Towards domain-specific extensibility of quality-aware software architecture meta-models

Sebastian Dieter Krach
krach@fzi.de
FZI Forschungszentrum Informatik, Karlsruhe

Abstract

The evermore extending presence of software systems in ubiquitous application domains requires current software architecture analyses to incorporate domain-specific concepts. Current model extension approaches are too general, do not provide sufficient support for multiple roles in the development process or impose high effort on the extension developer. In this paper, we present MDSD.tools Characteristics, a framework to enhance existing architecture description languages with easy-to-use quality modeling profiles. It facilitates capturing of domain expert knowledge concerning relevant attributes of architecture entities into reusable specifications. Furthermore, it comprises a notion of contextual information with a model-based specification for information propagation to simplify the integration with existing analyses.

1 Introduction

With increasing capabilities of ubiquitous technology, the size and inherent complexity of the required software systems grows. Already 2006, Broy estimated 50% to 70% of software and hardware development costs to be incurred by software [2]. Hence, also the relevance of a suitable up-front architecture design increases. Model-based software quality analyses, e. g. the Palladio approach [4], provide means to quantitatively assess architecture-level alternatives and thereby support cost-and-quality-efficient decisions. When designing domain-specific software, domain knowledge and domain-specific system properties become a significant part of the architecture.

Capturing domain-specific properties in architecture models requires to create meta-models for each domain and subsequently adapting existing analysis tooling to be reused. While this is acceptable for domain-specific analyses, it becomes a cumbersome task for reusing existing generic ones, e. g. performance simulation or reliability analysis. Alternatively, the software architect needs to know how to translate selected domain-specific properties into generic ones to continue using a generic architecture model. We envision taking a middle ground: use a generic structural architecture model and separately extend it with the properties of the domain. The approach is similar to a modularization of what Strittmatter et al. [6] refer to as Ω and Σ layer.

In this paper we present MDSD.tools Characteristics, an approach for domain-experts to extend meta-models with domain-specific properties while being able to transparently reuse existing analyses.

2 Running Example

Our running example focuses on using Palladio performance simulation for a simple perception system. It consists of two cameras which periodically take a picture and transmit it via messaging to a shared image processing component (see Figure 1).

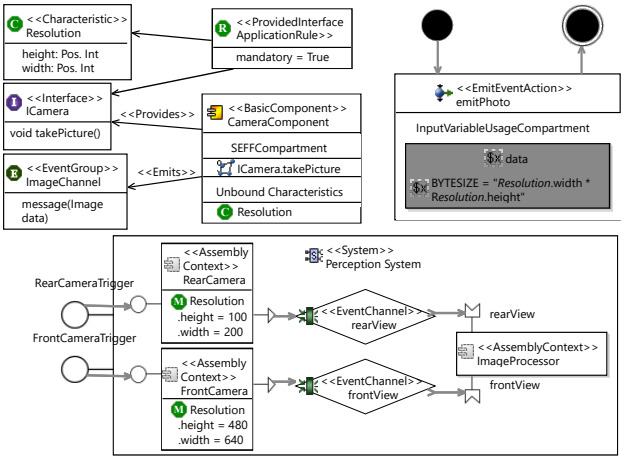

Figure 1: Excerpt of PCM Perception Example

We want to determine the network load and estimate the transmission latency for different *image resolution* settings. The image resolution determines the size of each image message and should be configurable for each camera on its own. Furthermore, the image resolution affects the probability of successful object recognition in subsequent stages.

This trivial example illustrates the use of domain-specific characteristics and does not strive to present a comprehensive analysis scenario.

3 Domain-specific characterizability

In the following, we discuss the requirements which need to be fulfilled by a meta-model extension mecha-

nism. We identified the requirements based on our experience with meta-modeling and model-based analyses of applications from different domains, in particular using the Palladio Approach [4]. Thereafter, we present the concepts of our framework to meet the identified requirements.

I. External extensibility The component developer of our *CameraComponent* should be able to specify the required additional properties for his component without the need to adapt the PCM meta model.

II. Multi-view refinement support The meta model extension framework should provide support for multi-view-based architecture models, particularly deferred initializations. In particular, property specification and initialization should be separate. In our example, the image resolution property needs to be specified for the component while concrete image resolution values would be specified during system assembly for each component instance. For the PCM, *Component Parameters* [4, p. 107] provide the required functionality. They do however lack support for generalization and automated validation.

III. Ahead of analysis-time validation support The system architect should be able to validate the completeness of extended models. For instance, assemblies of the camera component which do not specify an image resolution should be identified.

IV. Flexible constraint type system A domain expert who specified additional properties also wants to communicate the expected value range. For instance, the resolution property has two nested positive integer properties *width* and *height*. The specification is essential for editing and validation tool support.

V. Support for stochastic initializations Certain characteristics of a system entity cannot be expressed by a fixed quantity but appear to have a random component. Consequently, when initializing properties the modeller should be able to use stochastic means to express probabilistic distributions over potential values.

3.1 The Characteristics Approach

In order to achieve our desired extensibility, we distinguish three model-based view points: *Characteristics*, *Manifestations* and *Value Types* (see Figure 2). A *Characteristics* model captures the specification of properties and includes rules on where they can be applied on an existing meta-model. *Manifestations* are concrete values for a particular Characteristic and a model entity. They are stored inside the original model. Third, *Value Types* define the space of valid Manifestations. Value Types are defined using a simplified modeling language similar to Ecore. Primitive types directly map to their Ecore equivalents.

The applicability of a characteristic to a model entity, that is, the possibility to specify a manifestation on a concrete model element, is determined by *Application Rules*. Each rule can be regarded as function $rule : M \times C \rightarrow \{void, optional, mandatory\}$, with M, set of all model elements, and C, set of all characteristics. A *void* result refers to a rule not making a statement concerning the applicability. We intentionally refrained from supporting prohibiting rules to prevent rule conflicts. In our example, a rule specifies the mandatory requirement for the Resolution characteristic to each component which provides the *ICamera* interface. Multiple Characteristics and Application Rules are bundled to a *Characteristic Profile*. Similar to EMF Profiles [3], they are applied to a model.

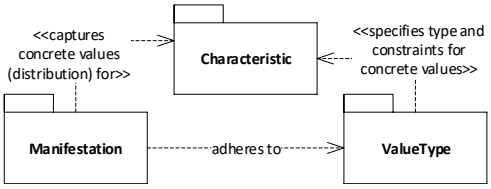

Figure 2: Overview of the Characteristics view points

When applying the MDSD.tools Characteristics framework we differentiate four phases: 1) enhancing the original meta-model with framework support, 2) defining profiles of characteristics and their applicability, 3) creating enriched models with manifestations according to the applied profiles and 4) using the enriched models in model-based quality analyses. The meta-model extension (phase 1) needs to be done by a developer familiar with the meta-model. Phase 2 targets domain experts which require additional properties. Starting with phase three the properties are integrated transparently into the model view points.

3.2 Hierarchical Manifestation Contexts

A *Context Model* is the central artifact in enhancing an existing meta model with Characteristics support. It captures which classes of a meta-model are eligible for characterization. In particular, it defines which model elements function as *Characterization Contexts* and their inter-context relationships. A context constitutes a name space containing a set of valid identifiers. Each identifier either refers to a Manifestation or links to a different but related context, e. g. a parameter name inside a RDSEFF. In our example, each BasicComponent and each AssemblyContext constitute a separate context. On the technical side, the context serves as storage container for Manifestations.

For each class which acts as Characteristic Context, the Context Model contains model queries, to find the Contexts which are *included* and to which there is a *refines* relationship. Identifiers in *included* contexts are resolved, as if they were contained in the including context. Unless specified otherwise, includes relationships are assumed along containment references.

Contextual refinements are not included by-

default, as their nature is conditional. It lies within the responsibility of the evaluating analysis to select the set of refining contexts which need to be taken into account. Manifestations in the selected refining contexts then overlay the target context. If the target context already contains a Manifestation, the refining Manifestation has precedence. For example, the Interpreter of a Palladio Performance simulation, when simulating a RDSEFF, identifies the currently active Assembly and includes selects the appropriate Assembly Context. Figure 3 visualized relevant contextual relations in our example.

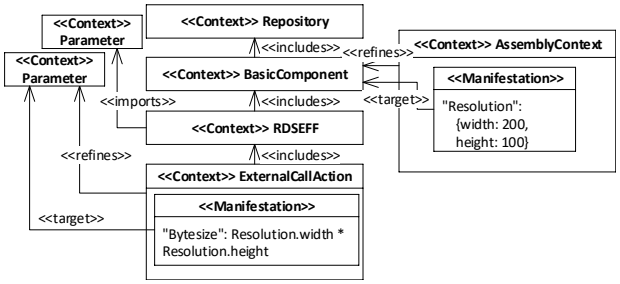

Figure 3: Excerpt of exemplary lookup hierarchy

To specify Manifestations, we extend the existing *Stochastic Expressions (StoEx)* approach [4] with support for contextual lookups. Every *Enhanced StoEx (eStoEx)* expression is evaluated in the context which contains it. We reuse the mathematical and stochastic capabilities of StoEx but refactor variable reference access. Henceforth, identifiers are resolved using the containing context. We continue to use dot separation for navigation between contexts. When resolving an identifier, e. g. by analyses, the evaluation starts with the context containing the expression and follows includes relationships until the identifier is found taking into account conditional context overlays.

4 Related Work

In the following, we discuss similar extension mechanisms. To the best of our knowledge, there is currently no approach which fulfills all of our requirements.

With Component parameters [4, p. 107], the Palladio Component Model already provides support for multi view modeling on selected elements. However, they are specific to the PCM and do not support external extensibility. Our approach provides a generic mechanism to extend existing meta-models. Component parameters provide no support for automated validation, which is a main requirement of ours.

CQML+ [1] is a generic quality modeling framework for component-based systems. It uses a similar separation between specifications and initializations of quality parameters as our framework. While CQML focuses on inter-component contracts, it does not provide support for multi-view refinements.

EMF Profiles [3] provide means to attach additional informations to existing meta-models. The elements which should be extensible are selected solely based on their meta-class. EMF Profiles does not provide support for constraint data types or contextual refinements. It could, however, serve to store Manifestations inside of existing model elements.

Architectural Templates [5] fulfill most of our requirements. ATs support complex model extensions, including structural adaptations. Their specification however requires the domain expert to create several complex artifacts, including model transformations. ATs are less intrusive to the meta-model to extend but still need similar extensions to the tooling.

5 Conclusion and Future Work

In this paper we presented foundational concepts of the MDSD.tools Characteristics framework. The framework aims to provide domain-specific extensibility to existing software architecture meta-models. It enables domain-experts to enhance enabled meta-models with additional properties without changing the meta-model or the underlying tooling itself. Furthermore, it provides a formalized mechanisms to allow for property refinements throughout the architecture design process. To provide this flexibility, meta-model and tooling need to be extended once.

The development of the framework is currently in progress[1]. Consequently, as part of future work we plan on conducting elaborate evaluations to determine the degree to which we were able to achieve our goals. We will provide more complete documentation on modeling concepts, technical aspects and application principles in an upcoming technical report.

Acknowledgements. The author acknowledges recurrent reviews and conceptual input of Stephan Seifermann and Dominik Werle (both Karlsruhe Institute of Technology). This work was partially funded by the German Federal Ministry of Education and Research under grant 16EMO0360 (SmartLoad).

References

[1] S. Röttger and S. Zschaler. "CQML+: Enhancements to CQML". In: *In Proceedings of the 1st International Workshop on Quality of Service in CBSE*. Cépaduès-Éditions, 2003, pp. 43–56.

[2] M. Broy. "Challenges in Automotive Software Engineering". In: *Proceedings of the 28th International Conference on Software Engineering*. ICSE '06. Shanghai, China: ACM, 2006, pp. 33–42.

[3] P. Langer et al. "EMF Profiles: A Lightweight Extension Approach for EMF Models". In: *Journal of Object Technology* 11.1 (Apr. 2012), 8:1–29.

[4] R. H. Reussner et al. *Modeling and Simulating Software Architectures – The Palladio Approach*. Cambridge, MA: MIT Press, Oct. 2016. 408 pp.

[5] S. M. Lehrig. "Efficiently Conducting Quality-of-Service Analyses by Templating Architectural Knowledge". PhD thesis. Karlsruher Institut für Technologie (KIT), 2018.

[6] R. Heinrich, M. Strittmatter, and R. H. Reussner. "A Layered Reference Architecture for Metamodels to Tailor Quality Modeling and Analysis". In: *IEEE Transactions on Software Engineering* (2019).

[1] `https://github.com/MDSD-Tools/Characteristics-Modeling`