The MULTI Process Challenge – EMISAJ Special Issue Version

João Paulo A. Almeida*, Thomas Kühne†, Adrian Rutle‡ and Manuel Wimmer§
*Federal University of Espírito Santo, Vitória, Brazil
†Victoria University of Wellington, New Zealand
‡Western Norway University of Applied Sciences, Bergen, Norway
§Johannes Kepler University Linz, Austria
jpalmeida@ieee.org, tk@ecs.vuw.ac.nz, Adrian.Rutle@hvl.no, manuel.wimmer@jku.at

***Abstract*—This challenge is intended to allow the demonstration of multi-level modeling techniques and enable the comparison of submissions, and hence framework/language capabilities. The multi-level modeling community is invited to respond to this challenge with submissions describing solutions to the challenge expressed in a technology of their choosing. Authors should emphasize the merits and limitations of their solution according to the criteria defined in this challenge description. This challenge closely follows the MULTI 2019/2020 Process Challenge with minor changes to Sections 1 and 3, and adds more specific presentation requirements.**

***Index Terms*—Multi-level modeling, challenge, process management, MULTI workshop.**

1. Introduction

Multi-level modeling (MLM) represents a significant extension to the traditional two-level object-oriented paradigm with the potential to improve upon the utility, reliability and complexity of models. In contrast to conventional approaches, MLM allows for an arbitrary number of classification levels and introduces further concepts that foster expressiveness, reuse and adaptability.

The modeling challenge described here is intended as a basis for demonstrating MLM capabilities and enabling the comparison of MLM alternatives. The challenge description closely follows the MULTI 2019/2020 Process Challenge [3] which in turn was inspired by the MULTI Bicycle Challenge used in MULTI 2017 [1] & MULTI 2018 [2]. Despite similarities in the criteria between the bicycle and process challenges, the subject domain has been changed entirely and new criteria have been added which are intended to increase opportunities for languages and tools to exercise their capabilities.

This challenge concerns the domain of process management [4], a domain in which one is not only interested in *particular occurrences* (i.e., “processes” = “process instances”, “tasks” = “task occurrences”), but also in universal aspects of *classes of occurrences* (“process types”, “task types”) and their relations to actor types and artifact types. Furthermore, so-called process metamodelling can be used to classify this universal level in turn.

This challenge is intended to elicit submissions which demonstrate how MLM technology can deal with such a multi-level domain. For example, domain-specific concepts may be defined in their dedicated branches of a hierarchy of models without polluting the general terminology of process management, allowing domain-specific behaviour to be defined for each branch of the hierarchy while allowing for the reuse/enforcement of common structure/behaviour.

Each submission will be reviewed against the following criteria: (i) Does the submission address the established domain as described in Section 2 and demonstrate the use of multi-level features? Note that it is not required to satisfy all requirements listed in Section 2, however any omissions should be flagged and discussed. (ii) Does it evaluate/discuss the proposed modeling solution against the criteria presented in Section 3? (iii) Does it discuss the merits and limitations of the applied MLM technique in the context of the challenge? Authors may suggest further requirements that clearly demonstrate the utility of their chosen approach.

Submitters should either include a complete solution in the form of a model as part of their submission or alternatively provide a supplementary artifact which completely details their solution using a standard format, such as PDF.

The proposed solution should be presented in an article with at least the following sections:

1) Technology (precise description of the technology / approach that is used);
2) Analysis (any disambiguations of the case description and assumptions made, any potentially added requirements);
3) Model Presentation (detailed presentation of a model, including justifications for design decisions);
4) Satisfaction of Requirements (demonstration of how the solution satisfies the challenge requirements);
5) Assessment of the Modeling Solution (discussing choices made, pointing out potential compromises / deficiencies);
6) Related Work (positioning and contrasting the presented solution with related work);
7) Conclusions (including lessons learned, impulses for future work, etc.).

2. Case description

This MLM challenge involves representing universal properties of process types along with task types, artifact types, actor types and their various relations and attributes (Section 2.2), and an application of this conceptualization in the scope of a particular software engineering process (Section 2.3). Submitted solutions should include bottom-level instances, at least for key types, exemplifying all attributes mentioned in the challenge description. Deviations from the case as described here should be documented in submissions. The case description may be extended but respective rationales should then be provided.

2.1. Overview

Process management is characterized by the prescription of rules concerning the execution of certain types of processes, tasks, actions or activities. It therefore involves regulating and keeping track of processes, i.e, the enactments of process types, with such process enactments often being referred to as "process instances" in the process management literature. For example, in the software engineering domain, it may be necessary to keep track of the results of certain tasks such as *testing*, e.g., the fact whether or not code has been *tested*, etc. Further rules impose requirements on the participation of business actors (humans, organizations) and artifacts (equipment, documents, tools) in certain tasks and specifies dependencies. For example, in the software engineering domain: (i) *testing* requires prior *test case design*; (ii) *test case design* is *performed* by a *senior analyst*, employs a *requirements specification*, and results in *test cases*; and (iii) *testing* is performed by a *tester*, employs *test cases*, and produces a *test report*.

In other contexts, such as the insurance domain, there may be a need to keep track of which *policy holder submitted an insurance claim*, when it was submitted, which *claims analyst authorized payment* of the insurance premium in response to the claim, how much was claimed, which claims are still pending assessment, how much was paid out for a particular claim, etc.

Submissions to the challenge should focus on the software engineering domain. They may optionally include the insurance domain as well. In the following, we are using the insurance domain for illustrative purposes only.

2.2. Processes, tasks, actors and artifacts

The following general rules pertaining to processes, tasks, actors and artifacts apply for the challenge:

P1) A *process type* (such as *claim handling*) is defined by the composition of one or more *task types* (*receive claim*, *assess claim*, *pay premium*) and their relations.

P2) Ordering constraints between *task types* of a *process type* are established through *gateways*, which may be *sequencing*, *and-split*, *or-split*, *and-join* and *or-join*.

P3) A *process type* has one *initial task type* (with which all its executions begin), and one or more *final task types* (with which all its executions end).

P4) Each *task type* is *created by* an *actor*, who will not necessarily perform it. For example, *Ben Boss created* the task type *assess claim*.

P5) For each *task type*, one may stipulate a set of *actor types* whose instances are the only ones that may *perform* instances of that *task type*. For example, in the *XSure* insurance company, only a *claim handling manager* or a *financial officer* may *authorize payments*.

P6) A *task type* may alternatively be assigned to a particular set of *actors* who are authorized (e.g., *John Smith* and *Paul Alter* may be the only *actors* who are allowed to *assess claims*).

P7) For each *task type* (such as *authorize payment*) one may stipulate the *artifact types* which are *used* and *produced*. For example, *assess claim* uses a *claim* and produces a *claim payment decision*.

P8) *Task types* have an *expected duration* (which is not necessarily respected in particular occurrences).

P9) *Critical task types* are those whose instances are *critical tasks*; each of the latter must be *performed* by a *senior actor* and the artifacts they produce must be associated with a *validation task*.

P10) Each *process type* may be enacted multiple times.

P11) Each *process* comprises one or more *tasks*.

P12) Each *task* has a *begin date* and an *end date*. (e.g., *Assessing Claim 123* has *begin date 01-Jan-19* and *end date 02-Jan-19*).

P13) *Tasks* are associated with *artifacts used* and *produced*, along with *performing actors*.

P14) Every *artifact used* or *produced* in a *task* must instantiate one of the *artifact types* stipulated for the *task type*.

P15) An *actor* may have more than one *actor type* (e.g., *Senior Manager* and *Project Leader*.)

P16) Likewise, an *artifact* may have more than one *artifact type*.

P17) An *actor* who *performs* a *task* must be authorized for that task. Typically, a class of actors is automatically authorized for certain classes of tasks.

P18) *Actor types* may *specialize* other *actor types* in which case all the rules that apply to instances of the specialized *actor type* must apply to instances of the specializing *actor type*. For example, if a *manager* is allowed to *perform tasks* of a certain *task type*, so is a *senior manager*.

P19) All modeling elements, at all levels, must have a *last updated* value of type *time stamp*. This feature should be defined as few times as possible, ideally only once. Respective definitions are exempt from the requirement to have a *last updated* value. Note that this requirement differs from the respective version in [3].

Note that it is not necessary for every *type* in the model to have an instance. It is useful, however, to illustrate the design with a number of instances.

2.3. Software engineering process

An application of the above described process management must be defined to capture domain-specific aspects of software engineering processes in the fictional *Acme Software Development Company*[1]. The *Acme software development process* is composed of: *requirements analysis*, *design*, *coding*, *test case design*, *test design review* and *testing* (conforming to the constraints indicated in Figure 1, where the bars represent an *and-split* and an *and-join* respectively).

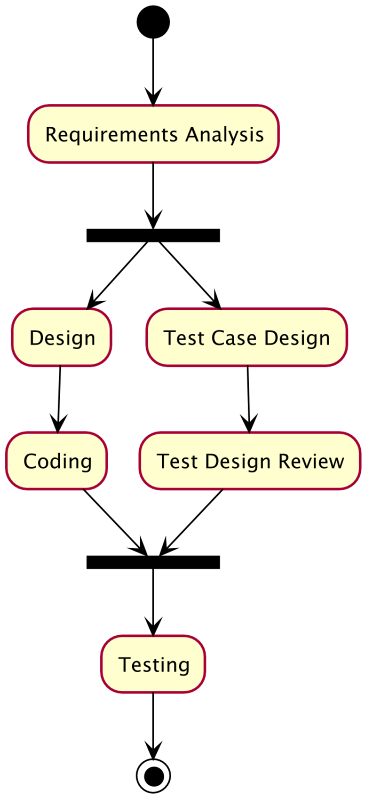

Figure 1. The Acme software engineering process.

The following rules for the software engineering domain apply:

S1) A *requirements analysis* is *performed by* an *analyst* and *produces* a *requirements specification*.

S2) A *test case design* is *performed by* a *developer* or *test designer* and *produces test cases*. Note that this requirement from [3] conflicts with S13. We have maintained it here for the record but ask submitters to let the information in S13 override S2, i.e. only *senior analysts* may perform a *test case design*. Note that *test case designs* still *produce test cases*.

S3) An occurrence of *coding* is *performed by* a *developer* and *produces code*. It must furthermore reference one or more *programming languages* employed.

1. As mentioned before, an additional incorporation of the insurance domain is purely optional.

S4) *Code* must reference the *programming language(s)* in which it was *written*.

S5) *Coding in COBOL* always produces *COBOL code*.

S6) All *COBOL code* is written in *COBOL*.

S7) *Ann Smith* is a *developer*; she is the only one allowed to perform *coding in COBOL*.

S8) *Testing* is *performed by* a *tester* and *produces* a *test report*.

S9) Each *tested artifact* must be associated to its *test report*.

S10) *Software engineering artifacts* have a responsible *actor* and a *version number*. This applies to *requirements specification*, *code*, *test case*, *test report*, but also to any future types of *software engineering artifacts*.

S11) *Bob Brown* is an *analyst* and *tester*. He has *created* all *task types* in this *software development process*.

S12) The *expected duration* of *testing* is *9 days*.

S13) *Designing test cases* is a *critical task* which must be performed by a *senior analyst*. *Test cases* must be validated by a *test design review*.

3. Solution presentation requirements

Submissions responding to the challenge should describe a multi-level model conforming to the case description, including justifications for non-trivial design decisions. In order to foster comparability between solutions, respondents are asked to make sure that concepts of the case description are explicitly represented by one or more model elements. Conformance of the model elements to each of the requirements (P1–P19 and S1–13) must be documented in a dedicated section of the article.

3.1. Mandatory discussion aspects

Challenge respondents must discuss their multilevel model solution with regard to the following aspects, each of which should be treated in a specific sub-section of the "Assessment" section of the article:

– **Basic modeling constructs**: Explain the basic modeling constructs used in the solution.
– **Levels (or other model content organization schemes employed)**: Explain the nature of "levels" in the model, how model elements are arranged on these levels and which relationships (such as "instance-of") may feature between elements at different levels. The nature of levels should be captured by explicitly stating the level segregation and the level cohesion principles used [5]. Avoid vague language such as "higher level concepts are more abstract" if the inter-level relationship is more specific. If the inter-level relationship is deliberately allowed to be vague, state this explicitly.
– **Number of levels**: Elaborate whether the submitted solution could have had more or fewer levels and explain how any potentially existing degrees of freedom were resolved.
– **Cross-level relationships**: Discuss if and how associations and links can connect model elements at different levels. State well-formedness constraints, if any apply.

– **Cross-level constraints**: Discuss if and how constraints can span multiple levels, especially with regard to cross-level relationships.
– **Integrity mechanisms**: Discuss how the integrity of level contents is preserved when changes to level contents occur.
– **Deep characterization**: Discuss if and how higher levels influence elements at lower levels with a level distance of two or more. Such an influence may be desired to ensure properties of lower level elements regardless of the design choices that modelers make at intermediate levels, including future extensions to intermediate levels.
– **Generality**: Discuss the generality of the solution. Is (part of) it applicable to other domains? Does it embody invariant principles of the domain(s) it covers with minimal redundancy?
– **Extensibility**: Elaborate how the solution would respond to further requirements, such as further special tasks that must be taken care of by special actors. Identify expected extension points in the solution, e.g., subtyping opportunities. If level insertion is a possibility in your chosen approach, explain how it would be performed.

Solutions should discuss the aspects listed above with respect to related work.

Submissions may reference prior publications for technical details but should strive to be self-contained regarding the explanation of the major aspects of the technology they employ.

3.2. Recommended discussion aspects

Challenge respondents are invited to:

– Indicate whether there are formalisms to establish the semantics of the MLM technique and/or tools that support the presented solution.
– Discuss model verification (e.g., consistency analyses) or other quality assessment mechanisms supported by the MLM technique employed.

4. Conclusions

Submissions should cover:

– Requirements the solution does not address, if any.
– Any extensions that may have been made to the case description or evaluation aspects.
– Advantages and drawbacks of the presented solution.
– Advantages and drawbacks of the presented MLM approach that may not be evident in the solution to the challenge but are worth mentioning.
– Lessons learned and their implications for future work.

Acknowledgments

We would like to thank Ulrich Frank and Tony Clark who authored the 2017/2018 MULTI challenges for establishing the format of the challenge, including some of the general requirements and many of the discussion aspects. We would also like to thank Colin Atkinson for reviewing the challenge and providing feedback. Parts of the challenge were inspired by a model published in [6]. João Paulo A. Almeida is partly supported by CNPq (407235/2017-5 and 312123/2017-5) and CAPES Finance Code 001 (23038.028816/2016-41).

References

[1] "MULTI 2017: 4th International Workshop on Multi-Level Modelling," https://www.wi-inf.uni-duisburg-essen.de/MULTI2017/, accessed January 12, 2021.

[2] "MULTI 2018: 5th International Workshop on Multi-Level Modelling," https://www.wi-inf.uni-duisburg-essen.de/MULTI2018/, accessed January 12, 2021.

[3] J. P. A. Almeida, A. Rutle, M. Wimmer, and T. Kühne, "The MULTI process challenge," in *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion 2019, Munich, Germany, September 15-20, 2019*, L. Burgueño, A. Pretschner, S. Voss, M. Chaudron, J. Kienzle, M. Völter, S. Gérard, M. Zahedi, E. Bousse, A. Rensink, F. Polack, G. Engels, and G. Kappel, Eds. IEEE, 2019, pp. 164–167. [Online]. Available: https://doi.org/10.1109/MODELS-C.2019.00027

[4] M. Dumas, W. M. van der Aalst, and A. H. ter Hofstede, *Process-aware Information Systems: Bridging People and Software Through Process Technology*. New York, NY, USA: John Wiley & Sons, Inc., 2005.

[5] T. Kühne, "A story of levels," in *Proceedings of the MODELS 2018 Workshops co-located with the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS 2018)*, ser. CEUR Workshop Proceedings, ISSN 1613-0073, vol. Vol-2245, 2018, pp. 673–682.

[6] J. D. Lara and E. Guerra, "Refactoring multi-level models," *ACM Trans. Softw. Eng. Methodol.*, vol. 27, no. 4, pp. 17:1–17:56, Nov. 2018. [Online]. Available: http://doi.acm.org/10.1145/3280985