Comparing Three Notations for Defining Scenario-based Model Tests: A Controlled Experiment

Bernhard Hoisl*†, Stefan Sobernig*, and Mark Strembeck*†
*Institute for Information Systems and New Media,
Vienna University of Economics and Business (WU Vienna)
†Secure Business Austria Research (SBA Research)
{firstname.lastname}@wu.ac.at

***Abstract*—Scenarios are an established means to specify requirements for software systems. Scenario-based tests allow for validating software models against such requirements. In this paper, we consider three alternative notations to define such scenario tests on structural models: a semi-structured natural-language notation, a diagrammatic notation, and a fully-structured textual notation. In particular, we performed a study to understand how these three notations compare to each other with respect to accuracy and effort of comprehending scenario-test definitions, as well as with respect to the detection of errors in the models under test. 20 software professionals (software engineers, testers, researchers) participated in a controlled experiment based on six different comprehension and maintenance tasks. For each of these tasks, questions on a scenario-test definition and on a model under test had to be answered. In an ex-post questionnaire, the participants rated each notation on a number of dimensions (e.g., practicality or scalability). Our results show that the choice of a specific scenario-test notation can affect the productivity (in terms of correctness and time-effort) when testing software models for requirements conformance. In particular, the participants of our study spent comparatively less time and completed the tasks more accurately when using the natural-language notation compared to the other two notations. Moreover, the participants of our study explicitly expressed their preference for the natural-language notation.**

I. Introduction

In model-driven development (MDD), models are used to define a software system's problem domain as well as corresponding solutions. Thereby, models abstract from the underlying implementation technology (see, e.g., [1], [2]). In this context, models are not only used for documentation purposes, but are first-class artifacts from which executable code is generated. Consequently, the quality of a software product considerably relies on the quality of the corresponding models. To ensure such high quality models, the predominantly code-centric testing strategies must be complemented with approaches for testing at the model-level. Such model-level tests help detect errors at an early development stage (see, e.g., [3], [4]).

In recent years, scenarios have become a popular means for capturing a software system's requirements (see, e.g., [5]–[7]). In MDD, scenario specifications can be used to test models for compliance with corresponding domain requirements (see, e.g., [3], [4], [8]). Different approaches have been proposed to provide notations for specifying scenarios, for example in a table-based layout, as message sequence charts, or via formal methods (see, e.g., [7], [9], [10]). However, no generally accepted standard notation exists and the different scenario notations vary with respect to the corresponding application domain and with the professional background of the stakeholder involved in a particular development project (see, e.g., [7]). Nevertheless, independent of the visualization technique, three cognitive activities are fundamental to determine the usability and quality of different notations: learnability, understandability, and changeability (see, e.g., [11]). These aspects have a significant influence on the productivity (see, e.g., [12]): the costs increase the longer a domain expert or software engineer needs to understand a scenario and the less software errors are corrected. Hence, by ensuring easy-to-understand scenario tests, the software-development costs can be decreased.

In this paper, we report on a controlled experiment for the comparison of three frequently used alternatives for scenario notations (see, e.g., [7]) to test Ecore-based structural models[1]. In particular, we compare a semi-structured natural-language notation, a diagrammatic notation, and a fully-structured textual notation. As the comprehension of a notation is a cognitive process, it cannot be observed directly (see [13]). Therefore, we compare the three notations with respect to the accuracy and the required effort for comprehending scenario-test definitions, as well as with respect to the detection of errors in the models under test (MUT). The evaluation consists of comprehension (learnability, understandability) and maintenance (changeability) tasks (see, e.g., [14]). As concrete measures, we use the response time and the correctness of a solution (see, e.g., [12]). We report our experiment according to the guidelines for empirical software studies defined by Jedlitschka et al. [15] and by Kitchenham et al. [16].

The remainder of this paper is structured as follows. In Section II, we describe the three notations that we compare in our study. Subsequently, Section III gives an overview of related work, before Section IV explains the planning and design of the experiment. Section V describes the execution of the experiment and Section VI provides an analysis of the study's results. Next, Sections VII and VIII interpret the results and discuss threats to validity. Section IX concludes the paper and points to future work. In the Appendix, we showcase a

[1] All materials used in the experiment as well as collected data, calculations, and computed results can be downloaded from http://nm.wu.ac.at/modsec.

complete task actually performed during the experiment.

II. Three Notations for Scenario-based Testing

In this section, we provide an overview of the three scenario notations. In particular, we show an example how the same test scenario is described in each of the three notations. The example is taken from the introductory tutorial given to the participants of our study at the beginning of each experiment session.

For the experiment, each scenario test consists of three parts: (1) preconditions setting up the scenario context, (2) events triggering an action, and (3) expected outcomes of the action (see, e.g., [7], [17]). All three parts of a scenario test must evaluate to true in order to make a scenario test pass as a whole. In the example, and as required by the design of our experiment, the MUT is an Ecore model[2].

N-notation: The semi-structured natural-language notation is adopted from Cucumber's set of syntax rules for scenario tests (called Gherkin; see [17]). The natural-language statements follow a declarative sentence structure. In Listing 1, line 1 initializes a new scenario (Scenario). The keywords Given, When, and Then define the start of a precondition, an event, and an expected outcome, respectively (lines 2–4). This natural language scenario definition can be automatically transformed into executable test scenarios which can be processed by the framework presented in [19]. The scenario test passes iff all test expressions are fulfilled, otherwise it fails.

```
1 Scenario:
2   Given "that ClassH owns exactly three structural
        features"
3   When "all supertype classes of ClassB are abstract
        "
4   Then "in ClassJ there shall be at least one
        attribute of type EnumA with an upper bound
        multiplicity of -1"
```

Listing 1. Semi-structured natural-language scenario notation (N-notation) exemplified.

D-notation: The second scenario-test notation is based on UML sequence diagrams [20]. It is inspired by related model-based scenario-test approaches (see, e.g., [9], [21]). Fig. 1 shows the example scenario test defined via this diagrammatic representation—it is semantically equivalent to the natural-language representation in Listing 1. Lifelines represent instances of a test runner, a test component, the MUT, and classes contained in the MUT which collaborate to realize the tested scenario. Interactions are shown as sequences of call and return messages between the corresponding instances on the lifelines. Events and expected outcomes are declared optional (see the CombinedFragments with opt InteractionOperatorKind in Fig. 1 [20]). The different scenario-test expressions are nested, so that an event can only be triggered iff the precondition(s) are fulfilled (see InteractionConstraint guards of CombinedFragments in Fig. 1). In the same way, an expected outcome is only tested iff all preconditions and events are fulfilled.

E-notation: The third notation defines scenario tests through an extension to a fully-structured language for model

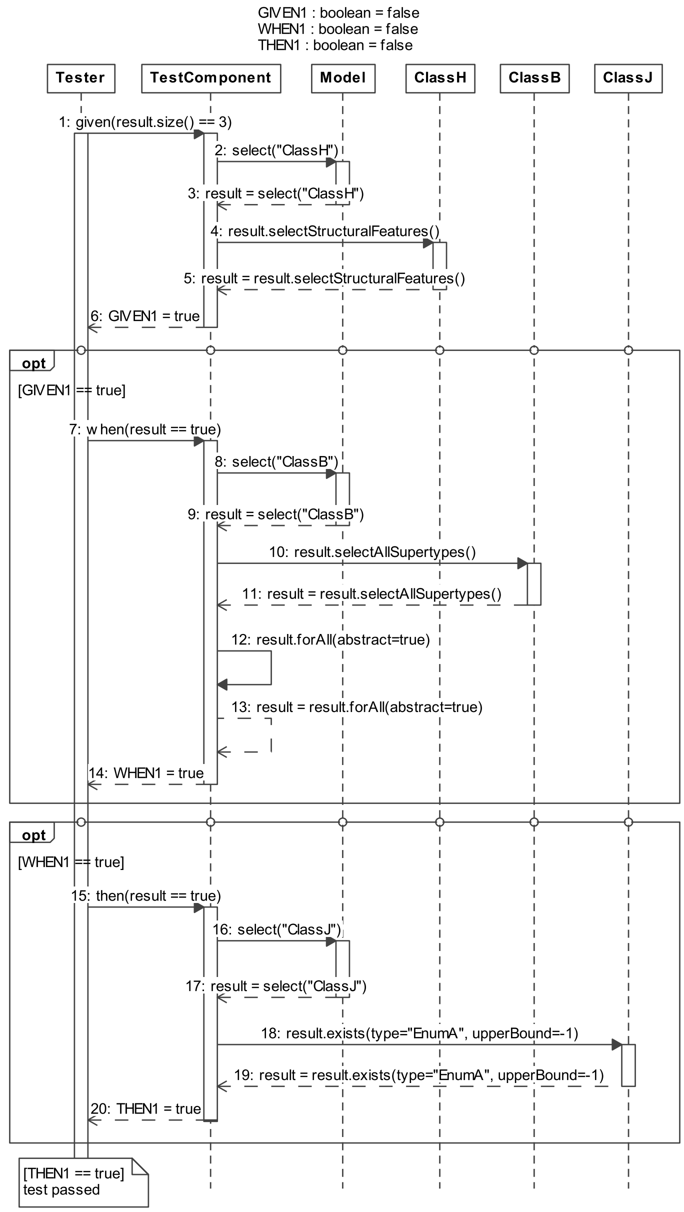


Fig. 1. Diagrammatic scenario notation (D-notation) exemplified.

management tasks (see Listing 2), called Epsilon [22]. In Epsilon, model testing is supported by the Epsilon unit testing framework (EUnit) which we extended to define scenario tests (see [4]). With this notation, scenario tests are imperatively defined via textual sequences of commands. Listing 2 shows our scenario-test example (again, semantically equivalent to Listing 1 and Fig. 1) including a precondition (line 6), an event (lines 8–10), and an expected outcome (line 11). Epsilon shares similarities with OCL [23]. For example, regarding operations on collections, a universal quantification can be expressed identically (forAll(); see line 9 in Listing 2).

```
1 @TestSuite
2 operation testSuite() {
3   @TestCase
4   operation testCase() {
5     @TestScenario
6     $pre Model!EClass.all().selectOne(x|x.name="
          ClassH").eStructuralFeatures.size() = 3
7     operation testScenario() {
```

[2]In this paper, we do not provide a background on Ecore models; for the remainder, it is sufficient to refer to generic object-oriented modeling constructs such as classes, inheritance relationships between classes, references, and attributes. For details on Ecore, see, e.g., Steinberg et al. [18].

```
        if (
          Model!EClass.all().selectOne(x|x.name="
               ClassB").closure(x|x.eSuperTypes).forAll
               (x|x.abstract=true)
        ) {
          assertTrue(Model!EClass.all().selectOne(x|x.
               name="ClassJ").eStructuralFeatures.
               exists(x|x.isTypeOf(EAttribute) and x.
               eType.name="EnumA" and x.upperBound=-1))
               ;
        } else {
          assertTrue(false);
        }
      }
    }
}
```

Listing 2. Fully-structured textual scenario notation (E-notation) exemplified.

III. Related Work

For this paper, we consider two categories of related work: (1) studies regarding the comprehension of the three notation types used in our experiment and (2) (empirical) evaluations of requirements-driven scenario notations.

Our first notation uses a text-based format that closely resembles natural-language expressions. Studies to measure text comprehension via problem solving tasks—comparable to the tasks performed in the experiment reported in this paper—have a long history (see, e.g., [24]). Emerging from this background, program comprehension experiments try to measure factors influencing the understandability of source code, such as scenario tests defined via EUnit. For instance, Kosar et al. [11] compare the accuracy and efficiency of domain-specific languages with general-purpose languages and Feigenspan et al. [12] evaluate the improved correctness in the development of software-product lines by introducing background-color-enhanced preprocessor directives. In both studies, a family of experiments was conducted in which participants had to perform comprehension-related tasks. As an example of a model-based notation, Mendling et al. [25] examine factors influencing how model viewers comprehend the syntactical information in process models (UML sequence diagrams fall in this category) in a four-part series of experiments. All related studies have in common that comprehension-influencing factors (i.e. dependent variables) are measured via response time and correctness of a solution.

As a representative for a non-empirical evaluation of requirements-driven scenario notations, Amyot and Eberlein [10] define evaluation criteria (e.g. decomposition and abstraction) and review fifteen scenario notations with respect to their suitability for the telecommunication domain. Ricca et al. [26] performed two controlled experiments to empirically evaluate a table-based notation for defining scenario tests. In particular, they investigate whether the notation contributes to the clarification of domain requirements and whether any additional comprehension effort is required. For requirements elicitation, scenarios are frequently documented via use cases. Gemino and Parker [27] explore the effectiveness of integrating use case models [20] with a set of textual use case descriptions (see, e.g., [7]). A study is conducted including comprehension, retention, and problem solving tasks measuring the level of understanding developed by the participants who either used textual uses cases or textual use cases with a supporting use case diagram. Likewise, Ottensooser et al. [28] report on an experiment for comparing the success of participants in interpreting business process descriptions for a graphical notation (BPMN) and for a textual notation (based on written use cases).

Our study complements existing contributions with a systematic comparison of three notations for scenario-based model testing.

IV. Experiment Planning

A. Goals and Variables

As mentioned above, the objective of our experiment is to compare three notations (see Section II) regarding their applicability for the scenario-based testing of models. For this task, we selected three notations from the body of previously screened literature on model-testing notations (see, e.g., [4], [17], [19], [20], [22]). The MUTs were defined using Ecore, a metamodeling language that provides the foundation of the Eclipse Modeling Framework (EMF; see, e.g., [18]).

Results obtained in related studies indicate that different scenario notations have an influence on the accuracy and effort of fulfilling model-testing tasks. For our experiment, the participants had to solve tasks which were identical except for the alternate usage of different scenario notations (our independent variable). Similar to related studies (see Section III), we use the response time and the correctness of the solution as dependent variables to measure the comprehension of a particular notation. A change in the independent variable is expected to have a measurable effect on one or both dependent variables.

To ensure that the scenario notation is the only independent variable, we had to control important confounding variables. In particular, we focused on maximizing the internal validity of our experimental setting to be able to draw sound conclusions from our results. Prior to the experiment, we asked all participants to fill in a questionnaire regarding their experience with topics that could have an influence on the tasks performed in the experiment (e.g., experience with different testing methods and testing notations relevant to the experiment). In addition, we collected the basic demographic profile of the participants. The process of building homogeneous groups was based on these information to keep the influence of confounding parameters on notation comprehension constant (see Section IV-B).

B. Participants of the Study

For the experiment, we contacted 30 software professionals (software engineers, testers, researchers) from different software companies, universities, and research institutions. For their participation, we offered a small reward in the form of a voucher for a well-known e-commerce company. 24 individuals agreed to contribute to the experiment, the remaining six did not respond to our request. We selected three participants to take part in a pilot study to pre-test the experimental setting (see also [12]). From the remaining 21, one participant could not take part due to illness. As we conducted the experiment at companies in different cities in Austria, we had to exclude the ill participant due to travel requirements. Hence, our data set consisted of 20 participants.

In our study we had five female and 15 male participants with an average age of 30.5±2.95[3] years. Each participant has obtained a computer-science degree or a related academic degree and currently works in the IT industry. The experience questionnaire consisted of five demographic and ten experience questions. To measure the level of confounding parameters, participants had to estimate their own experience level on a six-point Likert scale[4]. The ten questions collected evidence about the participants' experiences with regard to: scenario-based testing, Ecore modeling, natural-language testing, model-based testing, and test-case programming. We grouped and classified the questions according to a pre-defined schema (five groups, one to three questions per group). In a discussion with colleagues, we allocated weights to each question in a group. We classified general questions to be less important than specific ones. For example, experience in writing software in any programming language counted less than experience with the Epsilon language or model constraint/validation languages (such as OCL or EVL). The reason for this weighting was that we explicitly used dialects of the Epsilon language for validating models in our experiment. We assume, for instance, that general programming knowledge does not help as much in comprehending EUnit scenario tests as does knowledge in a language specifically designed for model management tasks (e.g. model navigation or element selection).

Our research design required to divide the participants into three groups (see Section IV-E). According to the participants' self estimations on the Likert scale, zero to six points ("no experience" to "very high experience") were allocated per question. Then, questions were grouped and weighted according to the schema explained above. The group building process homogeneously spread the participants so that the combined score of the weighted experience questions in each of the five question groups was similar. After the allocation, each group contained seven participants[5].

Via the experience questionnaire, we informed the participants that the collected data will be used solely for the process of conducting the experiment. Furthermore, we guaranteed that all published results will be anonymized and that personal data is neither made public nor given to any third-parties. In order to ask clarification questions after the experiment, we needed to know the participants' names as well as the name of their employer. As the collected data can be linked to the participants, we consulted the Center of Empirical Research Methods of WU Vienna which approved the ethical correctness of our design. The participants could opt-out of the experiment at any time, but no one did so.

C. Experimental Material

For the experiment, we prepared six different scenarios[6]. For each scenario, we provide an Ecore-based MUT, a scenario description specified in each of the three notations, and questions to be answered (see also the example task in the appendix). All materials were printed and fitted on one A4 sheet respectively. The Ecore models were created using the EMF project of Eclipse 4.2 [18]. The models were taken from real-world examples (e.g., the ATL code generator metamodel, the Ant metamodel), were obtained from the AtlanMod Metamodel Zoo[7], and adapted to fit the experimental setting (in terms of size and naming conventions). The adapted size of the six models were similar to each other and had been chosen to ensure a certain degree of complexity. In particular, it was intended that the participants should frequently consult the models during their tasks and should not be able to memorize their structure easily. Regarding model measures, on average, each model contained 23±1.17 classes, 8±1.03 of them declared abstract. These classes contained on average 44±3.21 structural features consisting of 18±3.51 attributes and 26±2.1 references. 12±1.67 of these references were defined as containment references. In a model, 19±3.14 inheritance relationships between classes were specified on average.

All scenario descriptions included the definition of two preconditions, two events, and two expected outcome specifications. The creation of all scenarios was supported by tools. The natural-language notation was defined via the infrastructure described in Hoisl et al. [19], the fully-structured notation via the infrastructure described in Sobernig et al. [4], and the diagrammatic notation by utilizing the UML sequence diagram editor of No Magic's MagicDraw 17.05. Each line of the two textual scenario descriptions were numbered on the left-hand side (see Listings 1 and 2 in Section II). The UML sequence diagram shows scenario interactions via messages passed between object instances. All call and return messages were sequentially numbered from top to bottom and these message numbers were displayed in front of each message expression (see Fig. 1 in Section II).

The experimental material also included language references for each of the three scenario notations as well as for Ecore models. The language references explained the syntactical format and the semantics of each notation and served as a documentation for the study participants (e.g., for the fully-structured notation, method calls and return values were explained).

For each scenario, five questions were asked with "yes", "no", and "don't know" answer choices. For each question, the possibility to insert line and message numbers was given. At the top of the answer sheet, participants had to insert their start time. An equivalent field was provided at the bottom for the end time. The task, the participants had to carry out, was explained on the sheet. At the bottom, space was reserved for comments.

To correctly answer the questions in the experiment the participants had to understand whether a scenario test passes or fails when elements in the corresponding model are changed. We classified the model changes referred to by the questions according to Wimmer et al. [30]. On the one hand, solutions to the questions required changes to existing modeling concepts: attributes (context, multiplicity, datatype), references (context, multiplicity, direction, containment), and inheritance

[3]The value after ± shows the standard deviation (σ) from the arithmetic mean ($\bar{x}$).

[4]Self estimation is expected to be a reliable way to judge experiences and is recommended, for example, in Siegmund et al. [29].

[5]Note that the group allocation was done before we learned that one of the individuals was not able to participate because of illness.

[6]All materials used in the experiment as well as collected data, calculations, and computed results can be downloaded from http://nm.wu.ac.at/modsec.

[7]http://www.emn.fr/z-info/atlanmod/index.php/Zoos

relationships (concreteness, depth, inheritance type). On the other hand, we also asked for creation and deletion of classes, attributes, references, and inheritance relationships (source-target-concept cardinality differences).

At the end of the experiment, an ex-post questionnaire had to be filled out. The participants should rate different criteria on a five point scale (1=worst, 5=best) for all three scenario-test notations. In particular, we asked the participants to evaluate the notations according to eight different criteria:

- *Clarity*: Is the notation understandable regarding its syntax and semantics?
- *Completeness*: Does the notation describe all necessary information to review the scenario test without consulting other information sources (e.g., the notation reference)?
- *Conciseness*: Are the resulting scenario tests precise (e.g., no ambiguities in the description, no misinterpretations when reviewing the description)?
- *Expressiveness*: Is the notation eligible to express model tests (e.g., tests on Ecore-based models)?
- *Generalizability*: Is the notation eligible to express tests other than model tests (e.g., code-unit tests)?
- *Practicality*: Does the notation require specific preparation (e.g., an extensive tutorial) and/or auxiliary materials (e.g., frequent use of notation reference)?
- *Scalability*: Does the notation support increasingly complex scenario tests well (e.g., increasing test and/or model size, increasing number of test conditions)?
- *General rating*: Overall, how did you like working with the notation?

Furthermore, participants should elaborate on their ratings (e.g. why did they favor a particular notation). Finally, we asked if they would like to add anything else (related to the scenario tests, improvements to the experiment etc.).

D. Tasks performed by the Participants

In total, each participant had to perform six tasks: one comprehension and one maintenance task per notation (i.e. for each of the three notations). All participants worked on the same six scenarios, but each group received a different sampling of scenario notations (according to the groups built via the experience questionnaire; see Section IV-B). Every task consisted of a scenario as described in Section IV-C: an Ecore-based MUT, a scenario description in one of the three notations, and five questions to be answered (see also the example task in the appendix). For every notation, the participants worked on the comprehension task first, followed by the maintenance task for the same notation.

Both tasks measure the participants' understanding of a scenario. The questions were designed in a way such that they could only be answered correctly if a subject fully understood the scenario. For the comprehension task, the participants were faced with a correct (i.e. passing) scenario test. We asked the participants to look at the scenario description and the corresponding MUT in order to answer five questions ("Does the scenario test fail if ..."). The second task was a maintenance task—here we provided an incorrect (i.e. failing) scenario test. The participants were asked to look at the scenario description and the corresponding MUT and find the line/message numbers which are responsible for the failure of the scenario test (for the maintenance task, an additional line/message number field was inserted in the answer sheet). After the initial error was found, the MUT had to be corrected in order for the scenario test to pass. To achieve this, the participants had to answer five questions whether changes in the model made the scenario test pass ("Does the scenario test pass if ..."). All questions were independent from each other (i.e. the participants had to answer each question on its own).

The process and the design of the tasks resemble the usual workflow of developing model-based software systems via scenarios. In a first step, requirements-based scenario descriptions are developed and executed to validate the MUT. When the requirements change, so do the scenario tests. Now, a discrepancy exists between the new scenario tests and the unchanged MUT. As a consequence, the scenario tests fail. To solve the problem, the MUT needs to be adapted in order to meet the new requirements. Again, the conformance of the (new) requirements is ensured via correct (i.e. passing) scenario tests.

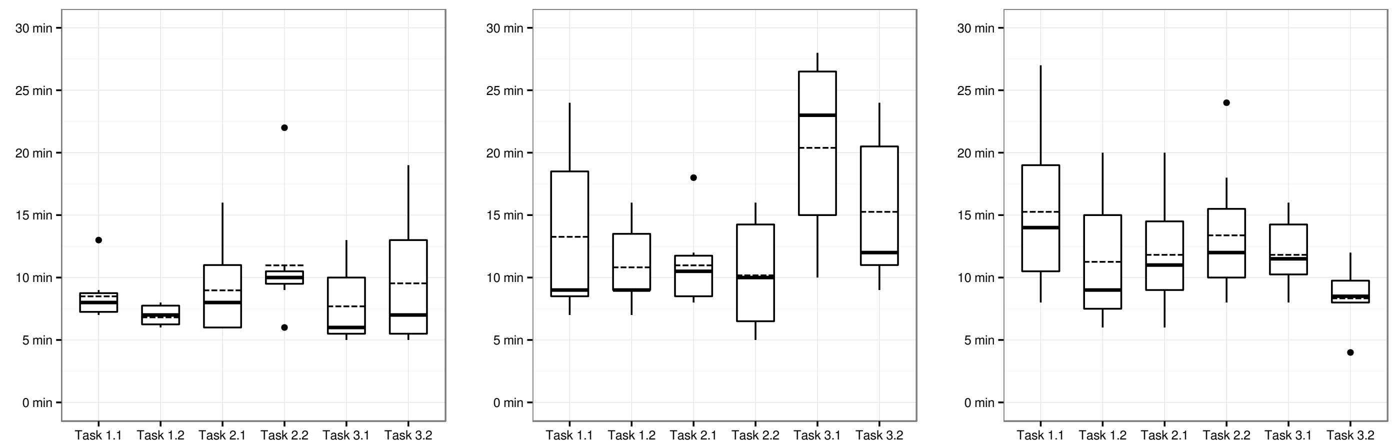
E. Design of the Experiment

Our controlled experiment has a between-subject design (see, e.g., [31]). In particular, every participant worked with all three notations (our independent variable) and on all six scenarios, but not with every notation alternative on every scenario (to exclude learning effects). The three homogeneous groups built via the experience questionnaire determined the notations which where given to each participant for a particular task. For instance, Group 1 used the natural-language notation for comprehension Task 1.1 and maintenance Task 1.2, Group 2 the diagrammatic notation, and Group 3 the fully-structured notation for the same tasks. For the remaining tasks (2.1, 2.2, 3.1, 3.2) the notations alternated between the groups so that every participant in each group worked with every notation and that every task had to be solved with every notation.

Every scenario was independently designed and did not reuse any elements from another scenario (i.e. different model, scenario description, and questions). To reduce potential learning effects, none of the participants worked on the same scenario twice. Weariness effects were eliminated by randomizing the order in which participants had to work on the different tasks. For further control, the questions per task were also randomly arranged for each participant individually. As domain knowledge has been shown beneficial for comprehension tasks (see, e.g., [32]), we factored out any domain bias by neutrally naming all model elements (e.g. ClassA, refB, attC). Hence, all participants had to use the same bottom-up approach (increasing the internal validity of our experiment), which means that the participants had to analyze the scenario descriptions statement by statement. Thus, the participants first had to understand a scenario statement and then gradually build up to an understanding of groups of statements until the whole scenario was understood completely (see, e.g., [12], [32]).

TABLE I. PARTICIPANTS' TIME SPENT AND ANSWERS GIVEN PER TASK AND PER NOTATION.

	Task 1.1			Task 1.2			Task 2.1			Task 2.2			Task 3.1			Task 3.2		
	N	D	E	N	D	E	N	D	E	N	D	E	N	D	E	N	D	E
Number of participants	6	7	7	6	7	7	7	6	7	7	6	7	7	7	6	7	7	6
Mean response time (in min.)	8.67	13.43	15.43	7.00	11.00	11.43	9.14	11.17	12.00	11.14	10.33	13.57	7.86	20.57	12.00	9.71	15.43	8.50
Standard deviation (in min.)	2.25	6.80	6.73	0.89	3.32	5.38	3.80	3.71	5.03	5.05	4.68	5.59	3.13	7.14	3.03	5.38	6.05	2.66
Min./max. response time (in min.)	7/13	7/24	8/27	6/8	7/16	6/20	6/16	8/18	6/20	6/22	5/16	8/24	5/13	10/28	8/16	5/19	9/24	4/12
Median response time (in min.)	8	9	14	7	9	9	8	10.5	11	10	10	12	6	23	11.5	7	12	8.5
Lower quartile (in min.)	7.25	8.5	10.5	6.25	9	7.5	6	8.5	9	9.5	6.5	10	5.5	15	10.25	5.5	11	8
Upper quartile (in min.)	8.75	18.5	19	7.75	13.5	15	11	11.75	14.5	10.5	14.25	15.5	10	26.5	14.25	13	20.5	9.75
Number of correct answers	29	29	32	27	21	28	35	26	30	27	23	22	29	28	26	26	27	19
Number of incorrect answers	1	5	1	3	5	7	0	4	5	8	7	8	5	4	4	7	8	11
Number of don't know answers	0	1	2	0	9	0	0	0	0	0	0	5	1	3	0	2	0	0
Percentage of correct answers	96.67	82.86	91.43	90.00	60.00	80.00	100.00	86.67	85.71	77.14	76.67	62.86	82.86	80.00	86.67	74.29	77.14	63.33



(a) N-notation (b) D-notation (c) E-notation

Fig. 2. Box plots of participants' time spent per notation and per task. Please note that the dashed horizontal line indicates the mean response time and the solid horizontal line the median response time.

Before the participants worked on the different tasks, an introductory presentation was given. The introduction explained the objectives of scenario-based testing, the structure of the scenario tests, and the syntax and semantics of Ecore models as well as of the three notations. To train the participants, three example tasks were presented—one for each notation (see Section II). Two questions were asked per task, the participants were motivated to answer the questions, and the solutions were presented. At the end of the introduction, we explained what the participants had to do for the experiment (see Section IV-D).

V. EXECUTION AND DEVIATIONS

To evaluate our experimental setting (time needed, difficulty of tasks etc.), we conducted a pilot study on March 4, 2014 (one participant) and on March 11, 2014 (two participants), respectively. In each round, we received feedback from the participants and revised the experimental material and the process. The experience questionnaires for building groups were submitted as editable PDFs to the participants on March 9, 2014 via e-mail. The participants had to return the questionnaires by e-mail until March 12, 2014 (the participants were assigned to groups on the same day). As we conducted the study at different locations and due to participants' time constraints, we had to execute the experiment multiple times (between March, 14 and March, 28 2014). The dates were arranged via e-mail communication. However, in every installment the experimental setting was identical (i.e. amount of introductory information presented, room with enough space for all participants and a projector for the introductory presentation etc.)—the experimental process did not differ as well. First, the participants were seated and told that they only need a pen to write and a clock for measuring time. After all participants had arrived, we gave the introductory presentation and performed the three warming up tasks which together took about 20 minutes. Then, each participant received a copy of their personal experimental material (according to the groups they belonged to and with randomized task and question orders; see Sections IV-D and IV-E) and were told that they had to work on the tasks in their predefined order. No supporting material or additional equipment was allowed. Subsequently, the participants worked on their tasks as explained above. Finally, each participant filled out the ex-post questionnaire and handed-in the material.

After the experiment, we had to contact four participants via e-mail and two personally because of ambiguous answers and were able to clarify all issues (e.g., problems reading the handwriting, unclear answer selections). After we evaluated the experiment's data, we informed the participants via e-mail that they could review their personal and, for comparison, the average results (correct answers, time needed etc.). Six participants responded and we transmitted the results to them.

Deviations from the plan occurred as one participant was ill and thus Group 1 consisted only of six participants instead of seven. Furthermore, it was hard to find suitable time-slots for the participants. Some participants took the experiment in

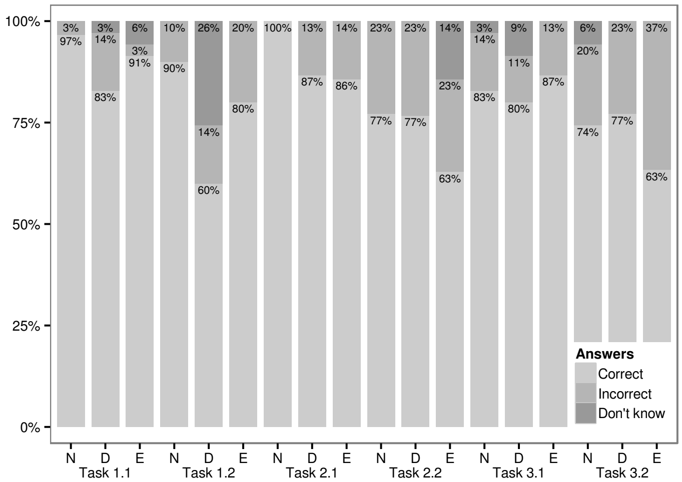


Fig. 3. Percentage of participants' correct, incorrect, and don't know answers per task and per notation.

the morning, some in the afternoon, and some in the evening. Although, the time of the day may influence the attention and concentration of participants, the different groups as well as the randomized tasks should compensate this deviation. The UML sequence diagrams needed the most space of all notations, but we fitted each of the diagrams on one A4 page (otherwise participants would have had to turn pages which would have required additional time). Five participants commented that the font sizes of the diagrammatic notation were too small and hard to read. This deviation could have an effect on the comprehension of the diagrammatic scenario notation.

VI. Analysis

Table I as well as Figs. 2 and 3 summarize our collected data in terms of the participants' time spent on the individual tasks and the answers given per task and per notation. The participants measured their time to complete a task in minutes (see Section IV-C). The correctness of a solution is derived from the ratio of correct answers to the sum of all answers (correct, incorrect, don't know answers). Table II shows the participants' average ratings per notation and per dimension from the ex-post questionnaire. The average least task-solving time and highest percentage of correct answers per task as well as the highest average rating per criterion are underlined in Tables I and II, respectively.

On average (arithmetic mean) the participants spent the following times for completing the six tasks: 53.52 min. for the natural-language notation (N), 81.93 min. for the diagrammatic notation (D), and 72.93 min. for the fully-structured notation (E). Furthermore, the percentage of correct answers over all tasks is 86.5% for the natural-language notation, 77% for the diagrammatic notation, and 78.5% for the fully-structured notation. On average, the participants spent 70.05±17.06 min. on the six tasks. As each task contained five questions, each participant had to answer 30 questions in total. On average, each participant answered 80.67% of the questions correctly (24.2±3.55 questions), 15.5% of the questions incorrectly (4.65±2.18 questions), and the participants did not know the answer to 3.83% of the questions (1.15±2.21 questions).

TABLE II. Ex-post average rating per notation (1–5, 5=best).

	N	D	E
Clarity	4.30±0.80	3.10±1.02	3.05±1.10
Completeness	4.40±0.82	3.45±0.89	3.15±1.04
Conciseness	4.00±0.79	3.35±1.14	3.80±1.15
Expressiveness	3.85±1.09	3.10±1.25	3.35±1.09
Generalizability	3.40±1.05	3.05±0.94	3.95±1.10
Practicality	4.65±0.59	3.00±0.97	2.50±1.00
Scalability	4.15±0.93	2.30±1.03	3.35±0.81
General	4.40±0.60	2.85±1.14	3.00±0.97

VII. Interpretation

Table I and Fig. 2 show that, except for Tasks 2.2 and 3.2, the participants required the least time to solve a task using the natural-language notation. Regarding these two tasks, the task-solving times of the natural-language notation were close to the times of the other notations: the diagrammatic notation in Task 2.2 (participants required on average 0.81 min. less time; an improvement by 7.27%) and the fully-structured notation in Task 3.2 (participants required on average 1.21 min. less time; an improvement by 12.46%). By using the natural-language notation, the participants required on average 28.4 min. less time to finish all six tasks than with the diagrammatic notation (an improvement by 34.7%) and 19.4 min. less time than with the fully-structured notation (an improvement by 26.6%). Using the fully-structured notation, it took the participants 9 min. less time than working on the tasks in diagrammatic notation (an improvement by 11%).

For the four Tasks 1.1–2.2, the participants reached most correct answers by using the natural-language notation (see Table I and Fig. 3). The fully-structured and the diagrammatic notations are comparable regarding answer correctness (78.5% and 77%). There was one participant who had a very low score of only 14 correct answers (46.7%). When removing this outlier from the data set, the natural-language notation accounts for the highest amount of correct answers in all tasks, including Tasks 3.1 and 3.2. The respective notation of the least response time remains unchanged, both per task and overall.

In total, participants using the natural-language notation required the least time to solve the tasks and answered the most questions correctly. These results in favor of the natural-language notation are also supported by the participants ex-post rating (see Table II). Except for its *generalizability*, the participants ranked the natural-language notation first in all remaining dimensions.

Via the ratings and comments in the ex-post questionnaire (see Table II), the study participants criticized the poor *scalability* of the diagrammatic notation (quote: "just overhead", "bloated"). Both the structure of the diagrammatic notation and the general syntax of UML sequence diagrams need comparably more space than the other two notations. Thus, for complex scenarios, the diagrammatic notation might incur the risk of comparatively large scenario-test definitions which, thereby, become hard to comprehend. Furthermore, the participants stated that the fully-structured notation requires the most specific preparation and/or auxiliary materials to be understood. Its *practicality* was therefore rated lowest (see Table II). However, despite its lower ranked practicality, the task-solving time was improved by 11% and the percentage of correct answers by 1.5% when using the fully-structured rather than the diagrammatic notation.

VIII. THREATS TO VALIDITY

Some threats to internal validity are caused by the deviations that occurred (see Section V). Furthermore, our group building process focused on a homogeneous distribution of participants with the same experience level. We tried to equally distribute demographic characteristics as well, and managed to do so for the participants' education and their age. Nevertheless, the data set did not allow for a homogeneous distribution of female participants (n=5) which is also a threat to internal validity. Our focus on internal validity limits external validity. The participants were all software professionals working with a selection of notations on a set of neutralized Ecore models to solve specific scenario tasks. In order to generalize our results and to improve external validity, we would need to repeatedly conduct the experiment with different participants from diverse professional backgrounds, different notation alternatives, different scenario descriptions and so forth.

IX. CONCLUSION AND FUTURE WORK

In this paper, we presented an experiment to evaluate three notations for their applicability to describe scenario-based model tests. In particular, the experiment compared the comprehensibility of three scenario-based notations and showed that the choice of a specific notation has an effect on the productivity when testing models for requirements conformance. The results of our experiment indicate that a natural-language-based approach for scenario-based model tests is recommended, as it required the least task-solving time and was the most accurate alternative as well as most favored by the participants who worked with it.

As future work, we will repeat the experiment with participants from a different professional background to be able to draw more general conclusions. For the replication study, we will revise the experimental setting and materials based on the feedback collected so far (e.g., conducting the experiment in one session, increasing the font size of the diagrammatic notation). Furthermore, we will run extended quantitative and qualitative analyses. For example, we will evaluate the line/message numbers recorded for the failing scenario tests by the participants.

APPENDIX
EXPERIMENT TASK 2.1

To illustrate the actual tasks performed during the experiment, we present the comprehension Task 2.1 of our experiment. For completing a task, we provided each participant with an Ecore-based MUT (for Task 2.1 the MUT is shown in Fig. 4) and a scenario-test definition in one of the three notations. For Task 2.1, the scenario was either specified as in Listing 3, as in Listing 4, or as in Fig. 5. Five questions were to be answered on each task, the ones about Task 2.1 are shown later in this section.

The MUT of Task 2.1 (see Fig. 4) is based on the Ant metamodel and was adapted to fit the experimental setting (in terms of size and naming conventions). The MUT of Task 2.1 contains 24 classes, 8 of them declared abstract. These classes contain 46 structural features consisting of 22 attributes and 24 references. 15 of theses references are defined as containment references. In the MUT of Task 2.1, 19 inheritance relationships between classes are specified.

The scenario descriptions of Task 2.1 for each notation are shown in Listing 3, Listing 4, and Fig. 5, respectively. Please note that all three scenario-test definitions are semantically equivalent; i.e. all three scenario representations per task describe the same scenario-based model test. Each participant of the experiment was provided with exactly one of these notations, depending on their group affiliation.

```
Scenario:
  Given "that it is possible to navigate from ClassO to ClassA via containment references refQ and refJ"
  And "that ClassY includes ClassF, ClassP, and ClassE in its hierarchy of supertypes"
  When "ClassD, ClassR, and ClassV are not abstract"
  And "exactly two attributes of ClassA are of type EString with a multiplicity of 0..1"
  Then "ClassY shall own exactly six structural features"
  And "ClassG shall have any kind of reference pointing to ClassR named refG"
```

Listing 3. Natural-language scenario notation of Task 2.1.

```
@TestSuite
operation testSuite() {
  @TestCase
  operation testCase() {
    @TestScenario
    $pre Model!EClass.all().selectOne(x|x.name="ClassO").eStructuralFeatures.selectOne(x|x.name="refQ").eType.eStructuralFeatures.selectOne(x|x.name="refJ").eType.name="ClassA"
    $pre Model!EClass.all().selectOne(x|x.name="ClassY").closure(x|x.eSuperTypes).includesAll(Model!EClass.all().select(x|x.name="ClassF" or x.name="ClassP" or x.name="ClassE"))
    operation testScenario() {
      if (Model!EClass.all().select(x|x.name="ClassD" or x.name="ClassR" or x.name="ClassV").forAll(x|x.abstract=false)
        and
        Model!EClass.all().selectOne(x|x.name="ClassA").eStructuralFeatures.select(x|x.isTypeOf(EAttribute) and x.eType.name="EString" and x.lowerBound=0 and x.upperBound=1).size() = 2
      ) {
        assertTrue(Model!EClass.all().selectOne(x|x.name="ClassY").eStructuralFeatures.size() = 6);
        assertTrue(Model!EClass.all().selectOne(x|x.name="ClassG").eStructuralFeatures.selectOne(x|x.name="refG").eType.name = "ClassR");
      } else {
        assertTrue(false);
      }
    }
  }
}
```

Listing 4. Fully-structured scenario notation of Task 2.1.

For the comprehension Task 2.1, the participants had to answer the following five questions with either "yes", "no", or "don't know" (please note that the questions were randomly ordered for each participant):

1) Does the scenario test fail if ClassF, ClassT, and ClassD are declared abstract?
2) Does the scenario test fail if the reference refG of ClassG is not a containment reference and has a multiplicity of 1..1?

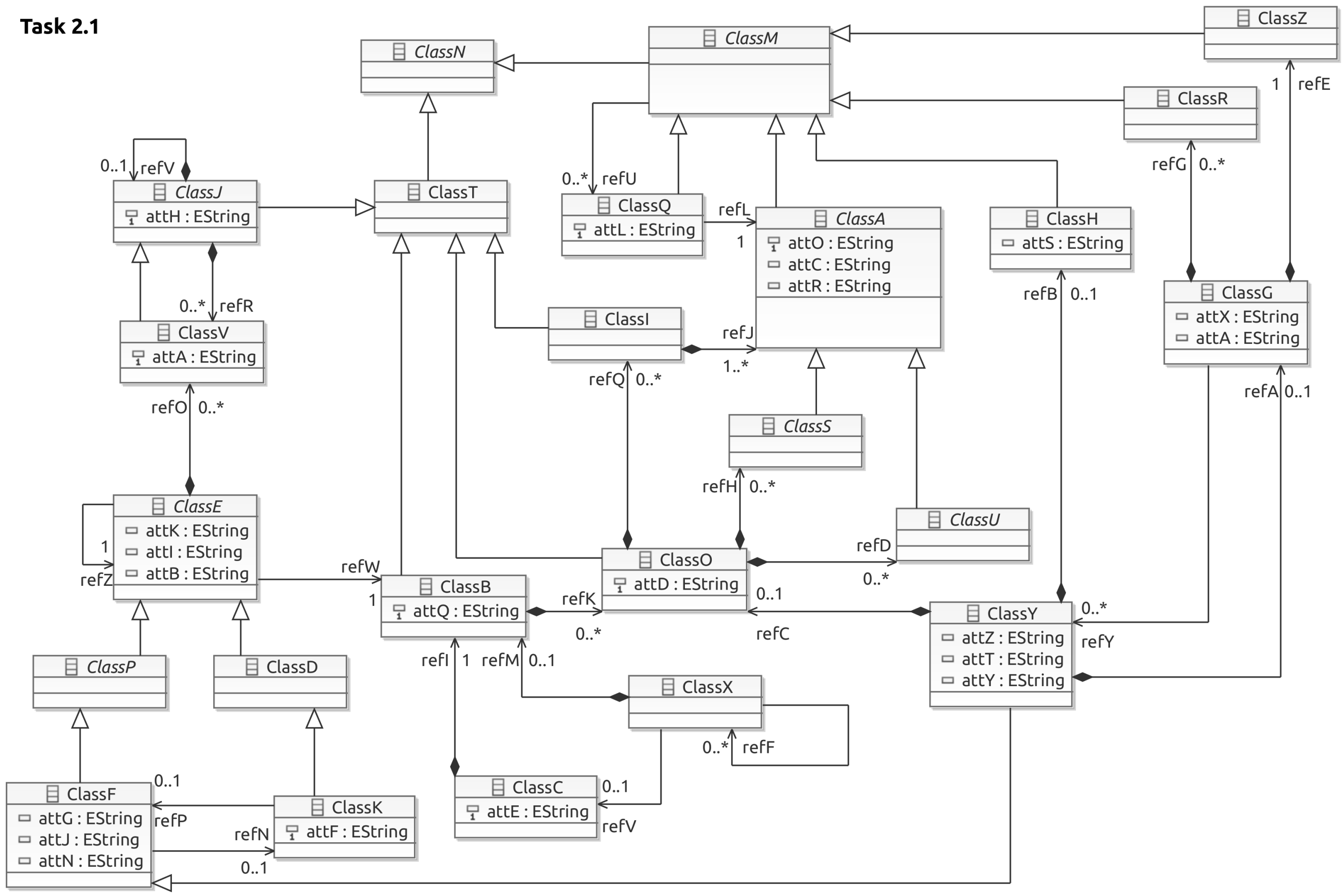


Fig. 4. Ecore-based MUT of Task 2.1.

3) Does the scenario test fail if in ClassI the navigability of reference refJ is inverted?
4) Does the scenario test fail if in ClassA the lower bound of attribute attR is changed to 1?
5) Does the scenario test fail if the reference refB of ClassY is inverted?

If a scenario test was deemed failed (i.e. a question was answered with "yes"), the participant had to mention the line number(s) (in case of the natural-language or fully-structured notation) or message number(s) (in case of the diagrammatic notation) in the scenario description considered responsible for failing the test. Furthermore, the participants had to fill in their start and end time per task (in minutes). At the end of the answer sheet, space was reserved for comments.

ACKNOWLEDGMENT

Thanks are due to Janet Siegmund for her advice on planning the experiment. This work has partly been funded by the Austrian Research Promotion Agency (FFG) of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT) through the Competence Centers for Excellent Technologies (COMET K1) initiative and the FIT-IT program.

REFERENCES

[1] D. Schmidt, "Guest editor's introduction: Model-driven engineering," *IEEE Computer*, vol. 39, no. 2, pp. 25–31, Feb 2006.

[2] B. Selic, "The pragmatics of model-driven development," *IEEE Softw.*, vol. 20, no. 5, pp. 19–25, Sept 2003.

[3] P. Brosch, U. Egly, S. Gabmeyer, G. Kappel, M. Seidl, H. Tompits, M. Widl, and M. Wimmer, "Towards scenario-based testing of UML diagrams," in *Tests and Proofs*, ser. LNCS. Springer, 2012, vol. 7305, pp. 149–155.

[4] S. Sobernig, B. Hoisl, and M. Strembeck, "Requirements-driven testing of domain-specific core language models using scenarios," in *Proc. 13th Int. Conf. Quality Softw.* IEEE Computer Society, 2013, pp. 163–172.

[5] I. Sommerville, *Software Engineering*, 9th ed. Addison-Wesley, 2010.

[6] A. Sutcliffe, *User-Centred Requirements Engineering*. Springer, 2002.

[7] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley, 2001.

[8] C. Nebut, F. Fleurey, Y. Le-Traon, and J.-M. Jézéquel, "Automatic test generation: A use case driven approach," *IEEE Trans. Softw. Eng.*, vol. 32, no. 3, pp. 140–155, 2006.

[9] A. Ulrich, E.-H. Alikacem, H. Hallal, and S. Boroday, "From scenarios to test implementations via Promela," in *Test. Softw. Syst.*, ser. LNCS. Springer, 2010, vol. 6435, pp. 236–249.

[10] D. Amyot and A. Eberlein, "An evaluation of scenario notations and construction approaches for telecommunication systems development," *Telecommun. Syst.*, vol. 24, no. 1, pp. 61–94, 2003.

[11] T. Kosar, M. Mernik, and J. Carver, "Program comprehension of domain-specific and general-purpose languages: Comparison using a family of experiments," *Emp. Softw. Eng.*, vol. 17, no. 3, pp. 276–304, 2012.

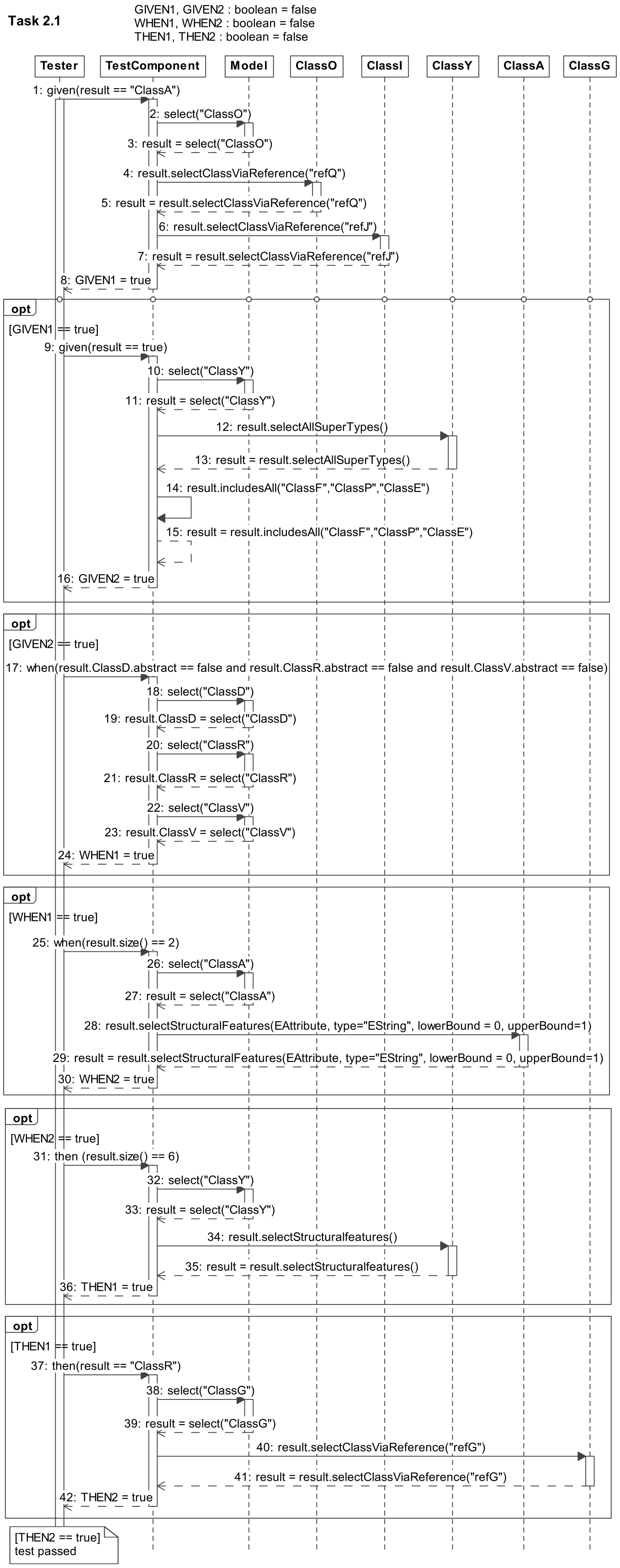


Fig. 5. Diagrammatic scenario notation of Task 2.1.

[12] J. Feigenspan, C. Kästner, S. Apel, J. Liebig, M. Schulze, R. Dachselt, M. Papendieck, T. Leich, and G. Saake, "Do background colors improve program comprehension in the #ifdef hell?" *Emp. Softw. Eng.*, vol. 18, no. 4, pp. 699–745, 2013.

[13] J. Koenemann and S. P. Robertson, "Expert problem solving strategies for program comprehension," in *Proc. Conf. Human Factors Comp. Sys.* ACM, 1991, pp. 125–130.

[14] A. Dunsmore and M. Roper, "A comparative evaluation of program comprehension measures," University of Strathclyde, Tech. Rep., 2000.

[15] A. Jedlitschka, M. Ciolkowski, and D. Pfahl, "Reporting experiments in software engineering," in *Guide Advanced Emp. Softw. Eng.* Springer, 2008, pp. 201–228.

[16] B. Kitchenham, H. Al-Khilidar, M. Babar, M. Berry, K. Cox, J. Keung, F. Kurniawati, M. Staples, H. Zhang, and L. Zhu, "Evaluating guidelines for reporting empirical software engineering studies," *Emp. Softw. Eng.*, vol. 13, no. 1, pp. 97–121, 2008.

[17] M. Wynne and A. Hellesøy, *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. Pragmatic Programmers, 2012.

[18] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*, 2nd ed. Addison-Wesley, 2008.

[19] B. Hoisl, S. Sobernig, and M. Strembeck, "Natural-language scenario descriptions for testing core language models of domain-specific languages," in *Proc. 2nd Int. Conf. Model-Driven Eng. Softw. Dev.* SciTePress, 2014, pp. 356–367.

[20] Object Management Group, "OMG unified modeling language (OMG UML), superstructure," available at: http://www.omg.org/spec/UML, Aug. 2011, v2.4.1, formal/2011-08-06.

[21] S. K. Swain, D. P. Mohapatra, and R. Mall, "Test case generation based on use case and sequence diagram," *Int. J. Softw. Eng.*, vol. 3, no. 2, pp. 21–52, July 2010.

[22] D. Kolovos, L. Rose, A. García-Domínguez, and R. Paige, "The Epsilon book," Available at: http://www.eclipse.org/epsilon/doc/book/, 2014.

[23] Object Management Group, "Object constraint language," available at: http://www.omg.org/spec/OCL, Feb. 2014, v2.4, formal/2014-02-03.

[24] W. Kintsch, "Learning from text," *Cognition Instruct.*, vol. 3, no. 2, pp. 87–108, 1986.

[25] J. Mendling, M. Strembeck, and J. Recker, "Factors of process model comprehension—findings from a series of experiments," *Decision Support Syst.*, vol. 53, no. 1, pp. 195–206, 2012.

[26] F. Ricca, M. Torchiano, M. D. Penta, M. Ceccato, and P. Tonella, "Using acceptance tests as a support for clarifying requirements: A series of experiments," *Inform. Softw. Tech.*, vol. 51, no. 2, pp. 270–283, 2009.

[27] A. Gemino and D. Parker, "Use case diagrams in support of use case modeling: Deriving understanding from the picture," *J. Database Mgmt.*, vol. 20, no. 1, pp. 1–24, 2009.

[28] A. Ottensooser, A. Fekete, H. A. Reijers, J. Mendling, and C. Menictas, "Making sense of business process descriptions: An experimental comparison of graphical and textual notations," *J. Syst. Softw.*, vol. 85, no. 3, pp. 596–606, 2012.

[29] J. Siegmund, C. Kästner, J. Liebig, S. Apel, and S. Hanenberg, "Measuring and modeling programming experience," *Emp. Softw. Eng.*, 2013.

[30] M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schoenboeck, and W. Schwinger, "Towards an expressivity benchmark for mappings based on a systematic classification of heterogeneities," in *Proc. 1st Int. Workshop Model-Driven Interoperability*. ACM, 2010, pp. 32–41.

[31] A. Dix, J. Finlay, G. Abowd, and R. Beale, *Human-Computer Interaction*, 3rd ed. Pearson Education, 2004.

[32] T. M. Shaft and I. Vessey, "The relevance of application domain knowledge: The case of computer program comprehension," *Inform. Syst. Res.*, vol. 6, no. 3, pp. 286–299, 1995.