Inferring Visual Contracts from Java Programs

Abdullah Alshanqiti[1], Reiko Heckel[2], Timo Kehrer[3]

Abstract: In this work, we report about recent research results on "Inferring Visual Contracts from Java Programs", published in [1]. In this paper, we propose a dynamic approach to reverse engineering visual contracts from Java programs based on tracing the execution of Java operations. The resulting contracts give an accurate description of the observed object transformations, their effects and preconditions in terms of object structures, parameter and attribute values, and their generalised specification by universally quantified (multi) objects, patterns, and invariants. We explore potential uses in our evaluation, including in program understanding and testing, and we report on experimental results w.r.t. completeness (recall) and correctness (precision) of extracted contracts.

Keywords: Visual contracts, graph transformation, model extraction, dynamic analysis, reverse engineering, specification mining

Summary

Visual Contracts provide a precise high-level specification of the object graph transformations caused by invocations of operations on a component or service. They link static models (e.g., class diagrams describing object structures) and behavioural models (e.g., state machines specifying the order operations are invoked in) by capturing the preconditions and effects of operations on a system's objects.

Visual contracts differ from contracts embedded with code, such as JML in Java or Contracts in Eiffel, as well as from model-level contracts in OCL. They are *visual*, using UML notation to model complex patterns and transformations intuitively and concisely; *abstract*, providing a specification of object transformations at a high level of granularity to aid readability and scalability; *deep*, capturing the transformation of internal object structures besides input / output behaviour; and *executable*, based on graph transformation they support model-based oracle and test case generation, run-time monitoring, service specification and matching, state space analysis and verification. However, the detailed specification of internal data states and transformations, referred to as deep behavioural modelling, is an error-prone activity.

In this paper, we report on a line of research whose key idea is a dynamic approach to reverse engineering visual contracts from sequential Java programs based on tracing the execution of Java operations. The resulting contracts give accurate descriptions of the observed object

[1] Department of Computer Sciences, University of Leicester. a.m.alshanqiti@gmail.com
[2] Department of Computer Sciences, University of Leicester. reiko@mcs.le.ac.uk
[3] Institut für Informatik, Humboldt-Universität zu Berlin. timo.kehrer@informatik.hu-berlin.de

doi:10.18420/se2019-11

transformations, their effects and preconditions in terms of object structures, parameter and attribute values, and allow generalisation by *multi objects and patterns* and general invariants. The restriction to sequential Java is due to the need to associate each access to a unique operation invocation.

Given a Java application, the process starts by selecting the classes and operations within the scope of extraction and providing test cases for the relevant operations. We proceed by (A) observing the behaviour under these tests using AspectJ instrumentation and synthesising rule instances as pre/post snapshot graphs of individual invocations; (B) combining the instances into higher-level rules by abstracting from non-essential context; (C) generalising further by introducing multi objects and patterns; (D) deriving logical constraints and assignments over attribute and parameter values; and (E) identifying universally shared conditions and structures as invariants captured separately.

First solutions to variants of (A) and (B) were developed in our earlier work and have been summarized, consolidated and extended in [2, 3] which also presents a prototype tool and explores its potential uses in program understanding, testing and debugging. In this paper, we raise further the level of abstraction by supporting, in addition to previous work, the inference of multi patterns in (C), attribute assignments in (D) and universal context in (E), and exploiting subtyping throughout the process. We support the use of visual contracts as *deep oracles* by exporting contracts to the model transformation tool Henshin [4] and controlling the (otherwise non-deterministic) model execution by comparing the effect of each test invocation with the rules in the operation's contracts. This technology is used in a new evaluation of completeness (recall) and correctness (precision) of extracted contracts. Finally, we report on improved tool support, and we discuss two further application scenarios for our approach and tool in the field of model-based software engineering, namely the use of visual contracts for model-based (or visual) debugging as well as the automated learning of complex model editing operations by examples [5].

References

[1] Abdullah Alshanqiti, Reiko Heckel, and Timo Kehrer. "Inferring visual contracts from Java programs". In: *Automated Software Engineering* (2018), pp. 1–40.

[2] Abdullah Alshanqiti and Reiko Heckel. "Extracting Visual Contracts from Java Programs". In: *ASE'15*. IEEE. 2015, pp. 104–114.

[3] Abdullah Alshanqiti, Reiko Heckel, and Timo Kehrer. "Visual contract extractor: a tool for reverse engineering visual contracts using dynamic analysis". In: *ASE'16*. ACM. 2016, pp. 816–821.

[4] Daniel Strüber et al. "Henshin: A usability-focused framework for emf model transformation development". In: *ICGT'17*. Springer. 2017, pp. 196–208.

[5] Timo Kehrer, Abdullah Alshanqiti, and Reiko Heckel. "Automatic inference of rule-based specifications of complex in-place model transformations". In: *ICMT'17*. Springer. 2017, pp. 92–107.