Applying Integrated Domain-Specific Modeling for Multi-Concerns Development of Complex Systems

Reinhard Pröll (ORCID 0000-0002-3979-5483), Adrian Rumpold* (ORCID 0000-0002-8248-7363), and Bernhard Bauer (ORCID 0000-0002-7931-1105)
{reinhard.proell, adrian.rumpold, bauer}@informatik.uni-augsburg.de

Institute for Software & Systems Engineering
University of Augsburg, Germany

Abstract. Current systems engineering efforts are increasingly driven by trade-offs and limitations imposed by multiple factors: Growing product complexity as well as stricter regulatory requirements in domains such as automotive or aviation necessitate advanced design and development methods. At the core of these influencing factors lies a consideration of competing non-functional concerns, such as safety and reliability, performance, and the fulfillment of quality requirements. In an attempt to cope with these aspects, incremental evolution of model-based engineering practice has produced heterogeneous tool environments without proper integration and exchange of design artifacts. In order to overcome these shortcomings of current engineering practice, we propose a holistic, model-based architecture and analysis framework for seamless design, analysis, and evolution of integrated system models. We describe how heterogeneous domain-specific modeling languages can be embedded into a common general-purpose model in order to facilitate the integration between previously disjoint design artifacts. A case study demonstrates the suitability of this methodology for the design of a safety-critical embedded system, a hypothetical gas heating, with respect to reliability engineering and further quality assurance activities.

Keywords: Domain-specific Modeling, Model Transformation, Model-based Analysis, Model-based Testing.

1 Introduction

A clear trend towards increasing complexity is visible in modern embedded systems, both with respect to hardware and software. This development is fueled by a variety of factors, with one major driver being the advent of stricter regulatory guidelines in diverse domains such as automotive (with the ISO 26262 standard), aviation (ARP4754A and DO-178C/DO-254), and industrial automation (IEC 61508, IEC 61511, among others).

* Corresponding author

Two major strategies have emerged that attempt to let system designers cope with this rise in product complexity:

More stringent engineering methodologies, notably model-based techniques, are becoming essential for the design of complex systems. However, most existing model-based methods place disproportionate focus on functional requirements, mostly disregarding non-functional and quality aspects, such as reliability, safety, and security.

At the same time, tooling vendors have provided a sizable amount of products for the analysis and management of non-functional engineering concerns (compare chapter 4.1.2 of [21]). No clear strategy exists as to how these heterogeneous tools can be integrated in a seamless workflow, in order to make their information base available throughout the entire product life cycle.

Due to these shortcomings caused by partial adoption of model-based techniques and inconsistent tooling environments, establishing traceability and consequent change management have emerged as two main challenges in systems engineering. The importance of these concerns can be seen clearly in the context of safety-critical systems: Here, regulatory standards and norms necessitate careful management of development processes and artifacts with respect to consistent traceability throughout the product life cycle. Non-compliance with these requirements may pose a significant financial risk (in the form of late changes required to attain safety certification) as well as a liability hazard for the manufacturer.

A similar argument holds for quality assurance activities during the development of such complex systems. Here, focus lies on a high degree of test coverage – some safety standards even mandate specific coverage requirements (e.g. the aviation norm DO-178C). The resulting need for careful manual review and management of traceability and consistency leads to sub-optimal process efficiency and ultimately a potential negative impact on product quality.

Problem Statement

Despite the advantages that stem from the use of state-of-the-art model-based engineering practice, a tighter integration between techniques and tools for functional and quality aspects is needed in order to conquer the difficulties of ever-increasing product complexity.

Some effort has been made towards artifact exchange between model-based engineering tools, e.g. through standardized interchange formats like XMI. However, the vision of truly seamless tool integration remains a fundamental challenge. The resulting need for manual process steps can delay quality-related design activities and consequently reduce overall product quality. As a result, quality defects discovered late in the development process drive costs and pose a hazard to timely product release (see [1]).

A consistent seamless design methodology is crucial when considering process artifacts such as documentation required for certification of safety-critical systems. It is immediately evident that consistency between the actual product

and its supporting artifacts is of crucial importance. However, although common modeling tools allow for generation of technical documentation from system models, the generation of more complex textual artifacts exceeds their limited capabilities.

In order to overcome the identified weaknesses we propose an approach which aims for a tight integration of all system modeling artifacts and a shift towards (semi-)automated integrated architecture analyses.

Based on an extensible set of domain-specific modeling languages, which make up a solid foundation for a more suitable description of quality aspects, we aim for a co-evolution of functional and quality architectures of the system under development. These modeling languages cover the domains of common non-functional requirements for embedded systems, for instance safety, reliability, and system integration. Further, we describe a reliable mechanism for quality assurance of systems developed using such heterogeneous modeling languages. Our approach aims to reuse existing design methodologies, as long as they generate artifacts that adhere to a formalized metamodel.

The model-level integration of multiple domain-specific aspects additionally enables developers to generate purpose-specific data from the system model, which offers the necessary flexibility for the development of complex embedded systems.

We foresee that this integrated modeling approach will lead to increased product quality and can thus support the development of safety-critical and similarly regulated systems.

Outline

This article is an extended and revised version of our earlier conference paper [17].

As a starting point, section 2 introduces the modeling concepts underlying our approach and describes their application in analysis scenarios within our proposed framework. Starting with general-purpose modeling languages, which are actively maintained by the system engineer, we describe a set of essential domain-specific views on the system and their embedding into the general-purpose language. Based on this definition of embedded domain-specific languages, we propose a model-based analysis framework in section 3, providing some insight into its technical background and implementation. In section 4, we demonstrate the feasibility of our approach using a realistic use case. There, we perform some exemplary design and analysis steps utilizing the previously introduced framework. Section 5 discusses related work regarding the integration of heterogeneous modeling tools, domain-specific modeling, and model-based analysis. Section 6 summarizes the key results of this paper and briefly outlines future applications extending our research.

2 A Domain-Aware Modeling Approach for Embedded System Engineering

To overcome the challenges identified in the introduction, we have developed a concept designed to integrate legacy development and modeling techniques with a new kind of domain-aware modeling approach and analysis framework. Based on the information embedded in an integrated system model, purpose-specific artifacts (e.g. certification- or test-related documentation), which had to be maintained manually before, can now be generated automatically. In order to switch between these representations and generate documents, we make use of model-to-model (M2M) and model-to-text (M2T), As a special case, we consider *x-to-code* (X2C) transformations, where X may stand for *text* (T2C) or *model* (M2C).

The high-level concepts and their relationships are illustrated in fig. 1 and will be elaborated in the following sections.

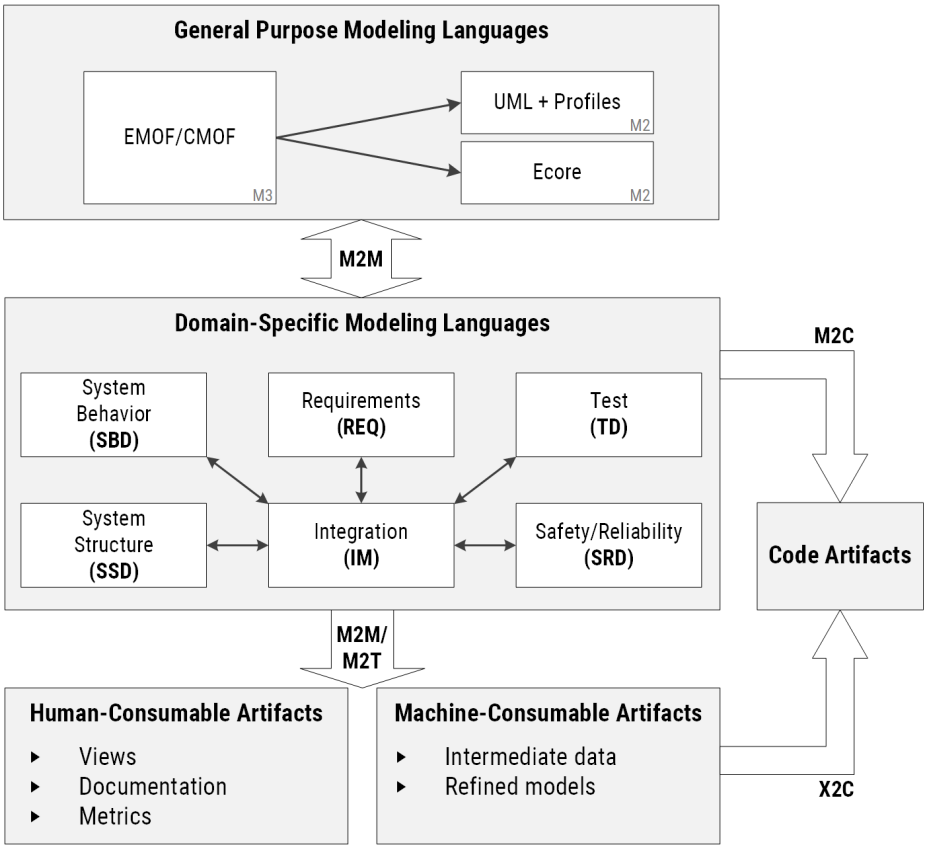

Fig. 1. Conceptual overview of the domain-aware integrated system modeling approach

2.1 General Purpose Modeling Languages

Following our goal of easy application and seamless integration into state-of-the-art development processes, we have decided to embed all relevant data for the development process within a *General Purpose Modeling Language* (GPML), such as UML or Ecore.

Using such modeling languages improves the applicability of our presented approach: GPMLs are widely accepted as state of the art, with many practitioners being familiar with their proper use. This familiarity allows for easier and faster adoption of new approaches based on general-purpose modeling languages. On the other hand, the general applicability of GPMLs has created a huge variety of available CASE tools for creating and viewing models. This rich tool environment can be reused within our newly propsed methodology, rather than developing yet another immature modeling tool.

In our scenario, these general-purpose languages serve a two-fold purpose: First, they provide a common modeling basis for all domain-specific models, as described in the following section. Second, the GPML can itself be used to cover certain subsets of the domain-specific modeling disciplines, if their expressive power is sufficient for a specific use case. We will see an example for this simplified domain modeling in the case study in section 4, where UML component diagrams and state machines are used to describe parts of the system architecture. Similarly, an extended version of the native UML activity chart is used for modeling of functional test models.

Our approach does not prescribe a certain GPML to be used for modeling the integrated system model. The only necessary requirement is the possibility to enhance the general-purpose language with metamodel extensions. In the case of UML this is achieved by defining profiles that leverage the stereotype mechanism. Similarly, we can extend the expressive capabilities of modeling languages which are themselves specified as UML profiles, for example SysML. Within the widely popular Eclipse Modeling Framework (EMF), metamodel extensions can be easily defined due to the reflexivity of the Ecore modeling language, which itself is its own meta-model.

2.2 Domain-Specific Modeling Languages

In order to accurately describe domain-specific aspects of the system under development, we embed them into the GPML mentioned above as *Domain-Specific Modeling Languages* (DSML). Our approach allows for any number of DSMLs to be used in conjunction with a general-purpose modeling tool to obtain an *integrated system model* (ISM) or *Omni model*.

These DSMLs preserve the separation of concerns, but enable developers to link information across domains in order to build up a holistic view of the system under development (SUD) and facilitate analyses based on domain-specific information. By using M2M transformations between the GPML and DSML representations, the distinct components of the ISM are in sync throughout the development process, yielding the best of both worlds.

In this section we briefly introduce some common domains pertinent to development of embedded systems and their focus. This lays the conceptual foundation for the following sections that will provide additional detail and demonstrate the application of these concepts.

The Requirements Domain (RQD) is related to the foremost engineering tasks of every modern software development process. As a result of these tasks, a set of requirements is extracted, which describes the desired system from a functional as well as a quality perspective.

Depending on the role of requirement specifications in the development process, an appropriate way of serialization must be chosen. In the early days of requirements engineering, Roman [16] pointed out its importance and already identified the need for knowledge integration.

In order to further make use of the generated set of requirements, a certain DSML needs to be specified. Natural language requirements with additional structuring capabilities as well as fully machine-processable requirement models are thinkable. For example, a ReqIF-based DSML (see [12]) can be used with most common CASE tools.

More advanced efforts propose ontologies as a suitable way of specifying requirements (e. g. [20]), which are also compatible with the basic concepts underlying our framework.

Being able to reference specific requirements in a model or parts of them, enables developers to use this semi-formal specification of the system for cross-domain traceability, thus extending the information base. For example, the availability of requirements information for a test engineer may result in a more transparent and effective test of the system under development. We give an example for this beneficial interaction our case study in section 4.

The System Structure Domain (SSD) contains the structural model of the system under development and reflects the architectural decomposition of the solution.

Our approach allows for a high degree of freedom regarding the actual implementation of the SSD model. For simple projects, the underlying GPML (see section 2.1) itself may be sufficiently expressive to model the system structure without any domain-specific additions. For more complex systems, a modeling language with more powerful abstractions, such as SysML, can be integrated to describe the structural domain more adequately.

It should be noted that the SSD model may also be derived from a prior system description in case of a brown-field project. Here, it is feasible to use either existing architecture models as a basis for the newly defined integrated system model, or to reverse engineer a system description from its code artifacts.

The System Behavior Domain (SBD) contains models that describe the functional behavior of the system under development. As described above for the SSD, a range of modeling languages can be used to implement the behavioral

domain within our approach. Natural choices are the behavior diagrams found in the Unified Modeling Language or its SysML extension.

If further usage includes the simulation or any kind of abstract interpretation of the behavioral parts of the system, an executable variant of UML, like *Foundational UML* (fUML, [10]), may provide suitable types of model artifacts.

However, different domain-specific modeling languages might be more familiar to designers of certain embedded systems; one example is the Function Block Diagram (FBD) notation for programmable logic controllers defined in the IEC 61131-3 and IEC 61499 international standards.

Given a suitable technology integration bridge (e.g. OSLC or ModelBus), it is conceivable to integrate behavioral models from widely used simulation tools like Simulink or Stateflow, as an intermediate step during re-engineering of legacy systems.

The Test Domain (TD) reflects the information specific to a tester's viewpoint on the system under development.

On the one hand it is used to simply formalize artifacts related to traditional quality assurance activities, such as test plans, test cases, and test execution reports. Depending on the expressiveness of the modeling languages used for the system description in the SSD and SBD, the test domain can sometimes be seen as an extension of these domains.

On the other hand, the benefit of a separate test domain can only fully be appreciated within a strict Model-Based Testing (MBT) approach. The main difference in this scenario is the purpose of the model artifacts: The artifacts related to a traditional testing process often represent the intermediate results of mostly manual, atomic steps. MBT in contrast, attempts to reduce the number of manual steps and the amount implicit knowledge in testing, thereby raising the efficiency, correctness, and reusability of artifacts. In this case it is conceivable to embed model languages specific to the testing domain, such as the OMG-maintained UML Testing Profile (UTP), which provides modeling facilities for test behavior description as well as quality assurance management activities. The UTP-affiliated Testing and Test Control Language (TTCN-3) may complete the palette of DSMLs of this domain in order to improve testing.

These considerations demonstrate only one possible solution and choice of DSMLs for the test domain – many others are conceivable, depending on the concrete use case. A possibly more intuitive solution is given by GPML-based test behavior specification via UML activity charts representing a set of test cases, i.e. a test suite. Once again, hybrids of the solutions mentioned above pose viable solutions for this domain and again encourage the use of our overall integrative approach.

Depending on whether test cases are generated or implemented manually, data specified or generated by other domain specific models, e.g. safety considerations, may guide this process. Further information on this topic can be found in section 2.2.

The Safety and Reliability Domain (SRD) covers the modeling and analysis of system reliability. Such analyses are invaluable and often mandated by regulations to demonstrate the system's expected failure behavior and obtain measures of reliability and availability, for example for safety-critical systems.

In order to quantify the reliability of a system, a thorough analysis of potential hazards and their associated risks is required. These hazard analyses require profound domain knowledge and experience and are therefore frequently performed as team efforts. Despite the interactive nature of these activities, their results can be formalized as a hazard model that describes identified hazards and the risks as well as possible faults and failures that can cause these hazardous events.

A major task in the design of safety-critical systems is the classification of hazards based on their associated risk. Risks that are deemed intolerable, either by societal or regulatory standards, have to be mitigated by deliberate risk reduction measures. Based on the necessary level of risk reduction, levels of safety integrity and associated safety requirements can be allocated to protective system components (*safety functions* in the terminology of the functional safety norm IEC 61508). This SIL allocation process requires the quantitative analysis of failure occurrence likelihoods.

Traditionally, quantitative reliability models are maintained in separate tool environments, decoupled from the actual system model. This disjoint setup can lead to inconsistencies in reliability models and decisions made based on them, unless proper care is taken during ongoing development of the system. However, many traditional reliability approaches can easily be adapted for use in model-based environments. For example, the widely used Fault Tree Analysis (FTA) technique defines a set of graphical elements to analyze failure causes in a system [22], and proves a suitable candidate for a domain-specific modeling language with a familiar graphical representation.

By embedding the reliability and hazard analysis models into the integrated system model, our approach allows to easily maintain full traceability between these models and their associated system model counterparts in the SSD and SBD. Moreover, change impact analyses can be easily performed based on this traceability information, whenever a modification to any part of the system model is made.

In the context of model-based systems engineering, it makes sense to move beyond the traditional FTA technique and incorporate a component-based extension, such as the Component Fault Trees as proposed in [7]. This hierarchical structuring of reliability information creates synergies with the end-to-end traceability provided by our modeling approach.

The Integration Model Domain (IMD), as illustrated in fig. 1, embodies the central mechanism to establish domain-specific model linking, mapping of artifacts, and cross-domain data accessibility.

In order to achieve this ambitious goal, its high-level structure represents an abstract, hierarchical breakdown of the instantiated system in a component-like fashion. Based on this abstract structure, cross-domain linking, represented

as bidirectional connectors in the model, on the one hand enables developers to make use of a solid and consistent traceability mechanism applicable throughout all development phases. On the other hand, the IM provides additional information to e.g. improve test related activities, previously out of scope. This holds for various combinations of domain-specific model data. Note that the IMD does not duplicate any information that has already been modeled in one of its connected domains.

Beside the linking and description of model interfaces, the IM holds analysis results generated by any kind of analysis executed by our proposed framework (see section 3 for a description of the framework). These results may represent the basis for ongoing processing steps, e.g. the scoping of a certain test model part, based on a set of criteria to be met.

In addition to the functionality of the IM presented above, it also plays the role of a early phase design artifact, reflecting an abstract decomposition of the proposed system functionality. For this reason, the IM may undergo constant change until it is connected to a concrete instantiation of SSD and SBD models and subsequently linked with other participating domain models.

Other Domains. The above modeling domains cover a wide range of engineering artifacts relevant during the design and construction of embedded and/or safety-critical systems. However, our modeling approach does not prescribe a fixed set of domain-specific modeling languages or domains and can easily be extended and tailored to a particular specific modeling use case.

The set of modeling domains presented above are especially relevant to the design of embedded systems. However, our framework may also take into account aspects of business and other applications. To this end, we envision domains addressing security and privacy considerations (e.g. to model information flows), timing models, description of data persistence, as well as usability models.

The next logical step based on such integrated domains is a holistic multi-concern consideration and a tightly coupled derivation of architecture optimization guidelines. These are beyond the scope of our current work and thus remain open as future topics.

2.3 Purpose-Specific Data

While the domain-specific models described above are derived from the GPML data through model-to-model transformations, our approach also covers the generation of purpose-specific data artifacts through model-to-text and model-to-model transformations. In contrast to the bidirectional transformation between GPML and DSML artifacts, the transformation into purpose-specific data (PSD) is unidirectional. This limitation is by design, since the GPML/DSML model should be regarded as the true information source, from which derived artifacts can be regenerated automatically.

Previous considerations only took into account the varying focus of domain-specific (human) developers. In order to address the increasing amount of generative (i.e. machine-based processing) steps, another distinction is chosen for

this information base: PSD artifacts fall into either two categories; *Human-Consumable Artifacts* and *Machine-Consumable Artifacts*, as shown at the bottom of fig. 1.

Human-Consumable Artifacts

As the name suggests, this kind of data centers on processing of information by humans. One can imagine a variety of scenarios where a tailored subset of the modeled information is desirable:

Views , a concept from software architecture, can be found again in our methodology. Since a system specified across several domain-specific models is not easily understood by non-technical stakeholders, a processed and condensed excerpt of the integrated system model is preferable. These views focus on specific aspects of the entire system and facilitate a better understanding and clearer communication.

Besides this more dynamic use case, which requires tool support, we also propose another variant of human-consumable artifacts:

Documentation, and in particular its automated generation, is an important factor in tightly integrated system engineering processes and plays a crucial role in quality-driven architecture.

The holistic nature of our proposed integrated system modeling approach facilitates document generation on a high abstraction level. For example, a common documentation requirement in safety-critical systems calls for seamless forward and backward traceability from system requirements down to the implementation level, and its proper documentation. Since the Omni model contains all necessary architectural elements and their relationships, generating such documentation consistently and in an easily navigable format (for example as hyperlinked HTML documents) is an effortless automated task. Using hypertext formats elegantly solves the traditional problem of limited traceability of these documents and makes them easily navigable.

The availability of usable, consistent, and up-to-date textual artifacts can help to reduce cost of safety certification by supporting high quality and early review of certification-related documents. Additionally, the same model-based document generation approach can be used to capture the results domain-specific analyses of the system that cover individual stakeholders' interests, leading to the next category of human-consumable artifacts:

Metrics are the concept of choice during analysis of certain Key Performance Indicators (KPI) of the system. From a project management perspective, we envision this approach to be useful for specification of (among others) test and requirement coverage metrics as an indicator of overall project progress.

A wholly different application scenario for metrics in the context of human-consumable artifacts is their use as a decision guidance in development processes: For example, a safety engineer may propose a change to the system model based

on the evaluation of a certain set of metrics. A similar use case is the improvement of test-related model artifacts based on a metric, reflecting the insights of a multi-concern consideration of the current application.

Machine-Consumable Artifacts

Our framework may also be used to export highly-specific data for further computation by external tools from the integrated system model. In contrast to the previous use case of human-consumable artifacts, this data is stored in a format optimized for machine processing. However, it also is subject to the limitation of unidirectionality, meaning that machine-consumable artifacts may not be part of a round-trip engineering approach. Such functionality is provided by the model-based analysis framework described in the following section.

We can distinguish different kinds of export formats, according to their intended application.

Intermediate Data, whose main purpose is to easily adapt to other tooling or the integration of libraries used for dedicated problem solving. For example, a processing step might provide input to an external optimization engine for the comparison of different architecture alternatives in the form of such intermediate data.

The term *intermediate* emphasizes the ephemeral nature of this kind of artifacts, representing a temporary result of a deterministic computation step based on the permanent information in the Omni model.

Refined Models, represent a special application of the previously mentioned *Intermediate Data*, where a domain-specific model is taken as a computational basis.

This model is either reduced to a limited scope, or enriched with supplementary information from another domain. For example, early applications of model-based testing did not separate the application model and the test model and instead only used the application model to generate test cases. Domain-aware approaches on the other hand favor the use of additional data in order to interface with existing tools to harness beneficial synergies.

While out of the immediate scope of our research, it should be noted that the final integrated system model is a suitable basis for generation of *source code*, as indicated by transformation steps on the right side of fig. 1. The integrated nature of the Omni model as well as purpose-specific data allows the code generation engine to make more educated decisions about the context of the source code to be synthesized. A possible scenario could be the automated application of defensive programming techniques in generated code, e.g. assertion of pre- and post-conditions or calculation of checksums, based on component contracts or safety requirements from the integrated system model.

3 A Model-Based Architecture and Analysis Framework

Based on the modeling approaches introduced in the previous section, we have developed a reference technology platform geared towards the domain-aware modeling of safety-critical systems and their quality attributes. In addition to the domain-specific metamodels, the prototype includes a framework for definition of model-based architecture analyses, introduced below in section 3.3.

3.1 Technical Foundations and Tooling

As shown in fig. 2, the analysis framework consists of three major components:

Enterprise Architect The commercially available Enterprise Architect (EA) is as a general-purpose modeling tool, providing the full range of UML modeling capabilities to the system designer. Domain-specific metamodels are integrated via EA's *Model-Driven Technologies* (MDG) feature

Model Repository A relational database system is used for persistent storage of the model repository and allows for external access to the system model without the need for tight coupling with EA.

Architecture and Analysis Framework The actual architecture and analysis framework, which offers model analysis services via a web service interface. Section 3.2 below describes the concrete execution model within the analysis framework.

For details on the various technologies involved and their interactions, see section 3 of [17].

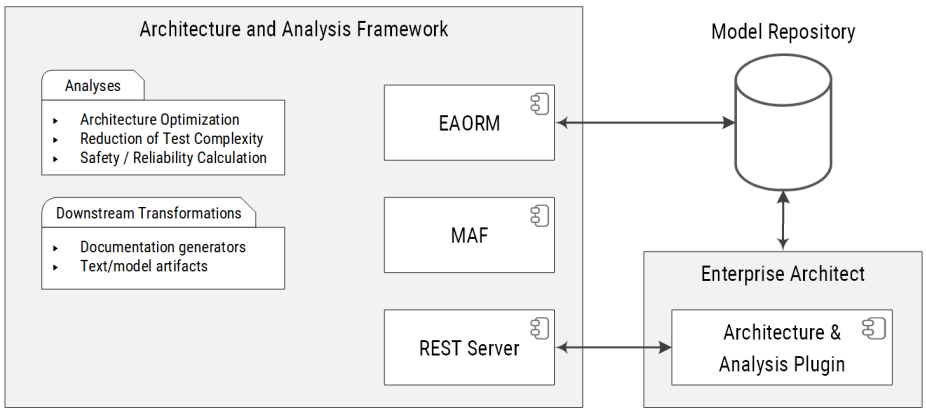

Fig. 2. Technical overview of the reference technology platform architecture

3.2 Analysis execution workflow

In order to execute an analysis request by the user, the Architecture and Analysis Framework passes through multiple execution phases. This section will give an overview of the necessary processing steps, as summarized in the activity diagram in fig. 3.

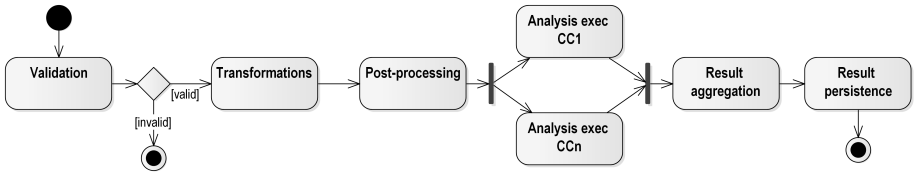

Fig. 3. Execution workflow within the Architecture and Analysis Framework

Validation Before any further processing takes place, the framework validates the analysis configuration supplied through the web service interface. This configuration adheres to a custom textual DSL and determines the types of analyses to be executed, their input model elements, as well as any additional parameters required to run the analysis.

The validation is carried out on the dependency graph between the analyses requested as part of the configuration. In order to qualify as a valid configuration, this graph must be acyclic and be closed under the transitive dependency relation. The second criterion assures that for each analysis, all its (transitive) prerequisites are also part of the configuration. If the configuration is found to violate these soundness assumptions, the execution engine aborts processing of the request and reports an error.

The actual execution order for all configured analysis is then calculated as a topological sorting of the dependency graph. As a subsequent optimization, the execution graph can be decomposed into its connected components to allow for parallel execution of multiple analyses: By definition, no dependencies exist between two analyses in different connected components, hence they are eligible to be processed by the framework simultaneously.

Figure 4 shows an example for such an execution graph from the case study described in detail in section 4. Transformations are shown as parallelograms, rectangles represent the requested analyses together with their position in the calculated execution ordering. Dashed arrows indicate a dependency on transformation outputs, while solid arrows denote the prerequisite relation between analyses. For readability reasons, transitive prerequisite arrows are omitted from the graph.

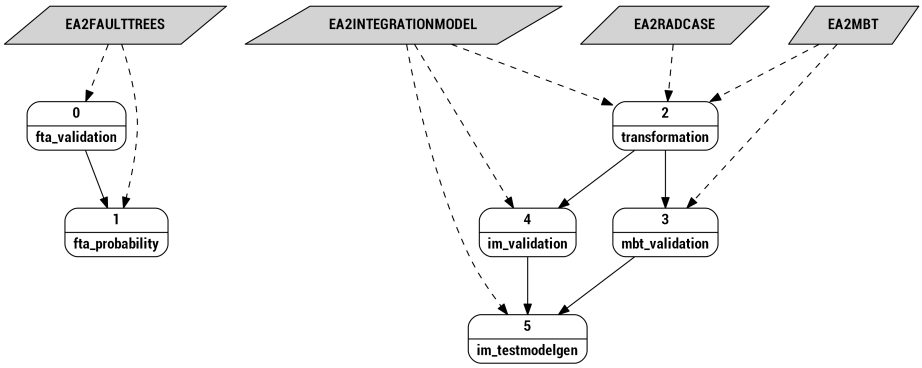

Fig. 4. Execution graph for the case study example (see section 4)

Transformation execution The Enterprise Architect input UML model is transformed into the corresponding domain-specific representations using a set of model-to-model transformations for each modeling domain. We have chosen the QVT Operational language (QVTo, see [8,11] for details) as the M2M transformation language for our prototype. Each domain-specific model can be obtained from the integrated system model by applying its associated transformation, mapping the extended general purpose modeling language (see section 2.1) onto the domain-specific modeling languages (section 2.2).

In order to simplify integration with the Eclipse EMF-based QVTo engine used for implementation of the transformation phase and model-based analyses, all domain-specific metamodels have been described using the Ecore metamodeling facilities.

Post-processing Optionally, an analysis may define arbitrary post-processing steps to be executed after the M2M transformation phase. Since the post-processing is implemented as regular application code, it can be used for additional calculations beyond the expressive capabilities of the transformation language. Examples include the handling of Java enumeration types and transformation of non-primitive value types into proper objects.

Analysis execution After inputs in the form of M2M transformation outputs are available, the execution engine iterates the pre-calculated execution order and runs all analyses specified by the configuration. Additional input parameters from the analysis configuration will be forwarded to the respective analysis.

For analyses that require functionality for data-flow based processing of a model, the execution phase can also delegate to the Model Analysis Framework (MAF in fig. 2). This framework allows the use of data-flow techniques originally researched in compiler construction for iterative, model-based analysis of design artifacts (see [18] for details).

Analyses report their execution status back to the framework, and may create or modify arbitrary model elements to represent the results of their calculation.

Result aggregation and persistence The execution engine tracks all modifications (object creation/deletion, attribute modification) to model elements performed in the analysis execution phase, as mentioned above. We have specified a compact domain-specific language for the description of reverse transformations of domain-specific models, while maintaining the integrity constraints of the original EA general-purpose model.

Note that only the general-purpose model is persisted in the model repository to remedy the problem of consistency across the transformation steps.

3.3 Processing of Integrated Model Data

Based on the execution mechanism previously described, we have developed a range of model analyses that can be applied to the integrated system model and its embedded domain-specific models. Consequently, one specific processing step is possibly made up out of multiple analyses, chained together. Most of the use cases mentioned in section 2 use this mechanism as a technical basis for intermediate computations.

Conceptually, we have identified three major classes of model analyses that can be distinguished by their responsibilities as well as the type of input and output models:

Validation analyses consume one or multiple input domain-specific models, but do not generate any new model elements as their output. Rather, a validation analysis verifies the syntactic and semantic well-formedness of its input models. In case this validation fails, the analysis produces a report of the identified violations and returns it as a separate result to the client.

Therefore, the purpose of validation analyses is the assurance of model integrity and quality. They are feasible candidates for tighter integration with the modeling tools used, and can be executed continuously without user interaction to provide rapid feedback about the state and quality of the model.

Note that the existence of this class of analyses is a testament to the state of metamodel extensibility in current general-purpose modeling tools. This shortcoming has previously been identified as the primary driver for so-called *descriptive stereotypes* [19]. If GPML tools provided first-class support for *restrictive stereotypes* or even full *restrictive metamodel extensions* instead, the syntactic and semantic constraints for a DSML could be directly validated as part of the metamodel extension.

Calculation analyses consume one or more input domain-specific models and calculate additional attributes for existing model elements, but do not add new elements.

These analyses can be seen as the formalization of a function application to their input models. Examples for this class of analysis are numerous, e.g. the

automated update of probability information in reliability models, risk classification, or the analysis of timing bounds in behavioral models.

In our analysis framework, calculation analyses are an obvious application point for the Model Analysis Framework (see above), since its feature set is well suited to the iterative nature of function evaluation on complex models.

Generative analyses both consume and produce model elements in one or more domain-specific metamodels. As such, they are similar to model-to-model transformations. However, they serve a broader purpose, and hence should be considered separately.

Generative analyses offer a consistent interface for the programmatic modification of the integrated system model as part of the execution workflow described in section 3.2 above. As opposed to ephemeral M2M transformations, their results are stored persistently.

Possible uses of this class of analyses are very broad: One possible example is the support of the system designer through wizard-type functionality, for example to generate skeleton reliability models from an existing structural model of a system. A different application scenario is the automated creation of a test model and test cases from the abstract description of system structure and functionality in the integration model.

While some generative analyses produce results that are intended for use inside the modeling loop centered on the integrated system model, the results of other analyses targets consumption outside the context of the analysis framework. This class of analyses is referred to as *downstream transformations* in fig. 2 and corresponds to the concept of purpose-specific data introduced previously in section 2.3.

The generation of source code from the integrated system model is a prime example for a downstream transformation. While the resulting code artifacts can still be regarded as a form of model, they are not persisted within the model repository and their main purpose lies outside the analysis framework.

Another important member of this class are model-to-text transformations in the form of document generators. They can make the creation of textual artifacts transparent to the client and encapsulate the actual invocation of the underlying M2T transformation engine.

As stated before, the three types of analyses can also be combined in order to handle more complex tasks. The processing of test model artifacts with the goal of reducing the final level of test complexity, for example composes an analysis for cross-domain calculation with a second analysis for generation of more specific test model artifacts. Again, this represents a common case of data preprocessing for further external use, subsumed under the category of *downstream transformations*.

4 Case Study: Design and Evaluation of a Gas Heating System

In the following section we will demonstrate the use of our domain-specific modeling approach to the reliability analysis of a gas heating system. Further, we take a closer look on related testing activities, which in turn benefit from the integrated model basis.

Gas boilers are commonly found in residential buildings to provide central heating by combustion of natural gas in a burner. A common extension to such heating systems is a reservoir to buffer a suitable amount of hot water. In case of a malfunction of the system, personal injury might arise. Therefore, the design of such a system must encompass an evaluation of the safety risks and include appropriate protection systems to reduce potential risks to an acceptable level.

The model artifacts shown in this paper represent a simplified version of a standard heating system to limit complexity to a manageable level. However, they nicely illustrate the application of our integrated modeling approach, its suitability for the development of safety-critical systems, and the improved efficacy of related quality assurance mechanisms.

4.1 System Structure and Behavior

Since our point of view on the system architecture is on a very abstract level, a plain UML component diagram is sufficiently expressive to describe the system structural domain for this case study. Figure 5 shows the main components of our exemplary gas heating system. Such a coarse-grained model can be derived in early design stages, as soon as the operational context of the system has been determined.

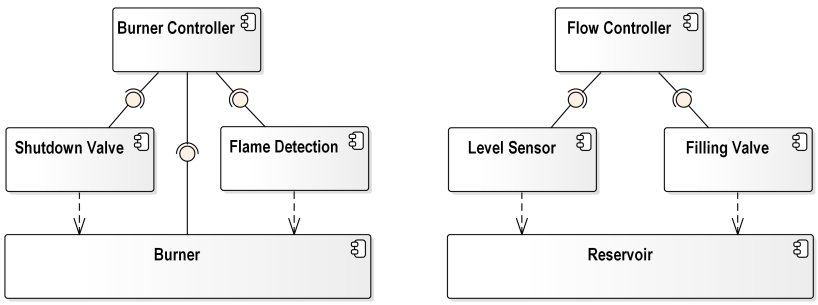

Fig. 5. Architecture of the gas heating system with integrated water heating circuit

We use UML state machines as well as other behavioral UML diagrams to model the internal functionalities of the elementary building blocks of the presented system. As an example, fig. 6 describes the main operating states of

the burner controller, which can be either operational or shut down in case a malfunction of the flame supervision mechanism has been detected.

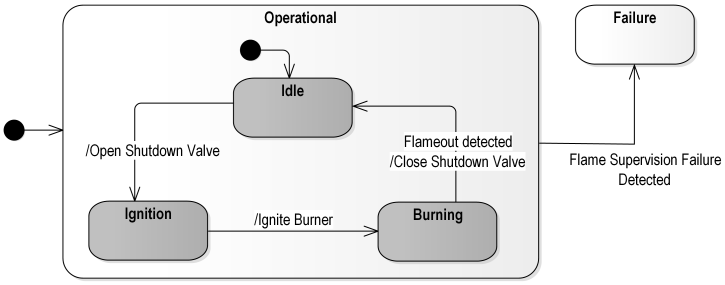

Fig. 6. Behavior model of the burner control logic

4.2 Reliability Model

An important early step during development of a safety-critical system is the assessment of potential hazards and risks associated with the system under development (see section 7.4 of [5] for details). This hazard and risk assessment, performed by a team of domain experts, can be documented inside the integrated system model.

As an illustrative example, we have chosen to analyze a potentially hazardous failure of the heating system, namely the presence of uncombusted gas in the burner chamber following a flameout. This situation can lead to rupture of the heating vessel due to over-pressurization as well as rapid deflagration or explosion of the uncombusted gas in the presence of an igniting spark. This hazard is assumed to occur with an intolerably high likelihood, which prompts the addition of a flame detection mechanism and an automatic safety shutdown valve as safety functions to the heating system.

The presence of a hot water reservoir in the heating system introduces an additional, unrelated hazard: If the filling valve malfunctions and becomes stuck in the open position, the reservoir might spill, posing the risk of severe scalding for anybody in its immediate vicinity.

A multitude of established engineering techniques exist for assessing the risk associated with a hazardous event and establishing the necessary risk reduction for an acceptable level of safety in the form of safety integrity levels. For this example, we have selected the risk graph technique described in appendix E of the IEC 61508-5 norm [6]. As shown in fig. 7, the results of a qualitative assessment of each hazard are embedded inside the hazard analysis model as *RiskGraphSpecification* instances.

The introduction of these new system components demands for another iteration of the hazard and risk assessment, to ensure an acceptable safety level.

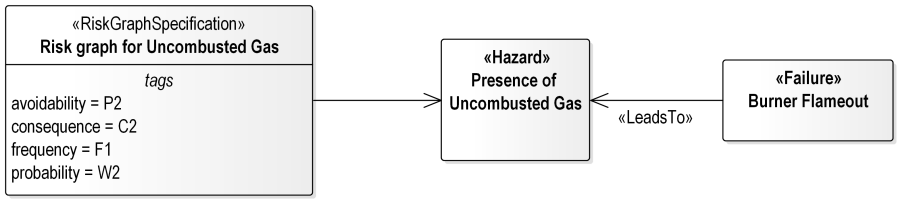

Fig. 7. Excerpt of hazard analysis and risk assessment model for the burner

4.3 Requirements Model

Given the initial qualitative hazard and risk assessment, the analysis framework is able to automatically identify the necessary risk reduction for each hazard and generate appropriate safety function and safety integrity requirements (see sections 7.5 and 7.6 of [5] for the regulatory background). The system engineer can subsequently allocate these requirements to appropriate safety functions.

Figure 8 shows a part of the safety requirements model for the previously discussed risk of uncombusted gas as well as the risk of a spillover of the respective reservoir. A safety function with a specified safety integrity level has been introduced to mitigate these risks, and is allocated to the related system components described earlier via the integration model. Both of the mentioned system components are implicitly tied together via the more abstract system component, the heating itself.

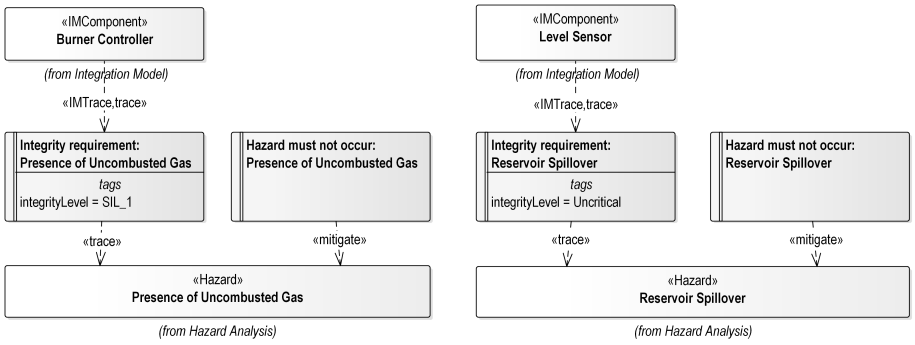

Fig. 8. Safety requirements model

Note that beside this domain-specific model of safety requirements, a complete integrated system model of the gas heating would also contain all functional requirements that govern the regular operation of the system.

4.4 Integration Model

The integration model for our use case ties together the system structure and behavioral domain with all additional domain-specific models like the reliability or related test models. Additionally this artifact forms a hierarchy of abstract components with the entire system under development at its root. Furthermore, the IM reflects the allocation of abstract functionality, e.g. the logic of the burner controller, to components and contains traceability information into the concrete behavioral model. In our example, the integration model associates the state machine for the burner control (see fig. 6) with the abstract control logic functionality.

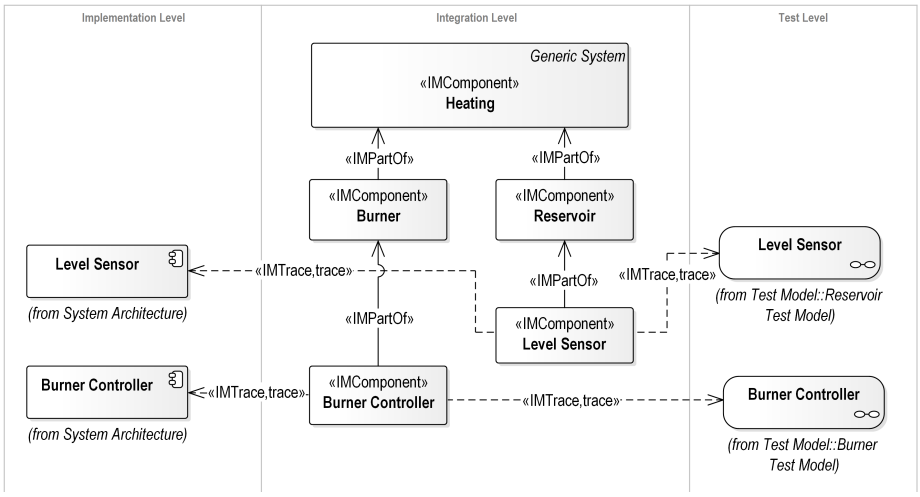

Fig. 9. Excerpt from Integration Model with links to SSD, SBD and TD

We can see this contribution of the integration model to seamless model traceability in fig. 10. The diagram emphasizes the trace relationship between the non-functional domains of the integrated system model for the heating system, in particular the requirements and hazard analysis domains. This traceability information is preserved by the model-based analysis framework and downstream transformations, especially during generation of source code. Therefore, based on this information, accompanying documentation can be generated that serves as evidence in safety certification of the burner control system.

4.5 Test Model

Beside the use case of generating comprehensive documentation for certification purposes, the results of such a safety and reliability analysis may also be utilized for test complexity reduction purposes. One might imagine a scenario,

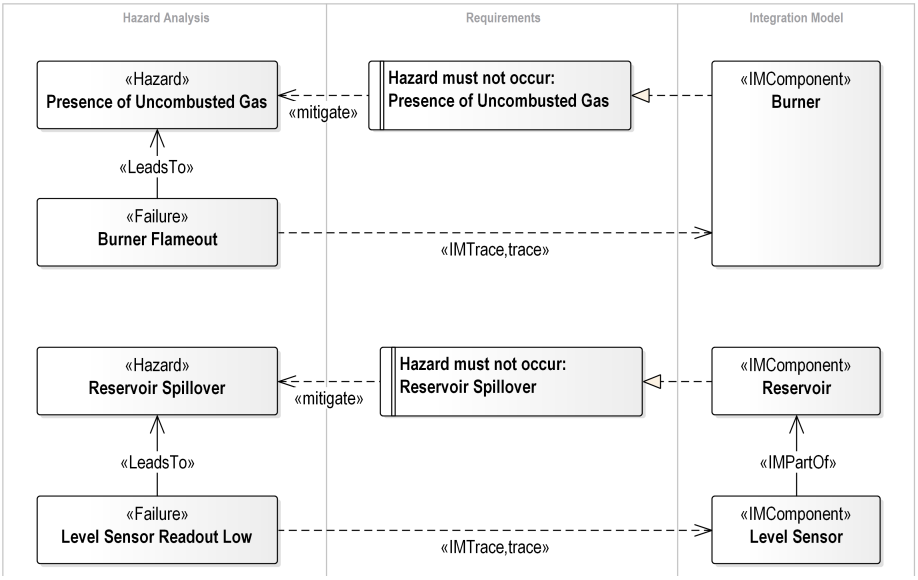

Fig. 10. Excerpt from Integration Model with trace links to reliability and requirements domains

where a management decision cuts down on the amount of time available for test-related activities. However, certification requires that safety-critical software components need to be tested extensively, achieving a certain degree of test coverage. It is evident that these two conflicting restrictions cannot be met by simultaneously, necessitating a trade-off. To overcome this problem, the tester makes use of the central model artifact, the integration model, specifically its aspect capabilities described earlier in this paper. This mechanism enables the tester to disregard certain parts of the holistic test model, thereby producing a reduced model for further test case generation. An integrated view on fig. 8 and fig. 9 illustrates the path of information flow across the domain-specific models. Applying the previously outlined scenario of focusing tests on safety-relevant system parts, for example might drop the test model for the *Level Sensor*, since its safety impact has been marked as *uncritical* (see fig. 8).

A combination of these flexible aspect configurations allows us to scope highly specific excerpts of the system model and thus focus on dedicated test cases with high impact on overall system quality.

5 Related Work

Our work relates to previous research in three related, but separate fields: Firstly, our approach provides a means of integrating various engineering disciplines into a coherent tool environment. Each of these disciplines brings with it its own set of domain-specific engineering artifacts and modeling languages. Finally, our

implementation of an architecture analysis framework based on an integrated system model relates to prior work in the field of model-based analysis.

The following sections give a short overview of the relevant literature in these three fields, as they relate to our current research.

5.1 Modeling Tool Integration

In his seminal work, Wassermann [23] describes an approach for integration of heterogeneous tools in a software engineering tool chain. He describes an integrated software engineering framework based on three cardinal dimensions of interoperability – presentation, data, and platform integration.

The EU-funded iFEST project (Industrial Framework for Embedded Systems Tools[1]) was aimed at developing an integrated framework for embedded systems, addressing both software and hardware concerns. The iFEST approach specifies a tool integration framework that leverages the OSLC specification to allow data exchange between heterogeneous modeling tools. Since it is focused exclusively on the aspect of tool integration, this approach does not address the field of model-based analyses of the integrated system model.

5.2 Domain-specific Modeling

Zschaler et al. [24] propose a generalization of DSLs to domain-specific modeling languages, in order to capture common concepts found in families of related DSLs and facilitate automation.

Similarly, de Lara et al. [9] describe an approach for domain-specific multi-level metamodeling languages, allowing for the definition of deep language hierarchies. Their approach contains a set of reusable metamodel transformations for management of multi-level metamodeling languages and describes approaches for code generation in such a setting.

The use of UML as a graphical visualization language for domain-specific modeling languages is proposed by Graaf and van Deursen [4]. Their work proposes model-to-model transformations as a means of deriving a visual representation from a domain-specific model. Conceptually, these transformations can be regarded as an embedding of the DSML into a generic-purpose modeling language, specifically into UML.

As stated by Dias et al. [2], this also holds for the testing domain. Their seminal survey showed that the majority of MBT approaches makes use of UML behavioral modeling capabilities, sometimes extended by certain domain-specific data. One of their conclusive remarks mentions the active use of UML-like languages due to the wide distribution of basic skills in this area.

5.3 Model-based Analysis

Papadopoulos and McDermid [13] introduce HiP-HOPS (Hierarchically Performed Hazard Origin and Propagation Studies), a methodology for model-based

[1] `http://www.artemis-ifest.eu/`

hierarchical reliability analysis of component-based systems. Based on the architecture of the system under analysis and certain failure annotations, HiP-HOPS allows for bottom-up generation of Fault Trees and so-called interface-focused FMEA results for a system. Under the HiP-HOPS methodology, components are enriched with additional model information about their failure behavior. Various classes of interface failures are defined that can be used to describe the black-box failure model of a system on a component level.

This approach has subsequently been extended to accommodate aspects of automatic architecture optimization. Papadopoulos et al. [14] further describe a conceptual approach for the automatic allocation of safety integrity levels to components of safety-critical systems. This work focuses on the automotive domain and uses the EAST-ADL2 modeling language for architecture description. Similarly, the authors propose a more generic architecture optimization technique based on the HiP-HOPS methodology and the use of genetic algorithms [15].

A complementary approach can be found in the EU-funded MBAT project (Combined Model-based Analysis and Testing of Embedded Systems[2]). This project aimed to provide a methodology and technology platform for specification of system analysis and V&V activities in the context of embedded system engineering. The central element of the proposed methodology is the so-called *A&T model* (short for [static] analysis and [model-based] testing), highlighting the focus of the approach to the quality-assurance domain.

An alternative path to overcome the constantly rising complexity of testing by utilizing information from other domains was proposed by Gebizli et al. [3]. The use of risk ratings of system components in combination with MBT in order to iteratively refine test models showed promising results. The resulting test suite boasts better fault detection capabilities in contrast to traditional MBT approaches, whereas the amount of time for testing was reduced.

6 Conclusions

We have proposed a valuable approach for integrated system modeling and model-based architecture analysis.

Our work introduces a solution to the challenge of integrating both system modeling and quality-related artifacts in the design and implementation of embedded systems. The resulting *integrated system model* or *Omni model* establishes explicit traceability between domain-specific modeling artifacts and enables consistent change management and change impact analyses.

This domain-centered view of systems engineering incorporates the fundamental challenge of multi-concerns design by unifying previously disjoint modeling domains.

Based on this holistic, model-based view on the system under development, complex model analyses can be performed to validate, process, or enhance the integrated system model. Analyses may also generate textual or model content for

[2] `http://www.mbat-artemis.eu/`

further processing outside our proposed methodology, for example as generated source code or management reports.

We have developed a reference technology platform that combines our proposed integrated system modeling approach with a model-based analysis framework. The suitability of this prototype is demonstrated through a case study, which illustrates the use of the framework to model the reliability and testing aspects of a residential gas heating burner, a simple safety-critical embedded system.

We envision a variety of possible application fields for our approach as a basis for future research work: The seamless availability of information across domain boundaries makes the integrated system model open for use in automated systems engineering processes. For example, suitable analyses could be developed to support the (semi-)automated optimization of certain architecture aspects under consideration of reliability and safety aspects. In order to better support these optimization heuristics, the system model can be enhanced with stricter, machine-comprehensible formalization, such as component safety contracts.

Overall, we predict that the consequent application of model-based design methodologies will help to cope with the current challenges of systems engineering and help to create safe and maintainable products.

Acknowledgements

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 numbers KF 2751303LT4 and 16KN044120.

References

1. Boehm, B.W.: Software Engineering. Tech. rep., TRW Systems and Energy Group (1976)
2. Dias Neto, A.C., Subramanyan, R., Vieira, M., Travassos, G.H.: A survey on model-based testing approaches: a systematic review. In: Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies. pp. 31–36. ACM (2007)
3. Gebizli, C.S., Metin, D., Sozer, H.: Combining model-based and risk-based testing for effective test case generation. In: Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on. pp. 1–4. IEEE (2015)
4. Graaf, B., van Deursen, A.: Visualisation of domain-specific modelling languages using UML. In: 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07). pp. 586–595. IEEE (2007)
5. Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 1: General requirements. Standard, International Electrotechnical Commission, Geneva, CH (2010)
6. Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 5: Examples of methods for the determination of safety integrity levels. Standard, International Electrotechnical Commission, Geneva, CH (2010)

7. Kaiser, B., Liggesmeyer, P., Mäckel, O.: A new component concept for fault trees. In: Proceedings of the 8th Australian workshop on Safety critical systems and software-Volume 33. pp. 37–46. Australian Computer Society, Inc. (2003)
8. Kurtev, I.: State of the art of QVT: A model transformation language standard. In: International Symposium on Applications of Graph Transformations with Industrial Relevance. pp. 377–393. Springer (2007)
9. de Lara, J., Guerra, E., Cuadrado, J.S.: Model-driven engineering with domain-specific meta-modelling languages. Software & Systems Modeling 14(1), 429–459 (2015), http://dx.doi.org/10.1007/s10270-013-0367-z
10. Semantics of a Foundational Subset for Executable UML Models, Version 1.2.1. Specification, Object Management Group (OMG), Needham, MA (2016)
11. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.3. Specification, Object Management Group (OMG), Needham, MA (2016)
12. Requirements Interchange Format (ReqIF), Version 1.2. Specification, Object Management Group (OMG), Needham, MA (2016)
13. Papadopoulos, Y., McDermid, J.A.: Hierarchically performed hazard origin and propagation studies. In: International Conference on Computer Safety, Reliability, and Security. pp. 139–152. Springer (1999)
14. Papadopoulos, Y., et al.: Automatic allocation of safety integrity levels. In: Proceedings of the 1st Workshop on Critical Automotive Applications: Robustness & Safety. pp. 7–10. ACM (2010)
15. Papadopoulos, Y., et al.: Engineering failure analysis and design optimisation with HiP-HOPS. Engineering Failure Analysis 18(2), 590–608 (2011)
16. Roman, G.C.: A taxonomy of current issues in requirements engineering. Computer 18(4), 14–23 (1985)
17. Rumpold, A., Pröll, R., Bauer, B.: 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 (MODELSWARD). pp. 157–168. SCITEPRESS (2017)
18. Saad, C., Bauer, B.: Data-Flow Based Model Analysis and Its Applications. In: Moreira, A and Schätz, B and Gray, J and Vallecillo, A and Clarke, P (ed.) Proceedings of the 16th International Conference on Model-Driven Engineering Languages and Systems. pp. 707–723. Springer, Berlin, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-41533-3_43
19. Schleicher, A., Westfechtel, B.: Beyond stereotyping: Metamodeling approaches for the UML. In: Proceedings of the 34th Annual Hawaii International Conference on System Sciences. p. 10 pp. IEEE (2001)
20. Siegemund, K., Thomas, E.J., Zhao, Y., Pan, J., Assmann, U.: Towards ontology-driven requirements engineering. In: Workshop Semantic Web Enabled Software Engineering at 10th International Semantic Web Conference (ISWC), Bonn (2011)
21. Sommerville, I.: Software Engineering. Pearson Education, 9th edn. (2011)
22. Vesely, W.E., Goldberg, F.F., Roberts, N.H., Haasl, D.F.: Fault tree handbook. Tech. rep., DTIC Document (1981)
23. Wasserman, A.I.: Tool integration in software engineering environments. In: Software Engineering Environments. pp. 137–149. Springer (1990)
24. Zschaler, S., Kolovos, D.S., Drivalos, N., Paige, R.F., Rashid, A.: Domain-specific metamodelling languages for software language engineering. In: International Conference on Software Language Engineering. pp. 334–353. Springer (2009)