A Model-based Test Case Management Approach For Integrated Sets Of Domain-Specific Models

Reinhard Pröll
Institute for Software & Systems Engineering
University of Augsburg, Germany
Email: reinhard.proell@informatik.uni-augsburg.de

Bernhard Bauer
Institute for Software & Systems Engineering
University of Augsburg, Germany
Email: bernhard.bauer@informatik.uni-augsburg.de

***Abstract*—Due to rapid improvements in the area of embedded processing hardware, the complexity of developed systems constantly increases. In order to ensure a high quality level of such systems, related quality assurance concepts have to evolve. The introduction of Model-Based Testing (MBT) approaches has shown promising results by automating and abstracting multiple activities of the software testing life cycle. Nevertheless, there is a strong need for approaches supporting scoped test models, i.e. subsets of test cases, reflecting specific test purposes driven by risk-oriented development strategies. Therefore, we developed an integrated and model-based approach supporting test case management, which incorporates the beneficial aspects of abstract development methodologies with predominant research for test case management in non-model-based scenarios. Based on a new model artifact, the integration model, tasks like cross-domain information mapping and the integration of domain-specific KPIs derived by analyses favor the subsequently applied constraint-based mechanism for test case management. Further, a prototypical implementation of these concepts within the Architecture And Analysis Framework (A3F) is elaborated and further evaluated based on representative application scenarios. A comparative view on related work leads to a conclusive statement regarding our future work.**

***Index Terms*—Model-Based Testing, Test Case Management, Test Selection, Test Prioritization, Test Suite Reduction, Test Model Scoping**

I. Introduction

The continuously increasing resources and processing power of embedded hardware enable developers to accomplish more complex tasks than in the early days of embedded computing. Moreover, the rising task-complexity of advanced applications may hardly be engineered by entrenched development techniques. In order to reduce the induced complexity to a manageable level, the concepts of abstraction and automation have to be applied, leading to model-based approaches.

Besides the pure system/software development approaches, an appropriate testing methodology is necessary. Especially in the embedded domain, the requirements for sufficient testing are much harder to achieve compared to non-embedded software applications, e.g. in the business domain. Here, the overhead for deploying new software to already released products, altogether with greater efforts for simulation, debugging and testing during development, reflect an excerpt of the domains challenges. In order to raise the quality of the currently manual testing efforts and reducing the overall complexity of testing, companies, no matter if software development is carried out in a model-based fashion or not, tend to switch over to Model-Based Testing (MBT) approaches [1].

On the one hand, these approaches mitigate the risk for weak testing, by serializing and strongly structuring the tester's mindset. Furthermore, the potential for automated test case generation and subsequent execution opens up new application scenarios that integrate such technologies into continuous integration (CI) and testing toolchains.

On the other hand, Pretschner and Philipps identified some methodological issues in MBT, potentially invalidating instantiations of such approaches [2]. One of these issues is related to the lack of redundancy and separation of concerns in the underlying model basis, leading to slightly divergent or unforeseen side effects. For example, a single model artifact used for automated code as well as test generation inevitably leads to tests for the applied code generator instead of tests for the developed functionality. Apart from that, fixing the redundancy concept on the model level introduces new challenges for the integration of participating model artifacts. Especially, processing steps evaluating data from various domain-specific models[1] artifacts, such as test case selection or intermediate impact analyses in the context of model evolution, may not easily be performed using such a modeling principle.

A. Problem Statement

Besides the methodological pitfalls of MBT in general, the pure adaption of traditional approaches for test case selection, prioritization, as well as reduction, to the context of MBT will probably fail or even perform worse, due to the following observations.

Firstly, state-of-the-art approaches in non-MBT scenarios are mostly limited to a very specific application context, as Engström et al.'s review reveals for the domain of regression testing [3]. Its internal adequacy criteria mostly focus on a small subset of development artifacts, such as the test specification or models of the system under development. Due to the limited context of these adequacy criteria, data modeled apart from these artifacts, potentially improving the overall result, may not even be taken into account. Although, the predominant approaches show promising results, extending the set of incorporated information is expected to perform better.

[1]domain-specific model: a purpose- and problem-specific development artifact

Secondly, the state-of-the-art arrangement of test case management related tasks in the testing life cycle, which are widely adapted from non-MBT scenarios, intensifies the complexity challenge. Nearly every state-of-the-art approach does the selection, prioritization, or reduction of the test suite, right before or during the generation of concrete test cases. While there are heuristic and algorithmic techniques for MBT scenarios to improve this subsequent concrete test case generation, there is no data-driven approach for test case management known to us, which is applied to the underlying model artifacts. [4]

As a consequence, the starting point for the predominant heuristic and algorithmic approaches may further be optimized, in terms of the pure size of the solution space. Therefore, a fix to the order or extension of processing steps is expected to mitigate this weakness and play off the beneficial effects of model-based approaches concerning the complexity challenge.

To overcome the identified problems in a model-based development and testing scenario, we aim for an extensible and model-based test case management methodology, which is based on an integrated heterogeneous set of domain-specific model artifacts. The information integration respecting the domain separation finally leads to manageable and highly specific sets of test cases, apart from that favoring analyses supporting a controlled co-evolution of models or even impact analyses for subsequent model-based debugging or regressing testing.

Based on our previous work addressing the application context-independent foundations of an integrated set of domain-specific model artifacts, this contribution focuses on the beneficial use of the underlying model basis in the context of model-based test case management [5]. Moreover, the herein proposed approach covers an intermediate process step of our vision of a consistently model-based Software Testing Life Cycle (STLC) [6].

B. Outline

The contribution addressing the challenges identified in the previous section outlines as follows: Section II baselines the set-up of conventional development processes and sets the intended integrated model basis in relation, additionally drawing a high-level sketch of the solution. In section III the abstract solution is transferred to a set of metamodels alongside with a methodology, leading to the prototypical implementation within the Architecture And Analysis Framework. Section IV further demonstrates the application along with a running example. In order to reveal pursuing applications of this approach, a heterogeneous set of scenarios is discussed in section V. Section VI sets the contribution in relation to predominant work of affected research domains. Finally, a conclusive statement together with future research of this area is arranged in section VII.

II. The Generic Approach

Addressing the previously defined problem statement with regard to state-of-the-art development processes, we present

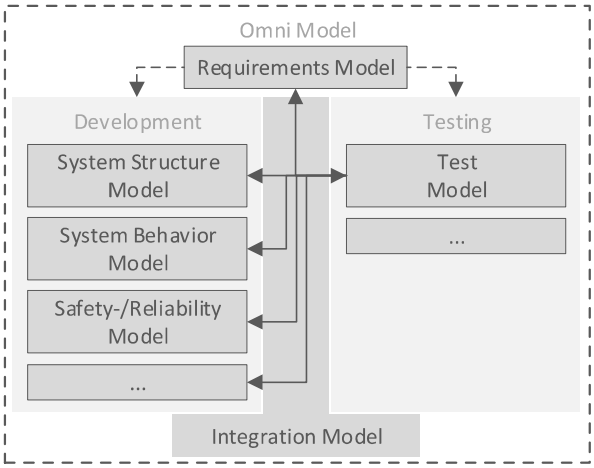


Fig. 1. The Omni Model and its relation to common development setups

an integrated model-based test case management approach incorporating heterogeneous domain-specific models.

Starting off with specifying requirements, every development process builds up its very first domain-specific model artifact, the requirements model. Based on this data, subsequent process steps, on the one hand, emerge the systems' architecture and functionality, on the other hand, verify and validate the development artifacts, to raise the belief in the softwares' correctness and appropriateness.

As shown in fig. 1, these three areas mark the basic building blocks of every software development setup. Depending on the systems' application environment, e.g. in an avionics, automotive or railway context, additional development artifacts are mandatory, in order to meet the requirements imposed. That means the set of model artifacts incorporated in a development process may vary.

Nevertheless, from a testers point of view, all of this information includes guiding information for subsequent testing activities. However, respecting the separation of concerns on the model level, for reasons of meaningful MBT efforts, prevents the tester from effectively using this information. To overcome this shortcoming, an additional domain-independent model artifact, namely the *Integration Model (IM)*, is introduced. Taking a test-centric view of the integration model, the new model artifact serves multiple purposes:

First, this model artifact tries to close the gap illustrated in fig. 1, by making up an intermediate representation of the system under development and test. Further, this structural breakdown of the instantiated system into components allows to explicitly link information across the heterogeneous landscape of domain-specific models. The so far inaccessible information may thereby be used for several purposes throughout the development and testing lifecycle.

Second, the integration model is meant to give developers the possibility to attach data, which is produced by analyses of domain-specific artifacts, such as results of risk assessments based on reliability models of the system. The input and output

data of such analyses should not compulsorily be mixed up, in order to preserve the separation of concerns on another level and its emerging beneficial effects. Beyond the pure serialization of such data, subsequent development steps may base upon these results to perform better, which leads us to the next purpose of the integration model.

Third, the previously mentioned analysis result data may serve as another information basis for automated test model processing, which is driven by the integration model. The processed model artifact is meant to reflect the testers focus, specified by a flexible set of aspects, which are taken into account. Further, supporting the vision of a largely automated and effective model-based testing methodology.

Beside the pure structure and information linking mentioned above, further steps of a model-based software testing life cycle have to deal with the behavioral characteristics of the system. Therefore, the models reflecting the behavior, potentially varying across the participating domains, are synchronized by a mapping, which is embedded in the integration model.

Overall, the described integration model together with the set of domain-specific development and test models makes up the so-called *Omni Model*, originally introduced by Rumpold et al. and conceptually embedded in the Architecture And Analysis Framework (A3F) [5].

Beside the model-/data-oriented view on the topic, the further on presented methodology may be put in a greater context. On the one hand, the presented omni model approach marks the conceptual basis for the consistent and strictly model-based software testing life cycle introduced by Pröll et al. [6].

On the other hand, the methodology for scoping a certain subset of the original test model (see section III-B), reflecting the current test focus, determines a certain process step of this model-based STLC. Instead of going into details of the methodology, we start off with a more detailed view of the meta-model concepts behind.

III. Instantiation Of The Generic Approach

Following the description of the generic approach in section II to overcome the challenges identified during section I-A, we further present an instantiation of this generic approach including the core metamodels, the methodology test model scoping, as well as some insights to the technical realization.

A. Meta Model Concepts

As stated in the introduction, development processes commonly consume and produce a huge amount of data artifacts. Especially in model-based approaches, these development artifacts represent the basic building blocks for ongoing automation purposes and therefore need to conform to respective meta models. Due to the openness of the omni modeling approach, we further focus on the essential set of meta models serving for the later on presented methodology (see section III-B).

a) The Integration Meta Model: This meta model introduces concepts essential for the solution vectors identified in previous sections. In order to close the conceptual gap between predominant development artifacts, an intermediate representation of the systems' structural breakdown together with explicit information mapping capabilities is initially focused.

Figure 2a shows an excerpt of the EMF metamodel, implementing related basics. On the right-hand side of the figure, concepts for specifying the hierarchical decomposition of the instantiated system take place. Thereby, the decomposition is represented by a tree-like structure consisting of nodes, either representing logical components of the system (*IMComponent*) or behavioral units of such components (*IMFunctionality*). Furthermore, the hierarchical structure reveals the granularity of the developed system parts (*IMPartOf*, *IMGeneralize*). Based on the hierarchical representation of the system, concepts for information mapping across the landscape of domain-specific model artifacts are defined. For coarse mappings, model artifacts may directly be connected to the previously introduced nodes (*IMTrace*). A more detailed mapping of model data is achieved by a set of anchor elements (*IMTraceAnchor*), enabling the developer to specify arbitrary information mappings.

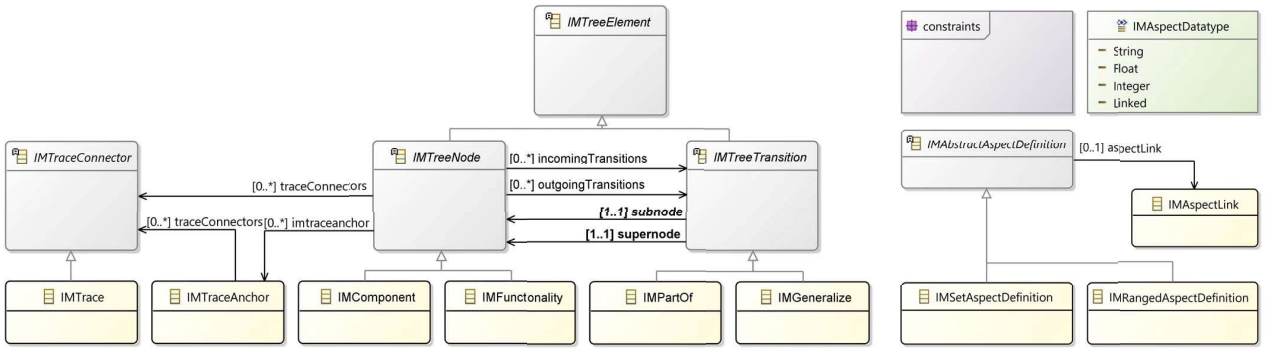
Beside the pure specification of mapping relations across models, the integration metamodel furthermore defines concepts for the definition and specification of quantifiable characteristics, hereinafter referred as *Aspects*. These aspects enable the test engineer to specify data, supporting the subsequent selection and prioritization of test cases. Figure 2b shows an excerpt of the EMF metamodel defining the aspects concept. The aspects subsequently attached to elements of the hierarchical decomposition initially need to be clearly defined (see lower part of fig. 2b). An aspect may either represent predominant data fields of connected domain-specific models, specified by an explicit link (*IMAspectLink*). The second option is to synthetically define aspects, in order to incorporate data produced by complex analyses or to represent impacting factors. For instance, not directly measurable, but quantifiable experience values of domain experts or results of previous risk evaluations represent potential aspects of the system. Their range of values may either be defined on a continuous scale (*IMRangedAspectDefinition*) or based on an explicit set definition (*IMSetAspectDefinition*), depending on the aspects nature. Listing III-A0a shows two snippets of our internal DSL for such definitions.

```
srname:linked req:REQRequirement:name;

sillevel:Integer:ranged [1,4];
```

Listing 1. Definition of a predominant and a synthetic aspect

The first aspect defined within this textual definition, named *srname*, aims for the incorporation of requirements' name attributes, to be found in the respective requirements model (`req:REQRequirement:name`). The second aspect deals with synthetic information addressing the results of a safety



(a) tree package (simplified) (b) aspect package (simplified)

Fig. 2. Excerpts of the integration meta model

integrity level (SIL) assessment (*sillevel*). Based on a purpose-specific model artifact, such as a Failure Modes And Effects Analysis (FMEA), the computed SIL levels are further on represented by this ranged aspect of the integration model.

As already mentioned, a concept for the specification of aspect values alongside the elements of the hierarchical decomposition is incorporated. At this point, a string-based specification of such values turned out to be more comfortable, therefore implemented by an attribute value of a node (omitted in fig. 2a due to detail level). Essentially, these specifications reflect key-value pairs, while *key* references the name of an aspect definition, e.g. *sillevel*, and *value* represents a set of values, which conforms to the definition.

The last metamodel concept responsible for determining a dedicated test focus based on the aspect data is embedded in the aspect constraint package (see fig. 2b). As its name implies, a set of concepts for specifying constraints for the initially defined aspects and subsequently specified aspect values is defined by the integration model. Thereby, an aspect constraint defines an expression for the subsequent evaluation against the aspect specifications embedded in the hierarchical decomposition of the system. These expressions may either address a single aspect and its respective values or reflect a combination of atomic constraints via boolean operators. Listing III-A0a shows a snippet of our constraint DSL.

```
srname:in ['SR3'] & sillevel:gre ['3']
```

Listing 2. Constraint Set for Aspects

The first constraint checks whether the focused integration model element maps a requirement, whose name attribute equals to “SR3”, or not, and consequently evaluates to *true* or *false*. The second constraint checks if the synthetic aspect specified alongside the integration model element is greater or equal to 3. The boolean values representing the results of the atomic constraints are further evaluated according to the logic of boolean expressions, for instance by the logical *AND* operator.

Overall, the presented excerpt of the metamodel concepts facilitates the further on detailed methodology. Beside this essential metamodel, which marks the basis for pruning activities, another metamodel is presented, facilitating the uniform processing of possibly heterogeneous test models, which is detailed in section III-B and section III-C.

b) The Execution Graph ++ Meta Model: As previously illustrated, the integration model together with the test model plays the central role in our contribution. Subsequent internal processing steps are based on a metamodel, which is independent of the original test modeling language. In general, test cases define a chain of sending stimuli to the SUT, receiving stimuli of the SUT, and comparing received stimuli against the tester’s expectations, conforming to the specification. Figure 3 represents the metamodel of an execution graph-like structure, serializing these chains of events occurring during the execution of test cases.

The execution chains are encapsulated in a graph (*EGPPGraph*) consisting of nodes (*EGPPNode*), i.e. fragments of a test case, and transitions between these nodes (*EGPPTransition*). Due to the fact, that test models commonly aggregate multiple test cases within one model artifact, several specializations of the node metamodel concept have been integrated. In order to establish a rudimentary structuring concept for sets of test cases within such a graph, intermediate nodes may also represent a subgraph. Beside the concepts making up the control flow from a defined start node (*EGPPInitialNode*) to one or more final nodes (*EGPPFinalNode*), additional information (*EGPPTaggedData*) may be attached to each of the presented elements. A common use case for this tagged data is given in section III-B, where domain-foreign information is introduced to the generic representation of the test model to improve the results of subsequent test generators.

Overall, the presented metamodel covers a versatile set of application scenarios. For instance, an integrated view of the structural and behavioral models of the system under development may be build up internally, in order to achieve a uniform internal representation for subsequent analyses.

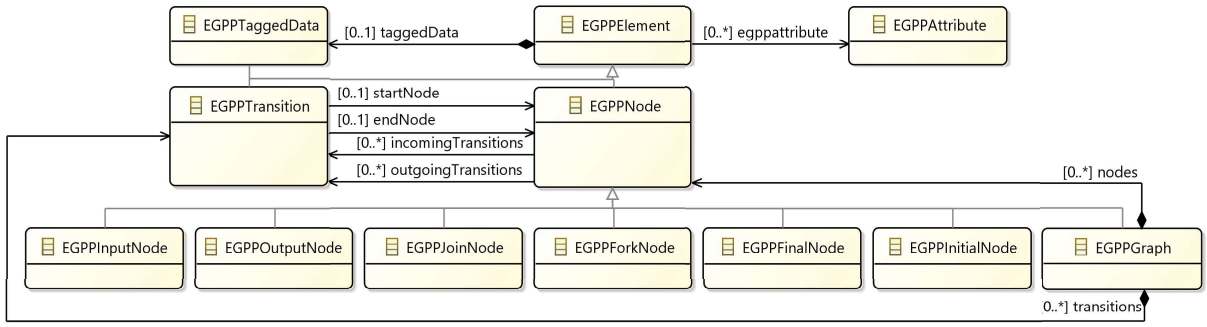


Fig. 3. Meta Model of the Execution Graph ++

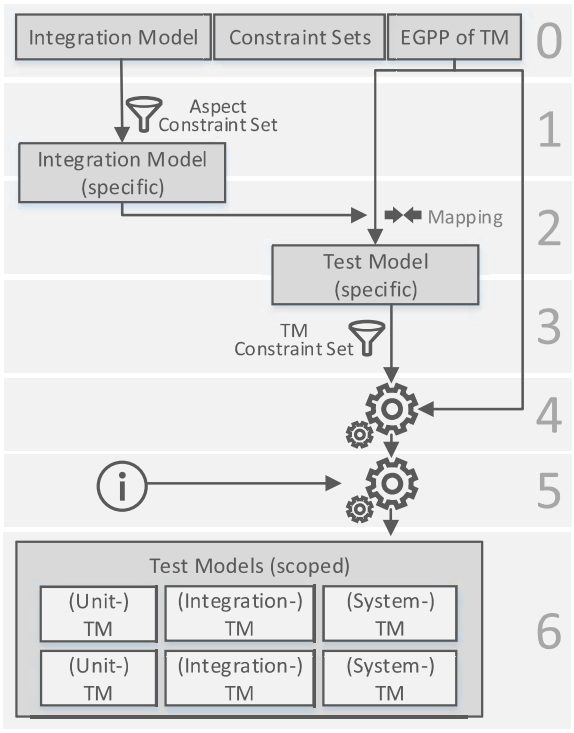


Fig. 4. Breakdown of the Test Model Scoping Process

B. Methodology For Scoped Test Models

Based on the concepts presented in section III-A, a methodology for model-based test case management is further presented. Altogether, we aim for an intuitive and flexible way for specifying the testers focus, consequently supporting the selection and prioritization of test cases in an automated way. In order to achieve the goal of scoped test models, reflecting the intended test focus, a multi-step process has been defined. Figure 4 shows the breakdown of the test model scoping (TMS) methodology detailed throughout this section.

Step 0 of the TMS processing chain represents the set of inputs incorporated in subsequent steps. A complete integration model, the internally used generic graph representation of the test model, further named test model, and two sets of constraints mark the starting point. The first constraint set is the aspect constraint set previously mentioned in section III-A, which is applied to the integration model artifact. However, the second set of constraints aims for the test model embedded in the omni model. Before the mentioned model artifacts as well as the constraint sets are processed by further steps, both a syntactic verification and a check for completeness is done, to ensure seamless processing.

Step 1 of the TMS process determines, which model elements of the integration model fulfill the previously specified aspect constraints. Further, incorporating the predominant model data linked via the IM, the set of aspects is evaluated. Model elements irrelevant for the defined scope are eliminated during this step by pruning the integration model. In case of a too restrictive set of aspect constraints, this reduction step possibly leads to an empty result. This terminates the processing chain consequently either forcing the modeler/tester to adjust the aspect-related data, specified in the integration model, or weaken the aspect constraints.

Step 2 further processes the intermediate integration model artifact generated during the previous step, as well as the original test model. Altogether, this step aims for a reduction of the test model with respect to the integration model reduction already accomplished. Therefore, the mapping information is used in a way, that each test model element sharing a related element in the reduced integration model, becomes part of the reduced test model. An important factor for the quality of the resulting model artifact is the level of detail of the modeled mapping relations. Further, this transformation step may lead to an intermediate test model, not necessarily conforming the proposed syntactic and semantic requirements. However, the subsequent processing step does not incorporate this as a precondition and therefore does not interrupt the processing at this point.

Step 3, quite similar to step 1, scopes dedicated parts of the intermediate test model by applying the second constraint set, which is specific to the test model artifacts. Depending on the

nature of the underlying test modeling approach and its meta-model, various constraints are conceivable. An obvious choice representing characteristics used to structure traditional testing approaches leads to constraining factors such as follows:

- test model parts representing certain test levels (unit, component, system, acceptance)
- extraction of particular sub test model(s) for specific parts of the SUD

Based on an intermediate scoped version of the test model artifact, these kind of constraints are sufficient for further scoping model parts and finally reflecting the tester's mindset. As stated above, the intermediate model artifacts are not necessarily correct in their syntax and semantic.

Step 4 fixes this weakness by combining the original version of the test model and the intermediate results of the previous processing step. Based on these two datasets a matching and completion of the potentially fragmented intermediate test models are performed, to restore the syntactic and semantic requirements. Besides the basic reconstruction of the test models, model parts obviously not representing any reasonable test description are eliminated by the Model Analysis Framework (MAF) [7].

Step 5 enriches the set of correct test models with additional information from other domain-specific models of the omni model. This appears to be a contradiction to the continuous reduction and scoping of the test models, but in fact, utilizes more sophisticated test case generators incorporating these additional properties to raise the quality of resulting test cases. Further usage scenarios include improved test documentation for subsequent reviews or test reports.

Step 6 marks the final step of the TMS process, where the test models generated by the previous processing steps are exported to an appropriate format. For further usage within the model-based STLC [6], the generic graph representation (EGPP) mentioned in section III-A is chosen for these test models.

Based on these scoped test models, the subsequent test case generation, which is guided by an adequacy criteria responsible for the completeness of the resulting test suite, produces a set of test cases, which reflects the specified test focus with no need for further manual processing.

C. Technical Realization Of the TMS Approach

Besides the underlying concepts and the methodology for test case management, the technical realization including its tool support decides about the acceptance and its success. Therefore, we take a closer look onto the technical realization of the contents presented in previous chapters.

As already mentioned in section II, the Architecture And Analysis Framework (A3F) implements the omni-modeling approach as its integrated model basis for subsequent computations. Further, the included mechanism for the specification of configurable analyses upon these models, together with its concept for chaining analyses, fits very well with the technical challenges of the presented approach. Figure 5 shows the building blocks incorporated by the Architecture And Analysis Framework, to implement the developed functionality.

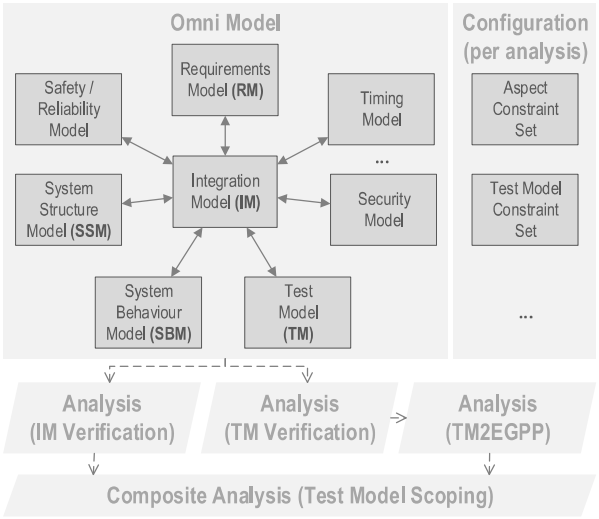


Fig. 5. Breakdown of the Technical Realization

The Omni Model, a heterogeneous and extensible set of domain-specific models, is realized by a loosely coupled landscape of EMF-based meta-models, making up the internal representation of the integrated model basis. The original model information, specified by domain-experts, thereby may be modeled in a general-purpose modeling language or any tool-specific language. This model information is transformed into the internal EMF-based metamodels by a set of QVTo-based Model-to-Model (M2M) transformations, to be ready for internal computation steps, encapsulated in the so-called *analysis* concept.

Depending on the set of analyses to be performed by the A3F, a set of configurations needs to be specified by an expert. The therein specified parameters determine and extend the information to be processed by the respective analysis. For the TMS-related analyses, the incorporated constraint sets represent an essential part of its overall configuration.

The lower part of fig. 5 deals with the chaining of analyses within the A3F. As remarked in section III-B the TMS process splits up into several analyses marking up the final functionality. First, the internal models are checked for syntactical and semantical conformance with the internal meta models together with a check for completeness in a sense of the minimal required set of information to perform ongoing processing steps. These kind of analyses are specified in a plain Java-based fashion. Second, the test model walks through another QVTo-based M2M transformation step, producing the generic graph representation of the test model (EGPP), to unify the subsequent TMS processing steps. Last, the composite analysis specifying the previously detailed TMS logic is triggered. Herein, the logical steps are either realized as plain Java-based analyses or MAF analyses for processing the model artifacts.

Overall, the TMS processing chain marks an excellent application scenario for the generation of purpose-specific data within the A3F.

IV. Test Model Scoping Applied To An Omni-Modeled Tank Control System

The concepts, as well as the prototypical implementation within the A3F, are furthermore demonstrated by means of an exemplary application to a tank control system. The system's purpose is to regulate the fill level of multiple, but possibly heterogeneous tanks. Depending on the type of the stored fluid, a varying set of regulatory requirements needs to be fulfilled, further determining the incorporated set of components making up the complete system. Our use case incorporates two different types of tanks, a water tank, and an acid tank. The water tank in our case study is not subjected to any safety regulations, whereas the acid tank is. Due to entrenched architectural patterns for such systems, some of the atomic components are implemented redundantly.

As mentioned in section III as well as in the contribution of Rumpold et al., the integration model serves as a bridging model artifact throughout an omni model-based development process [5]. As one of the first steps, a hierarchically decomposed, component-based, and instantiated sketch of the system under development is created, as included in fig. 6.

Starting with the top-level element *Tank Control System*, the IM incorporates the decomposition into sub-elements, namely the *Acid Tank*, the *Water Tank*, the *Display*, and the *Main Controller*. The latter represents a dedicated functionality assigned to the tank control system, whereas the other elements represent logically or physically isolated subsystems, contributing to the overall system. Depending on the level of detail served by the integration model, the decomposition may further be extended.

The second model artifact taken into account by the TMS processing chain is the test model, specifying test cases in an activity chart-like manner. As an alternative representation, we could think of a test case specification, which conforms to the UML Testing Profile (UTP) or even a mixture of test modeling languages. With respect to the hierarchical decomposition of the system, tests are modeled and linked to respective elements of the integration model. Apart from the test domain, other domain-specific information, as mentioned during section II, is further connected to the integration model. Especially, the set of requirements initially modeled for serving as the systems' specification is linked to elements of the integration model. Each time a requirement impacts a certain component or functionality, which is part of the integration model, a relation (*IMTrace*) is specified and to be maintained throughout ongoing development steps. Especially the requirements concerning the systems safety considerations are of special interest during further investigations.

Beside the domain-specific models previously presented the set of available aspects has to be defined and specified, which has already been conducted in section III-A. In case of a requirement-oriented testing effort, the aspect regarding the requirements information enables testers to scope tests, which are relevant to their current testing objective. A more safety/reliability oriented tester would in contrast to this prefer to align his testing activities alongside the second aspect, which relates to the computed SIL level of a dedicated element of the integration model.

Alongside his test focus, the test engineer develops the aspect constraint set (Listing III-A0a), which again serves as input for the TMS processing chain. He further limits testing activities to unit testing, by specifying a test model constraint, which is about to filter the unit tests out of the final graph representation for the test model artifact. Ultimately, this narrows down the testers current interests to unit testing the system parts, that are safety-relevant (constraining the *srname* aspect) and ranked with SIL greater equals 3 (constraining the *sillevel* aspect). In future iterations of the testing life cycle, the aspect constraints may be tweaked or either expanded, to fit best to the testers needs.

At this point, all of the input data for the TMS approach has been specified and may further be applied. The first step applies the set of aspect constraints (Listing III-A0a) to the original integration model. Figure 7 details the mappings across domain-specific models incorporating the aspect specifications of the relevant parts.

The upper part of the figure includes the requirement addressed by the first aspect constraint. The lower left part again incorporates a subset of integration model elements, which were previously analyzed by means of their safety relevance. Applying the *sillevel*-constraint, we see that the *water tank* model element is currently out of the testers scope. The result generated by the first processing step is represented by the highlighted model elements in fig. 6.

The second step investigates on the mapping of model elements between the integration model and the test model. Again illustrated by fig. 7, the test model parts *Acid Tank Test Model* and *Acid Tank Controller Test Model* are filtered according to their specified trace relations to elements of the integration model.

The third step applies the test model constraint set to the intermediate set of test model elements. This effectively reduces the number of test model elements to one, because of the fact, that the *Acid Tank* model element was not specified as an unit test for the system, also indicated by the mapping to a non-leaf element of the integration model.

Steps 4 to 6 of the process, responsible for the reconstruction of valid test models (in more complex cases), the incorporation of additional domain-foreign information, and the final purpose-specific data serialization do not impact on the result of this tiny running example.

In contrast to the original model artifacts, the scoped models are cut down to an almost minimal and highly specific extent, which favors targeted testing. Furthermore, the trend toward short development cycles in agile development contexts is also favored by such scoped test model artifacts.

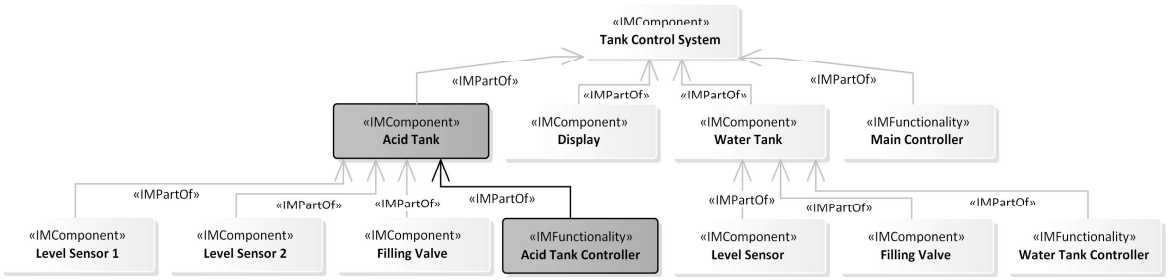


Fig. 6. Excerpt of the Integration Model of the Tank Control Systems' Omni Model

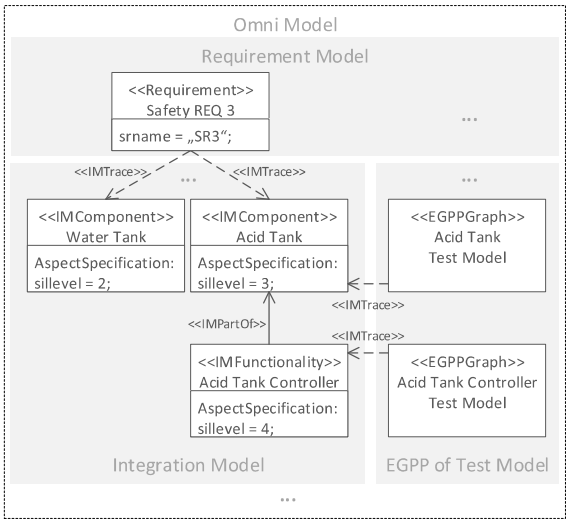


Fig. 7. Orchestration of Omni Model artifacts including aspect information

V. Scenario-Based Evaluation

Besides the exemplary application of our approach alongside a tank control system, aiming for the selection of an appropriate set of test cases, this section discusses further application scenarios. Thereby the adaptability and extendability of the approach are pointed out in a broader context.

Risk-Aware Test Case Prioritization

This scenario discusses the evolution of a model-based test process, to support risk-based testing strategies, by applying our approach. Therefore, a third-party framework for the assessment of heterogeneous sets of risks, potentially impacting the systems under development, is used. In order to reflect the results of such risk assessments, the computed risk values need to be assigned to the respective parts of the system. At this point, the integration model comes into play, embedding these risk ratings within appropriate synthetic or linked aspects. A subsequent and iteratively adapted definition of aspect constraints, regarding these new aspects, enables testers to do iterative and risk-based testing. Together with an appropriate test model or set of concrete test cases, also incorporated by the Omni Model, the application of the TMS process takes place. Executing the sequence of processing steps, enables testers to automate the risk-aware prioritization, or even selection, of test model artifacts.

Beside this analysis-based determination of prioritization data, alternative prioritization goals are thinkable, such as organizational factors imposed by management decisions.

Model-Based Regression Test Selection

The second scenario addresses a quite different use case, affecting the challenge of effective test case selection in regression testing. An essential challenge of regression testing is to identify the need to rerun a test in future iterations. In order to retrieve the knowledge about changes and their impact during development, impact analyses represent a state-of-the-art approach to such problems. Impact analyses on an integrated set of domain-specific models, such as the omni model, enable the automation of the regression test selection decision process.

For instance, the result of such impact analyses needs proper encoding within an aspect of the integration model consequently leading to scoped test models targeting the affected parts of the system. Regarding the dedicated steps of the TMS process, Step 0 may therefore include the definition of an aspect representing the amount of change introduced since the last run of related test cases. Furthermore, the aspect specification is performed by a change-impact-analysis performed on top of the integrated model basis, i.e. the omni model. Consequently, the set of aspect constraints, applied to the integration model during Step 1, needs to be extended in a sense, that the scoped version of the integration model represents the model parts focused for regression testing. Subsequent steps of the TMS process propagate these changes to the test model basis used for test case generation.

The practical realization of such a regression test case management approach may apart from that include the automation by modern CI Frameworks. Overall, this is expected to reveal the beneficial aspects of model-based process steps of the Software Testing Life Cycle.

Product line-Aware Test Case Management

The third and last scenario aims at model-based software product line development and test. Product line Engineering in

general addresses the challenge of developing a set of products in concert, while the products only vary in their choice of included features, but share a common set of software components (core features). For subsequent testing activities, the variety of products to be tested benchmarks the suitability of the instantiated testing life cycle. In order to optimize the test process steps following the test case generation, the test case selection and prioritization may be furthermore guided by yet, therefore, unused development artifacts, such as feature models.

The integration of such domain-specific model artifacts into our Omni Model enables the aspects concept on top to, e.g. determine test cases for a specific product of the product line or prioritize parts of the test model according to their cross-product relevance (core feature vs. optional feature). For instance, an aspect serving this purpose may incorporate the ID information of linked feature model elements. In combination with an aspect constraint determining an appropriate set of focused features via their ID, targeted testing for product line approaches is backed by the subsequent application of the TMS processing chain. Apart from the selection of test cases, an indicator for the prioritization of test cases may easily be introduced by additionally defining an aspect incorporating the type of the linked feature.

Especially, the automotive sector extensively uses product line-based approaches, in order to reduce engineering costs by unifying core components of the embedded software, as well as hardware. Beside the selection and prioritization of test cases, the reduction of test cases may either be achieved by a clever combination of specified aspects or an adapted choice of the incorporated test modeling language favoring holistic product line test models. This scenario may furthermore be seen in conjunction with the previous scenario for model-based regression test selection, therefore presenting a uniform solution for two previously disjoint problem domains, namely the integration model together with its embedded aspect concept.

Besides the three scenarios discussed, many other use cases such as security abuse-driven test case management or an application to legacy testing scenarios are thinkable, hereafter to be covered by this approach.

VI. Related Work

The proposed taxonomy for model-based testing approaches by Utting et al. categorizes the set of investigated approaches amongst others by its *Test Selection Criteria*, where various types of criteria are postulated by means of reflecting the tester's mindset for the subsequently generated test suite [8]. Apart from the *Requirement-Based Coverage Criteria*, which is an information-based selection approach, mostly algorithmic selection approaches are taken into account. This indicates that former approaches favored an algorithmic way of solving the problem, instead of aggregating various sets of information. In their conclusive statement, they clearly pointed out the research challenge of appropriate "domain-specific test selection criteria" [8], which we address with our presented work.

In contrast to the academic view on the topic, Hemmati et al. contributed an industrial case study on test case selection [9]. Within their research, the problem of test case explosion in combination with hardware-in-the-loop testing in a model-based testing context provoked a strong need for sufficient test case management. Based on their initial categorization into *Random- or semi-random selection*, *Coverage-Based selections*, and *Similarity-Based selections*, they chose to apply a genetic algorithm to minimize the similarity values of pairs of test paths. Regarding our omni modeling approach, the similarity minimization approach may be encoded by a quantification of uniqueness revealed by a test model analysis, subsequently constrained via the aspects concept. Moreover, this enables testers to use this methodology for the prioritization of test cases as well. Besides the approaches investigating on test case management activities in a model-based testing context, the majority of related research focuses a subset of test case management activities, categorized by Yoo et al. [10].

Further, we present a selection of related research, impacting our work on test model scoping. Utilizing and constraining the related requirements information further on mapped to concrete test cases, Harrold et al. presented a methodology for controlling the number of test cases [11]. Together with previous work on the requirements tracing across multiple domains of the development lifecycle [12], a flexible mechanism for appropriate test suite reduction/selection based on linked information is presented. The experimental results of Harrold et al. further reveal promising results by means of the final test suite size [11]. Based on a quite similar set of requirements information, Arafeen and Do demonstrate an approach for test case prioritization [13].

Apart from the specification based criteria for test case selection and prioritization, Herzig et al. presented a self-adaptive approach, evaluating the execution costs of a test as criteria to determine the final set of test cases [14]. In contrast to the previously mentioned research, this represents another use case for our synthetic aspects concept powered by analyses of artifacts impacting the development process.

Alternative approaches aiming at the prioritization of test cases identify other vectors of knowledge, such as results of risk assessments of related development artifacts. Risk-Based test case prioritization has been extensively investigated by Felderer and Schieferdecker [15], which represented a valuable information basis for our scoping mechanism, further generalizing their concepts, while taking into account Kasurinen et al.'s observations [16]. Beyond the risk oriented approaches, contributions aiming at fault detection capabilities (history- or search-based) show again show promising results [17] [18]. The methodology behind these kind of approaches may also be encoded by our processing chain, which takes a more generic and context-independent view on the topics challenges.

Summarizing our work on related research, we found no evidence for comparable studies performing test case management tasks based on information from multiple domains incorporated in a model-based development process.

VII. Conclusions and Future Work

Targeting the complexity challenges of current software under development and test, we have presented a valuable approach supporting test case management related tasks, such as test case selection, prioritization, and reduction. Addressing the identified set of problems, the proposed solution, on the one hand, aims for improving the knowledge basis, while simplifying the methodology on top. Therefore, the complete palette of domain-specific development artifacts is taken into account. Tying together these model-based information sinks, the integration model together with its aspects concept introduces a mechanism for testers to scope certain parts of the test model.

On the other hand, a fix to the process-level arrangement of the test case management tasks is addressed. Performing the selection, prioritization or reduction of test model artifacts, i.e. set of test cases on a higher level of abstraction, before the generation of concrete test cases, the exposure of the systems' complexity is delayed until the test cases are run. In most alternative approaches for test case management, the concrete set of test cases serves as a starting point for subsequent tasks representing an even harder challenge.

Apart from the conceptual and methodological view on the approach, we have presented insights to our prototypical implementation within the Architecture and Analysis Framework (A3F), further automating the formerly error-prone and complex steps of test case management. Our investigations on multiple heterogeneous application scenarios demonstrated beneficial effects, furthermore justifying the modeling and maintenance overhead introduced by the new model artifact.

Beside the positive impact on test case management related tasks of the software testing lifecycle, especially the integrated model basis may serve multiple purposes in early testing activities, which was already outlined by Pröll et al. [6]. The tasks integrated into this vision of a consistently model-based software testing life cycle represent the major topics of our future work on model-based testing.

Furthermore, an extensive evaluation of the methodology and its prototypical implementation within the Architecture and Analysis Framework in an industrial context represent major tasks of our future research. Beside the functional benchmark of the concepts presented during this contribution, the feasibility of the approach and the maintainability of incorporated model artifacts, i.e. the integrated model basis, are of special interest.

Acknowledgment

The research in this paper was funded by the German Federal Ministry for Economic Affairs and Energy under the Central Innovation Program for SMEs (ZIM), grant number 16KN044120.

References

[1] R. V. Binder, B. Legeard, and A. Kramer, "Model-based testing: where does it stand?" *Communications of the ACM*, vol. 58, no. 2, pp. 52–56, 2015.

[2] A. Pretschner and J. Philipps, "10 methodological issues in model-based testing." *Model-based testing of reactive systems*, pp. 11–18, 2005.

[3] E. Engström, P. Runeson, and M. Skoglund, "A systematic review on regression test selection techniques," *Information and Software Technology*, vol. 52, no. 1, pp. 14–30, 2010.

[4] E. N. Narciso, M. E. Delamaro, and F. D. L. D. S. Nunes, "Test case selection: A systematic literature review," *International Journal of Software Engineering and Knowledge Engineering*, vol. 24, no. 04, pp. 653–676, 2014.

[5] A. Rumpold, R. Pröll, and B. Bauer, "A Domain-aware Framework for Integrated Model-based System Analysis and Design," in *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,*, INSTICC. SciTePress, 2017, pp. 157–168.

[6] R. Pröll and B. Bauer, "Toward a Consistent and Strictly Model-Based Interpretation of the ISO/IEC/IEEE 29119 for Early Testing Activities," in *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development - Volume 1: AMARETTO*, INSTICC. SciTePress, 2018, pp. 699–706.

[7] Saad, Christian and Bauer, Bernhard, "Data-Flow Based Model Analysis and Its Applications," in *Proceedings of the 16th International Conference on Model-Driven Engineering Languages and Systems*. Berlin, Heidelberg: Springer, 2013, pp. 707–723.

[8] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Software Testing, Verification and Reliability*, vol. 22, no. 5, pp. 297–312, 2012.

[9] H. Hemmati, L. Briand, A. Arcuri, and S. Ali, "An Enhanced Test Case Selection Approach for Model-based Testing: An Industrial Case Study," in *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE '10. New York, NY, USA: ACM, 2010, pp. 267–276.

[10] S. Yoo and M. Harman, "Pareto Efficient Multi-objective Test Case Selection," in *Proceedings of the 2007 International Symposium on Software Testing and Analysis*, ser. ISSTA '07. New York, NY, USA: ACM, 2007, pp. 140–150.

[11] M. J. Harrold, R. Gupta, and M. L. Soffa, "A Methodology for Controlling the Size of a Test Suite," *ACM Trans. Softw. Eng. Methodol.*, vol. 2, no. 3, pp. 270–285, Jul. 1993.

[12] F. Abbors, D. Truscan, and J. Lilius, "Tracing requirements in a model-based testing approach," in *Advances in System Testing and Validation Lifecycle, 2009. VALID'09. First International Conference on*. IEEE, 2009, pp. 123–128.

[13] M. J. Arafeen and H. Do, "Test Case Prioritization Using Requirements-Based Clustering," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, March 2013, pp. 312–321.

[14] K. Herzig, M. Greiler, J. Czerwonka, and B. Murphy, "The Art of Testing Less Without Sacrificing Quality," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ser. ICSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 483–493.

[15] M. Felderer and I. Schieferdecker, "A taxonomy of risk-based testing," *International Journal on Software Tools for Technology Transfer*, vol. 16, no. 5, pp. 559–568, Oct 2014.

[16] J. Kasurinen, O. Taipale, and K. Smolander, "Test Case Selection and Prioritization: Risk-based or Design-based?" in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '10. New York, NY, USA: ACM, 2010, pp. 10:1–10:10.

[17] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: a family of empirical studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182, Feb 2002.

[18] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: an empirical study," in *Software Maintenance, 1999. (ICSM '99) Proceedings. IEEE International Conference on*, 1999, pp. 179–188.