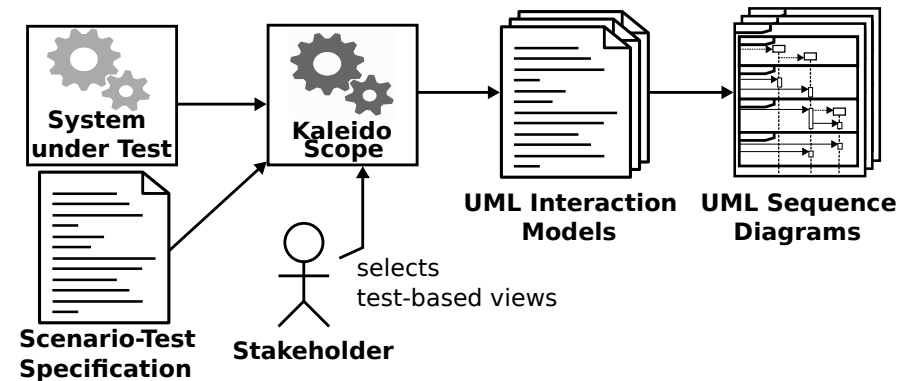


# **An Approach for the Semi-automated Derivation of UML Interaction Models from Scenario-based Runtime Tests**

by Thorsten Haendler, Stefan Sobernig, and Mark Strembeck

Institute for Information Systems and New Media  
Vienna University of Economics and Business (WU), Austria  
[thorsten.haendler@wu.ac.at](mailto:thorsten.haendler@wu.ac.at)

- Derivation of behavior documentation (in terms of UML interaction models) from runtime tests
- Runtime tests reflect exemplary and *intended* behavior of the system under test (SUT).
- Characteristic structure of scenario-based tests provides an option space for configuring views:  
→ resulting in partial models human-tailored for a specific task.



**Fig. 1.** Deriving tailored models from scenario-based runtime tests.

# Structure of the Talk

- Motivation
- Conceptual Overview
- Example
  - System under test (SUT)
  - Scenario-test specification
  - Test-execution trace model
  - Mappings between test and UML
  - Tailored sequence diagrams
- Option space for tailoring models (scenario-test viewpoint)
- Prototype implementation *KaleidoScope*
- Future Work
- Summary

- Behavior documentation, esp. by using graphical models, facilitates communication about and understanding of software systems.
- Manual creation (and maintenance) is an error-prone and time-consuming task (Rost et al., 2013).
- Multiple approaches exist for reverse-engineering behavioral models automatically from system execution (e.g., UML sequence diagrams: Briand et al., 2003).  
→ **Problem of model-size explosion** (e.g., Sharp and Rountev, 2005; Bennett et al., 2008)
- Common counter measures are, e.g., techniques of *sampling* and *hiding* of model elements (e.g., Hamou-Lhadj and Lethbridge, 2004; Bennett et al., 2008).

## **In this approach:**

- We leverage the characteristics of scenario-based runtime tests for deriving tailored interaction models (scenario-test viewpoint)
  - we provide configuration options for the system's stakeholders to fit the models to maintenance tasks (tailoring)
  - test-to-system traceability (behavioral slices)
- scenarios (e.g., Jacobson, 1992): structured stories describing sequences of actions and events
- scenario-based testing (e.g., Ryser and Glinz, 1999): automated execution and verification of scenarios that describe interactions with or within a software system

- Model-driven approach**
  - transformation based on mappings between the metamodels of scenario-based testing and UML2
- Semi-automated derivation**
  - manual selection of views conforming to a scenario-test viewpoint



# Example 1/5

## A) System under Test (SUT)

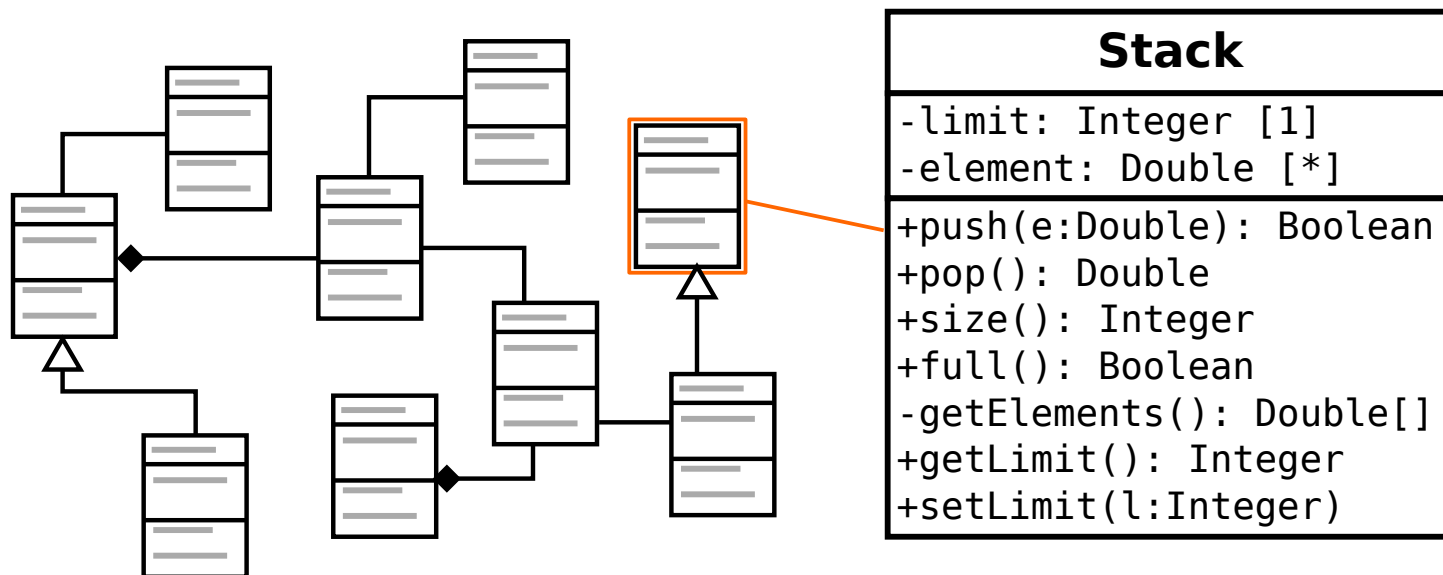



Fig. 3. Exemplary object-oriented system under test (SUT).

# Example 2/5

## B) Scenario-Test Specification

### Exemplary test scenario **pushOnFullStack**



```
1 # It is provided in the setup script of the owning test
   case pushElement that an instance of Stack exists
   containing the two elements 3.5 and 4.3
2 set fs [::STORM::TestScenario new -name pushOnFullStack
   -testcase pushElement]
3 $fs expected_result set 0
4 $fs setup_script set {
5   [::Stack info instances] limit set 2
6 }
7 $fs preconditions set {
8   {expr {[::Stack info instances] size} == 2}}
9   {expr {[::Stack info instances] limit get} == 2}}
10 }
11 $fs test_body set {
12   [::Stack info instances] push 1.4
13 }
14 $fs postconditions set {
15   {expr {[::Stack info instances] size} == 2}}
16 }
```

Fig. 4a. Excerpt from test specification.

- 1 **Given:** 'that a specific instance of Stack contains elements of the size of 2 and has a limit of 2'
- 2 **When:** 'an element is pushed on the instance of Stack'
- 3 **Then:** 'the push operation fails and the size of elements is still 2'

Fig. 4b. Natural-language description.



# Example 3/5

## C) Test-Execution Trace Model

```
<scenario name="pushOnFullStack">
  <block name="setup" call="//@trace/@call.11 //@trace/@call.12 " />
  <block name="checkPreConditions" call="//@trace/@call.13 //@trace/@call.15
//@trace/@call.14 //@trace/@call.16 //@trace/@call.17 " />
  <block name="test" call="//@trace/@call.18 //@trace/@call.21 //@trace
/@call.23 //@trace/@call.22 //@trace/@call.20 //@trace/@call.19 " />
  <block name="checkPostConditions" call="//@trace/@call.24 //@trace/@call.26
//@trace/@call.25 " />
</scenario>

<call source="//@instance.3" target="//@instance.2" caller="//@feature.1"
callee="//@feature.6" name="push" definedBySTF="true">
  <argument name="1.4" />
  <returnValue name="0" />
</call>
```

instance of

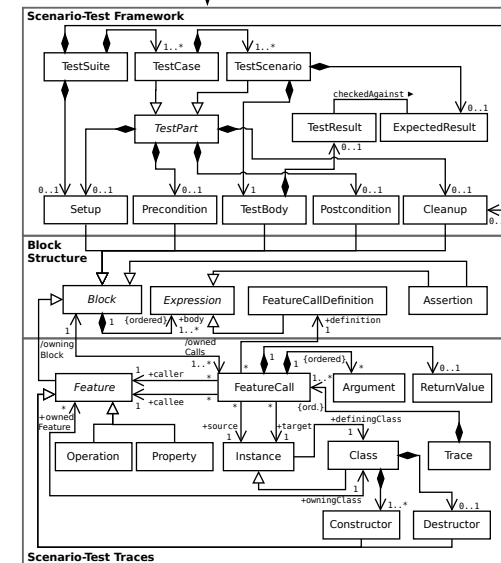
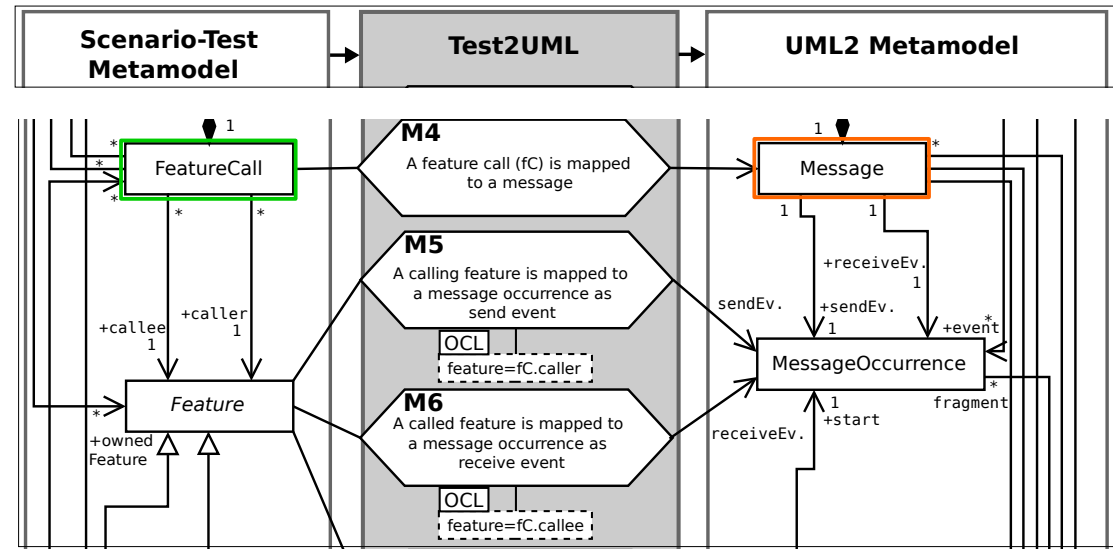


Fig. 5b. Test-execution trace meta-model.

# Example 4/5

## D) Mappings between Test and UML

- transML diagram (Guerra et al., 2012) technology- & language-independent and UML compatible
- in total, 18 mappings (12 for traces, 6 viewpoint mappings)
- mappings refined by OCL constraints



```
context M4 inv:
  message.name=featureCall.name and
  message.sendEvent.oclIsTypeOf (
    MessageOccurrenceSpecification) and
  message.sendEvent.name=featureCall.caller.name
```

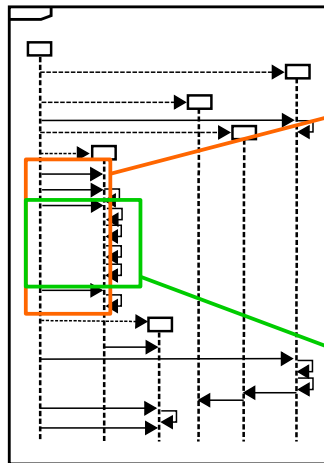
Fig. 6. Excerpt from transML mappings with excerpt from OCL consistency-constraints based on mapping M4.

# Example 5/5

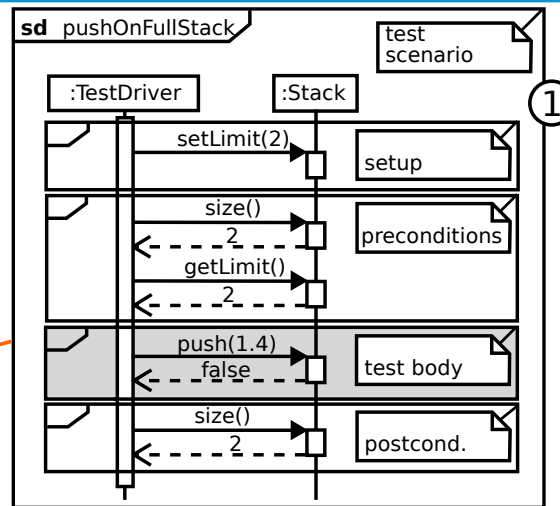
## E) Resulting Tailored UML Sequence Diagrams

Fig. 7. Exemplary stakeholders/tasks and derived diagrams.

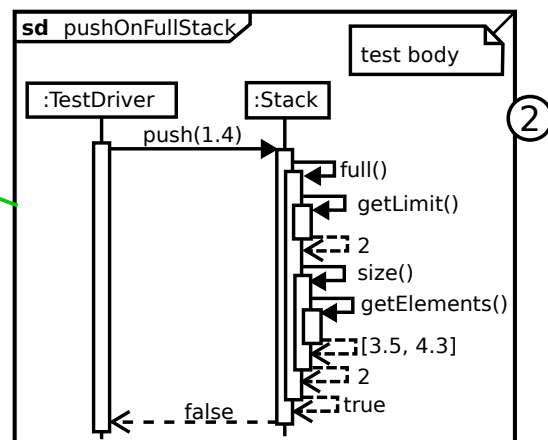
test engineer /  
test review



system developer /  
after code modification



- calls running from STF to SUT
- test scenario pushOnFullStack



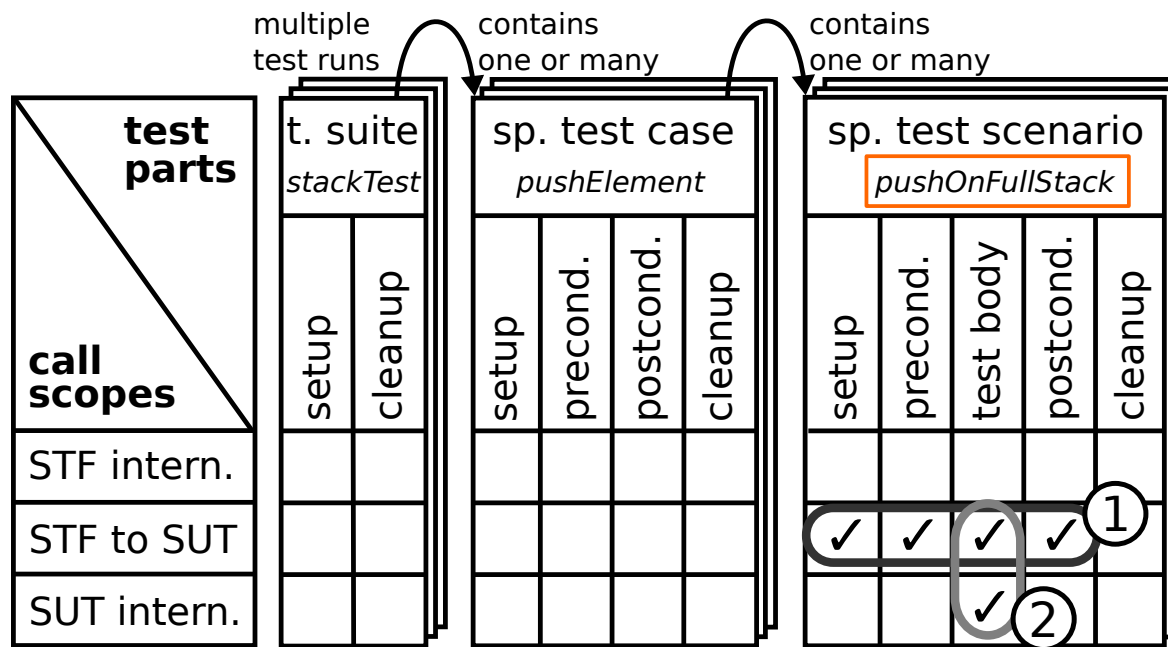
- calls running from STF to SUT and SUT internal calls
- test body of test scenario pushOnFullStack

# Scenario-Test Viewpoint

## - Structure of Scenario-based Tests

- Characteristics of scenario-based testing:
  - **Scenario-test parts** (test suite, test case, test scenario, as well as assertion and exercise blocks)
  - **Feature-call scopes** (calls running from STF to SUT, calls internal to the SUT, and calls internal to the STF)
- A view stipulates elements to be selected or not  
→ resulting in partial interaction models human-tailorable for a specific task.
- All views conform to a viewpoint (see, e.g., Clements et al., 2011).

# Option Space for configuring views

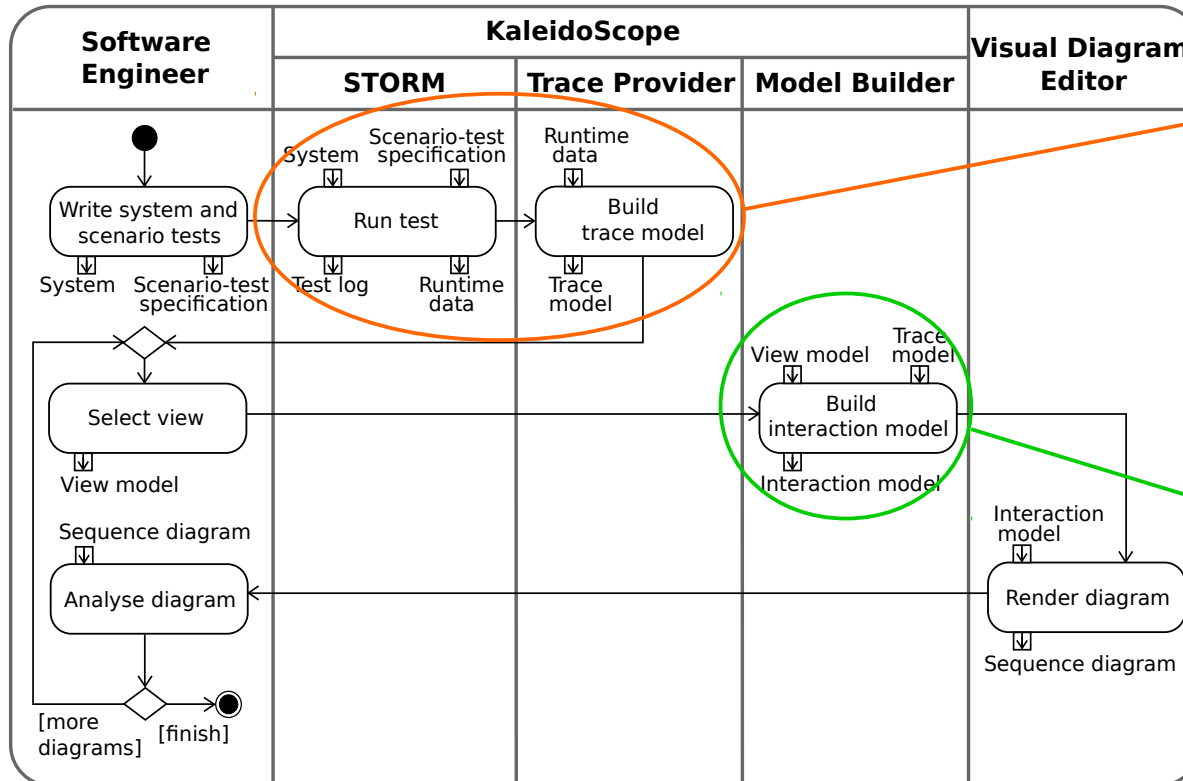


For the resulting derived diagrams corresponding to the configurations 1 and 2, see slide 11.

**Fig. 8.** Option space for configuring different views conforming to a scenario-test viewpoint.

# KaleidoScope 1/3

## - Derivation Process



**Instrumenting** the test framework and extracting execution-trace data

**Transforming** view and trace models to interaction models

**Fig. 9.** Process of deriving tailorable UML-based software-behavior documentation with KaleidoScope.

Available for download from our website <http://nm.wu.ac.at/nm/haendler>

# *KaleidoScope 2/3*

## - Used Technologies A

- **Test Framework:** *Scenario-based Testing of Object-Oriented Runtime-Models (STORM)* (Strembeck, 2011)
- **Instrumentation:** NX/Tcl (object-oriented extension of Tcl) provides introspection techniques (see Neumann and Sobernig, 2015):
  - message interceptors (Mixin and Filter)
  - extraction of trace data by using  
callstack introspection (e.g., `nx::current`) and  
structural introspection (e.g., `info method`)

# *KaleidoScope 3/3*

## - Used Technologies B

- Trace and view models are **transformed** to UML interaction models by using *Query View Transformation operational* (QVTo) mappings (in total 24 mapping actions).
- All models are stored and processed in their Ecore/XMI representation, which makes it possible to import the models via XMI-compliant diagram editors (e.g., *Eclipse Papyrus*).



- Derivation of **other model types**
  - structure models (e.g., UML class models)
  - component-based architecture documentation (e.g., inter-component interactions)
- **Extension of prototype**
  - integration of other filtering/abstraction techniques (e.g., constructor hiding, identification of loops)
  - instrumenting other testing frameworks (e.g., JBehave using AspectJ)
- Application in **large-scale projects**
  - usability for stakeholders/tasks

- Model-driven and semi-automated derivation of behavior documentation (in terms of UML2 interaction models)
- Scenario-test viewpoint (different views on the test-execution trace available for configuration)
- Prototype implementation *KaleidoScope* (proof of concept)

*Thank you for your attention!*

Q&A