Université Bretagne Loire
Ecole Doctorale MathSTIC
Mathématiques & STIC

Spécialité : INFO
Laboratoire : LS2N
Equipe : NaoMod

A Generic Testing Approach for Low-Code Development Platforms using TDL

Faezeh Khorram
Mél : faezeh.khorram@imt-atlantique.fr

Abstract: Low-code is a growing development approach supported by many platforms entitled Low-Code Development Platforms (LCDPs). It fills the gap between business and IT by supporting the active involvement of non-technical domain experts in the application development lifecycle through taking advantage of Domain-Specific Languages (DSLs). Although low-code introduces new concepts and characteristics, there is no formal structure to the concepts and research questions of this area, especially for testing low-code software. The objective of this thesis is to characterize low-code testing and to propose a generic testing approach that provides testing facilities usable by various LCDPs. Our first contribution is a feature list for low-code testing that can be used as a baseline for comparing low-code testing components and as a guideline for building new ones. We also identify the existing challenges and offer opportunities to overcome them.

One important identified challenge in providing testing facilities for LCDPs is the diversity of the DSLs used in different LCDPs. This raises the need for a generic testing approach applicable to various LCDPs by supporting the heterogeneity of their DSLs. In this context, a standard such as the Test Description Language (TDL) appears as a suitable foundation for the definition of such an approach, but it does not provide the domain-specific concepts required to write test cases and does not include any execution facilities. In our second contribution, we address these limitations and thereby provide a fully generic testing approach for DSLs based on TDL. The application of our approach on 5 different DSLs shows its generality and that it can successfully be used for testing various low-code software.

Keywords: *Low-code Development Platform, Domain-Specific Language, Testing, Test Description Language*
Collaborations : Lowcomote european project

1 Background

A Low-Code Development Platform is a development environment typically on the cloud that can be used by non-programmers to build software applications. We call Citizen Developers these end-users of LCDPs that typically do not have any IT background. Low-code migrates the application development style from manual coding using General Purpose Languages (GPLs) into interacting with graphical user interfaces, using prebuilt components, and setting configurations [1]. The user interfaces, the business logic, and the data services can be built through graphical and/or textual modeling in a high level of abstraction by using Domain-Specific Languages (DSLs).

A DSL is a computer language specialized in a particular application domain. Each DSL enables the experts of its domain to create a facet of a system using the concepts they are familiar with (instead of programming using GPLs). For instance, the Business Process Model and Notation (BPMN) is a well-known DSL for modeling business processes. The application domains supported by an LCDP, or more specifically, the aspects of a system that are modeled in that LCDP, defines which kind of DSLs the LCDP must provide to design and develop a system. For example, Mendix LCDP provides a DSL named Microflow[2] for modeling the behavior of the system which is a variant of BPMN. In this thesis, we focus on the DSLs used for expressing the behavior of the system as *behavioral models* and that are able to execute those models—referred to as *executable DSLs (xDSLs)*.

As depicted in Figure 1, an LCDP includes two main editors. First, the system editor provides one or several DSLs for the citizen developer to design the system through modeling. Taking Mendix LCDP as an example, in the top right corner of Figure 1, you can see an example model (named "promote") conforming to the Microflow xDSL. This model defines how to promote the `level` of a given `Employee` (e. g., if the `level` is `empty`, it will be promoted to `Junior` and so on). Second, the test editor offers facilities for the citizen developer to design test cases. A sample test case for the `promote` microflow can be specified as follows: **Given** an `Employee` that does not have any `level`, **When** sending this `Employee` to the `promote` microflow, **Then** the `level` of the `Employee` should be changed to `Junior`. Since Mendix LCDP provides unit testing facilities such as assertions, the citizen developer can model such a test case using the Microflow xDSL[3] as shown in the bottom right corner of the Figure 1. However, Mendix testing facilities cannot be reused by other LCDPs which use different DSLs.

[2]https://docs.mendix.com/refguide/microflows
[3]https://docs.mendix.com/howto/testing/

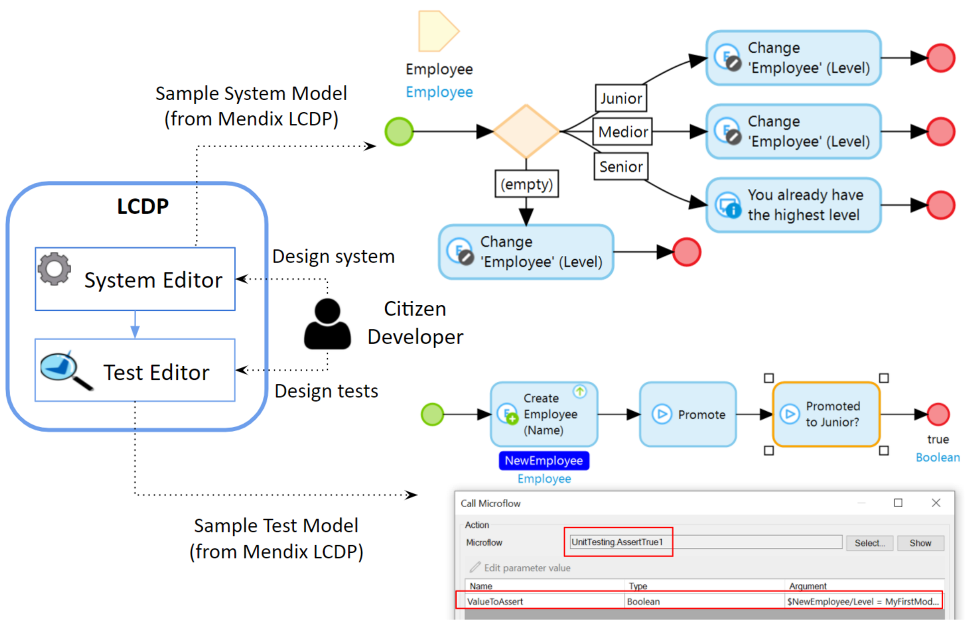

Figure 1: An example LCDP

2 Definition of Low-Code Testing

Abstraction and automation provided by xDSLs enable LCDPs to fill the gap between business and IT and thereby to improve the quality of the final product. Nevertheless, independently from the tool or platform used for the system development, the system must be tested to ensure all the requirements are realized. In an LCDP, the citizen developer is the one who defines the requirements since she is the expert on the system functionalities. As test cases are mainly derived from the requirements, the Citizen Developer is also able to define test cases and to evaluate test results. Therefore, LCDPs should provide testing facilities for her and it is necessary to support her full involvement in the testing activities, from design to evaluation. However, her low level of technical knowledge leads to the emergence of new requirements and challenges when providing a low-code testing component for an LCDP. As far as we know, there is no research at the moment which characterizes the requirements and the existing challenges. In this thesis, we perform several analyses to identify the low-code testing features and to investigate the existing challenges and opportunities for providing low-code testing components.

2.1 Low-Code Testing Features

Despite the lack of research in the context of low-code testing, there are some initial efforts made by commercial LCDPs. Accordingly, we studied the testing facilities of 5 well-known commercial LCDPs—Mendix, PowerApps, Lightning, Temenos Quantum, and OutSystems—to figure out what is offered by successful commercial platforms. Afterward, we propose a set of 16 features for the low-code testing along with possible values for them. They are defined based on the low-code principles [1] as well as the capabilities and deficiencies of the testing components of the investigated commercial LCDPs. The features can be used as criteria for comparing low-code testing components, and as a guideline for building new ones. Figure 2 presents the features, classified in 5 categories. Some of the features are general, while the rest are related to the different testing activities i. e., test design, test generation, test execution, and test evaluation. For more details, we refer the reader to our paper [2].

2.2 Challenges & Opportunities

Evaluating the investigated low-code testing components based on the proposed feature list enables us to identify the existing challenges that we classify into three categories: (1) *Role of Citizen Developer in Testing*: As the citizen developer does not have an IT background, the LCDPs should provide simple, familiar, and non-technical approaches for test design. Some LCDPs such as Mendix use the same DSL for modeling the system and the test cases (shown in Figure 1). However, since different LCDPs use different DSLs, the testing approach of an LCDP can rarely be reused by other LCDPs. (2) *The Need for High-level Test Automation*: As another consequence of the low-level technical knowledge of the citizen developer, most of the technical testing activities such as test configuration and test execution must be automated. In this context, many Model-Based Testing (MBT) tools

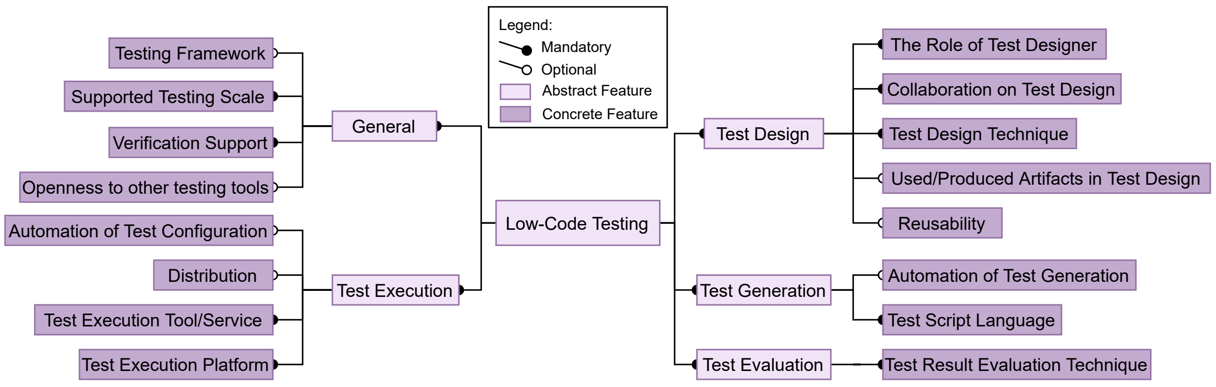

Figure 2: Low-code testing features[2]

are provided by the state-of-the-art which automates such activities. However, they are not suitable for low-code testing and cannot be reused due to the dependency on specific DSLs. (3) *Cloud Testing*: LCDPs are typically deployed on the cloud and they also support the building of cloud-based applications. Accordingly, a suitable testing approach should support cloud testing as well. To provide a roadmap for future research to overcome the mentioned challenges, we provided the existing academic research, the difficulties, and the opportunities for all the three categories in our paper [2].

3 Testing Support for xDSLs

One important identified challenge in providing testing facilities for LCDPs is the diversity of the xDSLs used in different LCDPs for system development. In a context where the engineering of new xDSLs is recurrent, a desirable solution would be an approach applicable to a wide range of xDSLs, i. e., a *generic* testing approach for xDSLs. This raises at least three interconnected challenges. First, to allow the domain expert (i. e., the citizen developer) to write test cases, a testing language must be defined, generated, or identified. In particular, this testing language must somehow allow the domain expert to use domain concepts to define how a model under test should be executed, and what results should be expected from the execution. Second, the testing language should be able to demand the execution of the models under test as needed. Third, this testing language must provide facilities to analyze the runtime state of the tested model, and to compare this state with the expected one.

A recent effort of the European Telecommunications Standards Institute (ETSI) led to the creation of the Test Description Language (TDL), a standardized language for the specification of test descriptions [3]. Since TDL is not specific to any specific GPL or xDSL, it represents an interesting candidate for generically writing test cases for executable models. Also, TDL was designed as a simple language for testers lacking programming knowledge, making it a good fit for domain experts working on models. Unfortunately, TDL fails to fully address the three aforementioned challenges: (1) because of its genericity, TDL requires the domain expert to first define the required domain-specific concepts, before being able to write test cases; (2) the TDL standard does not provide any clear way to make TDL test cases able to execute models conforming to a given xDSL; (3) the TDL standard does not provide any efficient way to analyze an arbitrarily complex runtime state of a tested model. As second contribution of this thesis, we address these limitations and thereby propose *a novel generic testing approach for xDSLs*.

Figure 3 presents an overview of the proposed approach. At the top left corner, we assume that an xDSL was implemented by an LCDP Developer. At the center, the domain expert (i. e., the citizen developer) uses the provided xDSL to define an executable model and wishes to write TDL test cases for this model.

At the top is shown the first component of the approach, namely the *TDL Library Generator*. Its purpose is to automatically generate a domain-specific TDL Library that the domain expert can then use to conveniently write test cases for executable models conforming to the xDSL. This library provides all the data types required for the specification of test data, a set of default test configurations, some elements for enabling TDL test cases to request the execution of the model under test, and some elements for writing Object Constraint Language (OCL) queries in the TDL test cases. As shown in Figure 3, the library generator requires as input data the definition of the xDSL.

At the bottom-right corner is shown the second component of the approach, namely the *TDL Interpreter*. This interpreter is based on the operational semantics for TDL, adapted to the testing of executable models. For this purpose, this operational semantics is connected to two external components: the *Execution Engine* and the *Query Evaluator*. We assume that the *Execution Engine* exists and provides services to trigger the execution of a model conforming to an xDSL. In particular, that this engine is able to load a model, load an xDSL, and execute the model

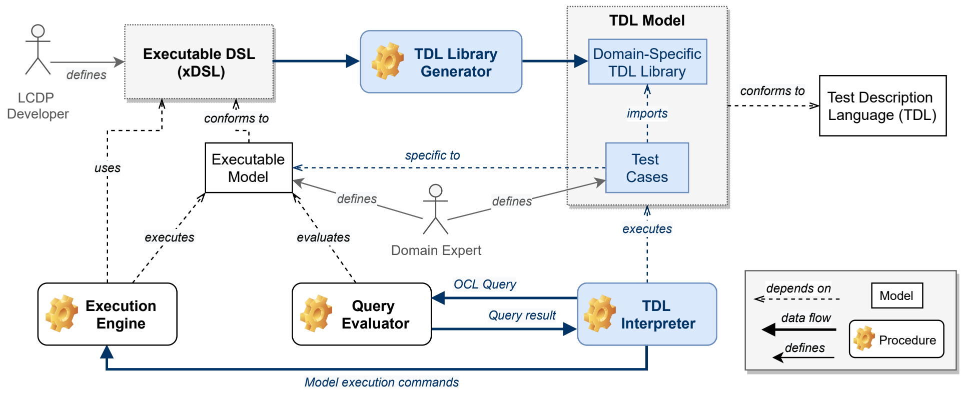

Figure 3: Overview of the proposed approach

using the semantics of the xDSL. Here, the Execution Engine is used by the TDL Interpreter to start the execution of a model under test, to get the content of the model, or to set the model in a specific runtime state. We also assume that the *Query Evaluator* can evaluate OCL queries. The Query Evaluator is used by the TDL Interpreter to evaluate queries written inside TDL test cases, so the query validation result can be used as part of the oracle of a TDL test case. With everything in place, the domain expert can use the generated domain-specific TDL library to write test cases, and can then use the TDL interpreter to execute these test cases.

We implemented the presented approach for the GEMOC Studio, a language and modeling workbench for xDSLs [4]. The source code is available on a public GitLab instance[4]. We conducted an evaluation to assess the *genericity* aspect of our proposed approach considering the diversity of xDSLs. To this end, we applied the approach on 5 xDSLs covering various domains and implemented it with different techniques. The evaluation results demonstrate that the genericity aspect is successfully realized. We submitted our work to the ECMFA conference and it is under review [5]

4 Conclusion

Due to the high trend toward low-code and the limited academic resources for low-code testing, we performed several analyses to characterize the features of the low-code testing. As a result, we provided a list of 16 features to be used when comparing or building low-code testing components. Our analysis reveals three main challenges in this domain, one of those the diversity of the xDSLs used in various LCDPs. It raises the need for a generic testing approach that supports this heterogeneity and provides a dedicated low-code testing component for a given xDSL. In this thesis, we propose such an approach using TDL language. We evaluated the approach by applying it for 5 different xDSLs—different in the supported application domain and in the techniques used for their implementation— and the result demonstrated that the genericity aspect is well-satisfied.

References

[1] Massimo Tisi, Jean-Marie Mottu, Dimitrios S. Kolovos, Juan De Lara, Esther M Guerra, Davide Di Ruscio, Alfonso Pierantonio, and Manuel Wimmer. Lowcomote: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms. In *STAF 2019 Co-Located Events Joint Proceedings*, 2019.

[2] Faezeh Khorram, Jean-Marie Mottu, and Gerson Sunyé. Challenges and opportunities in low-code testing. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–10, 2020.

[3] Philip Makedonski, Gusztáv Adamis, Martti Käärik, Finn Kristoffersen, Michele Carignani, Andreas Ulrich, and Jens Grabowski. Test descriptions with etsi tdl. *Software Quality Journal*, 27(2):885–917, 2019.

[4] Erwan Bousse, Thomas Degueule, Didier Vojtisek, Tanja Mayerhofer, Julien Deantoni, and Benoit Combemale. Execution framework of the gemoc studio (tool demo). SLE 2016, page 84–89, 2016.

[5] Faezeh Khorram, Erwan Bousse, Jean-Marie Mottu, and Gerson Sunyé. Testing support for executable dsls using tdl. *Manuscript submitted for publication*, 2021.

[4]https://gitlab.univ-nantes.fr/naomod/faezeh-public/xtdl