functional requirements. This will further improve the traceability model and provide a more robust coverage of requirements. In addition, methods to automate the steps in this process will be investigated and a fully automated process of recording traces in the trace model will be explored. This will create a faster and more effective process for test traceability.

As a result, non-functional requirements, generally not captured by UCM, cannot be used. In addition, the semi-automatic recording improved the process, yet it still requires manual work to complete the process.

VII. Conclusion

Our main contribution in this paper is the proposal and presentation of a model-based approach that leverages available methods to generate test artefacts based on model transformations. This approach enables creating traceability links among testing artifacts. It also extends the transformation methodology to create and document relationships as a set of metadata in a trace model through consecutive transformation steps. A traceability scheme with constraints that determines which testing artifacts and at which level of detail the traces can be recorded was defined. The proposed traceability scheme guides the recording of traces (manual) and makes them persistent. Relationships are created and made explicit among scenario definitions in UCM models, their test specifications in TDL notation, and the corresponding test suite scenario in TTCN-3 language. The documented relationships in a trace model enable the support for visualizing traceability, coverage analysis and test result evaluation. This paper shows the developed infrastructure and workflow for MBT that applies model transformation and test generation techniques to create test scenarios, test cases, and traceability models.

References

[1] Tanvir Hussain and Robert Eschbach, "Automated Fault Tree Generation and Risk-Based Testing of Networked Automation Systems," in Proceedings of 15th IEEE Conference on Emerging Technologies and Factory Automation (ETFA 10) Bilbao, Spain, 2010.

[2] Lago, P., Muccini, H., van Vliet, H.: A scoped approach to traceability management. J. Syst. Softw. 82(1), 168–182 (2009).

[3] Winkler, Stefan, and Jens von Pilgrim. "A survey of traceability in requirements engineering and model-driven development." Software & Systems Modeling 9.4 (2010): 529-565.

[4] Galvao, I., & Goknil, A. (2007, October). Survey of traceability approaches in model-driven engineering. In 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007) (pp. 313-313). IEEE.

[5] Aichernig, Bernhard K., Wojciech Mostowski, Mohammad Reza Mousavi, Martin Tappler, and Masoumeh Taromirad. "Model learning and model-based testing." In Machine Learning for Dynamic Software Analysis: Potentials and Limits, pp. 74-100. Springer, Cham, 2018.

[6] J. Dick, Faivre, A., Automating the Generation and Sequencing of Test Cases from Model-Based Specifications, Springer-Verlag, 1993.

[7] D. J. Richardson, Aha, S. L., O'Malley, T. O., Specification-based test oracles for reactive systems, Proceedings of the 14th international conference on Software engineering, ACM Press, Melbourne, Australia, 1992, pp. 105-118.

[8] Kesserwan, N., Dssouli, R., Bentahar, J., Stepien, B. and Labrèche, P., 2017. From use case maps to executable test procedures: a scenario-based approach. Software & Systems Modeling, pp.1-28.

[9] Pinheiro, F.A.C.: Requirements traceability. In: Sampaio do Prado Leite, J.C., Doorn, J.H. (eds.) Perspectives on Software Requirements, pp. 93–113. Springer, Berlin (2003).

[10] DO-178C, available from RTCA at www.rtca.org. Retrieved 01/22/2021.

[11] Bernhard Schatz. 2011. 10 Years Model-Driven -- What Did We Achieve?. In Proceedings of the 2011 Second Eastern European Regional Conference on the Engineering of Computer Based Systems (ECBS-EERC '11). IEEE Computer Society, Washington, DC, USA, 1-. DOI=http://dx.doi.org/10.1109/ECBS-EERC.2011.42.

[12] Kienzle, Jörg, et al. "A unifying framework for homogeneous model composition." Software & Systems Modeling 18.5 (2019): 3005-3023.

[13] Eclipse Modeling Framework (EMF), available at http://www.eclipse.org/modeling/emf/, retrieved 01/22/2021.

[14] DO-178A Software Considerations in Airborne Systems and Equipment Certification, Document Number: DO-178A, Issue Date: 3/22/1985, Committee: SC-152, Category: Software.

[15] Zaman, Qamar uz, Aamer Nadeem, and Muddassar Azam Sindhu. "Formalizing the use case model: A model-based approach." Plos one 15, no. 4 (2020): e0231534 Buhr, R.J.A.: Use case maps as architectural entities for complex systems. IEEE Trans. Softw. Eng. 24(12), 1131–1155 (1998).

[16] Buhr, R.J.A.: Use case maps as architectural entities for complex systems. IEEE Trans. Softw. Eng. 24(12), 1131–1155 (1998).

[17] ITU-T Z.151 - the International Telecommunication Union, available at https://www.itu.int/en/pages/default.aspx, retrieved 01/22/2021.

[18] https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.38.9896&rep=rep1&type=pdf.

[19] G. Spanoudakis, Zisman, A., Software Traceability: A Roadmap, Advances in Software Engineering and Knowledge Engineering, World Scientific Publishing, 2005.

[20] Philip Makedonski, Gusztav Adamis, Martti Käärik, Andreas Ulrich, Marc-Florian Wendland, Anthony Wiles. “Bringing TDL to users: A Hands-on Tutorial” User Conference on Advanced Automated Testing (UCAAT 2014), Munich.

[21] TTCN-3 Standards, available at http://www.ttcn-3.org/index.php/downloads/standards.

[22] F. Fraikin, Leonhardt, T., SeDiTeC — Testing Based on Sequence Diagrams, 17th IEEE International Conference on Automated Software Engineering, 2002, pp. 261 - 266.

[23] Gagarina, Larisa G., Anton V. Garashchenko, Alexey P. Shiryaev, Alexey R. Fedorov, and Ekaterina G. Dorogova. "An approach to automatic test generation for verification of microprocessor cores." In 2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), pp. 1490-1491. IEEE, 2018.

[24] J. Wittevrongel, Maurer, F., SCENTOR: Scenario-Based Testing of E-Business Applications, Tenth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2001, pp. 41 - 46.

[25] L. C. Briand, Labiche, Y., A UML-Based Approach to System Testing, 4th International Conference on the Unified Modeling Language (UML), Toronto, Canada, 2001, pp. 194-208.

[26] F. Basanieri, Bertolino, A., Marchetti, E., The Cow_Suite Approach to Planning and Deriving Test Suites in UML Projects, Proceedings of the 5th International Conference on The Unified Modeling Language, Springer-Verlag, 2002, pp. 383-397.

[27] W. Grieskamp, Nachmanson, L., Tillmann, N., Veanes, M., Test Case Generation from AsmL Specifications - Tool Overview, 10th International Workshop on Abstract State Machines, Taormina, Italy, 2003.

[28] Naslavsky, Leila, Hadar Ziv, and Debra J. Richardson. "Towards traceability of model-based testing artifacts." Proceedings of the 3rd international workshop on Advances in model-based testing. ACM, 2007.

[29] F. Basanieri, Bertolino, A., Marchetti, E., The Cow_Suite Approach to Planning and Deriving Test Suites in UML Projects, Proceedings of the 5th International Conference on The Unified Modeling Language, Springer-Verlag, 2002, pp. 383-397.

[30] Anquetil, N., Kulesza, U., Mitschke, R., Moreira, A., Royer, J. C., Rummler, A., & Sousa, A. (2010). A model-driven traceability framework for software product lines. Software & Systems Modeling, 9(4), 427-451.

[31] Arcelli, D., Cortellessa, V., Di Pompeo, D., Eramo, R., & Tucci, M. (2019, March). Exploiting architecture/runtime model-driven traceability for performance improvement. In 2019 IEEE International Conference on Software Architecture (ICSA) (pp. 81-90). IEEE.

[32] Bünder, H., Rieger, C., & Kuchen, H. (2017). A Model-Driven Approach for Evaluating Traceability Information. Model-Driven Software Development, 436.

[33] A. Hartman, Nagin, K., The AGEDIS tools for model based testing, 2004 ACM SIGSOFT international symposium on Software testing and analysis, ACM Press, Boston, Massachusetts, USA, 2004, pp. 129-132.

[34] W. Grieskamp, Nachmanson, L., Tillmann, N., Veanes, M., Test Case Generation from AsmL Specifications - Tool Overview, 10th International Workshop on Abstract State Machines, Taormina, Italy, 2003.

[35] F. Abbors, D. Truscan, and J. Lilius, "Tracing requirements in a model-based testing approach," in 2009 First International Conference on Advances in System Testing and Validation Lifecycle (VALID), Piscataway, NJ, USA, 2009, pp. 123-8.

[36] D. Arnold, J. P. Corriveau, and Shi Wei, "Modeling and validating requirements using executable contracts and scenarios," in 8th ACIS International Conference on Software Engineering Research, Management and Applications (SERA), CA, USA, 2010, pp. 311-20.

[37] A. Goel, B. Sengupta, and A. Roychoudhury, "Footprinter: Round-trip engineering via scenario and state-based models," in 31st International Conference on Software Engineering - Companion Volume - ICSE-Companion, Piscataway, NJ, USA, 2009, pp. 419-420.

[38] C. Pfaller, A. Fleischmann, J. Hartmann, et al., "On the integration of design and test: A model-based approach for embedded systems," in Proceedings of the 2006 international workshop on Automation of software test (AST) 2006, pp. 15-21.

[39] J. L. Boulanger and V. Q. Dao, "Requirements engineering in a model-based methodology for embedded automotive software," in IEEE International Conference on Research, Innovation and Vision for the Future in Computing 484 & Communication Technologies(RIVF), Ho Chi Minh City, Vietnam, 2008, pp. 263-268.

[40] M. Felderer, P. Zech, F. Fiedler, et al., "A Tool based Methodology for System Testing of Service-oriented Systems," in Second International Conference on Advances in System Testing and Validation Lifecycle (VALID), Los Alamitos, CA, USA, 2010, pp. 108-13.

[41] R. Marelly, D. Harel, and H. Kugler, "Multiple instances and symbolic variables in executable sequence charts," in 17th International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2002), USA, 2002, pp. 83-100.

[42] Kesserwan, N. (2020). Automated Testing: Requirements Propagation via Model Transformation in Embedded Software (Doctoral dissertation, Concordia University).

[43] http://istar.rwth-aachen.de/tiki-index.php?page=jUCMNav

[44] ETSI ES 203 119-1 V1.3.1 standard, available at http://www.etsi.org/deliver/etsi_es/203100_203199/20311901/01.03.01_60/es_20311901v010301p.pdf, retrieved 01/22/2021.

[45] Ulrich,A., Jell, S.,Votintseva, A.,Kull, A.: The ETSI TestDescription Language TDL and its application. In: 2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD), pp. 601–608. IEEE (2014, January).

[46] P. Stocks, Carrington, D., A Framework for Specification-Based Testing, IEEE Transactions on Software Engineering, 1996, pp. 777-793.

[47] M. Didonet Del Fabro, Bézivin, J., Valduriez, P., Weaving Models with the Eclipse AMW plugin, Eclipse Modeling Symposium, Eclipse Summit Europe 2006, Esslingen, Germany, 2006.

[48] Boniol, F.,Wiels, V.: The landing gear system case study. In: ABZ 2014: The Landing Gear Case Study, pp. 1–18. Springer (2014).