The MULTI Process Challenge

João Paulo A. Almeida[1], Adrian Rutle[2], Manuel Wimmer[3] and Thomas Kühne[4]
[1]Federal University of Espírito Santo, Vitória, Brazil
[2]Western Norway University of Applied Sciences, Bergen, Norway
[3]Johannes Kepler University Linz, Austria
[4]Victoria University of Wellington, New Zealand
jpalmeida@ieee.org, Adrian.Rutle@hvl.no,
manuel.wimmer@jku.at, tk@ecs.vuw.ac.nz

***Abstract*—This challenge is intended to allow submitters to demonstrate the use 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 papers describing solutions to the challenge. Authors should emphasize the merits of their solutions according to the aspects defined in this challenge description. The challenge follows up on the "MULTI Bicycle Challenge" which was used in MULTI 2017 and MULTI 2018, and reuses some criteria that were established in these previous editions. Despite the similar criteria, 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.**

***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. A key aspect of the MLM paradigm is the use of entities that are simultaneously types and instances, a notion which has consequences for conceptual modeling, language engineering and for the development of model-based software systems.

Research into MLM has increased significantly over the last few years, manifesting itself in lively debates in the literature, five international workshops (MULTI 2014–2018), a published journal theme issue (SoSyM), a special issue for the EMISA journal, a Dagstuhl Seminar (in 2017) [4], and an increasing number of tools and languages [3], with different capabilities and underlying theories.

The modeling challenge described in this paper is intended as a basis for demonstrating these capabilities and enabling their comparison. It follows up on the MULTI Bicycle Challenge which was used in MULTI 2017 [1] and MULTI 2018 [2], and reuses some criteria that were established in these previous editions. Despite the similar criteria, 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 [5], 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 definitions", "task types") and their relations to actor types and artifact types. This challenge is intended to elicit submissions which demonstrate the advantages of MLM not only but also in comparison to traditional solutions as they have been proposed in the context of BPMN, Workflow modelling, etc. 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 of common behaviour.

Each challenge response will be reviewed against the following criteria: (i) Does the response address the established domain as described in Section 2 and demonstrate the use of multi-level features? (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 an MLM technique in the context of the challenge?

Papers that clearly address the review criteria listed above will be accepted for presentation at the workshop and for inclusion in the workshop proceedings. Tool demonstrations that are suited to show the strengths of the proposed solution are appreciated as well. Authors are invited to suggest further requirements that clearly demonstrate the utility of multi-level modeling.

Challenge responses should be submitted as regular papers. Each submission must be subtitled: "A contribution to the MULTI Process challenge". Any artifacts (models, implementation) produced should be made available for review in material accompanying the paper, e.g., in a shared repository. The proposed solution should be presented in a paper with the following structure:

1) Introduction (presentation of the approach that is used);
2) Case analysis (analysis, interpretation, completion of case description, any additional requirements);
3) Model presentation (step-by-step presentation of model, including justifications for design decisions);
4) Discussion (reflecting on requirements that could not be addressed, etc.);
5) Conclusions.

2. Case description

This multi-level modeling 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). Challenge responses should include instances for all of the types defined, exemplifying all attributes mentioned in the challenge description. Deviations from the case as described here should be documented in challenge responses. Challenge participants may extend the case description in their responses but should provide a respective rationale for the extensions.

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 *test case designer*, employs a *requirements specification* and results in a *test suite*; and (iii) *testing* is performed by a *tester*, employs a *test suite* 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 include the insurance domain as well, but in the following, we are mainly using the latter for illustrative purposes. Making sure that submitted models cover both software engineering and insurance domains is optional.

2.2. Processes, tasks, actors and artifacts

The following rules for 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* that *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) *Artifacts*, *actors* and all *types* (including *process*, *task*, *artifact* and *actor types*) are given a set of alternative *names* (e.g., to cope with internationalization requirements or variation in terminology).

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).

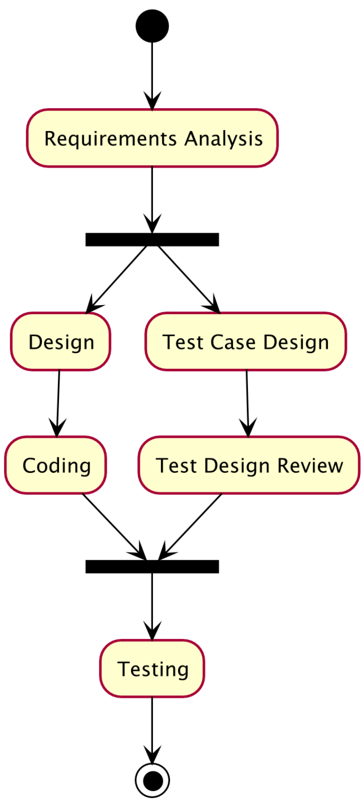

Figure 1. The Acme software engineering process.

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*.[2]
S3) An occurrence of *coding* is *performed by* a *developer* and *produces code*. It must furthermore reference one or more *programming languages* employed.
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*.

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

2. This requirement was found by one of the challenge submitters of MULTI 2019 to contradict S13. We have maintained it here for the record, along with all of the original text of the challenge. We recommend that future responses to the challenge ignore S2 and further reference the most up to date version of the challenge description.

3. Solution presentation requirements

Submissions responding to the challenge should describe a multi-level model conformant 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.

3.1. Mandatory discussion aspects

Challenge respondents should discuss their multilevel model solution with regard to the following aspects:

– **Basic modeling constructs**: Discuss the basic modeling constructs used in the solution.
– **Levels (or other model content organization schemes employed)**: Discuss 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 may be captured by explicitly stating the level segregation and the level cohesion principles used [6].
– **Cross-level relationships**: Discuss if and how associations and links can connect model elements at different levels.
– **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.
– **Abstraction**: Discuss the resulting level of abstraction of the solution. Is it applicable to other domains? Does it embody invariant principles of the domain(s) it covers with minimal redundancy?

– **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.
– **Reuse**: It should be possible to use the model (or parts of it) as a foundation for a software system that is suited for a wide range of process management requirements and adapt (customize) it to specific circumstances. Optionally, a submission may demonstrate how the insurance domain can be covered by the same process conceptualization.

3.2. Recommended discussion aspects

Respondents are invited to:

– Position their solution with respect to related work according to the aspects listed above.
– Indicate whether there are formalisms to establish the semantics of the MLM technique and/or tools that support the presented solution.
– Discuss model verification or other quality assessment mechanisms supported by the MLM technique employed.

4. Conclusions

Participants should summarize their submission and highlight:

– Limitations of the solution.
– Key advantages and drawbacks of the presented solution with regard to the challenge.
– Key advantages and drawbacks of the presented MLM approach that may not be evident in the solution to the challenge but are worth mentioning.
– Key extensions of the case description and considered discussion aspects.

Acknowledgments

We would like to thank Ulrich Frank and Tony Clark who authored the previous 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 [7]. 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 May 21, 2019.

[2] "MULTI 2018: 5th International Workshop on Multi-Level Modelling," https://www.wi-inf.uni-duisburg-essen.de/MULTI2018/, accessed May 21, 2019.

[3] "Multi-Level Modeling Wiki," http://homepages.ecs.vuw.ac.nz/Groups/MultiLevelModeling/, accessed May 21, 2019.

[4] J. P. A. Almeida, U. Frank, and T. Kühne, "Multi-Level Modelling (Dagstuhl Seminar 17492)," *Dagstuhl Reports*, vol. 7, pp. 18–49, 2018.

[5] 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.

[6] 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.

[7] 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