Information Systems 62 (2016) 104–135

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Combining unit and specification-based testing for meta-model validation and verification

Jesús J. López-Fernández *, Esther Guerra, Juan de Lara

Computer Science Department, Universidad Autónoma de Madrid, Spain

ARTICLE INFO

Article history:
Received 3 May 2016
Accepted 22 June 2016
Recommended by: D. Shasha
Available online 30 June 2016

Keywords:
Model-driven engineering
Meta-modelling
Domain-specific modelling languages
Validation & verification
Meta-model quality

ABSTRACT

Meta-models play a cornerstone role in Model-Driven Engineering as they are used to define the abstract syntax of modelling languages, and so models and all sorts of model transformations depend on them. However, there are scarce tools and methods supporting their Validation and Verification (V&V), which are essential activities for the proper engineering of meta-models.

In order to fill this gap, we propose two complementary meta-model V&V languages. The first one has similar philosophy to the *xUnit* framework, as it enables the definition of meta-model unit test suites comprising model fragments and assertions on their (in-) correctness. The second one is directed to express and verify expected properties of a meta-model, including domain and design properties, quality criteria and platform-specific requirements.

As a proof of concept, we have developed tooling for both languages in the Eclipse platform, and illustrate its use within an example-driven approach for meta-model construction. The expressiveness of our languages is demonstrated by their application to build a library of meta-model quality issues, which has been evaluated over the ATL zoo of meta-models and some OMG specifications. The results show that integrated support for meta-model V&V (as the one we propose here) is urgently needed in meta-modelling environments.

1. Introduction

Model-Driven Engineering (MDE) [13] is a Software Engineering paradigm that promotes an active use of models and transformations throughout all phases of software development. Hence, models are used to specify, test, simulate and generate code for the final application. The rationale of MDE is that models have a higher level of abstraction than code, with less accidental details, which promises higher levels of quality and productivity [48]. Sometimes, models are described using general-purpose modelling languages like the UML, but it is also frequent the use of Domain-Specific Modelling Languages (DSMLs) capturing the abstractions within a given domain in a more concise and intuitive way [25,29].

The creation of a new DSML involves the definition of its abstract syntax by means of a meta-model declaring the relevant primitives and relations within a domain. Hence, it is important to validate this meta-model with respect to specifications of the domain, or with the help of domain experts who can provide meaningful examples of correct and incorrect uses of the DSML. Moreover, meta-models are normally defined using an object-oriented approach and are implemented in specific platforms like the Eclipse Modelling Framework (EMF) [46]. Therefore, they should adhere to accepted object-oriented quality criteria and style guidelines in object-oriented conceptual schemas [2,3], as well as to framework-specific rules and conventions.

* Corresponding author.
E-mail addresses: jesusj.lopez@uam.es (J.J. López-Fernández), Esther.Guerra@uam.es (E. Guerra), Juan.deLara@uam.es (J. de Lara).

http://dx.doi.org/10.1016/j.is.2016.06.008
0306-4379/

However, while meta-models play a central role in MDE, they are often built in an ad-hoc way, without following a sound engineering process [33,37]. This lack of systematic means for their construction yields non-repeatable processes that may lead to unreliable results, with the aggravating factor that errors in meta-models may be propagated to all artefacts developed for them, like modelling editors, model transformations and code generators. Thus, it is of utmost importance to deliver proven meta-models of high quality. Unfortunately, most efforts on DSML research focus on the implementation aspect, while neglecting domain analysis and DSML validation [31]. Hence, there are scarce methods and tools to validate and verify meta-models against domain requirements, quality guidelines and platform-specific rules.

To fill this gap, we propose two complementary languages and tool support for meta-model Validation and Verification (V&V). Our first language, called *mmUnit*, is similar to the xUnit framework [9]. It enables writing conforming and non-conforming model fragments to check whether the meta-model accepts the former and rejects the latter. Model fragments can be defined either using a dedicated textual syntax or sketched by domain experts in drawing tools, thus involving domain experts in the meta-model validation process in a more direct way [26]. For non-conforming tests, it is possible to declare assertions that state the expected disconformities and reflect the intention of the test. This is useful for regression testing, when the meta-model evolves.

The second language, called *mmSpec*, allows expressing and checking expected meta-model properties that may arise from the domain, like the existence of a path of associations between two classes, and from the implementation platform, like the existence of a root container class (which is a common practice in frameworks like EMF). The language also permits the specification of quality criteria, like threshold values for the depth of inheritance hierarchies, and style conventions, like naming rules regarding the use of capitalized nouns for class names. The latter is enabled by the use of WordNet [38], a lexical database for English.

To analyse the usefulness of *mmUnit*, we compare it with the use of the JUnit testing framework to implement meta-model test cases. Moreover, we evaluate *mmSpec* from several perspectives. First, we provide a comparison with OCL to evaluate its conciseness and relative performance. To assess its expressiveness and usefulness, we have used *mmSpec* to develop a reusable library of 30 meta-model quality properties. The properties come from quality criteria of conceptual schemas [2], naming guidelines [3], or have been derived by us from experience. The library has been applied to a repository of 295 meta-models from the ATL zoo[1] and to a suite of 30 OMG specifications. The obtained results evince the need of this kind of support for the V&V of meta-models.

This paper extends our previous work with a detailed presentation of all features of *mmUnit* and *mmSpec* (which were only partially described in [35]), and proposing their integration with an example-driven meta-model construction approach [33]. In addition, we evaluate several aspects of our languages like their usefulness, conciseness, expressiveness and performance. The analysis in [34] has also been extended to include standardized meta-models from the OMG.

The remaining of this paper is organized as follows. First, Section 2 reviews the state of the art on meta-model validation and verification, identifying existing gaps. Section 3 introduces a running example, and Section 4 overviews our approach. Then, Section 5 presents our language for example-driven testing, while Section 6 describes our language for testing expected meta-model properties. Section 7 shows the tool support and illustrates its use in an example-based process for meta-model construction. Section 8 includes an evaluation of our languages. Finally, Section 9 finishes the paper with the conclusions and future work. An appendix gives the detailed encoding of *mmSpec* primitives in OCL.

2. Approaches to meta-model validation & verification

Most efforts towards the V&V of MDE artefacts are directed to test model management operations [22], like model transformations [41], but few works target meta-model V&V. In this paper, we take the classical view of V&V [12]. Thus, while meta-model validation tries to answer the question "are we building the *right* meta-model?", verification addresses the question "are we building the meta-model *right*?". The literature reports on three main approaches to meta-model V&V, which we classify as *unit testing*, *specification-based testing* and *reverse testing*.

Unit testing approaches: This branch of works supports the definition of test suites made of models or model fragments, and their validation against a meta-model definition. This is the most usual approach, which follows the philosophy of the xUnit framework [9]. For example, in [43], test models describe instances that the meta-model should accept or reject. In a different style, [18] proposes embedding meta-modelling languages into a host programming language like PHP, and then inject the meta-model back into a meta-modelling technological space. While this enables the use of existing xUnit frameworks for meta-model testing, it resorts to a programming language for meta-model construction. The proposal presented in [40] is similar, but using Eiffel as host language. None of these works provides support for asserting the expected test results, though having an assertion language tailored to meta-model testing would enable an intensional description of the test models, documenting and narrowing the purpose of the test.

Other proposals [28,49] expand general-purpose testing tools (e.g., JUnit) to enable the testing of DSML programs, not necessarily defined by meta-models. In [47], the authors present CSTL, a JUnit-like framework to test executable conceptual schemas written in UML/OCL. Test models in CSTL are described in an imperative way, lacking specialized assertions to check for disconformities. The de facto standard meta-modelling technology EMF [46] also provides some support for testing. Given a meta-model, the EMF synthesizes a Java API to instantiate the meta-model, as well as some classes to facilitate the construction of JUnit tests. Such tests must be actually encoded using Java and JUnit assertions by the meta-model developer. Java unit testing is also proposed in [6] as a way to test meta-models. However, we believe that it would be

[1] http://web.emn.fr/x-info/atlanmod/index.php?title=Ecore