Experimental Specification Mining for Enterprise Applications

Matthias Schur[*]
SAP Research
Darmstadt, Germany
matthias.schur@sap.com

ABSTRACT

Specification mining infers abstractions over a set of program execution traces. Whereas inductive approaches to specification mining rely on a given set of execution traces, experimental approaches systematically generate and execute test cases to infer rich models including uncommon and exceptional behavior. State-of-the-art experimental mining approaches infer low-level models representing the behavior of single classes. This paper proposes an approach for inferring models of built-in processes in enterprise systems based on systematic scenario test generation. The paper motivates the approach, sketches the relevant concepts and challenges, and discusses related work.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications

General Terms

Algorithms, Documentation, Experimentation, Verification

Keywords

Specification Mining, Test Case Generation, Model-based Testing, Enterprise Applications

1. INTRODUCTION

In recent years a multitude of software engineering techniques based on explicit models such as formal verification and model-based testing (MBT), have been developed. These techniques can be used to verify certain properties of complex software systems, but they rely on formal specifications of the systems to verify. However, in software engineering practice, specifications are often informal, outdated or entirely missing, elevating the implementation to the single source of truth. As large enterprise application vendors start to address the mid-market segment with core business functionality that can be extended by numerous partners, reliable system models become critical to ensure operational reliability. In general, the success of model-based software engineering (SE) techniques in industry largely depends on the tradeoff between modeling effort (incl. the effort for keeping the models up-to-date) and the expected benefit. Modeling large industrial software systems is a complex and time-consuming task, hampering broad adoption of model-based SE techniques in industry. Automated techniques actively supporting the modeling process are an essential prerequisite for the success of model-based SE techniques. By utilizing the advances in model-based testing, techniques for automatically inferring models from implementations could significantly decrease the modeling effort and thus make the adoption of additional model-based SE techniques such as formal verification more interesting for industry. Furthermore, inferred models could also be beneficial to documentation, monitoring and change management activities. From an industrial point of view, specification mining also can support the integration of legacy systems that have been adapted and modified over the years and come with outdated documentation.

This paper presents an approach for inferring models of built-in processes in enterprise systems by utilizing MBT methodology. It is organized as follows. *Section 2* discusses the problem to be solved in this research, as well as the underlying research hypothesis. *Section 3* sketches the proposed approach and briefly discusses some challenges. *Section 4* presents the research plan and planned contributions. *Section 5* discusses the state of the art in mining specifications. *Section 6* concludes the paper.

2. RESEARCH QUESTION

In state-of-the-art enterprise systems, most of the functionality is implemented in so called business objects (BOs) such as *Sales Quote* (SQ) and *Sales Order* (SO). BOs are classes with a well defined interface and an internal model describing the state of the object (e.g. *submitted* or *approved*) and the actions that are enabled in each state (e.g. *submit*, *approve* or *reject*) [9]. State-of-the-art approaches to specification mining can infer such models either based on a given set of execution traces or by generating and executing test cases. Especially the latter one, establishing a feedback loop between specification mining and test case generation, is a promising approach, where the quality of models can be improved based on the results of generated test cases and vice versa. (cf. [14])

[*]Advised by Andreas Zeller, Saarland University, Saarbrücken, Germany, zeller@cs.uni-saarland.de

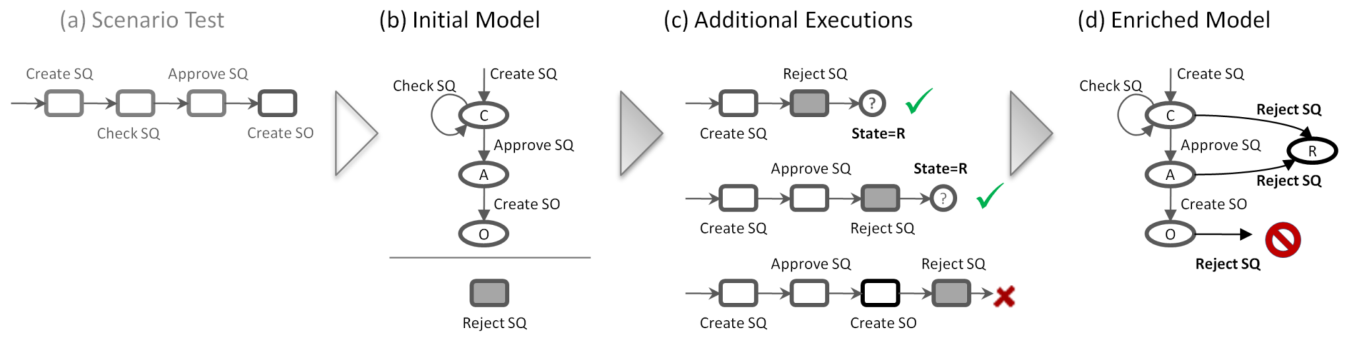

Figure 1: Example. An executable scenario step *Reject SQ* is selected to extend an initial model of the application behavior (b), which can optionally be inferred from a scenario test execution (a). Based on the initial model, scenario tests are generated that execute the selected step in each state of the model; after successful scenario test executions, state information is extracted (c). The initial model is enriched with the observed behavior (d).

Another kind of model that is more common to enterprise systems are business process models. These models represent common business processes implemented by the enterprise system. Figure 1 (a) illustrates such a process: a sales clerk creates a quote for a customer (step 1). The system checks the quote for consistency (step 2). The sales manager approves the quote (step 3). The customer accepts the quote and the sales clerk creates a sales order with reference to the quote (step 3). Despite the importance of process models in enterprise applications, in the majority of enterprise systems, process models are used solely for documentation while there is no direct link to the implementation like in business process execution engines. Compared to models inferred by state-of-the-art specification mining approaches, business process models have a much higher abstraction level; due to the heavy involvement of frameworks for persistency, access and life cycle control, a single process step usually comprises hundreds of method calls on several objects. The proposed research addresses the question *how to automatically enrich models of built-in processes in enterprise systems without relying on a given set of execution traces*. The underlying research hypothesis is that this goal can be achieved by *systematically generating and executing high-level scenario tests* for testing cross-unit functionality. Not relying on a given set of execution traces has several advantages: (1) the approach is also useful for non-productive systems, with no or only a small set of execution traces. (2) The approach is not restricted to a given set of traces. (3) Execution traces are produced in a controlled way. On the other side, this implies a number of challenges that will be discussed in the following section.

3. APPROACH

In enterprise application development, scenario tests are one of the major quality assurance techniques. Software vendors maintain large repositories of scenario tests, usually constructed with the same degree of abstraction as business process models. Scenario tests comprise multiple steps, where each step is an abstract test script that can be equipped with test data and executed on the system under test (SUT). Each step may provide arguments for following steps.

The general idea of the presented approach is to leverage scenario tests to infer models of built-in processes in enterprise systems. In a nutshell, the approach works as follows (see Figure 1). A domain expert selects an executable scenario step e.g. *Reject Sales Quote* (SQ) to extend an initial model of the application behavior (b). Since executable process models are often missing in industrial practice, the initial model can optionally be inferred from a given scenario test (a), which consists of several steps such as creating a sales quote, checking the quote for consistency, approving the quote and creating a *sales order* (SO) with reference to the quote. Therefore the scenario test is executed and after each step the state of the core objects affected by the current step is extracted from the SUT using the tracing functionality provided by modern enterprise systems. States in the initial model represent the state of the main objects the scenario steps operate on such as *SalesQuote(submitted|approved|rejected)* and transitions are labeled with scenario step names such as *Approve SQ*. Based on the initial model, additional scenario tests are generated that execute the selected step in each state of the model; after successful scenario test executions, state information is extracted (c). The observations made from the execution of the generated scenario tests are used to enrich the initial model (d). Note that step (c) and (d) are executed iteratively, i.e. that for each new legal state discovered by the generated scenario tests, further tests are generated that execute the selected step in the new state, until no new states are discovered by the generated tests. If each of the selected scenario steps has been executed in each state of the inferred model or a predefined timeout has been reached, the mining process stops. The inferred model can then be used to compare the implemented system behavior with the intended behavior.

There are a number of challenges that will be briefly discussed in following.

Combinatorial explosion: a naïve approach that generates and executes all combinations of a set S of selected scenario steps has to execute $|S|!$ scenario tests. For the small example in Figure 1 with $|S| = 5$ this means a total of 120 tests comprising of 600 scenario steps to execute. This is only feasible for scenarios with a small number of selected steps and a short average step runtime.

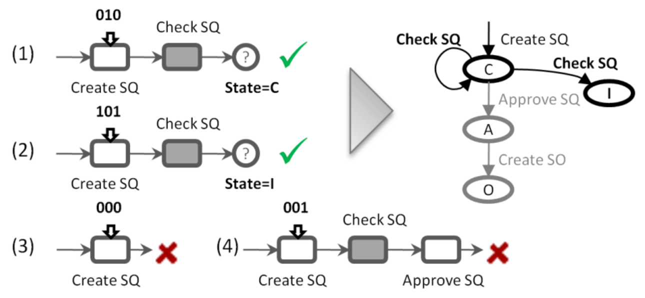

Figure 2: Influence of test data. Executing the same steps may lead to different states (1, 2), or even fail (3, 4), which results in nondeterministic models.

Long running test scripts: unlike existing approaches that leverage test case generation to infer models, this approach generates and executes scenario tests instead of unit tests. A scenario test consists of several steps that typically comprise a multitude of operations. This allows abstracting from single operations, but also increases the runtime of the approach compared to approaches that mine models with transitions representing single method calls.

Test data provisioning: scenario tests comprise multiple steps, i.e. parameterized test cases that have to be provided with concrete arguments to execute them. Figure 2 illustrates the influence of test data on experimental specification mining approaches. Depending on the selected test data for the scenario step parameters, executing the same steps may lead to different states - *consistent* (1) or *inconsistent* (2), or even fail (3). Another, more business related example where test data influences test execution is the approval step in scenario test (4); depending on the order value provided as an argument for the *Create SQ* step, an approval may become unnecessary and the execution of the *Approve SQ* step may even fail. If test data is not fixed and data dependencies are not inferred, this results in nondeterministic models. However, inferring relations over arguments requires executing each scenario test multiple times with different test data. For a scenario with s parameterized steps and n argument lists for each step, testing all data combinations results in n^s executions for a single scenario test.

Parameter mapping: another challenge that is an exacerbated version of the previous one is to find arguments for steps that are not represented in the initial model such as *Reject SQ* in Figure 1 (b), and therefore are missing a parameter mapping; this means it is not clear which arguments are generated by preceding steps (e.g. *SQ Id*), and which arguments are independent from previous steps (e.g. rejection comments). Searching for script usage in existing scenario tests and comparing parameter naming and types could help but may not solve the problem in all cases. Therefore user interaction might be necessary where automated techniques do not provide good results.

Anonymous vs. labeled states: a number of specification mining approaches [3, 1] define states solely by their outgoing transitions, i.e. states are anonymous and provide no information except for their outgoing transitions. Approaches with anonymous states do not need to extract state information from the SUT, but the algorithm for building the FSM and merging states is more complex [2]. For the approach presented in this paper, inferring FSMs with anonymous states increases the number of scenario steps to execute in order to distinguish states, whereas an approach based on explicit state information requires extracting the system state after each scenario test execution.

State abstraction: considering an approach based on explicit state information leads to the question of finding the right level of abstraction for the system state. Due to the enormous state space of business application systems, it is hardly possible to extract and represent the full system state. Apart from that the inferred models would be too detailed to be useful.

4. RESEARCH PLAN

Being at the beginning of my research, I did some first experiments with SAP enterprise systems. I achieved to extract selected state information from the system; the aforementioned long runtime of single scenario steps turned out to be a major challenge.

I plan to proceed by implementing a tool in the domain of enterprise systems. In the first iteration I will exclude the test data aspect by fixing the data. In order to evaluate the proposed approach, I plan to analyze a number of processes in SAP enterprise systems, using the tool and comparing the results with manually created models.

The expected contributions of this research are: (1) a technique to mine high-level models of built-in processes from enterprise systems. (2) A tool implementing the mining technique for SAP enterprise systems. (3) An evaluation of the technique based on experiments with the tool.

5. RELATED WORK

The first approach that infers finite-state machines (FSMs) from process event traces was published by Cook and Wolf [3]; the *Ktail* algorithm, which builds upon the work of Biermann and Feldman [2], mines FSMs with anonymous states and transitions labeled with process events. States reached by process events are defined by the future behavior that can occur from it, i.e. events whose successor events are equal belong to the same equivalence class and therefore lead to the same state in the FSM. Ammons et al. [1] mine nondeterministic finite automata (NFAs) with anonymous states capturing temporal and data dependencies of API call. They use a variation on the classic *k-tails* algorithm [2] to mine probabilistic finite state automata (PFSAs) from execution traces annotated with flow dependencies; each edge in the PFSA is labeled with an interaction and weighted by how often it is used. The PFSA is then transformed into an NFA by removing rarely-used edges, unreachable states and the weights. The *GK-Tail* algorithm by Lorenzoli et al. [10] mines extended FSMs from execution traces that capture the relations between data values and component interactions. The algorithm uses *k-tails* [2] to mine interaction patterns and extends the inferred FSMs with invariants over data values extracted with the *Daikon* tool by Ernst et al. [6].

The *ProM* framework [12] provides an extensible environment for analyzing process event logs and comes with a multitude of plug-ins for process model inference: the *α-algorithm* by van der Aalst et al. [11] mines Petri nets. The *fuzzy miner* [8] infers abstractions over less structured processes by aggregating and abstracting less significant behavior based on several metrics. The *flexible heuristics miner* [13] infers dependency graphs with the most frequent dependencies between events in the log and annotates less frequent dependencies in form of causal nets, making the algorithm robust to noise.

The specification mining approaches [3, 1, 10, 11, 8, 13] can be classified as inductive approaches (cf. [16]), i.e. they build abstractions over a given set of program executions. In contrast, the proposed approach systematically generates and executes scenario tests and therefore does not rely on a given set of execution traces. The idea of building a feedback loop between specification inference and test generation was first published by Xie and Notkin [14]. The *Obstra* tool [15] implements this idea by executing a given test case to infer an initial specification. This specification is used to automatically generate new test cases, which are executed in a subsequent iteration to extend the specification. The *Tautoko* tool by Dallmeier et al. [4] mines typestate automata based on systematic test case generation. Typestates are FSMs with anonymous states and transitions labeled with method names that encode the legal usage of a class under test. The approach builds on their previous work [5] for inferring *object behavior models* (OBMs), which have a lower level of state abstraction than typestates. Given a Java program and a concrete test case, Tautoko infers an initial OBM by executing the test case; the initial OBM is enriched by observing the execution of additional test cases generated by mutating the given test case such that each method in the test case is called in each state of the OBM. The enriched OBM is then transformed into a typestate automaton by replacing each state label, except for *start* and *exception*, with a unique number. The proposed approach as well as [14, 15, 4] control program executions by generating test cases and therefore can be considered as experimental techniques to program analysis (cf. [16]). However, the proposed approach infers models with a higher level of abstraction granularity.

Another field related to the proposed approach is model-based robustness testing [7], which mutates an initial system specification to produce test cases violating the initial specification. While the purpose of robustness testing is checking that certain behavior is inhibited by the system e.g. a high volume sales quote may not be released without manager approval, the proposed approach observes the test case execution and extends the specification accordingly.

6. CONCLUSION

To the best of my knowledge, the proposed approach is the first one that infers models of built-in processes in enterprise systems based on systematic scenario test generation and therefore does not rely on a given set of process execution logs. The approach builds on research in specification mining and model-based testing and will be implemented and evaluated in the domain of enterprise systems.

Acknowledgments. I would like to thank Andreas Zeller, Andreas Roth, Sebastian Wieczorek, Wei Wei and the anonymous reviewers for their valuable comments.

7. REFERENCES

[1] G. Ammons, R. Bodik, and J. R. Larus. Mining Specifications. *ACM Sigplan Notices*, 37(1):4–16, 2002.

[2] A. W. Biermann and J. A. Feldman. On the Synthesis of Finite-State Machines from Samples of Their Behavior. *IEEE Transactions on Computers*, C-21(6):592–597, 1972.

[3] J. E. Cook and A. L. Wolf. Discovering Models of Software Processes from Event-based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, July 1998.

[4] V. Dallmeier, N. Knopp, C. Mallon, S. Hack, and A. Zeller. Generating Test Cases for Specification Mining. In *Proc. of the 19th International Symposium on Software Testing and Analysis*, ISSTA '10, pages 85–96, New York, NY, USA, 2010. ACM.

[5] V. Dallmeier, C. Lindig, A. Wasylkowski, and A. Zeller. Mining Object Behavior with Adabu. In *Proc. of the 2006 International Workshop on Dynamic Systems Analysis*, WODA '06, pages 17–24, New York, NY, USA, 2006. ACM.

[6] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin. Dynamically Discovering Likely Program Invariants to Support Program Evolution. *IEEE Trans. on Software Engineering*, 27(2):99–123, 2001.

[7] J.-C. Fernandez, L. Mounier, and C. Pachon. A Model-based Approach for Robustness Testing. In *TestCom*, pages 333–348, 2005.

[8] C. W. Günther and W. M. P. van der Aalst. Fuzzy Mining - Adaptive Process Simplification Based on Multi-perspective Metrics. In *Proc. of BPM*, pages 328–343, 2007.

[9] S. Kätker and S. Patig. Model-driven Development of Serviceoriented Business Application Systems. In *Wirtschaftsinformatik (1)*, pages 171–180, 2009.

[10] D. Lorenzoli, L. Mariani, and M. Pezzè. Automatic Generation of Software Behavioral Models. In *Proc. of the 30th International Conference on Software Engineering*, ICSE'08, pages 501–510, Leipzig, Germany, 2008.

[11] W. M. P. van der Aalst, T. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142, 2004.

[12] B. F. Van Dongen, A. K. A. De Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. Van Der Aalst. The ProM Framework. *Lecture Notes in Computer Science*, 3536/2005:1105–1116, 2005.

[13] A. J. M. M. Weijters and J. T. S. Ribeiro. Flexible Heuristics Miner. 2011. Accepted for the SSCI2011.

[14] T. Xie and D. Notkin. Mutually Enhancing Test Generation and Specification Inference. In *Proc. 3rd International Workshop on Formal Approaches to Testing of Software*, FATES 03, volume 2931 of *LNCS*, pages 60–69, October 2003.

[15] T. Xie and D. Notkin. Automatic Extraction of Object-oriented Observer Abstractions from Unit-Test Executions. In *Proc. 6th International Conference on Formal Engineering Methods*, ICFEM 2004, pages 290–305, November 2004.

[16] A. Zeller. Program Analysis: A Hierarchy. In *Proc. Workshop on Dynamic Analysis*, May 2003.